Data Management and Analytics for UF Smart Testbed

Anand Rangarajan, Sanjay Ranka

anand@cise.ufl.edu, sranka@ufl.edu

Date: September 2020

Deliverable 7: Final Report

FDOT Contract BDV31-977-77

Disclaimer

The work was supported in part by the Florida Department of Transportation. The opinions, findings, and conclusions expressed in this publication are those of the author(s) and not necessarily those of the Florida Department of Transportation or the U.S. Department of Transportation.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Data Management and Analytics for UF Smart Testbed		5. Report Date September 2020
		6. Performing Organization Code
7. Author(s) Tania Banerjee, Dhruv Mahajan, Xiaohui Huang, Anand Rangarajan, Sanjay Ranka		8. Performing Organization Report No.
9. Performing Organization Name and Address, Department of Computer Information Science and Engineering University of Florida Gainesville, FL 32611		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. BDV31-977-77
12. Sponsoring Agency Name and Address Florida Department of Transportation 605 Suwannee Street, MS 30 Tallahassee, FL 32399		13. Type of Report and Period Covered Final Report 08/15/2017 – 12/15/2020
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

We describe the data collection, management mechanisms, and analysis mechanisms for data collected from the I-STREET testbed, which comprises selected roads on the University of Florida campus and adjoining City of Gainesville roads. The test bed is instrumented with roadside units that deliver a variety of data about traffic. In this project, we provide the requirements for applications using the collected data. We further describe a three-tiered architecture consisting of edge, local servers, and cloud-based components that we use to deploy our applications. Subsequently, we describe each application in detail in terms of the sensors and data sources, the data ingestion methods, and the core applications, enabling us to have a deeper understanding of the data, thereby helping us identify strategies for improvements and optimizations.

17. Key Word	18. Distribution Statement
Big Data Analytics, Cloud Computing, Edge Computing, Transportation Applications	No restrictions.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price.
Unclassified.	Unclassified.	82	

Form DOT F 1700.7 Reproduction of completed page authorized

Acknowledgments

The authors would like to thank the Florida Department of Transportation (FDOT) for providing high resolution datasets and for insightful discussions during the course of the project.

Executive Summary

The I-Street Testbed comprises selected roads on the University of Florida campus and adjoining City of Gainesville roads. The testbed is instrumented with roadside units that deliver a variety of data about traffic. The Trapezium, a key portion of the I-Street infrastructure, was developed as a collaboration between the University of Florida (UF) and its Transportation Institute (UFTI), the Florida Department of Transportation (FDOT), and the City of Gainesville (COG). FDOT has taken a leadership role in the development and deployment of autonomous vehicles, and the development of this testbed is among FDOT's highest priorities to provide further opportunities to industry and researchers to test and deploy advanced transportation technologies. The UF campus and its surroundings host a significant presence of various transportation modes and a diverse population. Hence, the smart testbed will enable the deployment and evaluation of several advanced technologies, including autonomous vehicles, smart devices, and sensors.

In this report, we describe the data collection, management mechanisms, and analysis mechanisms. We also provide the requirements for applications using the collected data. We further describe a three-tiered architecture consisting of edge, local servers, and cloud-based components that we use to deploy our applications. Subsequently, we describe each application in detail in terms of the sensors and data sources, the data ingestion methods, and the core applications, enabling us to have a deeper understanding of the data, thereby helping us identify strategies for improvements and optimizations. Our models leverage machine learning methodologies for data collected from a large number of intersections to derive key spatiotemporal traffic patterns in a city and interactively allow a traffic engineer to focus on critical challenges or improvements be carried out to alleviate them.

Contents

List of Figures

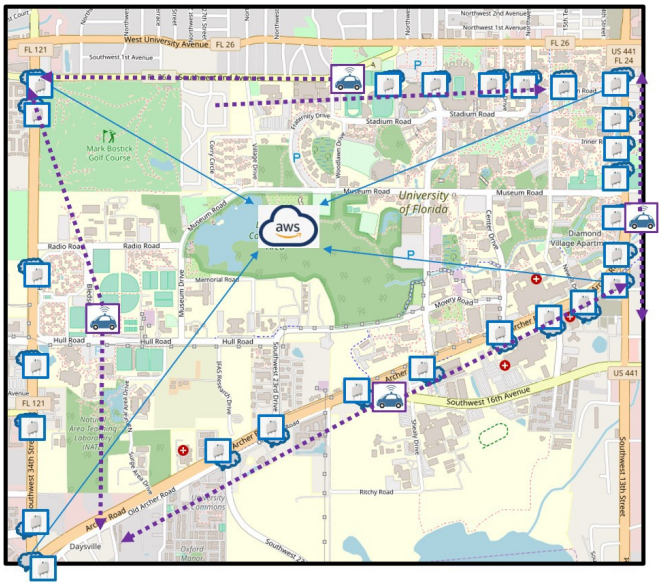
List of Tables

1 Introduction

In this document, we report the findings of the project “Data Management and Analytics for UF I-Street Testbed.” The I-Street Testbed comprises selected roads on the University of Florida campus and adjoining City of Gainesville roads. The testbed is instrumented with roadside units that deliver a variety of data about traffic. The Trapezium (a key portion of I-Street) was developed as a collaboration between the University of Florida (UF) and its Transportation Institute (UFTI), the Florida Department of Transportation (FDOT) and the City of Gainesville (COG) (Figure 1). FDOT has taken a leadership role in the development and deployment of autonomous vehicles; and the development of this testbed is among FDOT’s highest priorities to provide further opportunities to industry and researchers to test and deploy advanced transportation technologies.

UF campus and its environs have a significant presence of various transportation modes and a diverse population. There is heavy pedestrian flow, extensive bicycle facilities, transit buses, scooters and mopeds, and a variety of roadways, with different lane counts and allowed speeds. Hence, the smart testbed will enable the deployment and evaluation of several advanced technologies, including autonomous vehicles, smart devices, and sensors, as well as to develop novel applications for their use.

In this report, we describe the various devices installed and data collection mechanisms and outlined the requirements for applications using the collected data. We further describe a three-tiered architecture consisting of edge, local servers, and cloud-based components that we use to deploy our applications. Subsequently, we describe each application in detail in terms of the sensors and data sources, the data ingestion methods, and the core applications, enabling us to have a deeper understanding of the data, thereby helping us identify strategies for improvements and optimizations. Our models leverage machine learning methodologies for data collected from a large number of intersections to derive key spatiotemporal traffic patterns in a city and interactively allow a traffic engineer to focus on critical challenges or improvements be carried out to alleviate them. Additionally, we provide an analysis of traffic interruptions by observing changes in traffic at detectors, approaches, and intersections.



The rest of the document is organized as follows:

1. Chapter 2 is a detailed summary of all sensors and data sources that are installed in the urban road networks today. It also covers ways to convert this raw data into performance measures and the relevant performance measures used in the literature and field. The section details previous work that applied data analytics techniques at these measures.

2. Chapter 3 provides a description of the architecture we have developed based on edge components, local servers, and cloud servers. This section also presents a detailed description of how these different components are used to implement our ATSPM data analysis pipeline, video analysis pipeline, Siemens RSU data analysis pipeline, and Bluetoad data collection pipeline.

3. Chapter 4 summarizes the main issues with the high-resolution logs and describes essential data cleaning techniques. It also represents a model to derive or infer crucial metadata (detector to phase mappings) when they are not present. These mappings are a prerequisite for calculating most performance measures. This section also contains a description of the intersection ranking, clustering, and change detection techniques developed as part of this work and also details ways to use these techniques to build a decision support system.

4. Chapter 5 presents our video analysis algorithms. In particular, we have proposed a two-stream convolutional networks architecture that performs real-time detection, tracking, and near-accident detection for vehicles in traffic video data. The two-stream convolutional network comprises a spatial stream network and a temporal stream network. The spatial stream network detects individual vehicles and likely near-accident regions at the single-frame level by capturing appearance features with a state-of-the-art object detection method. The temporal stream network leverages motion features of detected candidates to perform multiple object tracking and generates individual trajectories of each tracked target. We detect near-accidents by incorporating appearance features and motion features to compute probabilities of near-accident candidate regions.

5. Chapter 6 provides our work in conjunction with Siemens Mobility, Inc., and the City of Gainesville to develop software for collecting signal phasing and timing (SPAT) data from the intersections on a corridor.

We conclude in Chapter 7. Appendix A provides details on the software used for our development. Appendix B provides a comparison of the cloud service providers at the time of this work.

2 Transportation Datasets and Requirements

Big Data for transportation applications is increasing at a rapid rate. The amount of data collected has to support multiple modalities with variable accuracy and periodicity. The requirements for these applications need greater data processing agility, multiple analytical tools, real-time analytics, and temporally and spatially aligned and consistent views of the data. In this section, we describe the different data sources of interest and the processing and storage requirements of the target applications. This section is broadly divided into two parts. Section 2.1 describes the available data sources, while Section 2.2 presents the requirements for the applications that would use these datasets.

2.1 Data Sources

This section documents the different types of traffic controllers, the sensors that gather data, their accuracy, rate of data collection, potential missing data, and the information that can be gathered using the current state of the art infrastructure.

2.1.1 Fixed-point Systems and Sensors

Controllers and invasive detectors. Traditionally, all sensors used for traffic observations had to be installed at a fixed location, and these are also called fixed point sensors. Sensors such as loop detectors that are widely prevalent in many major roads in cities throughout the world are a prime example of these. The major generations of controllers in the field today are based on NEMA standards and communicate using the National Transportation Communications for ITS (Intelligent Transportation Systems) Protocol (NTCIP) (National Electrical Manufacturers Association (NEMA), 2005). These include:

- TS2/2070 Controllers (Versions 50.x, 60.x, 61.x, and 65.x)
- TS2 Controllers (Version 61.x)
- TS2 Controllers (Version 65.x)
- Advanced Transportation Controllers (ATC Version 76.x)

Out of these, the ATC controllers are the most widely used today. ATC-complaint controllers have the following noteworthy capabilities. The control unit of the ATC controller maintains the following timing accuracy for all the reported split timing intervals. Any interval timed does "not deviate by more than 100 milliseconds from its set value at a power source frequency of 60 hertz." The oscillator has a timing accuracy of ±0.005% over the entire NEMA-specified temperature & humidity range (specified above) as compared to the Universal Coordinated Time Standard (Institute of Transportation Engineers, 2018). This enables highly acurate data logging. Induction loop detectors and other kind of sensors (magnetometers) are often paired with actuated signal controllers (Kotwal et al., 2013). These detection technologies can be used in determining phase cycle lengths and sequence of green activation. Inductive loop detectors and magnetometers make up the majority of traditional traffic detection sensors that are already deployed at scale. Both kinds of sensors

enable the analysis of key factors, including vehicular speed, volume, and presence. As these sensors are the conventional method for vehicular detection, industry professionals are often most comfortable working with induction-based loops, and as a consequence, they are the primary means of detection. A major advantage of such systems is that they are mostly insensitive to weather and environmental factors and, if they are properly installed and maintained, are the most accurate means of traffic detection. (Mimbela and Klein, 2007) However, installing and maintaining inductive sensors has in recent times become much more expensive compared to more modern video and LIDAR detection. This is mainly because inductive sensors require cutting and patching the road for installation or maintenance (Walton et al., 2009; Middleton et al., 2009).

Bluetooth and Wi-Fi based Roadside Units (RSUs). Bluetooth- or Wi-Fi-based sensors are unique because they can maintain and track the profile or identity of a vehicle over extended periods of time and across the instrumented network. These Bluetooth- or Wi-Fi-based sensors can provide arterial travel time estimates by tracking specific vehicles travelling on specific paths (Allström et al., 2017). The major limitation of this approach is that the user needs to manually specify the paths of interest. There are mainly two kinds of Bluetooth detectors on roadsides or at signal intersections:

- Wired, Ethernet-based boxes that only track discoverable Bluetooth devices
- Boxes tracking both discoverable & non-discoverable Bluetooth devices by using passive scanning and that can be deployed in wireless setup mode by using the cellular network.

Any kind of passive Bluetooth scanning is limited by user preferences on the devices being tracked. Passive scanning technology cannot track devices that do not broadcast any Bluetooth LE (Low Energy) or Wi-Fi information, specifically Media Access Control address (MAC) address.

Moreover, it is possible to enable software MAC address randomization for both the Wi-Fi and Bluetooth MAC address on a device. End users can do this to specifically block MAC address-based tracking. This randomization of MAC address is also part of the Bluetooth low energy (LE) security standard. Due to the concerns surrounding Bluetooth tracking devices, a Bluetooth LE privacy standard (Woolley, 2015) was released by the Bluetooth Special Interest Group (SIG), which is the non-profit organization supporting and improving the Bluetooth standard.

Current systems (TrafficCast) report 10% to 20% as their peak match rates for their latest RSUs. This number is completely dependent on the OEMs that make the entertainment units in the car or the mobile device makers. Often, they make the choice about implementing the Bluetooth LE privacy specification on their devices. Moreover, it is conceivable that regulation may make such tracking a lot more difficult to accomplish in the future. Hence, the results obtained by use of MAC address based-tracking are completely dependent on implementation (or a lack of) of MAC randomization in the firmware of the devices being tracked.

2.1.2 Probe and GPS Data

An alternative source of traffic data is probe-based systems. These can be contrasted with fixed point sensors and data gathering systems. The most prevalent examples of probe-based systems are navigation systems on cars and smart devices, fleet management systems, and black-box systems installed by insurance providers. Travel time and speed estimates from GPS devices have been extensively used for real-time traffic state estimation (Tao et al., 2012). An important factor to consider with all probe or GPS-based systems is that the client type may impact the quality or accuracy of data that can be collected and may introduce biases that need to be corrected for. An example of this can be a local or professional driver avoiding the slowest parts of a network, and hence, the travel time between his or her origin and destination (via the main arterial or the most obvious route) might be underestimated. Some probe systems, for example, local government vehicles, may be available only at certain time of the day and other systems, like taxis or ride-share services, may use dedicated lanes or HOV lanes, and hence, the reported travel times can be significantly lower than what normal vehicles experience. Another possible issue is that certain systems like those installed on heavy vehicle fleets and buses might not report accurate travel times because these kinds of vehicles are often restricted in terms of mobility and speed limits. Taking into account all the above, it is crucial that the system type be considered when estimating the travel times using probe data. It is generally possible to normalize most such data and get a useful estimate of travel time. The characteristics of data that are collected can also be affected by the device type. As an example, consider the case of GPS data collected from navigation devices located inside an automobile vs. the data collected from cellular phones. Taking this example further, some devices will have poorer GPS receivers or internal clocks, and hence, the location reported from these devices will have lower accuracy, and this can impact the results of any travel time estimation algorithm, especially when trying to estimate the travel time over relatively short distances (Rose, 2006; Ferman et al., 2005).

Coverage is the frequency and number of nodes (and state measurements) that are available in a dataset based on probe vehicles. This is one aspect of the of these datasets that is very difficult to control or influence. For certain applications and decision support systems this can be very problematic. In sharp contrast, fixed sensors which are placed based on the intuition and understanding of local traffic managers can generate constant, high quality measurements for the applications and places under consideration.

The primary measurements that can be gathered from most automobile navigation systems are the current speed and travel time estimates. Some automobiles may not report speed measurements, which vastly reduces the data about the road network state and information that can deduced from each node in the system. If the central server is capable of tracking and storing an identifier of the probes as they report location, it is still possible to estimate travel times but with a much lower fidelity. However, if this is not possible, the only useful information is via density estimations based on current probe locations. This can be extremely challenging to implement with an accuracy, such that the information is usable, with the penetration rates of most commercial systems. The strategy used to sample data from the probes is also critically important when trying to estimate travel time and road traffic. Using longer sampling times can cause two separate problems. Firstly, in any such system, there is an inherent delay which is equal to the sampling time;

this becomes exaggerated and reduces accuracy of the travel time estimate. Next, there is the problem of path interference with long sampling times. Dense urban road networks may have a collection of route choices for point-to-point travel. Path interference becomes a major issue in such cases, especially when paired with longer sampling times. Rahmani and Koutsopoulos (2013) and Hunter et al. (2014) explore the problems of path inference and map matching (respectively) for devices with slow sampling rates in more detail and propose possible solutions. Normally, the ideal location sampling rate for GPS probes is between 30 seconds and two minutes.

To summarize, four main properties determine the performance of probe-based systems:

- Penetration level or coverage: The number of vehicles that report information (probes) vs. the total number of vehicles in the network
- Rate of sampling: The frequency at which measurements are recorded by the probe vehicle and how often these measurements sent to the central system
- Type of reporting device or system: It is important to take into account the types of sensors that the GPS probes are equipped with, the types of vehicles or devices these data are being collected from, and what data collection is possible using the communication protocols
- Measurement accuracy: The accuracy that the positioning devices on automobiles or the probe systems can achieve.

2.1.3 Video Data

Videos of traffic for monitoring and surveillance applications can be recorded by cameras mounted on masts at traffic intersections. These devices are generally connected and powered by Power-over-Ethernet (PoE) or optical fiber cables that run underground and connect to traffic cabinets. Cameras allow streaming of video feeds with the Real-Time Streaming Protocol (RTSP) over the Web. The RTSP feed can be encoded and compressed into the MP4 format, or it can be recorded frame-by-frame using JPG or MPEG compression for fine-grained analysis. Each camera has its own system clock that must be synchronized with an Internet time server; if this is not possible, the video data can be timestamped by the system which stores the data in real-time as it comes in. Depending on the method of compression used and the resolution, the amount of storage needed for video data can vary widely. At 20-30 frames per second (fps), high-definition video from a single camera can require over 2 Gigabytes (GB) of storage per hour. To minimize the amount of storage needed, one can capture fewer frames per second and reduce the resolution (Jodoin et al., 2016).

To automatically obtain information about traffic volume or speeds, post-processing must be applied to the raw video data. Using image processing techniques, vehicles can be detected and counted in this manner. To obtain information about the speed and position of the vehicle on the road from image data, the traffic camera must be calibrated to allow transformation of vehicle detections from pixel coordinates to world-frame coordinates. Quantifying the accuracy of volume and speed data extracted from image processing

techniques on traffic video data is difficult, as it is highly dependent on the algorithm and implementation, as well as various environmental factors. Some of the main challenges faced by traffic video are the sensitivity to variations in lighting and weather and the occlusion of vehicles (Roy et al., 2011). Improving image processing techniques for traffic surveillance is an ongoing area of research in the computer vision and intelligent transportation systems (ITS) communities for these reasons.

2.1.4 Radar and Lidar

The main type of radar used for ITS applications is frequency-modulated continuous-wave (FMCW) radar. Continuous-wave radar uses Doppler to prevent interference from large stationary objects and slow-moving clutter. A typical commercial traffic radar sensor is robust to adverse weather conditions, is unaffected by sunlight, and can operate in widely varying temperatures. These radars are forward-fire and can be mounted on mast arms facing oncoming traffic. The sensor reports a list of all tracked objects. Each track in the track list contains the x position, y position, x component of velocity, and y component of velocity of the vehicle. LIDAR sensors have recently become feasible for real-time traffic monitoring with the development of compact and cost-effective 3D sensors. The distance and speed measurements are highly accurate. The number of points returned by each object in the field of view decreases with distance from the sensor, which can make interpreting these points difficult. Due to the high volume of data returned by the sensor, processing in the form of object segmentation is generally carried out to identify objects of interest, such as all moving objects, that should be stored. The main advantage of lidar over radar and video sensors is itsr accuracy and robustness to variations in illumination and interference (Guerrero-Ibañez et al., 2018; Fedele et al., 2017).

2.1.5 Vehicle-to-Infrastructure Communication

Traffic data can be collected from vehicles that transmit safety messages using vehicle-to-infrastructure (V2I) communication. The most prominent protocol for one-way and two-way short to medium range wireless communication is Dedicated Short Range Communication (DSRC). DSRC operates on the 5.9 GHz band allocated by the FCC for vehicle safety applications. Vehicles equipped with a DSRC transmitter (On-Board Unit (OBU)) can send messages to entities with a receiver (either another OBU, or a roadside unit (RSU)). The most easy outcome of V2I communication is in safety-related applications. This is explored in detail in Nwizege et al.(2014) and Xu et al. (2004).

2.1.6 ATMS and Other Data AggregationSystems

Current and historical signal timing data and all the associated metadata for the controllers are usually available through ATMS systems. The data available through the ATMS system can be classified into the signal timing data which are available in the form of split reports and volume-occupancy data, which can be exported as volume reports. Based on this data, ATMS can also create various kinds of performance reports (these are explored in Section 4). All versions of the controllers report the signal timing data. However, the volume data and Purdue reports are only generated by controllers which are version 76.x. For ATC or Version 76.x controllers, a single signal controller generates about 2 GB of data in a one-year period. These data are generated as one row every decisecond, or 10 rows of data every second. Older controllers generate one

row of the split report every second and do not generate the rest of the data. However, ATMS does not directly provide access to this high-resolution data and only 15-minute bins are available for export through the reporting tools. Direct access to the controllers may be required to record all the data in real time.

2.2 Application Requirements

The applications that use the datasets that are described in the previous sections have the following high level requirements:

1. Scalable for data diversity and device diversity: The number and type of sensors are growing exponentially. Along with this, there is a growing diversity in structure and scale of data. Generally, each of these datasets can be represented as time series with different periods between multiple samples. Managing scalability and extensibility are necessary requirements as application developers recognize new opportunities from ever-increasing sensors, devices, and vehicles.

2. Distributed and decentralized systems: Underlying sensors, devices, applications are expected to evolve over time. The underlying data storage systems and applications should be able to handle this evolution so that subsystems can function seamlessly. The speed and frequency with which data will need to be accessed by different applications may change frequently. The database and storage system should support offline historical analyses as well as real-time analyses. Thus, there is a need to dynamically manage the optimal structure of associated databases on an ongoing basis.

3. Intelligent and dynamic analytics: cycle timing, pedestrian and bus signal phases, and sensor management depend on models of the queuing information, turn count values, pedestrian count models, distance of pedestrians to the signal, and other information. These models vary both spatially (intersection) and temporally (day of the week, hour of the day). The performance of all these algorithms depends on the accuracy of these models and, in many cases, their size. Analytics methods need to cluster the time intervals and intersections in an spatiotemporal hierarchy (or multiple such hierarchies) such that the lowest levels are more similar to each other and are aggregated at higher levels.

4. Real-time requirements: Many applications have to be executed in real time for them to have a practical use and will require real-time data reduction and fusion. Data reduction and fusion techniques match, aggregate, and consolidate several heterogeneous datasets into representations that can then be used for data mining and machine learning. This is an important step in using inputs from multiple data streams coming in real time. For example, creating a precise representation of location of an autonomous driving vehicle requires fusing information from a variety of sensors: accelerometers, magnetometers, cameras, GPS chips, and wheel movement sensors. After fusion, only a small fraction of data (vehicle location, size, direction, and speed) may need to be stored. Although data fusion techniques for structured data are well advanced, high level data fusion tasks merging multiple unstructured data remains difficult because different data sources may have different semantics and inconsistencies in resolution. This also involves application of semantics technologies, recognizing common vocabularies or linked data, as well as overcoming the heterogeneity of the data and providing

a unified view.

5. Security and privacy: The vast data collection and storage for transportation applications is also accompanied by increasing concerns related to privacy due to personally identifiable nature of mobility-related data. This is further accentuated by the fact that some of this information is often collected without informed consent of the person. In many cases, the data being collected many seem innocuous and anonymous. For example, unencrypted transmissions from automobile monitoring systems can be combined with other datasets to link vehicles to locations and trajectories and then to individuals. Location-based data are vulnerable to breaches because of their geospatial content. Even with concerns, these data can significantly improve services to individuals and contribute to traffic planning, safety, and operations. For example, this information can be used to send emergency services automatically in case of a crash or provide real-time route planning in heavy traffic areas.

3 Cloud and Edge Computing Architecture

Transportation data is heterogeneous, widely distributed, highly temporal, and event-based. The datasets may be structured or unstructured (ranging from Twitter feeds to video streams). The value of cloud computing enables seamless interoperability between the heterogeneous datasets and provides the computing power necessary for bulk data processing. Specifically, the variety of sensors deployed throughout the transportation network today creates a need for (a) ubiquitous data ingestion and storage, (b) seamless data processing, and (c) the ability to handle different kinds of data representations and their attendant visualizations. Given the variety of datasets, we need a framework to identify the representation, storage, management, and processing areas of cloud datasets. In Appendix B, we summarize the current cloud data storage methods, focusing on cloud platforms that support the underlying pipelines to support the heterogeneous sensors present in the transportation infrastructure of today.

Edge or fog computing is centered around the concept of adding a layer of elements between the so-called "IoT" devices and servers on the "cloud." The goal of the in-between layer is to improve performance and reduce costs (in an open and interoperable way). This computing model is also called the Cloud, Edge, and Beneath (CEB) model. The following are some of the requirements of the target transportation applications:

1. Data sources and device management: It is vital that the architecture can integrate all devices already deployed in the field. Data ingestion can be done by directly connecting these devices to the cloud or by introducing an edge layer (as discussed below). The cloud service provider is required to support receiving data from live data streams. Unfortunately, not all devices can be directly connected to the cloud. In these cases, the edge layer acts as an intermediary.

2. Storage: Given that the system is expected to handle multiple tabular and video data streams, it needs to have different storage mechanisms for (a) tabular storage for sensor data and (b) file storage with smart archival for video data.

3. Data Fusion: Several transportation applications require spatial and temporal fusion of multiple data streams. For example, deriving whether a car crossed an intersection during a red traffic light requires fusion of video streams with signal timing.

4. Compute Engines: Data analytics and machine learning algorithms will be implemented as part of the software stack. To ensure maximum flexibility and efficiency and to make the code platform-independent, a custom stack running on self-managed VMs on the cloud is required.

5. Streaming services on the devices: This is a critical aspect of the overall system architecture. A well-designed streaming service enables the developers to work with live streams of data from various sources and perform basic analytics on the fly on the devices. The basic analytics will include aggregation operations before storing tabular data coming from sensors and handling metadata from video streams, and they will be a crucial part of enabling operator alerts

6. Visualization: This is an alternate way to present the data using a pictorial or graphical format. It will help us identify patterns and perform analytics, which has innumerable benefits such as identifying areas that need attention or improvement and making decisions about where new infrastructure and connectivity are required.

In this section, we detail the hardware infrastructure and software platform used for performing data analytics on ground sensor data, video data and other datasets. We used a combination of locally hosted software and services from leading cloud providers (AWS). We implemented a local solution, a cloud-based solution, and a hybrid approach (Edge Computing). We describe our cloud and edge setups, along with some cost comparisons. The data we used for this section was provided by FDOT District 5 (batch data) and video and ATSPM data from the City of Gainesville.

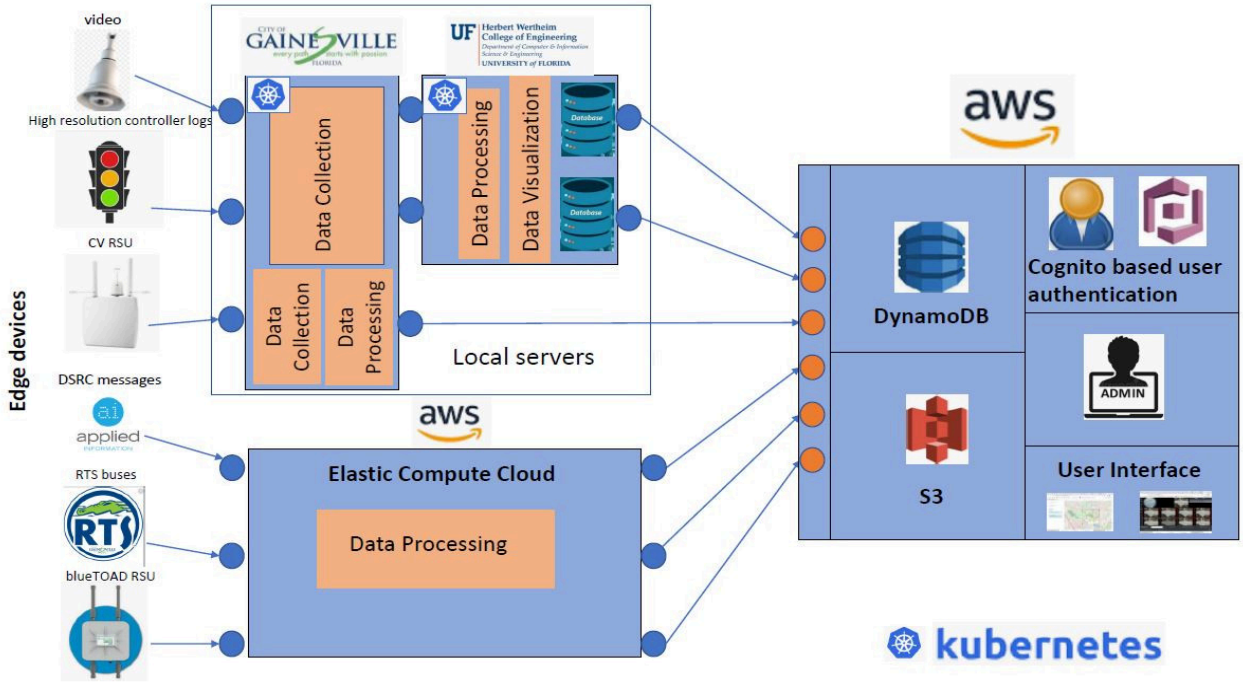


Figure 2: *Overall architecture diagram including edge devices, local servers, and cloud components.*

Figure 2 shows the overall architecture that we have used, and it encompasses the following components:

1. Edge servers: These devices collect information regarding pedestrians, vehicles, etc. and connect to a network or directly communicate with each other. For each edge device, metadata information is generally available. Examples of this information include: Device identifier, make, model, serial number, location information, and state information. There are several advantages of having an edge server. First, it significantly decreases the volume of data that must be moved and hence reduces the network traffic and also the distance the data must travel. The overall result is reduced transmission costs and latency. This means that the overall volume of data that needs to be moved to the cloud is significantly lowered. Second, it allows total control over the devices and sensors and also enables connecting already deployed devices to the cloud through an edge layer. Finally, it makes data more secure as data can be both compressed and encrypted before they are uploaded to the cloud.

2. Local Servers: The streams of information collected from edge servers are sent to locally operated servers. Most of the processing for real-time applications will be conducted on the local servers. The local servers will have much of the functionality of the cloud (as described in the section below). However, a local server may provide a shorter response time and lower latency. In our context, this can also be considered as a super-gateway. In particular, real-time analytics are performed on these servers. These servers are also responsible for sending the information for archival and other processing to the cloud.

3. Cloud: A cloud environment provides an efficient, scalable, and cost-effective way of storing, processing, and analyzing large amounts of data (big data). The following challenges are addressed:

    a. Data Ingestion: Data from local servers or gateways will be ingested in the cloud environment.

    b. Publisher/subscriber: Different components of applications will subscribe to a subset of streams or channels. This subsystem allows for effective matching by creating topics corresponding to different streams. It should also act as an impedance matcher by allowing spikes in information for different time periods by suitable buffering.

    c. Processing pipelines: Extensive format conversion, aggregating and summarizing data, enriching data from third-party source scan be performed.

An application involves a subset of the above components and requires differential amounts of processing at different levels based on latency requirements and storage and computation constraints. On one extreme, we expect edge-level computations to be low latency but using a few local data streams. On the other extreme, cloud-level computations are high latency but have a global view and can process big data. The local servers are in the middle for latency and processing requirements. Local servers will have high bandwidth connections to the cloud and to the edge servers. Having an edge+local servers+cloud architecture greatly reduces bandwidth requirements and potentially lowers costs. Local autonomy leads to reduced dependence on third parties and greater availability and better support for real-time functionality

because of a shorter decision loop. It becomes easy to communicate with or integrate multiple cloud providers in one solution. It speeds up the required time to adopt newer sensors with much higher data rates and leads to a reduction in complexity of management of sensors (resetting, managing drift, etc.).

In the following, we will describe the pipelines we have developed as part of this project and how they utilize the edge-cloud infrastructure.

1. ATSPM pipeline: Data ingestion for this pipeline happens as we connect directly with the controllers at the traffic intersections via FTP once daily and download the high-resolution controller logs. The logs contain information about detector activation and signal changes. These logs are further stored in AWS S3, in a bucket named intersection-atspm.

2. Video pipeline: Data ingestion for this pipeline happens as we connect to the Gridsmart fisheye cameras at the traffic intersections using FFmpeg and store the videos in the MP4 file format. These video files are then transferred to the UF CISE network for video analysis and creation of trajectories. Once done, the subsequent stages of the pipeline are triggered. The trajectories are smoothed, fused with signaling data, clustered, and analyzed for determining speed, gap, response time, and anomalies in shape and timing. The video pipeline stores trajectory coordinate data, trajectory properties data, and trajectory statistics data in the following AWS S3 buckets: intersection tracks, intersection track properties, and intersection statistics, respectively.

3. Siemens RSU pipeline: Siemens roadside units (RSUs) must be enabled to forward a message stream to the client via the XFER interface. Once enabled, our Java message receiver uses WebSocket to send subscriptions to and receive message waves from the RSU. The receiver will read a script file with a message sent on each line and custom control command. In the DSRC message collector, we send the subscription command and use the "!wait" command to let the WebSocket stay connected to receive the message wave from the RSU. The Java receiver is forked by a Python program whenever there's no ongoing connection to the server. After a message is received, the message is written into the standard output, and this message is forwarded to a Python interpreter using the Python subprocess PIPE. Besides, the control message is also written to the stdout for the Python interpreter to check whether the connection is alive or not. Upon receiving the messages, the Python interpreter first extracts the byte-encoded XML message from the raw message wave and decodes it. The syntax and semantics of the XML message may be found in standard SAE J2735. After decoding the message, these are written to AWS RDS and AWS S3 bucket uf.dsrc.bucket for storage.

4. Bluetoad pipeline: We connect to bluetoad.trafficcast.com to set up reporting schedules at a specific frequency. These reports containing information about speed and travel times of different routes are emailed to an address specified in the schedule. Once received, the data is cleaned to remove rows that have no information, the date and time columns are merged, and the data are stored in AWS S3 bucket Bluetoad data.

4 High Resolution Ground Sensor Data

In the previous sections, we explored the efforts towards data collection and data logging. In this section, we explore the motivation for transforming this vast amount of data into metrics. Any set of performance measures offers a means to evaluate a system and at the same time manage and potentially improve its operations. By doing them, the system achieves its objectives. In the context of traffic signals, performance measures have many roles. They help improve the city's or the operator's situational awareness and give them a means to measure and quantify the impact of any action they may take in the field. They also help the operating agency find problems faster, optimize mobility, and improve safety.

Traffic signal systems have been used on streets and highways for about 100 years (FHWA, 2008), but historically these have been difficult to automate across a large system inventory. It was necessary to collect data manually or to develop custom-made data collection systems. A number of early systems were developed for collecting detailed data and extracting useful performance information from them (Ehrlich et al., 1994). The effort needed to develop such systems – particularly the lack of any standard set of data definitions – probably limited these developments. In the absence of robust, widespread data collection, agencies often evaluated their performance in terms of the amount of investment they were making in their systems, such as the number of traffic signals retimed within a year or the amount of money spent on system upgrades. In turn, the lack of good information from the field may also limit traffic signal practitioners and the traffic control more generally from being able to ask for more resources. There is certainly a need for improved performance measurement practices in traffic signal systems. From 2007 through 2012, the National Transportation Operations Coalition (NTOC) conducted a self-assessment called the Traffic Signal Report Card where voluntary participants were scored on a variety of categories. During each survey, the area of traffic monitoring received a score of "F" on a national scale (Denney et al., 2012).

To measure performance of a signalized intersection in a meaningful manner or effect substantial performance improvements, it is critical to have vehicle detection sensors deployed. This can be the traditional inductive loop detectors or the newer video- or radar-based detection methods in Section 2.1. Another prerequisite for computing most performance measures in a semi-automated or fully automated manner is to have high fidelity (or resolution) data logging solutions deployed for all signal controllers. However, as detection technologies continue to improve, it is becoming less difficult to add new detection zones, as in the case of minimally "invasive" systems such as video or infrared detection or wireless magnetometers (Yang and Pun-Cheng, 2018).

A few developments have taken place in the past few years that have enabled automation to be deployed in a saleable manner. One of these is the emergence of high-resolution data (Day and Bullock, 2010; Day et al., 2014). This type of dataset comprises a listing of timestamped events as they occur in a signal controller, such as phase and detector state changes. These data have actually been in existence since the very beginning of traffic signal operation because the electrical or logical states have always existed in some form

or another within the controller, but they were not logged.

For correctly interpreting data gathered at traffic intersections and computing performance measures at intersections, a certain secondary set of data or information is needed. The most critical requirement for interpreting the high-resolution signal event data logged in a controller is the detector mapping information. Because any detector event must be understood as a vehicle arriving at some right-of-way and distance at the intersection and any phase event must be understood as an interval of time elapsed for some movement, the detector mapping process is the critical step in guiding how signal output states and vehicle arrivals are to be interpreted relative to each other. Without the provision of such information, it would be impossible to generate most of the signal performance measures from the detector and phase data independently (Day et al., 2016).

Other things that might be needed are the timing sheet and location of the signal. The timing sheet includes the manually programmed timing plan which is unique to each signal. This is specified by the cycle plan column of the split report.

4.1 Key Performance Measures and ATSPM

Most of the early attempts at developing standardized performance measures were limited by the availability of reliable data with a high enough granularity and coverage to enable meaningful inference at any scale bigger an few intersections or corridors. Remias et al. (2013) compared how vehicle-based probe data sources can be used to identify appropriate adaptive signal control objectives and to assess the performance of these adaptive systems. However, for this to be done at scale, the probe data need to be available at a high granularity and the probes need to be installed in many cars to provide the required coverage. Other work (Sharifi et al., 2013; Hu et al., 2016) also examined outsourced vehicle probe data with the aim of assessing the quality of and accuracy of travel time estimates on signalized roadways for supporting real-time operations as well as computation of various performance measures. Subsequent to this work, there were studies (Day et al., 2015) that explored the use of high resolution detector data for computation of measures that enabled not just performance but also optimization, calibration and maintenance-based objectives. Argote-Cabanero et al., (2015) explored the use of connected vehicles (CVs) and improving the estimation of measures of effectiveness (MOEs) for near-real-time traffic operations and management. However, as with any new paradigm, CV adoption rates will increase gradually. Hence, this study also explores the minimum CV penetration rates that would guarantee reasonable MOE estimates on signalized Intersections. Based on the extensive research into traffic signal performance measures, the Traffic Signal Timing Manual (FHWA, 2008) and other studies (Day et al., 2015) contain a comprehensive guide of the most widely used performance measures and the operational objectives they achieve.

4.2 Related Research

Evaluating the performance of traffic signal systems is important for identifying any problems and addressing them, as well as for assessing and planning enhancements to these systems. Traffic engineers often face the challenge of quantifying the performance of signalized intersections and effectively troubleshooting day-to-day operational issues. In many cases, problems that are particularly difficult to troubleshoot are the ones that reoccur only at a certain time of day or because of exclusive traffic patterns. The portable toolbox for signal operations (Sunkari et al., 2012) is one of the first attempts to reduce the manual intervention needed to detect such problems. Other work (Day et al., 2016) centers around exploring how traffic signal performance measures can be integrated into the day-to-day operations of agencies responsible for maintaining signalized intersections. Under the current workflow, Purdue coordination diagrams (PCDs), arrivals on green vs. red and other such measures (Gettman et al., 2013; Day et al., 2014) explored in Section 4 aim to give practitioners a precise idea of the quality of traffic progression in a corridor. However a practitioner must manually generate and analyze these diagrams for each direction of movement at every intersection to understand and analyze signal performance. There have been past attempts to address these shortcomings.

Specifically, data analytics techniques have been previously applied to traffic flows, and we now present the relevant application areas. Li et al. (2017) present a heuristic based on system-wide split failure identification and evaluation. By using this heuristic, they demonstrated performance improvements for specific corridors. Freije et al. (2014) demonstrate the use of graphical performance measures based on detector occupancy ratios to verify reported split failures and other shortcomings in signal timing that are often reported to traffic engineers by the public. Wemegah et al. (2017) present techniques for management of big data for analyzing traffic volumes and congestion, addressing all the steps in the analytics pipeline, namely data acquisition, data storage, data cleaning, data analysis, and visualization. Amini et al. (2017) describe an architecture for real-time traffic control. In the past, there have been some attempts to apply machine learning techniques for predicting traffic flows and, thereby, traffic congestion. Horvitz et al. (2005) presents a probabilistic traffic forecasting system using Bayesian structure search. Huang et al. (2018) propose a set of new, derived MOEs that are designed to measure health, demand, and control problems in signalized intersections. The newly proposed MOEs are based on approach volume and platooning data derived from ATSPMs (UDOT, 2017).

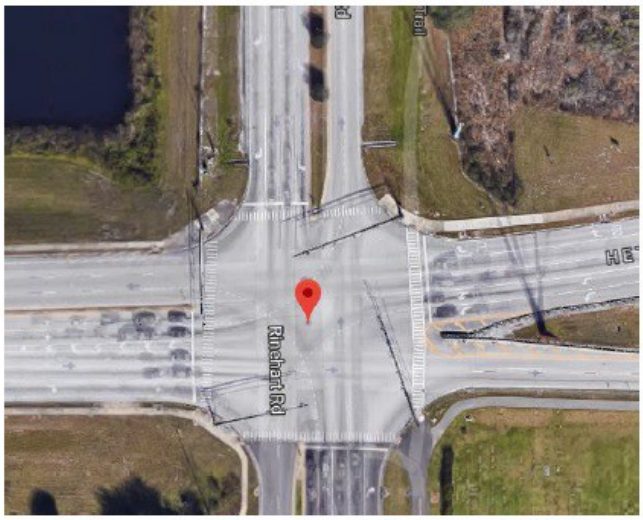


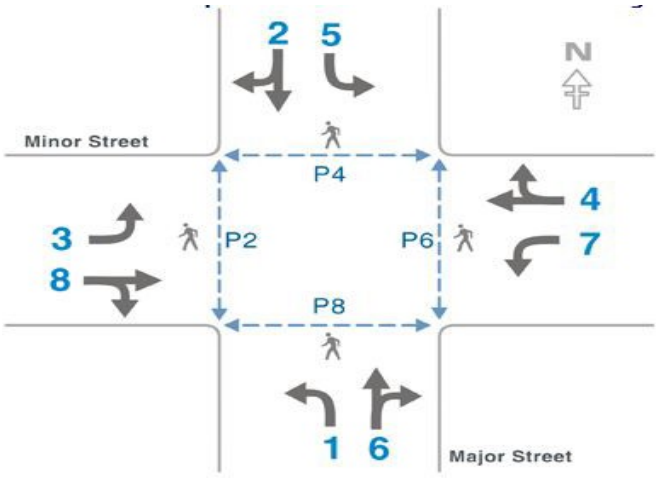
Figure 3: *Example of a four-way Intersection.*

To summarize, Section 2.1 aims to summarize all the different sources of raw data collection in the urban road network with a particular focus on raw data that can be transformed and used to quantify the performance of signalized intersections in the road network. The next Section (4.3) covers the research that

has gone into converting the data (that is generated and gathered via the sensors) into a set of measures that are indicative of the performance of individual signalized intersections, arterials, or corridors (viewed as a combination of interdependent intersections). Section 4.4 explores previous efforts which apply analytics on these performance measures with the goal of simplifying workflows for traffic engineers and managers.

4.3 Data Cleaning, Preprocessing, and Categorization

The new generation of signal controllers, based on the latest Advanced Transportation Controller (ATC) (ITE, 2018) standards, are capable of recording signal events as well as vehicle arrival and departure events at a very high resolution (10 Hz). As a result of the high resolution data rates, it is now possible to compute signal performance metrics such as arrivals on red, arrivals on green, and platooning ratios on a cycle-by-cycle or subcycle basis (Gettman et al., 2013). A standard four-way intersection with actuated or semi-actuated control has eight ***phases*** (or directions of vehicular movement) and 4 ways to ***approach*** the intersection as shown in Figure 4. Each approach usually has multiple lanes and can have one or two permitted phases. There may be more than one vehicle ***detector*** on each lane. There are usually many detectors (e.g., stop bar detectors and advanced detectors) at an intersection, with more than one detector per lane. These can be of different types; however, most detectors (at the very least) report vehicle arrivals and departures. Given a detector identifier in the log file, it is essential to know where the detector is located in the intersection to make meaningful observations about traffic behavior and the performance of the intersection.



(FHWA, 2008)

Both signal timing split reports and traffic volume reports from a single controller often contain information (metadata) about the location of the controller. But these data are often poorly formatted and sometimes even missing. They must be correctly formatted. The metadata need to be stored separately, and different split reports may need to be merged.

4.3.1 Faulty Data

The following are some of the common problems that can occur with controllers deployed in the field. The data coming in from the controllers and detectors can be checked for these problems before it is stored. Some of the controller error indicators are:

- No reported data communication from a controller for an extended period. This period can vary from one day to one week.
- Odd-hour pedestrian activations: If there is repeated pedestrian activity detected at odd hours, it may

indicate a problem with the controller.

Indicators for faulty lane detectors are:

- Max presence at odd hours: If a traffic volume detector reports maximum presence very late at night or early morning, the detector and the data reported from it may be faulty.
- No activity: If a detector reports no activity on a lane for an extended duration of time, it may indicate a fault in the detector.
- Chatter: Chatter is a condition in which the lane detectors continuously report presence at the maximum frequency (10 Hz). This may also indicate a problem with the detector.

For all these diverse sources of data, the metadata needed for effectively using them is needed in form of a metadata database. This should include locations of each of the distinct kind of detector and the effective range of the detectors.

4.4 Data Summarizing and Clustering

The primary function of the traffic controllers is to ensure safety by eliminating (or managing) conflicts and maximizing the flow of traffic while adhering to the safety constraints. The newest generation of traffic signal controllers, which are ATC compliant (ITE, 2018), can record both signal phasing information and vehicle arrivals and departures at a very high resolution (10 Hz). As discussed above, this has resulted in the development of many new measures of effectiveness (MOEs), including the development of the ATSPM (UDOT, 2017) and other related efforts. These tools that enable traffic engineers to quantitatively study the performance of signalized intersections are being developed and deployed in cities and regions all across the United States. This newer generation of controllers, combined with the ATSPM systems, provides vastly improved monitoring capabilities as compared to the older generation systems, which were based on coarse-grained data (typically at 15-minute intervals).

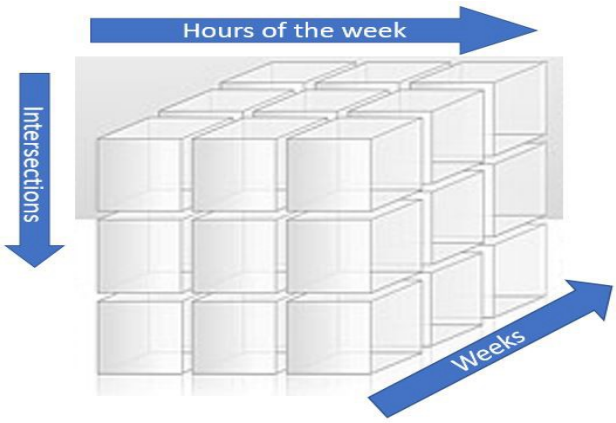
This section presents a novel framework that combines processing of high resolution controller log data pertaining to certain performance measures with modern data mining and machine learning techniques to produce the following outcomes:

1. Automatically learn a compact representation of signal performance and behavior both in terms of the signal's capacity to serve demand and coordination of signal with upstream and downstream intersections
2. The compact representation enables the grouping of signals based on similar demand patterns and performance over time and space as well as the ordering of these groups in terms of observed performance.
3. Computing the evolution of these performance-based groups over space and time can help in deriving potential time periods for coordinated signal timing plans.
4. Our models can be used to discover significant changes in signal performance and hence to estimate the need to update signal timing plans. In other words, detect temporal changes in signal performance over multiple weeks or months and detect periods with changes in many signals.

5. Automatic generation of human understandable descriptions for each of these performance groups or clusters and ways of further compacting the performance measure for each of these groups.

4.4.1 Measures of Effectiveness

In order to quantify the performance of intersections, we use a combination of two performance measures from the performance measurement literature, namely split failures and ratio of arrivals on green to arrivals on red (AOR/AOG). Split failures can be of two types: max-outs and force-offs. When a signal reaches the maximum allocated green time due to high demand, a max-out is said to have occurred, whereas force-offs occur when an intersection reaches the maximum allocated green time without being able to fulfill the demand. High split failures are indicators of high demand on a particular phase (movement direction) of the intersection. Arrivals on green and arrivals on red are the number of vehicles arriving at an intersection during a green phase vs. a red phase. Higher AOG are a positive indicator for signal coordination.



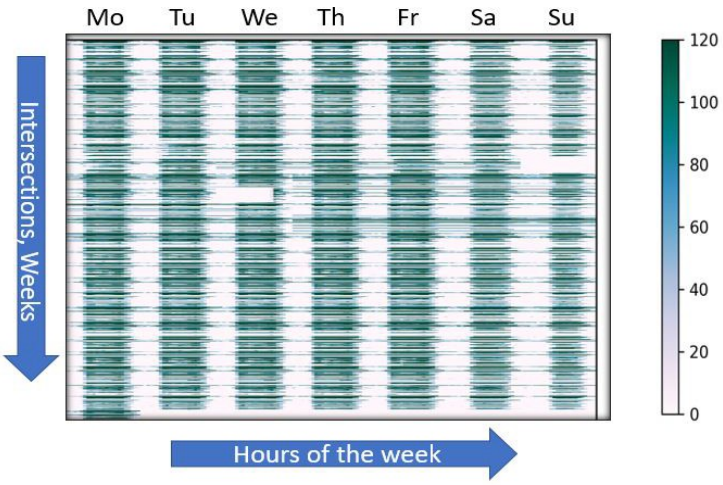
with similar performance).

To conclude, intersections with high arrivals on green and low split failures are, in general, well timed and utilized, whereas a large number of split failures and a low AOR/AOG generally indicates congestion.

Split Failures in Coordinated Corridors. We now explore these measures in more detail. A split failure is, almost always, an indicator of high demand. It can also indicate a situation where the demand is near or over the capacity of the phase. Most of the time, sets of intersections on the same corridor are coordinated to maximize the flow of traffic. In such a situation, the phases along the primary direction end up using the maximum possible green time and hence max out (report split failures). To ensure the usefulness of the data gathered on coordinated corridors, we take into account the detector-on events right before and after a split failure is reported. This is similar to computing the red occupancy ratio (ROR) and the green occupancy ratio (GOR), metrics used both in the literature and in the field. Henceforth, the term split failures refers only to these **demand-based split failures**.

In our methodology, the first step is to aggregate the high resolution (10 Hz) data into minute-by-minute buckets. For split failures reported on a phase, we record a 1 if that phase fails during the minute under consideration. More than one split failure in a minute is also recorded as a 1. If there are no split failures reported for the phase, we record a value of 0. We ignore the split failures reported in some coordinated

corridors when there is no demand (ROR/GOR). Hence, we compute the duration of split failures in minutes and store that information as a time series vector. Similarly, we compute the arrivals on red, arrivals on green, and the ratio of the two, every minute. The output of the first step is two vectors with 24 × 60 (1,440) entries representing the behavior of a particular phase over the entire day. The next step in our methodology is to take these 1,440-digit feature vectors and aggregate into one-hour bins (we experimented with 30 min, 60 min, and 120 min bins). The dimensionality of the new vectors is 24, i.e., each vector has 24 entries representing one intersection for each hour of the day. We now create 24 × 7 or 168-dimension vectors for each intersection, representing the intersection for the whole week. After the aggregation, we combine the vectors representing the various phases of an intersection. In our analysis, we have considered only the primary directions (phases 2 and 6) while creating these vectors. The end result is a (168-dimension) vector representing the performance of an intersection for the whole week. These 168-dimension vectors with (raw) performance measures are called $fav1$ (SF) and $fav2$ (AOR/AOG) in the rest of the document. Figure 5 represents a three-dimensional way of organizing many such vectors. This is the vector space subsequently studied from compression and clustering perspectives. This step concludes the first stage of processing the controller log data. Thus far, we have summarized the split failures and laid the foundation for further processing.



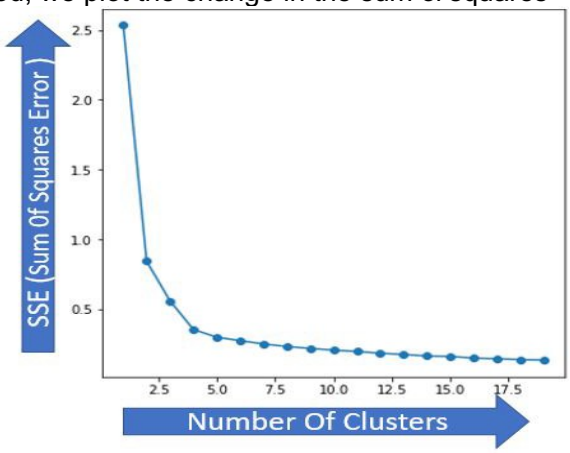
a week of data for a single intersection (fav vector).

4.5 Clustering

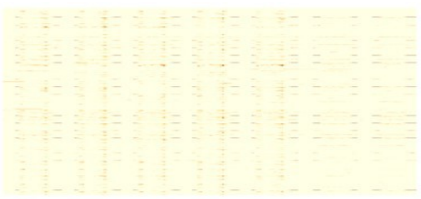
The basic goal is the discovery of regularities across these dimensions. Because the road network geometry is known to us, we can also use this domain knowledge to find regularities in signal performance along corridors of interest.

Spectral Clustering. Nonlinear dimensionality reduction (NLD) (Belkin and Niyogi, 2003) and spectral clustering are closely related if Laplacian eigenmaps are used for NLD. This is briefly summarized. A weighted graph is first constructed from the data cube with the binary relations in the graph set, based on distances between the vectors. Because we already have a linear vector space embedding of the raw data (from the previous section), this metric is used as the distance measure. NLD proceeds by computing the generalized eigenvectors corresponding to a suitable (usually the lowest) set of eigenvalues of the graph Laplacian. The rows of the eigenvector matrix form the nonlinear embedding. Given this embedding structure, we can then perform standard clustering in this space (K-means or equivalent) to produce the clusters of interest. If necessary, the tight relationship between spectral clustering and graph partitioning can be exploited to obtain a different set of clusters, but we have found that in practice, standard K-means clustering in the embedded space works well.

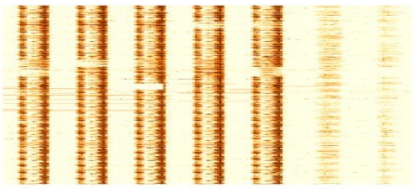
Model Selection, Number of Clusters. Using the Elbow method, we plot the change in the sum of squares distance between all the data points and their assigned cluster centers as we increase the number of clusters. This reduction in SSE (sum of squares error) is shown in Figure 7. From the plot, we can conclude that the number of clusters K must be at least 5. However, this method does not give us an upper bound on K. After $K = 5$, the reduction in SSE is mostly linear. After trying various values above 5, we picked $K = 10$ because this allows us to order the clusters by performance observed at the cluster centroid. This can be seen in Figure 8.



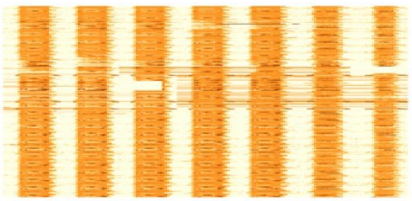
Spatial Information. Sometimes, the intersections belonging to a cluster are spread over geographic regions many miles apart. While these intersections may be performing similarly, there is no real value in having such distant intersections in the same cluster if we want to modify signal plans, for example. So, we may do a second round of processing where we split a cluster of intersections into multiple disjoint clusters based on a geographical indicator like primary road names, distance, or the hop distance between the intersections.



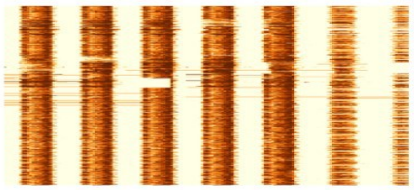
1



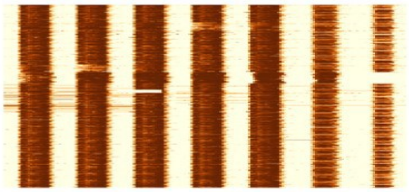
2



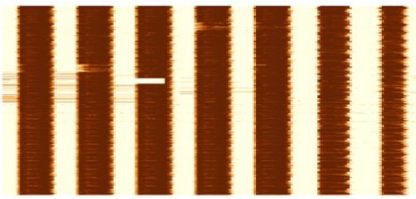
3



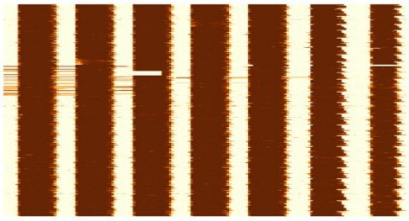
4



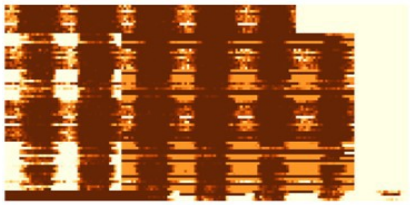
5



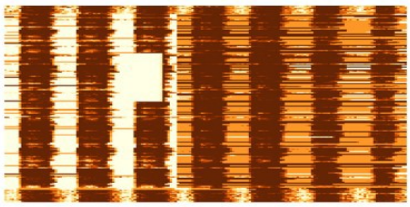
6



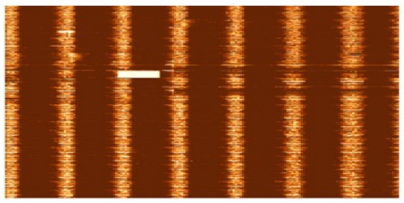
7



8



9



10

ID	Norm	Percentage	Variation
1	12	0	1%
3	61	18	3%
4	75	28	4%
8	83	34	6%
9	97	47	9%
0	104	54	8%
6	111	61	11%
5	115	66	13%
7	118	70	13%
2	133	88	15%

5 Video Analytics

The rapid changes in growth of exploitable and, in many cases, open data have the potential to mitigate traffic congestion and improve safety. Despite significant advances in vehicle technology, traffic engineering practices, and analytics based on crash data, the number of traffic crashes and fatalities are still too many. Many drivers are frustrated due to long (but potentially preventable) delays at intersections. The use of video or lidar processing, big data analytics, artificial intelligence, and machine learning can profoundly improve the ability to address these challenges. The collection and exploitation of large datasets is not new to the transportation sector. However, the confluence of ubiquitous digital devices and sensors, significantly lower hardware costs for computing and storage, enhanced sensing and communication technologies, and open-source analytics solutions have enabled novel applications. The latter may involve insights into otherwise unobserved patterns that may positively influence individuals and society.

Sensor data from smart devices or video cameras can be analyzed immediately to provide real-time analysis for intelligent transportation systems (ITS). At traffic intersections, there is a greater volume of road users (pedestrians and vehicles), traffic movement, dynamic traffic event, near-accidents, etc. It is a critically important application to enable global monitoring of traffic flow, local analysis of road users, and automatic near-accident detection.

As a new technology, vision-based intelligence has a wide range of applications in traffic surveillance and traffic management (Coifman et al., 1998; Valera and Velastin, 2005; Buch et al., 2011; Kamijo et al., 2000; Veeraraghavan et al., 2003; He et al., 2017). Among them, many research works have focused on traffic data acquisition with aerial videos (Angel et al., 2002; Salvo et al., 2017); the aerial view provides better perspectives to cover a large area and focus resources for surveillance tasks. Unmanned aerial vehicles (UAVs) and omnidirectional cameras can acquire useful aerial videos for traffic surveillance, especially at intersections, with a broader perspective of the traffic scene and with the advantage of being both mobile and present in both time and space. A recent trend in vision-based intelligence is to apply computer vision technologies to these acquired intersection aerial videos (Scotti et al., 2005; Wang et al., 2006) and process them at the edge across multiple ITS architectures.

Object detection and multiple object tracking are widely used applications in transportation, and real-time solutions are significant, especially for the emerging area of big transportation data. A near-miss is an event that has the potential to develop into a collision situation between two vehicles or between a vehicle and a pedestrian or bicyclist. These events are important to monitor and analyze for preventing collisions in the future. They also serve as a proxy for potential timing and design issues at the intersection. Cameras monitoring intersections produce video data in gigabytes per camera per day. Analyzing thousands of trajectories collected per hour at an intersection from different sources in order to identify near-miss events quickly becomes challenging, given the amount of data to be examined and the relatively rare occurrence of such events.

In this project, we investigated the use of traffic video data for near-accident detection. However, to our best

knowledge, a unified system that performs real-time detection and tracking of road users and near-accident detection for aerial videos is not available. Therefore, we have collected video datasets and presented a real-time deep-learning-based method to tackle these problems.

Generally, a vision-based surveillance tool for ITS should meet several requirements: (1) segment vehicles from the background and from other vehicles so that all vehicles (stopped or moving) are detected; (2) classify detected vehicles into categories: cars, buses, trucks, motorbikes, etc.; (3) extract spatial and temporal features (motion, velocity, and trajectory) to enable more specific tasks, including vehicle tracking, trajectory analysis, near-accident detection, anomaly detection, etc.; (4) function under a wide range of traffic conditions (light traffic, congestion, and varying speeds in different lanes) and a wide variety of lighting conditions (sunny, overcast, twilight, night, and rainy, etc.); and (5) operate in real time.

Over the decades, although an increasing amount of research on vision-based systems for traffic surveillance has been proposed, many of the criteria listed above still cannot be met. Early solutions (Hoose, 1992) do not identify individual vehicles as unique targets and progressively track their movements. Methods have been proposed to address individual vehicle detection and vehicle tracking problems (Koller et al., 1993; McLauchlan et al., 1997; Coifman et al., 1998) with tracking strategies including model-based tracking, region-based tracking, active-contour-based tracking, feature-based tracking, and optical flow employment. Compared to traditional hand-crafted features, deep learning methods (Ren et al., 2015; Girshick et al., 2016; Redmon et al., 2016; Tian et al., 2016) in object detection have illustrated the robustness of specialization of the generic detector to a specific scene. Recently, automatic traffic accident detection has become an important topic. One typical approach uses object detection or tracking before detecting accident events (Sadeky et al., 2010; Kamijo et al., 2000; Hui et al., 2014; Jiang et al., 2007; Hommes et al., 2011), using methods such as histogram of flow gradient (HFG), hidden Markov model (HMM), or Gaussian mixture model (GMM). Other approaches (Liu et al., 2010; Ihaddadene and Djeraba, 2008; Wang and Miao, 2010; Wang and Dong, 2012; Tang and Gao, 2005; Karim and Adeli, 2002; Xia et al., 2015; Chen et al., 2010; Chen et al., 2016) use low-level features (e.g., motion features) to demonstrate better robustness. Neural networks have also been employed for automatic accident detection (Ohe et al., 1995; Yu et al., 2008; Srinivasan et al., 2004; Ghosh-Dastidar and Adeli, 2003).

Intersections tend to experience more, and potentially more severe, near-accidents, due to factors such as angles and turning collisions. The first type of intersection video considered here is drone video which monitors an intersection with a top-down view. The second type is real traffic video acquired by omnidirectional fisheye cameras that monitor small or large intersections. This is widely used in transportation surveillance. These can be directly used as inputs for any vision-based framework.

We propose a unified vision-based framework for near-incident detection that has the following novel ideas:

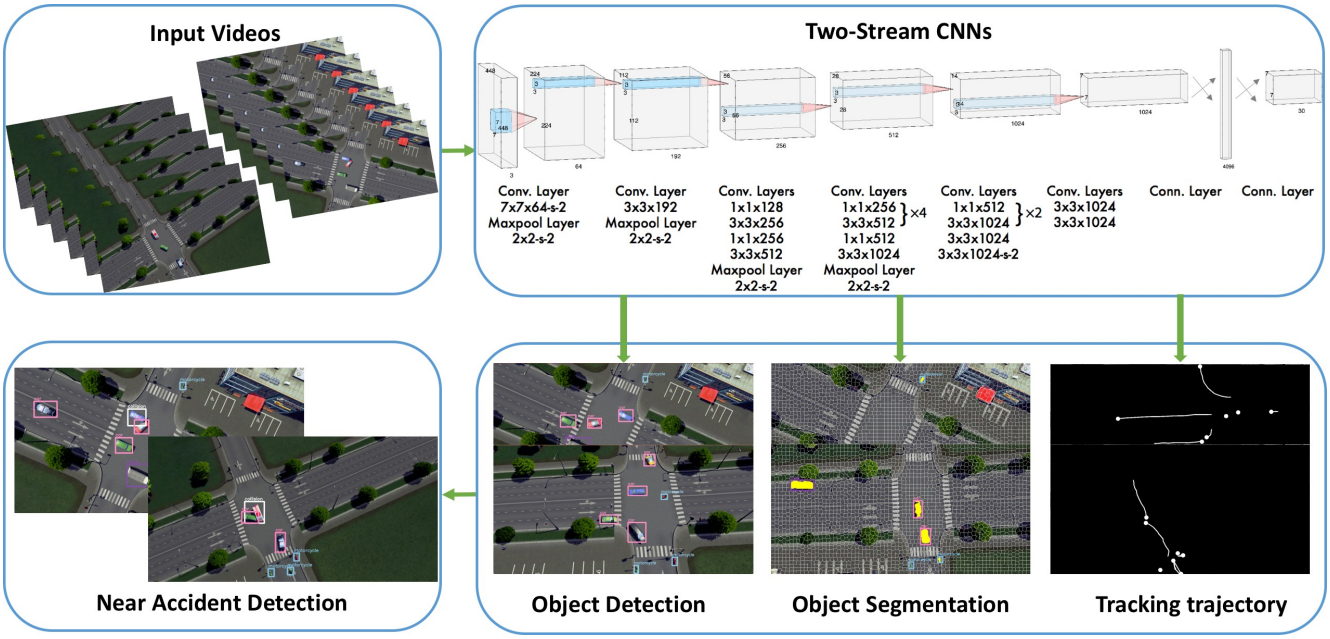


Figure 9: *The overall pipeline of our two-stream convolutional neural networks for near-accident detection.*

1. A combined spatial and temporal stream convolutional networks architecture that performs object detection, classification, and tracking. The spatial stream network detects individual vehicles and likely near-accident regions at the single frame level by capturing appearance features with a state-of-the-art object detection method. The temporal stream network leverages motion features extracted from detected candidates to perform multiple object tracking and generates corresponding trajectories of each tracked target.

2. A frame-based spatial pixel segmentation approach to derive accurate bounding boxes for multiple objects. This approach is considerably more accurate as compared to rectangular bounding boxes approaches.

3. A metric learning approach for improved tracking in the presence of occlusion. Occlusion can lead to several frames where the object goes undetected.

4. A novel distance measure between tracks to detect near-misses between objects. This measure detects near-accidents by incorporating appearance features and motion features to compute probabilities of near-accident candidate regions.

In addition, we show that this can be executed on a GPU at the rate of 33+ frames per second, which is useful in real-time video applications. We also note that these methods are executed on fisheye lens-based video, which is considerably more difficult than standard rectilinear video frame geometry. Experiments demonstrate the advantage of our framework with an overall competitive performance at high frame rates. The overall pipeline of our method is depicted in Figure 9.

5.1 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) have shown strong capabilities in representing objects, thereby boosting the performance of numerous vision tasks, especially when compared to traditional features (Dalal and Triggs, 2005). A CNN is a class of deep neural network which is widely applied in image analysis and computer vision. A standard CNN usually consists of an input and an output layer, as well as multiple hidden layers (convolutional layers, pooling layers, fully connected layers, and normalization layers) as shown in Figure 10. The input to a convolutional layer is the original image $\boldsymbol{X}$. We denote the feature map of the i-th convolutional layer as $\boldsymbol{H}_i$ and $\boldsymbol{H}_0 = \boldsymbol{X}$. Then $\boldsymbol{H}_i$ can be described as

$$\boldsymbol{H}_i = f\left(\boldsymbol{H}_{i-1} \otimes \boldsymbol{W}_i + \boldsymbol{b}_i\right) \tag{1}$$

where $\boldsymbol{W}_i$ is the weight for the i-th convolutional kernel for the (i−1)-th image or feature map and $\otimes$ is the convolution operation. The output of the convolution operation includes a bias $\boldsymbol{b}_i$. Then, the feature map for the i-th layer can be computed by applying a standard nonlinear activation function. We briefly describe a 32 × 32 RGB image with a simple ConvNet for CIFAR-10 image classification (Krizhevsky, 2009):

- Input layer: the original 32 × 32 image with three color channels (R,G,B)
- Convolutional layer: local operations in the image passed through nonlinear activation functions. If we use 12 filters, the size of the result is [32 × 32 × 12].
- Pooling layer: a downsampling operation, resulting in a volume such as [16 × 16 × 12]
- Fully connected layer: output scores for each class, resulting in a volume of size [1 × 1 × 10] for 10 classes.

In this way, CNNs transform the original image into multiple high-level feature representations layer by layer, finally obtaining class-specific outputs or scores.

5.2 YOLO Object Detection

You Only Look Once (YOLO) (Redmon et al., 2016) is a state-of-the-art, real-time object detection system. This end-to-end deep learning approach does not make use of region proposals and is therefore quite different from the region-based methods. The pipeline of YOLO (Redmon et al., 2016) is rather straightforward: YOLO (Redmon et al., 2016) passes the whole image through the neural network only once, as its name implies (You Only Look Once), and returns bounding boxes and class probabilities for predictions. Figure 12 demonstrates the detection model and system of YOLO (Redmon et al., 2016). YOLO (Redmon et al., 2016) is orders of magnitude faster (45 frames per second) than other object detection approaches,

.

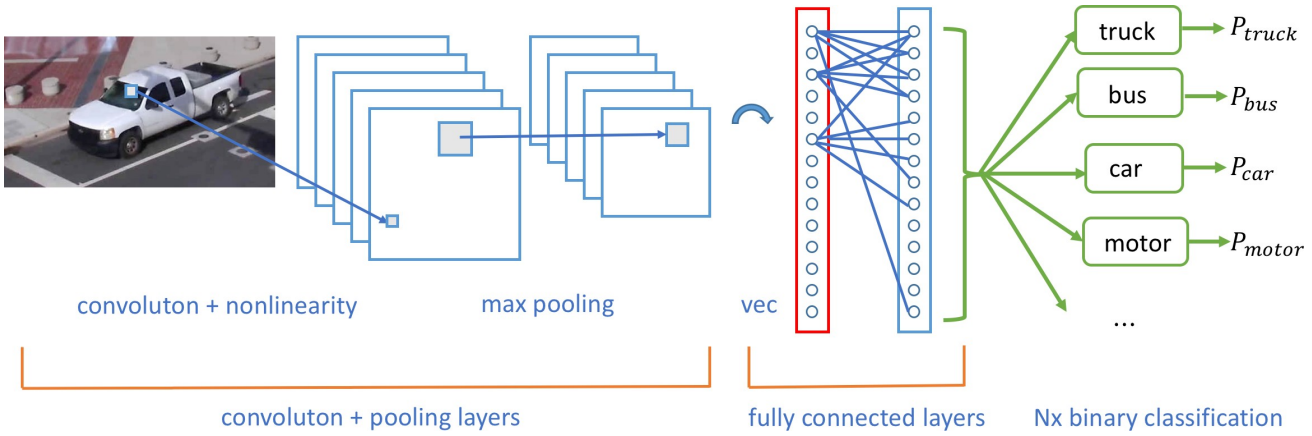


Figure 10: *Architecture of convolutional neural networks for image classification. This figure has been adapted from a figure that appeared on the blog article "A Beginner's Guide to Understanding Convolutional Neural Networks (https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/) by Adit Deshpande).*

which means it can process streaming video in real time and achieve more than twice the mean average precision of other real-time systems. In this work, we leverage the extension of YOLO (Redmon et al., 2016), Darknet-19, a classification model used as the basis of YOLOv2 (Redmon and Farhadi, 2017). Darknet-19 (Redmon and Farhadi, 2017) has 19 convolutional layers and five max-pooling layers, and it uses batch normalization to stabilize training, speed up convergence, and regularize the model (Ioffe and Szegedy, 2015).

5.3 Simple Online Real-time Tracking(SORT)

Simple Online Real-time Tracking (Bewley et al., 2016) is a simple, popular, and fast multiple object tracking (MOT) algorithm. The core idea is to perform Kalman filtering (Kalman, 1960) in image space and then frame-by-frame data association using the Hungarian method (Kuhn, 1955) with an association metric that measures bounding box overlap. Despite only using a rudimentary combination of the Kalman Filter (Kalman, 1960) and Hungarian algorithm (Kuhn, 1955) for the tracking components, SORT (Bewley et al., 2016) achieves an accuracy comparable to state-of-the-art online trackers. Moreover, due to its simplicity, SORT (Bewley et al., 2016) can update at a rate of 260 Hz on a single machine, which is over 20 times faster than other state-of-the-art trackers.

5.4 DeepSORT

DeepSORT (Wojke et al., 2017) is an extension of SORT (Bewley et al., 2016) which integrates appearance information through a pre-trained association metric to improve the performance of SORT (Bewley et al., 2016). It combines a conventional single hypothesis tracking methodology with recursive Kalman filtering (Kalman, 1960) and frame-by-frame data association. DeepSORT helps to solve a large number of identity switching problems in SORT, and it can track objects through longer periods of occlusion. During online application, it establishes measurement-to-track associations using nearest-neighbor queries in visual appearance space.

associations using nearest-neighbor queries in visual appearance space.

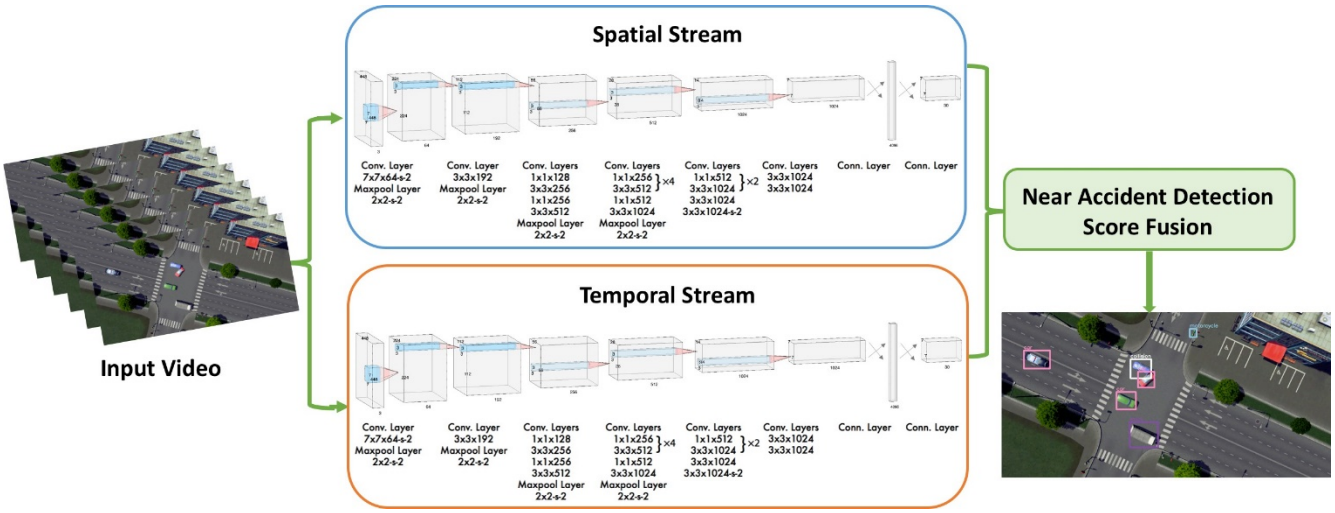


Figure 11: *Two-stream convolutional neural networks architecture for near-accident detection.*

5.5 Two-Stream Architecture for Near-Accident Detection

In this section, we present our computer vision-based two-stream architecture for real-time near-accident detection. The architecture is primarily driven by real-time object detection and multiple object tracking (MOT). The goal of near-accident detection is to detect likely collision scenarios across video frames and report these near-accident records. Because videos have both spatial and temporal components, we divide our framework into a two-stream architecture as shown in Figure 11. The spatial aspect comprises individual frame appearance information about scenes and objects appearing in the video. The temporal aspect comprises motion information of objects. For the spatial stream convolutional neural network, we utilize a standard convolutional network designed for state-of-the-art object detection (Redmon et al., 2016) to detect individual vehicles and mark near-accident regions at the single-frame level. The temporal stream network leverages object candidates from object detection CNNs and integrates their appearance information with a fast MOT method to extract motion features and compute trajectories. When two trajectories of individual objects intersect or come closer than a certain threshold (whose estimation is described below), we label the region covering the two objects as a high probability near-accident region. Finally, we take the average near-accident probability of both the spatial stream network and the temporal stream network and report the near-accident record.

5.5.1 Object Detection and Classification

In our framework, each stream is implemented using a deep convolutional neural network. Near-accident scores are combined by averaging. Since our spatial stream ConvNet is essentially an object detection architecture, we base it on recent advances in object detection—essentially the YOLO detector (Redmon

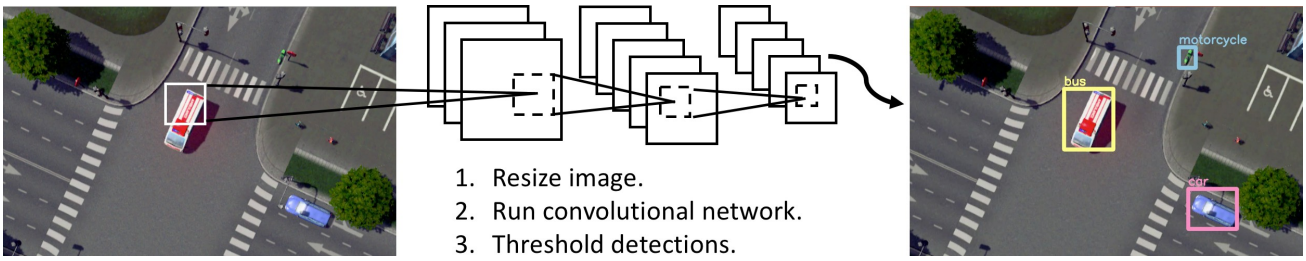


Figure 12: *Object detection pipeline of our spatial stream: (1) resizes the input image to 448 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detection by the model's confidence.*

et al., 2016)—and pre-train the network from scratch on our dataset containing multiscale drone, fisheye, and simulation videos. As most of our videos contain traffic scenes with vehicles and traffic movement captured in a top-down view, we specify different vehicle classes such as motorbike, car, bus, and truck as object classes for training the detector. Additionally, near-accidents or collisions can be detected from single still frames even at the beginning of a video or from stopped vehicles associated in an accident after collision. Therefore, we train our detector to localize these likely near-accident scenarios. Since static appearance is a useful cue, the spatial stream network performs object detection by only operating on individual video frames.

The spatial stream network regards object detection as an end-to-end regression problem and uses a single convolutional network to predict the bounding boxes and the class probabilities for these boxes. It first takes the image and splits it into a $S \times S$ grid. Within each grid, it sets up m bounding boxes. For each grid cell,

- It predicts B boundary boxes, and each box has a confidence score,
- It detects one object only, regardless of the number of boxes B,
- It predicts C conditional class probabilities (one per class for the likelihood of the object class).

For each bounding box, the convolutional neural network (CNN) outputs a class probability and offset values for the bounding box. Then, it selects bounding boxes which have the class probability above a threshold value and uses them to locate the object within the image. In essence, each boundary box contains five elements: (x, y, w, h) and a box confidence. The (x, y) are coordinates which represent the center of the box relative to the bounds of the grid cell. The (w, h) parameters are the width and height of the object. These elements are normalized such that $x, y, w,$ and h lie in the interval [0, 1]. The confidence prediction represents the intersection over union (IoU) between the predicted box and any ground truth box, which reflects the likelihood that the box contains an object (objectness) and the accuracy of the boundary box. The network architecture of the spatial stream simply contains 24 convolutional layers followed by two fully connected layers, reminiscent of AlexNet and even earlier convolutional architectures.

Some convolution layers use 1 × 1 reduction layers alternatively to reduce the depth of the features maps. For the last convolution layer, it outputs a tensor with shape (7, 7, 1024), which is flattened. It performs a

linear regression using two fully connected layers to make boundary box predictions and to make a final prediction using the threshold of box confidence scores. The final loss adds the localization (the 1st and the 2nd terms), confidence (the 3rd and the 4th terms), and classification (the 5th term) losses together.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \quad (2)$$

In equation 2, $\mathbb{1}_{i}^{\text{obj}}$ denotes if the object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the j-th bounding box predictor in cell i is "responsible" for that prediction.

5.5.2 Multiple Object Tracking

In general, MOT can be regarded as a multivariable estimation problem (Luo et al., 2014), and the objective of MOT can be modeled by performing MAP (maximum a posteriori) estimation in order to find the *optimal* sequential states of all the objects from the conditional distribution of the sequential states, given all the observations:

$$\widehat{\mathbf{S}}_{1:t} = \arg\max_{\mathbf{S}_{1:t}} P(\mathbf{S}_{1:t} | \mathbf{O}_{1:t}) \quad (3)$$

where s_t^i denotes the state of the i-th object in the t-th frame. $\mathbf{S}_t = (s_t^1, s_t^2, \ldots, s_t^{M_t})$ denotes states of all the M_t objects in the t-th frame. $\mathbf{S}_{1:t} = \{\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_t\}$ denotes all the sequential states of all the objects from the first frame to the t-th frame. In tracking-by-detection, o_t^i denotes the collected observations for the i-th object in the t-th frame. $\mathbf{O}_t = (o_t^1, o_t^2, \ldots, o_t^{M_t})$ denotes the collected observations for all the M_t objects in the t-th frame. $\mathbf{O}_{1:t} = \{\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_t\}$ denotes all the collected sequential observations of all the objects from the first frame to the t-th frame.

The spatial stream network is not able to extract motion features and compute trajectories due to single-frame inputs. To leverage this useful information, we present our temporal stream network, a ConvNet model which implements a tracking-by-detection MOT algorithm (Bewley et al., 2016; Wojke et al., 2017) with a data association metric that combines deep appearance features. The inputs are identical to the spatial stream network using the original video. Detected object candidates (only vehicle classes) are used for tracking, state estimation, and frame-by-frame data association using SORT (Bewley et al., 2016) and DeepSORT (Wojke et al., 2017) — the real-time MOT method. MOT models the state of each object and describes the motion of objects across video frames. The tracking information allows us to stack trajectories of moving objects across several consecutive frames, which are useful cues for near-accident detection.

Figure 13: *Vehicle samples selected from the VeRi dataset. All images are the same size, and we resize to 64 × 128 for training. Left: Diversity of vehicle types and colors; right: variation resolutions, viewpoint, and occlusion. Source: Large-scale vehicle re-identification in urban surveillance videos, by Liu et al. published in* IEEE International Conference on Multimedia and Expo (ICME)*, IEEE, New York, NY. Pg. 1–6, 2016.*

Estimation Model. The state of each target is modeled as

$$\mathbf{x} = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T \tag{4}$$

where u and v represent the horizontal and vertical pixel location of the center of the target, while s and r represent the scale (area) and the aspect ratio (usually considered to be constant) of the target's bounding box, respectively. When a detection is associated with a target, it updates the target state using the detected bounding box, where the velocity components are solved optimally via a Kalman filter framework (Kalman, 1960). If no detection is associated with the target, its state is simply predicted without correction using the linear velocity model.

Creation and Deletion of Track Identities. When new objects enter or old objects vanish in video frames, unique identities for objects need to be created or destroyed accordingly. For creating trackers, we consider any detection with an overlap less than IoU_{min} to signify the existence of an untracked object. Then the new tracker undergoes a probationary period during which the target needs to be associated with detections to accumulate enough evidence in order to prevent tracking of false positives. Tracks could be terminated if they are not detected for T_{Lost} frames to prevent an unbounded growth in the number of trackers and localization errors caused by predictions over long durations without corrections from the detector.

Track Handling and State Estimation. Using Kalman filtering (Kalman, 1960) for track handling and state estimation is mostly identical to SORT (Bewley et al., 2016). The tracking scenario is defined using an eight-dimensional state space $(u, v, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$ that contains the bounding box center location (u, v), aspect ratio γ, height h, and its velocities relative to other bounding boxes in image coordinates. It uses a standard Kalman filter (Kalman, 1960) with a constant velocity motion and linear observation model, where it takes the bounding coordinates (u, v, γ, h) as direct observations of the object state.

Data Association – To solve the frame-by-frame association problem between the predicted Kalman states and the newly arrived measurements, SORT uses the Hungarian algorithm (Kuhn, 1955). It integrates both motion and appearance information through a combination of two appropriate metrics. For motion information, the (squared) Mahalanobis distance between predicted Kalman states and newly arrived measurements is utilized:

$$d^{(1)}(i,j) = (\boldsymbol{d}_j - \boldsymbol{y}_i)^{\boldsymbol{T}} (\boldsymbol{S}_i)^{-1} (\boldsymbol{d}_j - \boldsymbol{y}_i) \tag{5}$$

where the projection of the i-th track distribution into measurement space is $(\boldsymbol{y}_i, \boldsymbol{S}_i)$ and the j-th bounding box detection is $\boldsymbol{d}_j$. The second metric measures the smallest cosine distance between the i-th track and j-th detection in appearance space:

$$d^{(2)}(i,j) = \min \left\{ 1 - \boldsymbol{r}_j{}^T \boldsymbol{r}_k^{(i)} | \boldsymbol{r}_k^{(i)} \in \mathcal{R}_i \right\} \tag{6}$$

Then, this association problem is built with a combination of both metrics using a weighted sum, for which the influence of each metric on the combined association cost can be controlled through a hyperparameter λ.

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j) \tag{7}$$

Matching Cascade. Rather than solving measurement-to-track associations in a global way, SORT adopts a matching cascade, introduced in (Wojke et al., 2017) to solve a series of subproblems. In some situations, when an object is occluded for a longer period of time, the subsequent Kalman filter (Kalman, 1960) predictions would increase the uncertainty associated with the object location. Consequently, the probability mass spreads out in state space, and the observation likelihood becomes less peaked. Intuitively, the association metric should account for this spread of probability mass by increasing the measurement-to-track distance. Therefore, the matching cascade strategy gives priority to more frequently seen objects to encode the notion of probability spread in the association likelihood.

5.6 Metric Learning for Vehicle Re-identification

Metric learning and various hand-crafted features are widely used in the field of person re-identification. We apply a cosine metric learning method to train a neural network for vehicle re-identification and use it to make our temporal stream more robust in terms of tracking performance. Metric learning aims to solve the clustering problem by constructing an embedding in which the metric distance corresponding to the same identity is likely to be closer than features from different identities. The cosine metric measures the degree of similarity by calculating the cosine distance between two objects. We observe that in traffic video, especially in crowded traffic scenes or scenes with heavy occlusion, our tracking algorithm produces switched road user identities. In order to generate accurate and consistent track data, we introduce a deep cosine metric learning method to learn the cosine distance between objects. The cosine distance involves

appearance information that provides useful cues for recovering identities in crowded scenes or after long-term occlusion when motion information is less discriminative. Through this deep network, the feature expression vector obtained by any object is placed in the cluster corresponding to the nearest neighbor. We trained the network on a vehicle re-identification dataset (VeRi dataset) (Liu et al., 2016) (Figure 13) and integrated it as the second metric measure for the assignment problem of our temporal stream.

Our idea is to use a unified end-to-end framework that automatically learns the best metrics. Compared to the standard distance metric (L1, L2), the learned metric can obtain more discriminative features for re-identification and is more robust to cross-view vehicle images. We adopt the architecture of (Wojke and Bewley, 2018) and train a deep network with a cosine softmax classifier which can generate feature vectors of fixed length (128) for the input images (vehicles). The cosine metric finds the nearest cluster exemplar and matches the vehicle on (sometimes far-flung) different video frames to solve tracking problems in the presence of heavy occlusion.

Given a re-identification dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ of N training images $\boldsymbol{x}_i \in \mathbb{R}^D$ and associated class labels $y_i \in \{1,\ldots,C\}$, deep metric learning is to find a parametrized encoder function (deep neural network) $\boldsymbol{r} = f_\Theta(\boldsymbol{x})$ with parameter Θ which projects the input images $\boldsymbol{x} \in \mathbb{R}^D$ into a feature representation $\boldsymbol{r} \in \mathbb{R}^d$.

The cosine softmax classifier can be adapted from a standard softmax classifier and is expressed as

$$p(y = k \mid \boldsymbol{r}) = \frac{\exp\left(\kappa \cdot \tilde{\boldsymbol{w}}_k^{\mathrm{T}} \boldsymbol{r}\right)}{\sum_{n=1}^{C} \exp\left(\kappa \cdot \tilde{\boldsymbol{w}}_n^{\mathrm{T}} \boldsymbol{r}\right)} \tag{8}$$

where κ is a free scale parameter. To generate compact clusters in the feature representation space, the first modification is to apply the ℓ_2 normalization to the final layer of the encoder network so that the representation has unit length $||f_\Theta(\boldsymbol{x})||_2 = 1, \forall \boldsymbol{x} \in \mathbb{R}^D$. The second modification is to normalize the weights to unit length as well, i.e., $\tilde{\boldsymbol{w}}_k = \boldsymbol{w}_k / ||\boldsymbol{w}_k||_2, \forall k = 1,\ldots,C$. The training of the encoder network can be carried out using the cross-entropy loss.

5.7 Using Image Segmentation to Determine ObjectGaps

To present our method in a fully end-to-end fashion, we discard the original, manually determined distance threshold for near-accident detection. We introduce a threshold learning method by estimating the gap between objects using object segmentation. We present a supervised learning method, using gap distance between road users in the temporal stream to determine a near-accident while continuing to update the threshold for more convergence and precision. The straightforward way is to compute the gap using bounding boxes from detections, but the bounding boxes are not accurate in terms of representing boundaries of objects, particularly when rotation is involved. We need a more compact representation and therefore combine detections with superpixel segmentation on the video frame to get an object mask for gap distance estimation.

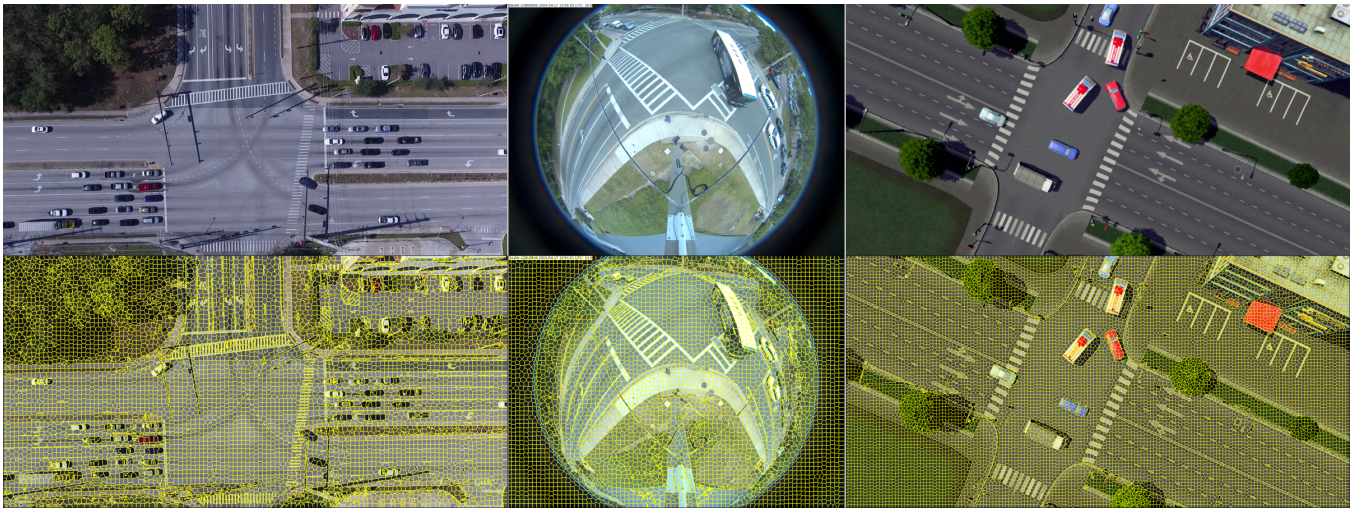


Figure 14: *2D image superpixel segmentation using SLIC. Top: original image; bottom-left: SLIC segmentation for drone video (1,600 superpixels); bottom-middle: SLIC segmentation for fisheye video (1,600 superpixels); bottom-right: SLIC segmentation for simulation video (1,600 superpixels).*

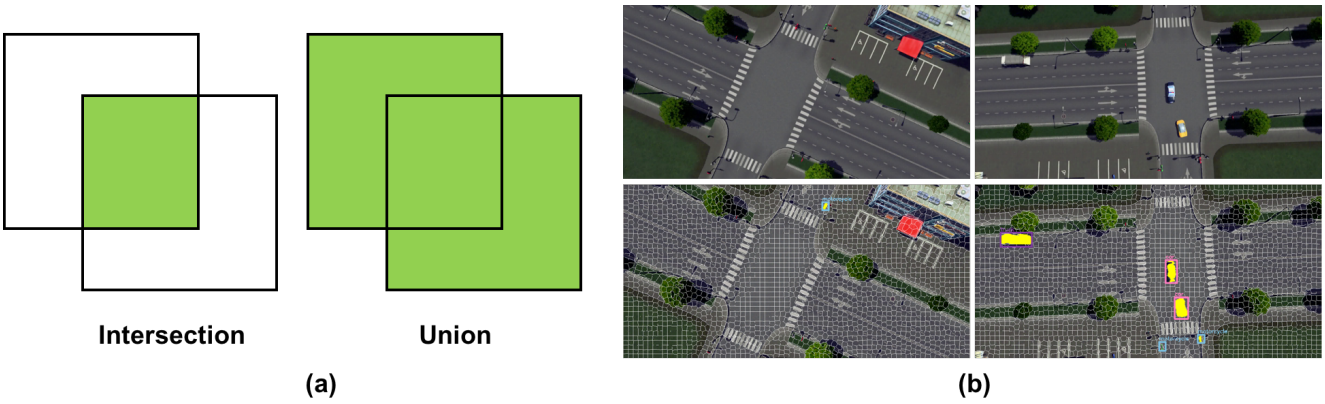


Figure 15: *(a) Illustration depicting the definitions of intersection and union. (b) Top: original video frames; bottom: object detections and object masks produced by our method.*

The main steps of our gap estimation method can be summarized as follows:

1. Extract object detections (bounding boxes) from the spatial stream network
2. Tessellate the video frame into homogeneous superpixels
3. Compute the Intersection over Union (IoU) metric of bounding boxes and superpixels at frame level
4. Generate object mask per detections (using the IoU metric), and compute the gap between objects using the mask and bounding boxes
5. Update the gap threshold while processing the video.

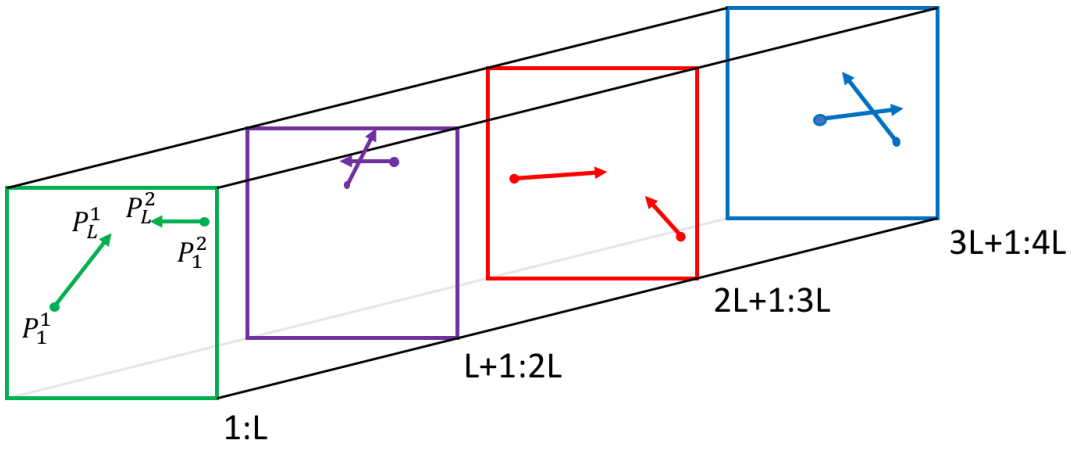


Figure 16: *The stacking trajectories extracted from multiple object tracking of the temporal stream. Consecutive frames and the corresponding displacement vectors are shown with the same color.*

In our gap estimation methods, the Intersection over Union (IoU) metric of boxes and superpixels is given by

$$IoU = \frac{area(B_{1:t}) \cap area(S_{1:t})}{area(B_{1:t}) \cup area(S_{1:t})} \tag{9}$$

where b_t^i denotes the detection for the i-th object in the t-th frame. $\mathbf{B}_t = (\mathrm{b}_t^1, \mathrm{b}_t^2, \ldots, \mathrm{b}_t^{M_t})$ denotes the collected detection observations for all the M_t objects in the t-th frame. $\mathbf{B}_{1:t} = (\mathbf{B}_t^1, \mathbf{B}_t^2, \ldots, \mathbf{B}_t^{M_t})$ denotes all the collected sequential detection observations of all the objects from the first frame to the t-th frame. Similarly, s_t^i denotes the i-th superpixel in the t-th frame. $\mathbf{S}_t = (\mathrm{s}_t^1, \mathrm{s}_t^2, \ldots, \mathrm{s}_t^{N_t})$ denotes all the N_t superpixels in the t-th frame. $\mathbf{S}_{1:t} = \{\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_t\}$ denotes all the collected sequential superpixels from the first frame to the t-th frame.

In 2D superpixel segmentation, the popular SLIC (simple linear iterative clustering) (Achanta et al., 2012) and ultrametric contour map (UCM) (Arbelaez et al., 2011) methods have established themselves as the state of the art. Recently, we have seen deep neural networks (Jampani et al., 2018) and generative adversarial networks (GANs) (Yan et al., 2018) integrated with these methods. In our architecture, we adopt the GPU-based SLIC (gSLICr) (Ren et al., 2015) approach to produce the tessellation of the image data to achieve an excellent frame rate (400 fps). SLIC is widely applicable to many computer vision applications, such as segmentation, classification, and object recognition, often achieving state-of-the-art performance. The contours forming the homogeneous superpixels are shown in Figure 14. In Figure 15, we show an illustration for the definition of intersection and union, and present some object masks we generated from detections and superpixels.

Algorithm 1: Collision Detection

Input: current frame $t_{current}$, collision state list $Collision$
Output: collision state list $Collision$
for $t_L \leftarrow t_{previous}$ *to* $t_{current}$ *in steps of* L *frames* **do**
  for *each pair of object trajectory* $(\mathbf{p}^1_{:t_L}, \mathbf{p}^2_{:t_L})$ **do**
    if $(\mathbf{p}^1_{:t_L}$ *intersects* $\mathbf{p}^2_{:t_L}$ *as of* $t_L)$ **then**
      add $\mathbf{o}_1, \mathbf{o}_2$ to $Collision$
    end
  end
  if *(Collisions)* **then**
    $t_{previous} \leftarrow t_L$; return TRUE
  end
end
$t_{previous} \leftarrow t_d$; return FALSE

5.8 Near-Accident Detection

The most typical motion cue is optical flow, which is widely utilized in video processing tasks such as video segmentation (Huang et al., 2018). A trajectory is defined as a data sequence containing several concatenated state vectors from tracking and an indexed sequence of positions and velocities over a given time window.

When utilizing the multiple object tracking algorithm, we compute the center of each object in several consecutive frames to form stacking trajectories as our motion representation. These stacking trajectories can provide accumulated information through image frames, including the number of objects, their motion history, and timing of their interactions, such as near-accidents. We stack the trajectories of all objects for L consecutive frames, as illustrated in Figure 16, where p_t^i denotes the center position of the i-th object in the t-th frame. $\mathbf{P}_t = (\mathrm{p}_t^1, \mathrm{p}_t^2, \ldots, \mathrm{p}_t^{M_t})$ denotes trajectories of all the M_t objects in the t-th frame. $\mathbf{P}_{1:t} = \{\mathbf{P}_1, \mathbf{P}_2, ..., \mathbf{P}_t\}$ denotes the sequence of trajectories of all the objects from the first frame to the t-th frame. For L consecutive frames, the stacking trajectories are defined by the sequence

$$\mathbf{P}_{1:L} = \{\mathbf{P}_1, \mathbf{P}_2, ..., \mathbf{P}_L\}, \mathbf{P}_{L+1:2L} = \{\mathbf{P}_{L+1}, \mathbf{P}_{L+2}, ..., \mathbf{P}_{2L}\}, \cdots \tag{10}$$

$\mathbf{O}_t = (\mathrm{o}_t^1, \mathrm{o}_t^2, \ldots, \mathrm{o}_t^{M_t})$ denotes the collected observations for all the M_t objects in the t-th frame. $\mathbf{O}_{1:t} = \{\mathbf{O}_1, \mathbf{O}_2, ..., \mathbf{O}_t\}$ denotes all the collected sequential observations of all the objects from the first frame to the t-th frame. We use a simple detection algorithm which finds collisions between simplified forms of the objects, using the center of bounding boxes.

Our algorithm is depicted in Algorithm 1. Once a collision is detected, we set the region covering collision-associated objects to be a new bounding box with class probability of near-accident of 1. By averaging the near-accident probabilities from the spatial stream network and the temporal stream network, we are able to obtain a final confidence measure of near-accident detection.

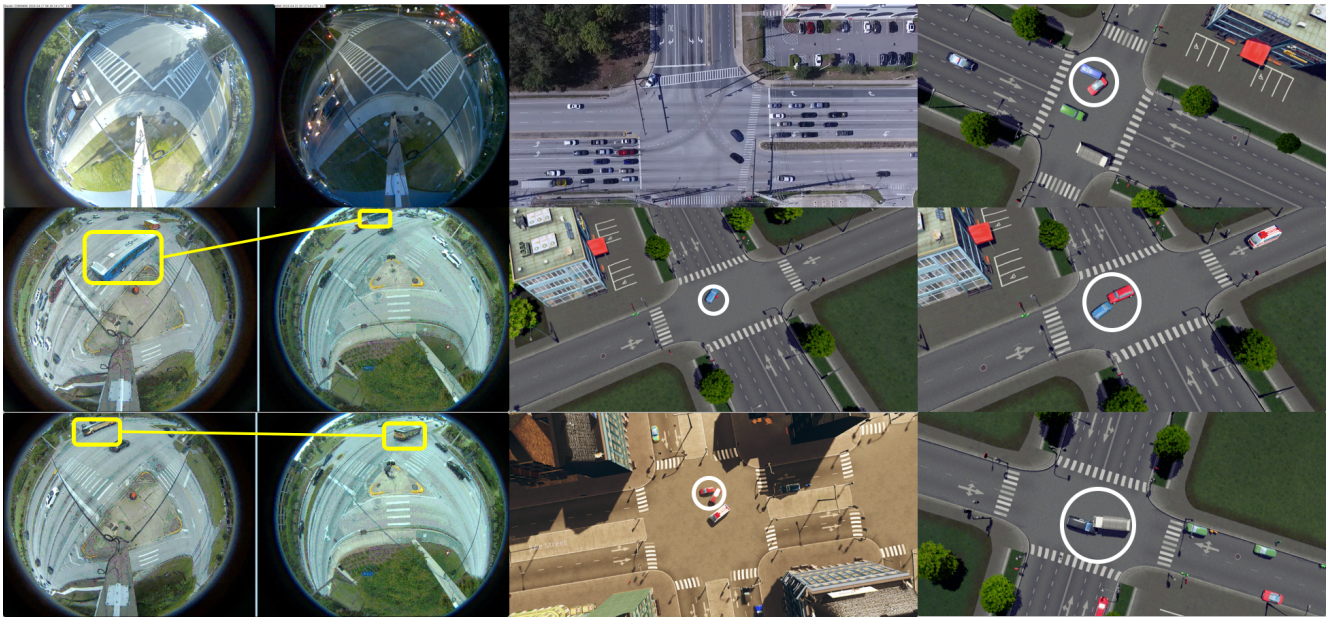


Figure 17: *Samples of traffic near-accident: Our data consist of a large number of diverse intersection surveillance videos and different near-accidents (cars and motorbikes). Yellow rectangles and lines represent the same object in video from multiple cameras. White circles represent the near-accident regions.*

5.9 Experiments

Here, we present qualitative and quantitative evaluation in terms of the performance of object detection, MOT, and near-accident detection, and a comparison between other methods and our framework.

5.9.1 A Traffic Near-Accident Dataset (TNAD)

To our best knowledge, we know of no comprehensive traffic near-accident dataset containing top-down-view videos such as drone or UAV videos or omnidirectional camera videos for traffic analysis. Therefore, we have built our own dataset, the traffic near-accident dataset (TNAD), which is depicted in Figure 17. Intersections tend to experience more near-accidents and more potentially severe ones due to factors such as angles and turning collisions. The traffic near-accident dataset contains three types of video data from traffic intersections that could be utilized for not only for near-accident detection but also for other traffic surveillance tasks, including turn movement counting.

The first type is drone video that monitors an intersection with a top-down view. The second type of intersection video is real traffic video acquired by omnidirectional fisheye cameras that monitor small or large intersections. They are widely used in transportation surveillance. These video data can be directly used as input for our vision-based intelligent framework. Furthermore, pre-processing and fisheye correction can be applied for better surveillance performance. The third type of video is video game-engine simulations

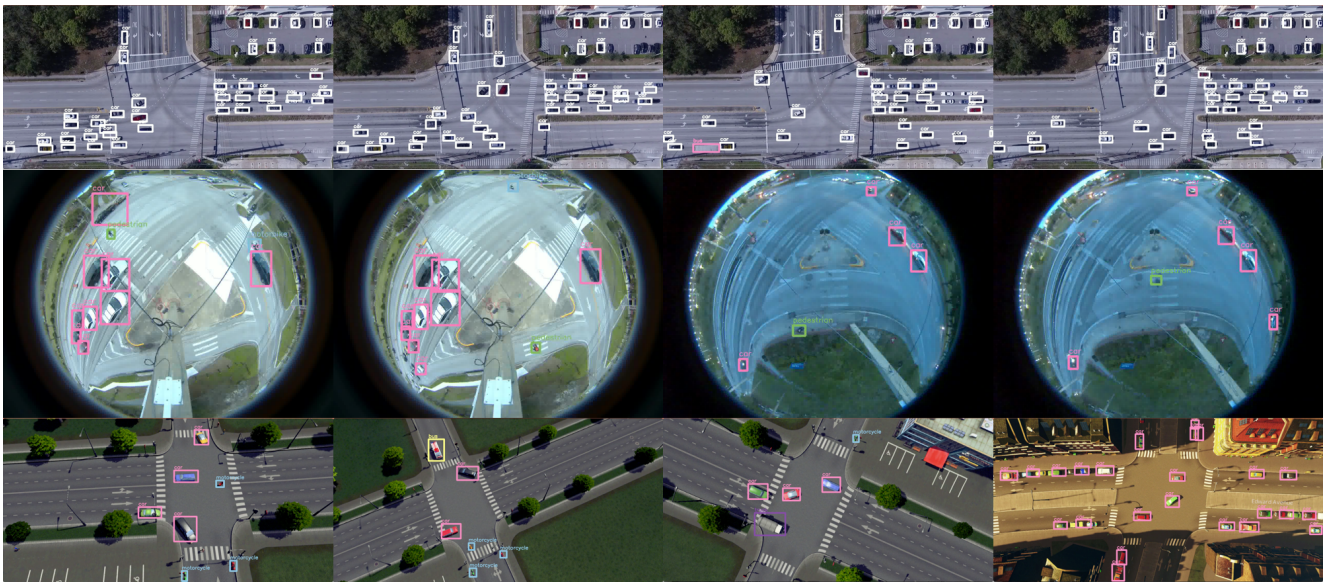


Figure 18: *Object detection result samples of our spatial network on TNAD dataset. Top: samples for drone video; middle: samples for fisheye video; bottom: samples for simulation video.*

that train on near-accident samples as they are accumulated. The dataset consists of 106 videos with total duration over 75 minutes, with frame rates between 20 fps and 50 fps. The drone video and fisheye surveillance videos were recorded in Gainesville, Florida, at several different intersections. Our videos are more challenging than videos in other datasets for the following reasons:

- Diverse intersection scene and camera perspectives: The intersections in drone video, fisheye surveillance video, and simulation video are very diverse. Additionally, the fisheye surveillance video has distortion, and a fusion technique is needed for multicamera fisheye videos.
- Crowded intersection and small object: The number of moving cars and motorbikes per frame is large, and these objects are relatively smaller than normal traffic video.
- Diverse accidents: Accidents involving cars and motorbikes are all included.
- Diverse lighting conditions: Lighting conditions such as daylight and sunset are included.

We manually annotated the spatial location and temporal location of near-accidents and the still or moving objects and their vehicle class in each video. Thirty-two videos with sparsely sampled frames (only 20% of the frames in these 32 videos are used for supervision) were used, but only for training the object detector. The remaining 74 videos were used for testing.

5.9.2 Fisheye and Multicamera Video

We have large amounts of fisheye traffic videos from across the city. The fisheye surveillance videos were recorded from real traffic data in Gainesville. We collected 29 single-camera fisheye surveillance videos and 19 multicamera fisheye surveillance videos monitoring a large intersection. We conducted two experiments: one directly using these raw videos as input for our system and another in which we first

preprocessed the video to correct for fisheye distortion and then fed them into our system. As the original surveillance video has many visual distortions, especially near the circular boundaries of the cameras, our system performed better on these after preprocessing. In this paper, we do not discuss issues related to fisheye unwarping, leaving these for future work.

For large intersections, two fisheye cameras placed opposite to each other are used for surveillance, and each of them shows almost half the roads and real traffic. In this paper, we do not investigate the stitching problem (and this is planned for future work). First, we correct fisheye distortion and then combine the two videos using points of similarity. Then, we apply a simple *object-level* stitching method by assigning the object identity for the same objects across the left and right video using similar features and appearing and vanishing positions.

5.9.3 Model Training

We adopt Darknet-19 (Wojke et al., 2017) for classification and detection with DeepSORT, using a data association metric that combines deep appearance features. We implement our framework on Tensorflow and perform multiscale training and testing with a single GPU (Nvidia Titan X Pascal). Training a single spatial convolutional network takes one day on our system with one Nvidia Titan X Pascal card. For classification and detection training, we use the same training strategy as YOLO9000 (Wojke et al., 2017). We train the network on our dataset with four classes of vehicle (motorbike, bus, car, and truck) for 160 epochs, using stochastic gradient descent with a starting learning rate of 0.1 for classification and 10^{-3} for detection (dividing it by 10 at 60 and 90 epochs.), weight decay of 0.0005, and momentum of 0.9 using the Darknet neural network framework (Wojke et al., 2017).

5.9.4 Qualitative Results

We present some example experimental results of object detection, MOT, and near-accident detection on our traffic near-accident dataset (TNAD) for drone videos, fisheye videos, and simulation videos. For object detection (Figure 18), we present some detection results of our spatial network with multi-scale training based on YOLOv2 (Redmon and Farhadi, 2017). These visual results demonstrate that the spatial stream is able to detect and classify road users with good accuracy even if the original fisheye videos have large distortions and occlusions. Different from drone and simulation video, we have added pedestrian detection on real traffic fisheye video. From Figure 18, we see that our detector has good performance on pedestrian detection on daylight and dawn fisheye videos. The vehicle detection capabilities are good on top-down view surveillance videos, even for small objects. In addition, we can achieve a fast detection rate at 20 to 30 frames per second. Overall, this demonstrates the effectiveness of our spatial neural network.

For MOT (Figure 19), we present comparison of our temporal network based on DeepSORT (Wojke et al., 2017) with Urban Tracker (Jodoin et al., 2014) and TrafficIntelligence (Jackson et al., 2013). We also present more visual MOT comparisons of our baseline temporal stream (DeepSort) with cosine metric learning-based temporal stream (Figure 20). For the tracking part, we use a tracking-by-detection paradigm, thus our methods can handle still objects and measure their state. This is especially useful since Urban Tracker

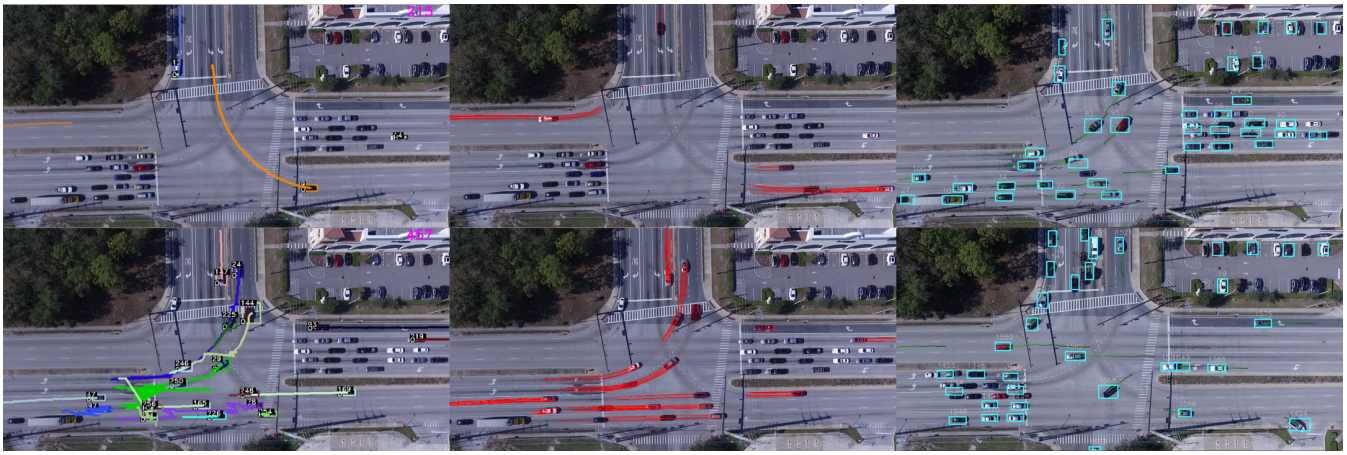


Figure 19: *Tracking and trajectory comparison with Urban Tracker and TrafficIntelligence on drone videos of TNAD dataset. Left: tracking results of Urban Tracker (BSG with Multilayer and Lobster Model); middle: tracking results of TrafficIntelligence; right: tracking results of our temporal stream network.*

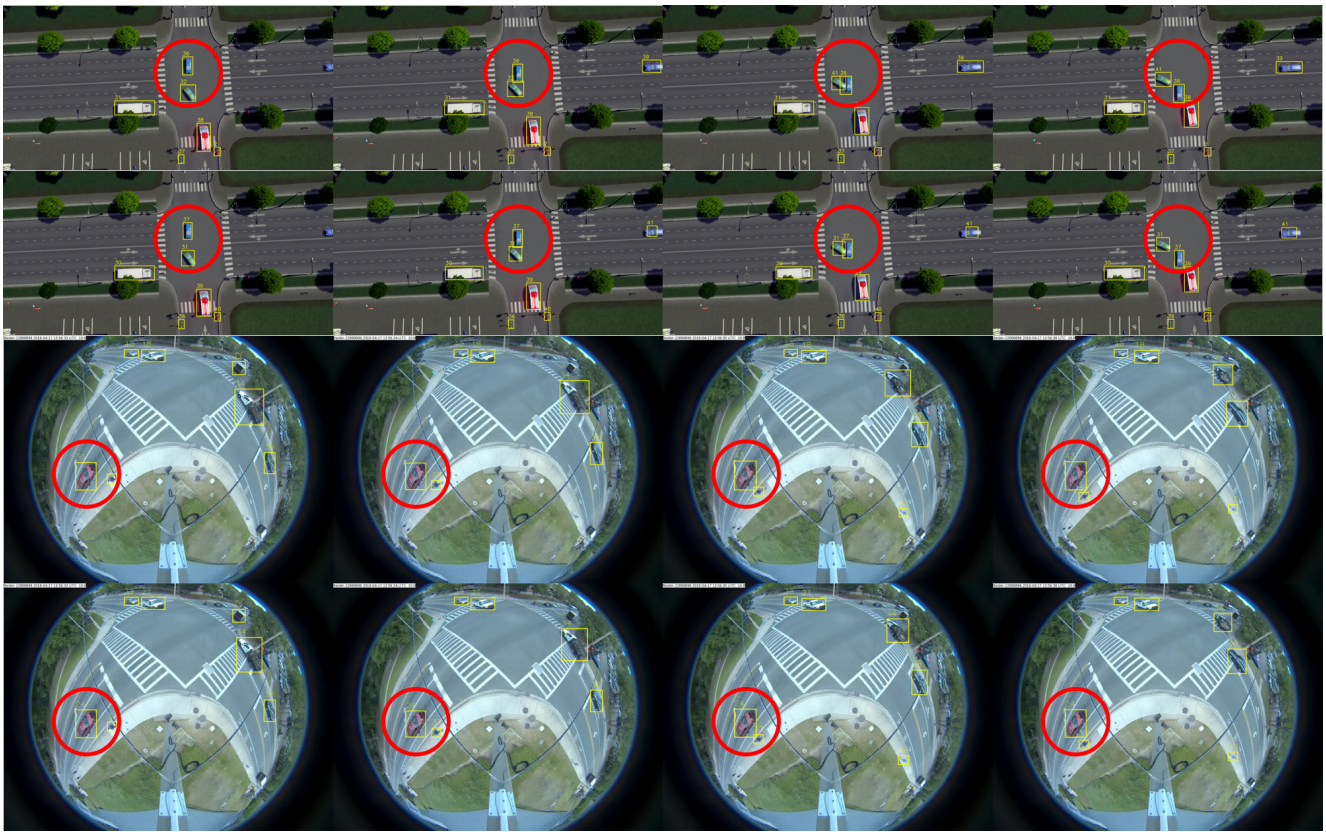


Figure 20: *Multiple object tracking comparisons of our baseline temporal stream (SORT) with cosine metric learning-based temporal stream. Row 1: results from SORT (ID switching issue); row 2: results from cosine metric learning + SORT (ID consistency); row 3: results from SORT (ID switching issue); row 4: results from cosine metric learning + SORT (ID consistency).*

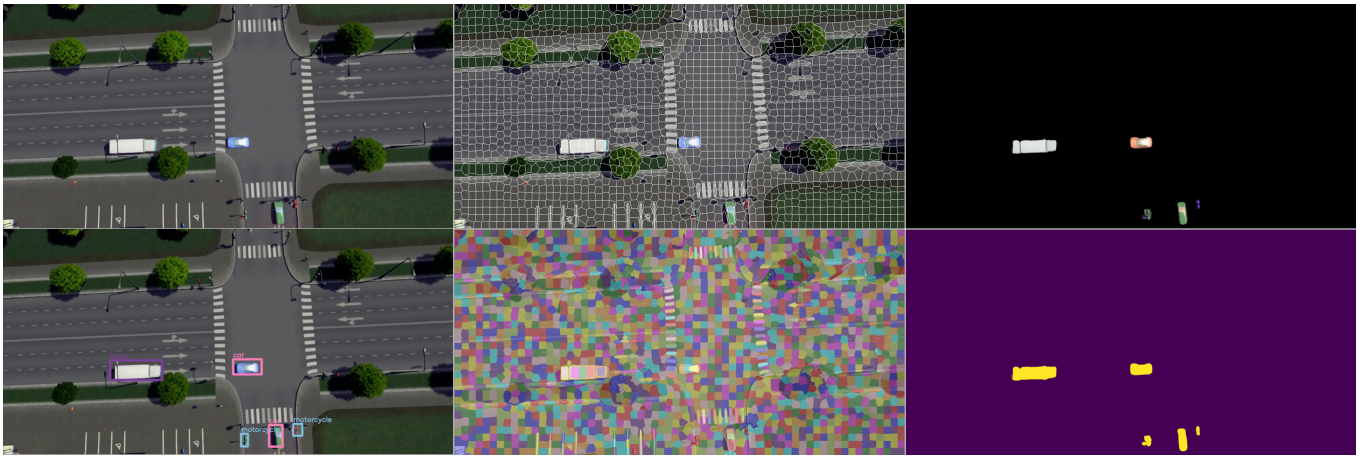


Figure 21: *Object segmentation using detections and superpixels. Top left: original image; top middle: superpixels generated by SLIC; top right: final object mask; bottom left: object detections; bottom middle: colored superpixels generated by SLIC; bottom right: final binary object mask.*

(Jodoin et al., 2014) and TrafficIntelligence (Jackson et al., 2013) can only track moving objects. On the other hand, Urban Tracker (Jodoin et al., 2014) and TrafficIntelligence (Jackson et al., 2013) can compute dense trajectories of moving objects with good accuracy, but they have slower tracking speed—around one frame per second. For accident detection, our two-stream convolutional networks are able to do spatial localization and temporal localization for diverse accident regions involving cars and motorbikes. The three subtasks (object detection, MOT, and near-accident detection) can always achieve real-time performance at a high frame rate—40 to 50 frames per second—and this depends on the frame resolution (e.g., 50 fps for 960×480 image frames).

In Figure 20, we demonstrate the effectiveness of applying cosine metric learning with the temporal stream for solving ID switching (and for the case where new object IDs emerge). For example, the first row shows that the ID of the green car (inside the red circle) got changed from 32 to 41. The third row shows that the IDs of the red car and a pedestrian (inside the red circle) were both changed (from 30 to 43 and from 38 to 44, respectively). The second and fourth row demonstrate that with cosine metric learning, the IDs of tracks are kept consistent. In Figure 21, we show an example of more accuracy and compact road user mask generated by superpixel segmentation and detections for the learning-based gap estimation. With segmentation mask, we can estimate the gap distance between road users in a more accurate way than directly using object detections. Segmentation would be more useful for fisheye videos where the distortion is large and occlusion is heavier. For near-accident detection (Figure 22), we present final near-accident detection results along with tracking and trajectories using our two-stream convolutional networks method. Overall, the qualitative results demonstrate the effectiveness of our spatial and temporal networks.

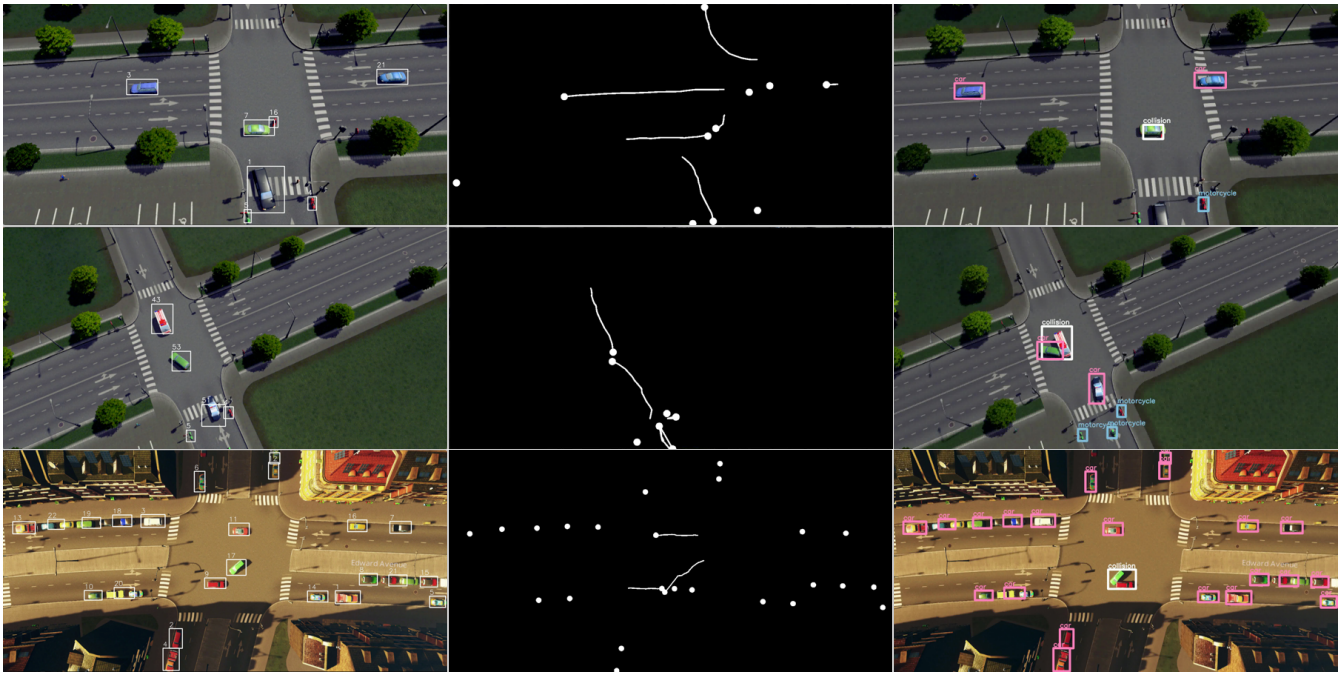


Figure 22: *Sample results of tracking, trajectory, and near-accident detection of our two-stream convolutional networks on simulation videos of TNAD dataset. Left: tracking results from the temporal stream; middle: trajectory results from the temporal stream; right: final near-accident detection results from the two-stream convolutional networks.*

5.9.5 Quantitative Results

Speed Performance. We present timing results for the tested methods in Table 1. All experiments have been performed on a single GPU (NVIDIA TITAN V). The GPU-based SLIC segmentation (Achanta et al., 2012) has excellent speed and runs from 110 fps to 400 fps on videos of different resolutions. The overall two-stream CNNs (object detection, MOT, and near-accident detection) can achieve real-time performance at a high frame rate, 33–50 fps, depending on the frame resolution.

Improved Multiple Object Tracking with Cosine Metric Learning. We present some quantitative results for gap estimation and cosine metric learning in Figure 23. The results for cosine metric learning have been established after training the network for a fixed number of steps (100, 000 iterations). The batch size was set to 120 images and the learning rate was 0.001. All configurations after training have fully converged as depicted in Figure 23.

5.9.6 Object Segmentation

For learning the gap threshold with superpixel segmentation and detections, the plot (top left) demonstrates the converging evolution of gap distance for three types of video. The initial gap threshold is set to be 50 pixels, the average gap thresholds from experiments are about 47 pixels, 45 pixels, and 42 pixels for simulation video, fisheye video, and drone video, respectively.

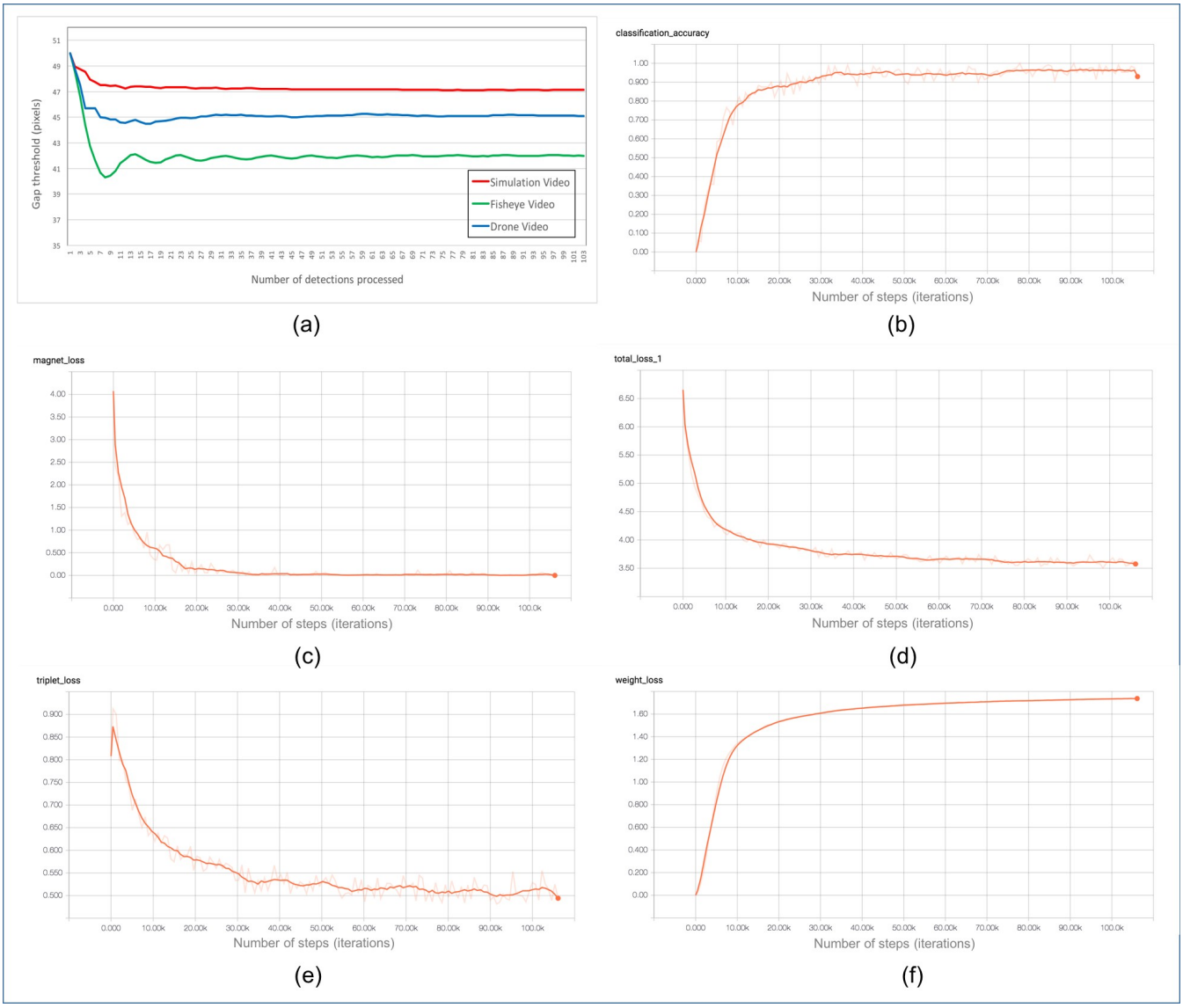


Figure 23: *Gap estimation and cosine metric learning: (a) evolution of gap threshold for three types of videos; (b) classification accuracy for training VeRi dataset; (c) evolution of total loss for training; (d) evolution of magnet loss for training; (e) evolution of triplet loss for training; (f) evolution of weight loss for training.*

Table 1: *Quantitative evaluation of timing results for the tested methods.*

Methods	GPU	Drone video	Simulation video		Fisheye video
		1920×960	2560×1280	3360×1680	1280×960
SLIC segmentation	NVIDIA TITAN V	300 fps	178 fps	110 fps	400 fps
Two-Stream CNNs	NVIDIA TITAN V	43 fps	38 fps	33 fps	50 fps

Table 2: *Quantitative evaluation of all tasks for our two-stream method.*

Task or Methods	Video Data	Precision	Recall	F1-score
Object Detection	Simulation	0.92670	0.95255	0.93945
	Fisheye	0.93871	0.87978	0.90829
Multiple Object Tracking	Simulation	0.89788	0.86617	0.88174
	Fisheye	0.91239	0.84900	0.87956
Multiple Object Tracking (cosine metric learning)	Simulation	0.91913	0.90310	0.91105
	Fisheye	0.92663	0.87725	0.90127
SLIC Segmentation Mask	Simulation	0.93089	0.81321	0.86808
Near-Accident Detection (Spatial Stream Only)	Simulation	0.90395	0.86022	0.88154
Near-Accident Detection (Temporal Stream Only)	Simulation	0.83495	0.92473	0.84108
Near-Accident Detection (Two-Stream)	Simulation	0.92105	0.94086	0.93085

The evaluation method we use for instance segmentation in this case calculates IoU of masks instead of bounding boxes.

Because our framework has three tasks and our dataset is quite different from other object detection datasets, tracking datasets, and near-accident datasets such as dashcam accident dataset (Chan et al., 2016), it is difficult to compare the individual quantitative performance for all three tasks with other methods. One of our motivations was to propose a vision-based solution for ITS; therefore, we focus more on near-accident detection and present quantitative analysis of our two-stream convolutional networks. In Table 2, we present quantitative evaluations of three subtasks: object detection; multiple object tracking (with and without cosine metric learning); SLIC segmentation (mask vs. bounding box); and near-accident detection (spatial stream only, temporal stream only, and two-stream model). For the fisheye video, we present evaluations regrading object detection and multiple object tracking. The simulation videos are for the purposes of training and testing with more near-accident samples, and we have 57 simulation videos totaling over 51,123 video frames. We sparsely sample only 1,087 frames from them for training processing. We present the analysis of near-accident detection for 30 testing videos (18 had positive near-accident; 12 had negative near-accident). To evaluate our prediction of near-accidents, we'll compare each of our predicted detection with each ground truth for a given input.

- A true positive is observed when a prediction-target detection pair has an IoU score which exceeds some predefined threshold (we set it to 0.7).
- A false positive indicates a predicted near-accident had no associated ground truth near-accident.
- A false negative indicates a ground truth near-accident had no associated predicted detections.

5.10 Conclusion

We have proposed a two-stream convolutional networks architecture that performs real-time detection, tracking, and near-accident detection for vehicles in traffic video data. The two-stream convolutional network comprises a spatial stream network and a temporal stream network. The spatial stream network detects

individual vehicles and likely near-accident regions at the single-frame level by capturing appearance features with a state-of-the-art object detection method. The temporal stream network leverages motion features of detected candidates to perform multiple object tracking and generates individual trajectories of each tracked target. We detect near-accidents by incorporating appearance features and motion features to compute probabilities of near-accident candidate regions. Experiments have demonstrated the advantage of our framework with an overall competitive qualitative and quantitative performance at high frame rates. Future work will include image stitching methods which will be deployed on multicamera fisheye videos.

6 Connected Vehicles

In this section, we will detail the progress and achieved objectives in our work with Siemens Mobility, Inc., and the City of Gainesville to develop software for collecting signal phasing and timing (SPAT) data from the intersections on a corridor. Siemens Mobility, Inc., furnished, installed, and integrated 27 roadside units (RSUs) at the 27 Trapezium signals. These RSUs were tested initially with six vehicles equipped with onboard units (OBUs). The RSUs and OBUs connect with each other using dedicated short range communication (DSRC) which is an 802.11p-based wireless communication technology operating in the 5.9-GHz band. DSRC enables highly secure, high-speed direct communication between vehicles and the surrounding infrastructure, without involving any cellular infrastructure.

DSRC enables the transmission of data at high speeds (over one-way or two-way short to medium range wireless channel) which is critical for communication-based active safety applications to prevent accidents. There are two types of DSRC: Vehicle-to-Vehicle(V2V) and Vehicle–to-Infrastructure (V2I). DSRC makes it possible to have a protected wireless interface constancy with short time delays and latency, while being highly robust under extreme weather conditions. Various traffic applications using DSRC will enhance public safety for normal vehicles and especially in connected vehicles, by providing an ideal setting for vehicle safety and mobility applications.

In the rest of this section, we will describe the software we developed for processing the signal phase and timing data received from the RSUs. The various steps were to connect to RSUs and download the upstream and downstream messages, store the relevant messages to a database in the cloud and also the development of an UI to display a summary of the messages received from the various RSUs and the connectivity status of each RSU. Each of these steps will be described in detail in this report.

6.1 Application Requirements

In this section, we present the application requirements for SPAT collection. The goal of the application for batch or real-time processing is to provide the user with the ability to download and use the signaling information at various intersections through RSUs built by Siemens Mobility, Inc.

6.1.1 Real-time Processing

The value of this DSRC-based application lies in the ability to support real-time processing. The messages carrying information about the signal phasing and timing information are received at a frequency of about 10 messages per second (10 Hz), which are processed as soon as we receive them to extract the current signaling state at an intersection. The functional requirements for the real-time processing are to maintain a live connection to the RSUs and continuously collect the upstream and downstream encoded messages. These messages after collection would be decoded and parsed. The most relevant fields will then be stored in the database. The application will support a user interface that will let the user see the basic statistics of the received messages.

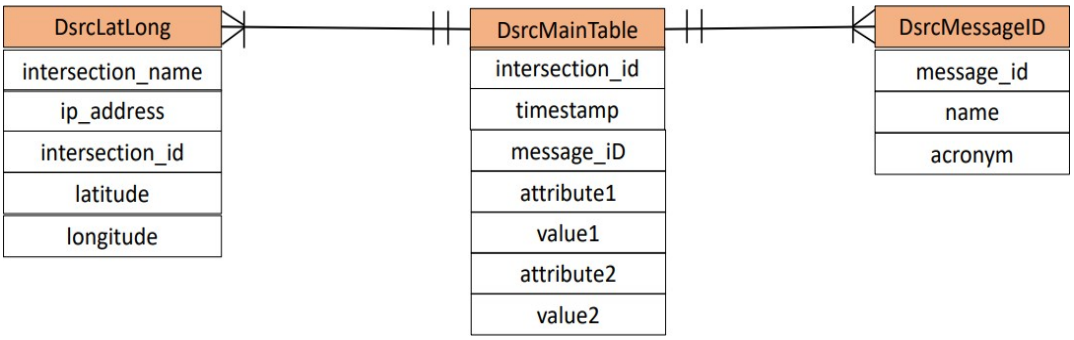


Figure 24: *Entity relationship diagram of the DSRC database.*

The application should be scalable to be able to receive messages from all the intersections in the city. The component of the software that will most likely be challenged in terms of reliability is the connection to an RSU. The connections made to the WebSocket could get disconnected occasionally. The application should be able to detect that situation and automatically try a reconnect. For security of the application, the computing device and the RSUs are behind the firewall at the CoG.

6.1.2 Batch Processing

Having noted that the real value of DSRC lies in real-time processing, we nevertheless describe the batch processing requirements which help us perform before-and-after studies. In this connection, it may be mentioned that we have an app that collects the signaling information from the advanced traffic controllers that capture signaling events at a 10-Hz frequency. Data may be collected from the controllers, with the smallest batch size being 1 minute. Historical values of the signal phases and timings may be downloaded by an application by directly connecting to the database.

The security of the application is ensured because the RSU and the computing device are behind the firewall at the CoG. The data collected is stored in AWS which helps us access historical data and apply the encryption algorithms for encryption of content at transit and at rest.

6.1.3 Imputing information

Though the WebSocket connection is reliable, there may be a few messages lost occasionally. Since SPAT is generated very frequently, at about ten messages per second, a few lost packets do not matter in our experience as long as we can continue processing the subsequent packets.

6.1.4 Entity-Relationship Diagram

An entity-relationship (ER) diagram of the DSRC tables is shown in Figure 24. There are three main entities described as follows: DsrcMainTable, which stores the messages received from an RSU; DsrcLatLong, which saves geographical latitudes and longitudes of intersections along with other properties such as intersection ID and IP address of the RSU unit at that intersection; DsrcMessageID, which stores the possible DSRC messages received by their ID which is an integer and their descriptive names. The DsrcMessageID table is

populated using the messages described in Section 7.40 of SAE J2735.

The ER diagram is drawn in Crow's foot notation where the entities are represented as boxes and relationships are represented as lines between the boxes. The relative cardinality of a relationship is described as different shapes at the ends of these lines. In this diagram, a vertical line represents one; a pair of perpendicular lines represent one and only one; and a crow's foot, or two forked lines, represent many. Thus, a DsrcMainTable may store DSRC messages received from many intersections with many different message IDs. A particular message with a specific message ID and from a certain intersection is stored in one and only one DsrcMainTable.

6.2 Receiving Messages from RSUs

There are two components to the receiver that receives messages from an RSU: a Java-based receiver and a Python-based message interpreter. We reuse the code Siemens developed for the Java-based receiver and developed the Python interpreter.

The Java message receiver is used to connect to the RSU WebSocket to send subscription to and receive message wave from the RSU. The receiver will read a script file with a message received on each line and a custom control command. The receiver sends the subscription command to specify reception of messages from both upstream and downstream message flows relative to the RSU. The code uses the "!wait" command to let the WebSocket stay connected to receive the message wave from the RSU.

6.3 Processing Messages Received from RSUs

The Python interpreter, upon receiving the messages, first extracts the byte-encoded XML message from the raw message wave and decodes it. The syntax and semantics of the XML message may be found in standard SAE J2735.

In this report, we focus on the SPAT messages. For the other message types, we extract the attribute values for two of the most important attributes and store each message in a row.

```
mysql> select * from DsrcMainTable where dsrcMsgID=19 and intersection_id=7359 limit 10;
+-----------------+----------------------------+-----------+------------+--------+------------+--------+
| intersection_id | timestamp                  | dsrcmsgID | attribute1 | value1 | attribute2 | value2 |
+-----------------+----------------------------+-----------+------------+--------+------------+--------+
|            7359 | 2020-05-24 18:43:49.548000 |        19 | eventState | 1122cc | revision   | 39     |
|            7359 | 2020-05-24 18:43:49.808000 |        19 | eventState | 0033cc | revision   | 41     |
|            7359 | 2020-05-24 18:43:53.206000 |        19 | eventState | 0000ff | revision   | 71     |
|            7359 | 2020-05-24 18:43:56.403000 |        19 | eventState | 840873 | revision   | 99     |
|            7359 | 2020-05-24 18:44:04.396000 |        19 | eventState | 048873 | revision   | 42     |
|            7359 | 2020-05-24 18:44:08.532000 |        19 | eventState | 0408f3 | revision   | 79     |
|            7359 | 2020-05-24 18:44:10.493000 |        19 | eventState | 448833 | revision   | 97     |
|            7359 | 2020-05-24 18:45:42.571000 |        19 | eventState | 00cc33 | revision   | 18     |
|            7359 | 2020-05-24 18:45:46.705000 |        19 | eventState | 0000ff | revision   | 55     |
|            7359 | 2020-05-24 18:45:48.805000 |        19 | eventState | 2102dc | revision   | 73     |
+-----------------+----------------------------+-----------+------------+--------+------------+--------+
10 rows in set (0.04 sec)
```

Figure 25: *Table showing SPAT messages for intersection 7359. The first entry is '1122cc' which encodes a condition where phases 4 and 8 are green, whereas phases 3 and 7 are yellow (permissive movement allowed), the rest of the phases are red. The attribute2 is unimportant for SPAT messages.*

6.3.1 SPAT

The SPAT messages contain the signal status for each approach, including their current state and the remaining times for the next change. The state of a signal in the J2735 format is one among 'stop-and-remain', 'protected-movement-allowed', 'permissive-movement-allowed', 'protected-clearance', or 'permissive-clearance'.

In our application, we store only the first SPAT message after a signal change and ignore the rest of the messages that just contain remaining time for a signal state. We encode the signal state information in a SPAT message using three octets (a 24-bit binary number) and store its hexadecimal equivalent. The first, second, and third octets are reserved for recording green, yellow, and red status, respectively. For example, if phases 2 and 6 are green and every other signal is red, then, the encoding is 0100 0100 0000 0000 1011 1011. The equivalent hexadecimal representation is 4400bb.

Further, upon connecting to an RSU, we only process the downstream SPAT messages that contain information about the intersection where the RSU is installed and ignore the upstream SPAT messages as they contain information about the neighboring intersections.

6.3.2 Cloud Database

We store the SPAT data along with the other DSRC messages received by the RSU in a MySQL database. The table to store SPAT and the other RSU messages store two attributes per message.

Figure 25 shows an example query to the table to display the SPAT messages. The message ID for SPAT messages as assigned in SAE J2735 is 19. The first entry, '1122cc', encodes a condition where phases 4 and 8 are green, whereas phases 3 and 7 are yellow (permissive movement allowed). The rest of the phases are all red. The second entry registered almost immediately is '0033ff' which encodes a condition of yellow on phases 3, 4, 7, and 8 and red on phases 1, 2, 5, and6. Similarly, Figures 26 and 27 show respectively

```
mysql> select * from DsrcLatLong limit 10;
+---------------------------------+--------------+-----------------+------------------+-------------------+
| intersection_name               | ip           | intersection_id | latitude         | longitude         |
+---------------------------------+--------------+-----------------+------------------+-------------------+
| SW 2nd Ave @ 13th St - FYA      | 10.23.51.11  |            5360 | 29.6502523325507 | -82.3393195764231 |
| SW 5th Ave/Inner Rd @ 13th St   | 10.23.51.31  |            5660 | 29.6473291784845 | -82.3393261805068 |
| SW 8th Ave @ 13th St - FYA      | 10.23.51.41  |            5960 | 29.6448944828607 | -82.3393181808896 |
| SW Archer Rd @ Newell Dr - FYA  | 10.23.51.71  |            7359 | 29.6397045628961 | -82.3418188124768 |
| SW Archer Rd @ 18th St - FYA    | 10.23.51.91  |            7357 | 29.6382287432322 | -82.3460025921795 |
| SW Archer Rd @ 23rd Dr - FYA    | 10.23.61.41  |            7353 | 29.6336693457120 | -82.3587624232869 |
| SW 2nd Ave @ 34th St            | 10.23.81.101 |            5350 | 29.6504017946502 | -82.3723638055293 |
| SW Archer Rd @ 34th St          | 10.23.81.51  |            7350 | 29.6269354535710 | -82.3724968751956 |
| Windmeadows Blvd @ 34th St - FYA | 10.23.81.61 |            7250 | 29.6283746553919 | -82.3725925377656 |
| SW 20th Ave @ 34th St - FYA     | 10.23.81.71  |            7050 | 29.6342806512682 | -82.3725326069981 |
+---------------------------------+--------------+-----------------+------------------+-------------------+
10 rows in set (0.01 sec)
```

Figure 26: *A table storing the details about the intersections.*

the tables storing intersections in Gainesville, FL, and the possible types of DSRC messages.

The DsrcMainTable contains all the received messages. However, to keep the table from growing too fast, we count the MAP and unique TIM messages only once over a five-minute interval. Similarly, we store SPAT only when a signal changes state, and for BSM/PSM we store only one message per unique device ID over a five-minute interval. The interval is parameterized in the code as a heartbeat interval and may be changed by the user to any other preferred value. Further, we only store those SPAT, MAP, and TIM messages that are transmitted by a RSU, and not necessarily those that are received by a RSU.

```
mysql> select * from DsrcMessageID limit 10;
+------+----------------------+----------+
| id   | name                 | acronym  |
+------+----------------------+----------+
|   18 | mapData              | MAP      |
|   19 | signalPhaseAndTiming | SPAT     |
|   20 | basicSafetyMessage   | BSM      |
|   21 | commonSafetyRequest  | CSR      |
|   22 | emergencyVehicleAler | EVA      |
|   23 | intersectionCollisio | ICA      |
|   24 | nmeaCorrections      | NMEA     |
|   25 | probeDataManagement  | PDM      |
|   26 | probeVehicleData     | PVD      |
|   27 | roadSideAlert        | RSA      |
+------+----------------------+----------+
10 rows in set (0.00 sec)
```

Figure 27: *A table storing the possible DSRC message types.*

6.4 Basic User Interface

We developed a user interface to count the number of messages received from each RSU for each message type. Figure 28 shows a sample of the UI, and the salient features of the UI are described as follows:

- The intersections are marked as red, green, or yellow, depending on whether the RSU at the signal was respectively online, offline during the entire time, or if the signal was a combination of online and offline during that duration.
- The user may select the time interval of interest by specifying the start and end time of the interval in the two boxes on the left-hand side of the page.
- After specifying the start and end times, the user must click on submit so that the app may initiate a query to the database, calculate the number and type of messages received during the user specified

time interval.

- The number of distinct intersections in the chosen duration are displayed in the blue box that says ‘Number of Intersections’.
- The last box on the left displays the start and end times of the available data.
- Upon hovering the mouse over each intersection marker, a box pops up that contains the number of each message types received. A message type named ‘testConnection’ is the only non-DSRC message that we added to be able to report the status of the connection after pinging the corresponding RSUs.

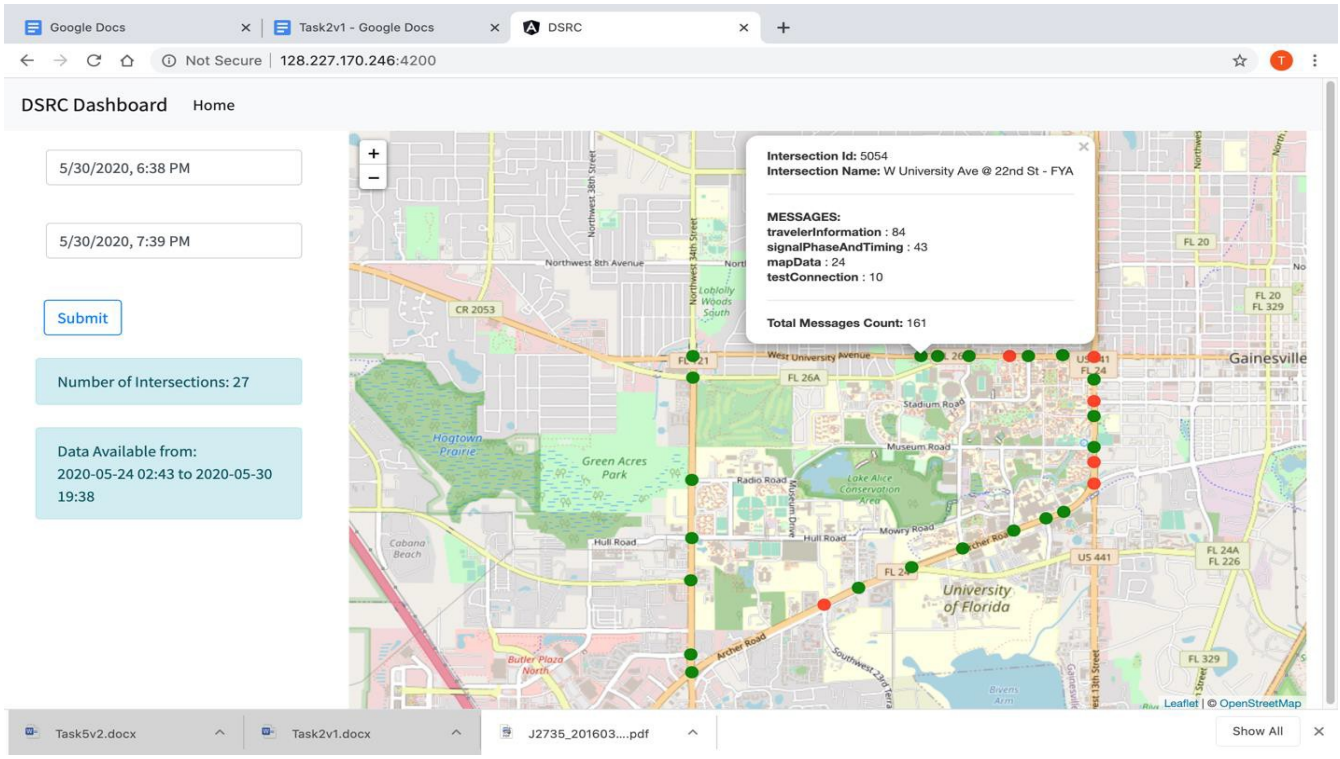


Figure 28 *UI for tracking whether RSUs at intersections are online or offline and a summary of the messages received from each RSU.*

Analysis of the Number of Received Messages

A pie chart of the number of received messages is presented in Figures 29 and 30 for the morning and afternoon times, respectively. In counting these messages, we use the log files to calculate the total number of transmitted and received messages of a given type at an intersection. The morning interval did not see any PSMs, while the afternoon interval did see a few, but their number was so small that the percent of the pie was nearly zero. This distribution is during COVID-19 and may change substantially post-COVID-19.

6.5 Performance

In this section, we present performance, latency, and storage requirements for our application.

6.5.1 Batch Performance

Currently, a query for one day's worth of data for all 21 intersections that had online RSUs resulted in the download of about 130,000 rows (or messages) on an average, and this download takes about 2.06 seconds. It is to be noted that the actual number of rows and messages will vary based on the activity at the intersections and is expected to increase post-Covid-19.

6.5.2 Real-time Performance

We measure the performance along the following dimensions:

1. Number of queries: Currently, the database supports about three queries per second. Once all the RSUs are up and our UI system is fully functional, we expect the number of queries to increase, and that would reflect the true performance of the database.

2. Network bandwidth: Each RSU connection, depending on the activity at the corresponding intersection, requires a bandwidth of between 50-150 kbps, with an average between 70-80 kbps.

3. Latency: The difference between the time a message is generated at an RSU and stored in our database is about 0.725 seconds.

4. Storage: The size of the main table of the database increases at a rate of about 500 MB per month.

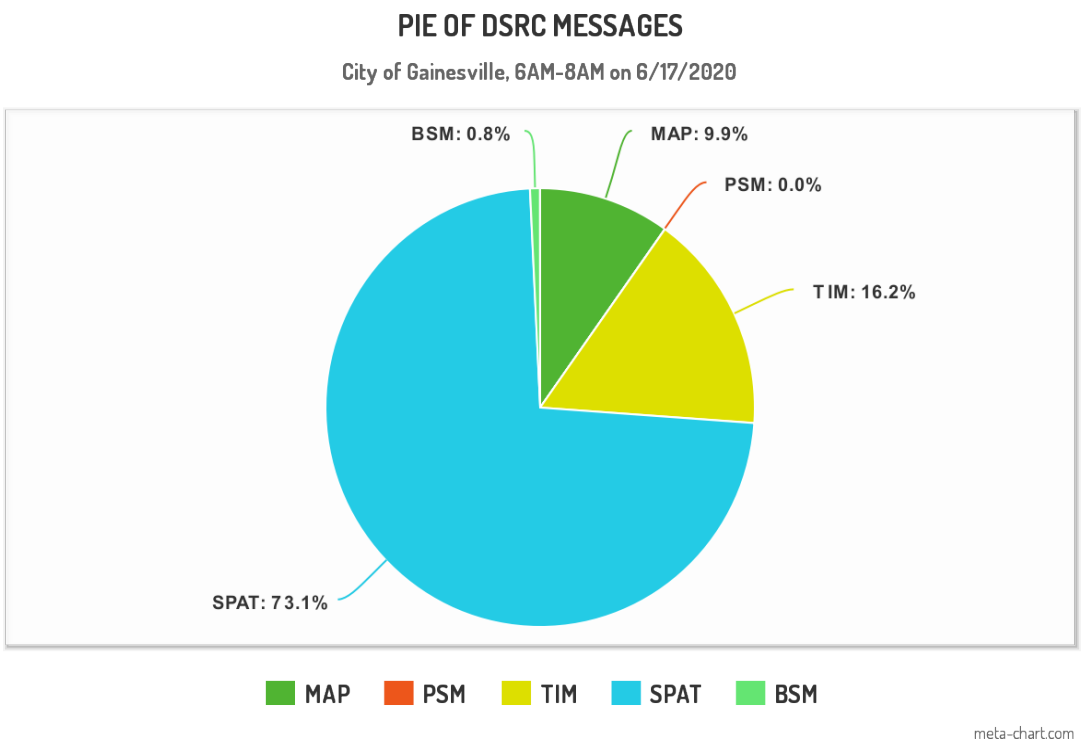


Figure 29 *Distribution of messages collected in the morning between 6AM–8AM.*

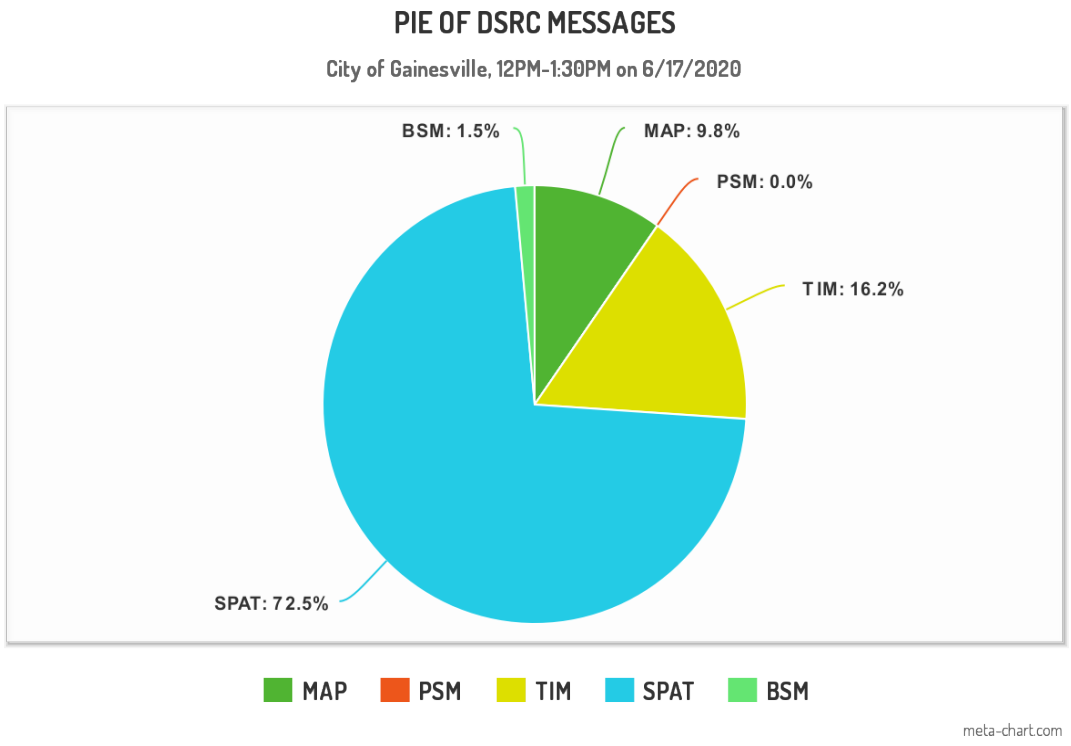


Figure 30 *Distribution of messages collected in the morning between 12PM–1:30PM.*

6.6 Conclusion

In this section, we detailed the progress and achieved objectives in our work with Siemens Mobility, Inc., and the City of Gainesville to develop software for collecting SPAT data from the intersections on a corridor. We described the software we developed to collect various DSRC messages received from the RSUs, with a particular emphasis on the application to collect, process SPAT messages, and process them to derive the signal phase and timing data for the intersections. The app we developed in this project to collect SPAT data may be used by other applications that require signal phase and timing data for data analysis.

The various steps to get the data were described: connect to RSUs and download the upstream and downstream messages; store the relevant messages to a database in the cloud; and develop a user interface to display a summary of the messages received from the various RSUs and the connectivity status of each RSU.

We presented the database schema used to store the information received from the RSUs and an analysis of the number and types of messages received. We also did a study of batch and real-time performance analysis of our application and show that batch queries on the table take about a couple of seconds. In contrast, real-time queries finish under a second.

7 Conclusions

The collaboration between University of Florida (UF) and its Transportation Institute (UFTI), the Florida Department of Transportation (FDOT) and the City of Gainesville (CoG) have developed a smart testbed on the UF campus and adjoining city streets. The testbed is established to deploy and evaluate numerous advanced technologies, including connected and autonomous vehicles, smart devices, and sensors, as well as to develop novel applications for their use. These technologies and their application will work within the existing network and will accommodate the presence of conventional vehicles. In order to facilitate the development of the testbed, we described the transportation datasets we collect and use and outlined the application requirements. At a high level, the applications must be scalable as the number and type of sensors are growing exponentially. Further, for better performance and maintainability, the applications must be supported on distributed and decentralized systems. The applications should be able to process data in real-time and apply intelligent and dynamic analytics. Further, the applications must address all security and privacy issues.

After outlining the datasets and requirements for applications using these datasets, we described the computing architecture for these applications. Given the nature of the applications we deploy a fully fledged stack of edge servers, local servers, and cloud servers to run our application pipelines. We presented our overall architecture which includes the edge, local, and cloud components. We described the architectures of our ATSPM analysis, video analysis, Siemens RSU, and Bluetoad data pipelines.

Subsequently, we described the sensors and data sources that are installed in modern urban road networks. We also described ways to convert this raw data into performance measures. We developed algorithms and models that leverage machine learning methodologies for data collected from a large number of intersections to derive key spatiotemporal traffic patterns in a city. We further summarized the main issues with the high resolution logs and described key data cleaning techniques. We described an intersection ranking, clustering and change detection techniques and how to use these to build a decision support system.

As a part of our video analysis pipeline, we have proposed a two-stream convolutional networks architecture that performs real-time detection, tracking, and near-accident detection for vehicles in traffic video data. The two-stream convolutional network comprises a spatial stream network and a temporal stream network. The spatial stream network detects individual vehicles and likely near-accident regions at the single-frame level by capturing appearance features with a state-of-the-art object detection method. The temporal stream network leverages motion features of detected candidates to perform multiple object tracking and generates individual trajectories of each tracked target. We detect near-accidents by incorporating appearance features and motion features to compute probabilities of near-accident candidate regions. Experiments have demonstrated the advantage of our framework with an overall competitive qualitative and quantitative performance at high frame rates. Future work will include image stitching methods which will be deployed on multi-camera fisheye videos.

As part of our work to support connected vehicles, we detailed the progress and achieved objectives in our work with Siemens Mobility, Inc., and the City of Gainesville to develop software for collecting SPAT data from the intersections on a corridor. We described the software we developed to collect various DSRC messages received from the RSUs, with a particular emphasis on the application to collect, process SPAT messages, and process them to derive the signal phase and timing data for the intersections. The app we developed in this project to collect SPAT data may be used by other applications that require signal phase and timing data for data analysis. We presented the database schema used to store the information received from the RSUs, and an analysis of the number and types of messages received. We also did a study of batch and real-time performance analysis of our application and show that batch queries on the table take about a couple of seconds. In contrast, real-time queries finish under a second.

References

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2012). SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282.

Allström, A., Barcelo, J., Ekström, J., Grumert, E., Gundlegård, D., and Rydergren, C. (2017). Traffic management for smart cities. In Angelakis, V., Tragos, E., Pöhls, H.C., Kapovits, A., Bassi, A. (Eds.), *Designing, Developing, and Facilitating Smart Cities: Urban Design to IoT Solutions*, Springer Publishing Co., New York, NY. Pp. 211–240.

Amini, S., Gerostathopoulos, I., and Prehofer, C. (2017). Big data analytics architecture for real-time traffic control. In *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, IEEE, New York, NY. Pp. 710–715.

Angel, A., Hickman, M., Mirchandani, P., and Chandnani, D. (2002). Methods of traffic data collection, using aerial video. In *Proceedings of the IEEE 5th International Conference on Intelligent Transportation Systems*, IEEE, New York, NY. Pp. 31–36.

Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2011). Contour detection and hierarchical image segmentation.

IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(5):898–916.

Argote-Cabanero, J., Christofa, E., and Skabardonis, A. (2015). Connected vehicle penetration rate for estimation of arterial measures of effectiveness. *Transportation Research Part C: Emerging Technologies*, 60:298–312.

Belkin, M., and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396.

Bewley, A., Ge, Z., Ott, L., Ramos, F., and Upcroft, B. (2016). Simple online and real-time tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, New York, NY. Pp. 3464–3468.

Buch, N., Velastin, S. A., and Orwell, J. (2011). A review of computer vision techniques for the analysis of urban traffic. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):920–939.

Chan, F.-H., Chen, Y.-T., Xiang, Y., and Sun, M. (2016). Anticipating accidents in dashcam videos. In *Asian Conference on Computer Vision*, Springer, New York, NY. Pp. 136–153.

Chen, L., Cao, Y., and Ji, R. (2010). Automatic incident detection algorithm based on support vector machine. In

2010 Sixth International Conference on Natural Computation, volume 2, IEEE, New York, NY. Pp. 864–866.

Chen, Y., Yu, Y., and Li, T. (2016). A vision-based traffic accident detection method using extreme learning machine. In *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*, IEEE, New York, NY. Pp. 567–572.

Coifman, B., Beymer, D., McLauchlan, P., and Malik, J. (1998). A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research Part C: Emerging Technologies*, 6(4):271–288.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Proceedings of the IEEE*

conference on Computer Vision and Pattern Recognition, volume 1, . IEEE, New York, NY. Pp. 886–893

Day, C., and Bullock, D. (2010). *Arterial Performance Measures, Volume 1: Performance Based Management of Arterial Traffic Signal Systems (Final Report, NCHRP 3-79A)*. Transportation Research Board, Washington, D.C.

Day, C., Bullock, D., Li, H., Remias, S., Hainen, A., Freije, R., Stevens, A., Sturdevant, J., and Brennan, T. (2014). *Performance Measures for Traffic Signal Systems: An Outcome-Oriented Approach*. Purdue University, West Lafayette, IN.

Day, C. M., Bullock, D. M., Li, H., Lavrenz, S., Smith, W. B., and Sturdevant, J. R. (2016). *Integrating Traffic Signal Performance Measures into Agency Business Processes*. Purdue University, West Lafayette, IN.

Day, C. M., Remias, S. M., Li, H., Mekker, M., McNamara, M. L., Cox, E. D., and Bullock, D. M. (2015). *Performance Ranking of Arterial Corridors Using Travel Time and Travel Time Reliability Metrics*. Purdue University, West Lafayette, IN.

Denney, R., Curtis, E., and Olson, P. (2012). The national traffic signal report card. *ITE Journal*, 82(6):22–26.

Ehrlich, D., Lacey, P., Colwell, C., Byer, M., et al. (1994). *Transportation Infrastructure: Benefits of Traffic Control Systems Are Not Being Fully Realized (Report RCED-94-105)*. U.S. Government Accountability Office, Washington, D.C.

Federal Highway Administration. (2008). Traffic signal timing manual (FHWA-HOP-08-024). Federal Highway Administration, Washington, D.C. https://ops.fhwa.dot.gov/publications/fhwahop08024/index.htm. (Accessed on 2/10/2020).

Fedele, R., Praticò, F., Carotenuto, R., and Della Corte, F. (2017). Instrumented infrastructures for damage detection and management. In *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, IEEE, New York, NY. Pp. 526–531.

Ferman, M., Blumenfeld, D., and Dai, X. (2005). An analytical evaluation of a real-time traffic information system using probe vehicles. *Journal of Intelligent Transportation Systems*, 9:23–34.

Freije, R., Hainen, A., Stevens, A., Li, H., Smith, W., Summers, H., Day, C., Sturdevant, J., and Bullock, D. (2014). Graphical performance measures for practitioners to triage split failure trouble calls. *Transportation Research Record*, 2439:27–40.

Gettman, D., Fok, E., Curtis, E., Kacir, K., Ormand, D., Mayer, M., and Flanigan, E. (2013). *Measures of Effectiveness and Validation Guidance for Adaptive Signal Control Technologies*. Federal Highway Administration, Washington, D.C.

Ghosh-Dastidar, S., and Adeli, H. (2003). Wavelet-clustering-neural network model for freeway incident detection.

Computer-Aided Civil and Infrastructure Engineering, 18(5):325–338.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2016). Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):142–158.

Guerrero-Ibañez, J., Zeadally, S., and Contreras Castillo, J. (2018). Sensor technologies for intelligent transportation systems. *Sensors*, 18:1212.

He, P., Huang, W., He, T., Zhu, Q., Qiao, Y., and Li, X. (2017). Single shot text detector with regional attention. In *Proceedings of the IEEE International Conference on Computer Vision*, IEEE, New York, NY. Pp. 3047–3055.

Hommes, S., State, R., Zinnen, A., and Engel, T. (2011). Detection of abnormal behaviour in a surveillance environment using control charts. In *2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, IEEE, New York, NY. Pp. 113–118.

Hoose, N. (1992). Impacts: an image analysis tool for motorway surveillance. *Traffic Engineering & Control*, 33(3):140–147.

Horvitz, E., Apacible, J., Sarin, R., and Liao, L. (2005). Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, ACM, New York, NY. Pp. 275–283.

Hu, J., de la Fontaine, M., and Ma, J. (2016). Quality of private sector travel-time data on arterials. *Journal of Transportation Engineering*, 142(4). Article 04016010. 11 pp.

Huang, X., Yang, C., Ranka, S., and Rangarajan, A. (2018). Supervoxel-based segmentation of 3D imagery with optical flow integration for spatiotemporal processing. *IPSJ Transactions on Computer Vision and Applications*, 10(1). Article 9.

Hui, Z., Yaohua, X., Lu, L., and Jiansheng, F. (2014). Vision-based real-time traffic accident detection. In *Proceedings of the 11th World Congress on Intelligent Control and Automation*, IEEE, New York, NY. Pp. 1035–1038.

Hunter, T., Abbeel, P., and Bayen, A. (2014). The path inference filter: Model-based low-latency map matching of probe vehicle data. *IEEE Transactions on Intelligent Transportation Systems*, 15(2):507–529.

Ihaddadene, N. and Djeraba, C. (2008). Real-time crowd motion analysis. In *2008 19th International Conference on Pattern Recognition*, IEEE, New York, NY. Pp. 1–4.

Institute of Transportation Engineers. Advanced transportation controller. http://www.ite.org/standards/atc/. (Accessed on 12/10/2019).

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning, Volume 37*, page 448–456.

Jackson, S., Miranda-Moreno, L. F., St-Aubin, P., and Saunier, N. (2013). Flexible, mobile video camera system and open source video analysis software for road safety and behavioral analysis. *Transportation Research Record*, 2365(1):90–98.

Jampani, V., Sun, D., Liu, M.-Y., Yang, M.-H., and Kautz, J. (2018). Superpixel sampling networks. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer International Publishing, New York, NY. Pp. 352–368.

Jiang, F., Wu, Y., and Katsaggelos, A. K. (2007). Abnormal event detection from surveillance video by dynamic hierarchical clustering. In *2007 IEEE International Conference on Image Processing*, volume 5, IEEE, New York, NY. Pp. 145–148.

Jodoin, J., Bilodeau, G., and Saunier, N. (2016). Tracking all road users at multimodal urban traffic intersections.

IEEE Transactions on Intelligent Transportation Systems, 17(11):3241–3251.

Jodoin, J.-P., Bilodeau, G.-A., and Saunier, N. (2014). Urban tracker: Multiple object tracking in urban mixed traffic.

In *IEEE Winter Conference on Applications of Computer Vision*, IEEE, New York, NY. Pp. 885–892.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45.

Kamijo, S., Matsushita, Y., Ikeuchi, K., and Sakauchi, M. (2000). Traffic monitoring and accident detection at intersections. *IEEE Transactions on Intelligent Transportation Systems*, 1(2):108–118.

Karim, A., and Adeli, H. (2002). Incident detection algorithm using wavelet energy representation of traffic patterns.

Journal of Transportation Engineering, 128(3):232–242.

Koller, D., Daniilidis, K., and Nagel, H.-H. (1993). Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer*, 10(3):257–281.

Kotwal, A. R., Lee, S.-J., and Kim, Y. J. (2013). Traffic signal systems: A review of current technology in the United States. *Science and Technology*, 3(1):33–41.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report, Self-published.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

Li, H., M. Richardson, L., Day, C., Howard, J., and Bullock, D. (2017). Scalable dashboard for identifying split failures and heuristic for reallocating split times. *Transportation Research Record*, 2620:83–95.

Liu, C., Wang, G., Ning, W., Lin, X., Li, L., and Liu, Z. (2010). Anomaly detection in surveillance video using motion direction statistics. In *2010 IEEE International Conference on Image Processing*, IEEE, New York, NY. Pp. 717–720.

Liu, X., Liu, W., Ma, H., and Fu, H. (2016). Large-scale vehicle re-identification in urban surveillance videos. In *2016 IEEE International Conference on Multimedia and Expo (ICME)*, IEEE, New York, NY. Pp. 1–6.

Luo, W., Xing, J., Milan, A., Zhang, X., Liu, W., Zhao, X., and Kim, T.-K. (2014). Multiple object tracking: A literature review. *arXiv preprint arXiv:1409.7618*.

McLauchlan, P., Beymer, D., Coifman, B., and Mali, J. (1997). A real-time computer vision system for measuring traffic parameters. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, New York, NY. Pp. 495–501.

Middleton, D., Charara, H., and Longmire, R. (2009). Alternative vehicle detection technologies for traffic signal systems. Texas Transportation Institute, College Station, TX.

Mimbela, L. and Klein, L. (2007). *A summary of vehicle detection and surveillance technologies used in intelligent transportation systems*. The Vehicle Detector Clearinghouse, Las Cruces, NM.

National Electrical Manufacturers Association (NEMA) (2005). *National Transportation Communications for ITS Protocol Object Definitions for Actuated Traffic Signal Controller (ASC) Units – Version 02*. NEMA, Washington, D.C.

Nwizege, K. S., Bottero, M., Mmeah, S., and Nwiwure, E. D. (2014). Vehicles–to-infrastructure communication safety messaging in DSRC. *Procedia Computer Science*, 34:559–564.

Ohe, I., Kawashima, H., Kojima, M., and Kaneko, Y. (1995). A method for automatic detection of traffic incidents using neural networks. In *Pacific Rim TransTech Conference. 1995 Vehicle Navigation and Information Systems Conference Proceedings, 6th International VNIS: A Ride into the Future*, IEEE, New York, NY. Pp. 231–235.

Rahmani, M. and Koutsopoulos, H. (2013). Path inference from sparse floating car data for urban networks. *Transportation Research Part C: Emerging Technologies*, 30:41–54.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, IEEE, New York, NY. Pp. 779–788.

Redmon, J., and Farhadi, A. (2017). Yolo9000: better, faster, stronger. *arXiv:1612.08242*.

Remias, S., Hainen, A., Day, C., Brennan, T., Li, H., Rivera-Hernandez, E., Sturdevant, J., Young, S., and Bullock, D. (2013). Performance characterization of arterial traffic flow with probe vehicle data. *Transportation Research Record*, 2380:10–21.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems,* volume 28, pages 91–99. MIT Press, Cambridge, MA.

Rose, G. (2006). Mobile phones as traffic probes: Practices, prospects and issues. *Transport Reviews*, 26:275–291.

Roy, A., Gale, N., and Hong, L. (2011). Automated traffic surveillance using fusion of Doppler radar and video information. *Mathematical and Computer Modelling*, 54(1):531–543.

Sadeky, S., Al-Hamadiy, A., Michaelisy, B., and Sayed, U. (2010). Real-time automatic traffic accident recognition using HFG. In *2010 20th International Conference on Pattern Recognition*, IEEE, New York, NY. Pp. 3348–3351.

Salvo, G., Caruso, L., Scordo, A., Guido, G., and Vitale, A. (2017). Traffic data acquirement by unmanned aerial vehicle. *European Journal of Remote Sensing*, 50(1):343–351.

Scotti, G., Marcenaro, L., Coelho, C., Selvaggi, F., and Regazzoni, C. (2005). Dual camera intelligent sensor for high definition 360 degrees surveillance. volume 152, IEEE, New York, NY. Pp. 250–257.

Sharifi, E., Young, S., Eshragh, S., Hamedi, M., Juster, R., and Kaushik, K. (2013). Quality assessment of outsourced probe data on signalized arterials: Nine case studies in mid-Atlantic region. In *TRB 95th Annual Meeting Compendium of Papers*, volume 2380. Transportation Research Board, Washington, D.C.

Srinivasan, D., Jin, X., and Cheu, R. L. (2004). Evaluation of adaptive neural network models for freeway incident detection. *IEEE Transactions on Intelligent Transportation Systems*, 5(1):1–11.

Sunkari, S., Charara, H., and Songchitruksa, P. (2012). Portable toolbox for monitoring and evaluating signal operations. *Transportation Research Record*, 2311:142–151.

Tang, S. and Gao, H. (2005). Traffic-incident detection-algorithm based on nonparametric regression. *IEEE*

Transactions on Intelligent Transportation Systems, 6(1):38–42.

Tao, S., Manolopoulos, V., Rodriguez, S., and Rusu, A. (2012). Real-time urban traffic state estimation with a-GPS mobile phones as probes. *Journal of Transportation Technologies*, 2:22–31.

Tian, Z., Huang, W., He, T., He, P., and Qiao, Y. (2016). Detecting text in natural image with connectionist text proposal network. In *European Conference on Computer Vision*, Springer, New York, NY. Pp. 56–72.

TrafficCast. *Bluetoad spectra*. http://www.trafficcast.com/bluetoad.html. (Accessed on 12/10/2019).

Utah Department of Transportation (UDOT) (2017). UDOT automated traffic signal performance measures. https://udottraffic.utah.gov/atspm/. (Accessed on 2/26/2020).

Valera, M., and Velastin, S. A. (2005). Intelligent distributed surveillance systems: a review. *IEE Proceedings: Vision, Image and Signal Processing*, 152(2):192–204.

Veeraraghavan, H., Masoud, O., and Papanikolopoulos, N. P. (2003). Computer vision algorithms for intersection monitoring. *IEEE Transactions on Intelligent Transportation Systems*, 4(2):78–89.

Walton, C. M., Persad, K., Wang, Z., Svicarovich, K., Conway, A., and Zhang, G. (2009). *Arterial Intelligent Transportation Systems*. Center for Transportation Research, Austin, TX.

Wang, L. and Dong, M. (2012). Real-time detection of abnormal crowd behavior using a matrix approximation-based approach. In *2012 19th IEEE International Conference on Image Processing*, IEEE, New York, NY. Pp. 2701–2704.

Wang, M.-L., Huang, C.-C., and Lin, H.-Y. (2006). An intelligent surveillance system based on an omnidirectional vision sensor. In *2006 IEEE Conference on Cybernetics and Intelligent Systems*, IEEE, New York, NY. Pp. 1–6.

Wang, S. and Miao, Z. (2010). Anomaly detection in crowd scene. In *IEEE 10th International Conference on Signal Processing Proceedings*, IEEE, New York, NY. Pp. 1220–1223.

Wemegah, T. D. and Zhu, S. (2017). Big data challenges in transportation: A case study of traffic volume count from massive radio frequency identification (RFID) data. In *2017 International Conference on the Frontiers and Advances in Data Science (FADS)*, IEEE, New York, NY. Pp. 58–63.

Wojke, N. and Bewley, A. (2018). Deep cosine metric learning for person re-identification. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, New York, NY. Pp. 748–756.

Wojke, N., Bewley, A., and Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In

2017 IEEE International Conference on Image Processing (ICIP), IEEE, New York, NY. Pp. 3645–3649.

Woolley, M. (2015). *Bluetooth technology – protecting your privacy*. Bluetooth SIG (online). https://www.bluetooth.com/blog/bluetooth-technology-protecting-your-privacy/.

Xia, S., Xiong, J., Liu, Y., and Li, G. (2015). Vision-based traffic accident detection using matrix approximation. In

2015 10th Asian Control Conference (ASCC), IEEE, New York, NY. Pp. 1–5.

Xu, Q., Mak, T., Ko, J., and Sengupta, R. (2004). Vehicle-to-vehicle safety messaging in DSRC. In *VANET '04:*

Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks, ACM, New York, NY. Pp. 19–28.

Yan, Y., Huang, X., Rangarajan, A., and Ranka, S. (2018). Densely labeling large-scale satellite images with generative adversarial networks. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing, and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, IEEE, New York, NY. Pp. 927–934.

Yang, Z., and Pun-Cheng, L. S. (2018). Vehicle detection in intelligent transportation systems and its applications under varying environments: A review. *Image and Vision Computing*, 69:143–154.

Yu, L., Yu, L., Wang, J., Qi, Y., and Wen, H. (2008). Back-propagation neural network for traffic incident detection based on fusion of loop detector and probe vehicle data. In *2008 Fourth International Conference on Natural Computation*, volume 3, IEEE, New York, NY. Pp. 116–120.

Appendix A: Software Packages

A.1 General Programming Tools and Languages

NumPy – NumPy is a Python package using for scientific computing and array processing. Some key features of NumPy are:

1. A powerful N-dimensional array object
2. Sophisticated (broadcasting) functions
3. Tools for integrating C/C++ and Fortran code
4. Useful linear algebra, Fourier transform, and random number capabilities.

Pandas – Pandas stands for "Python Data Analysis Library." It allows for processing and analyzing multidimensional arrays. The data and be easily exported or imported to a CSV format.

MySQL – MySQL is a fast, easy-to-use relational database management system (RDBMS) that allows implementation of a database with tables, columns, and indexes. While it is free and open source, it can still handle a large subset of the functionality of the most expensive and powerful RDBMSs. MySQL works on many operating systems and with many languages, including PHP, PERL, C, C++, JAVA, etc. It uses a standard form of the well-known SQL data language. It works well even with large datasets. It supports large databases of up to 50 million rows or more in a table. The default file size limit for a table is 4 GB, but it can be increased to a theoretical limit of 8 million terabytes (TB). MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

RAID – Redundant Array of Independent Disks (RAID) is a data storage technology that combines multiple independent physical disks into one virtual disk with the aim of improving performance and increasing data redundancy.

Tableau – Tableau is a suite of software that allows users to explore and visualize data. Typical workflow for Tableau is:

1. Desktop: The main development environment for Tableau. This is where a Tableau developer will create dashboards that can then be published and viewed.
2. Server and Cloud: Tableau server and cloud are publishing platforms for tableau. Both the visualization dashboards and the data that they need can be published to both platforms.
3. Mobile and Web Clients: Once the Tableau developer publishes a dashboard to the server or cloud, it can be viewed from a Web browser or from any mobile platform (mobiles, tablets, etc.).

A.2 Video Processing Software

The following is a summary of image processing and machine learning algorithms appropriate for video-based collision detection.

PyTorch – PyTorch is an open source machine learning library for Python, based on Torch. Torch is an open

source machine learning library, which provides a wide range of algorithms for deep learning and uses the scripting language LuaJIT and an underlying C implementation. PyTorch was primarily developed by Facebook's artificial intelligence research group. PyTorch provides two high-level features: (1) tensor computation (like NumPy) with strong GPU acceleration and (2) deep neural networks built on a tape-based autodiff system.

OpenCV – OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. The library is cross-platform and free for use under the open source BSD license.

CUDA – CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows practitioners to utilize the powerful CUDA-enabled graphic processing unit (GPU) for parallel computation. The platform is designed to work with programming languages such as C, C++, and Fortran.

YOLO – YOLO is the acronym of You Look Only Once. It is library for real-time object detection and classification. It is based on Python3, Tensorflow 1.0, NumPy, and OpenCV 3.

DeepSort – DeepSort is a simple online and real-time object tracking library. It is based on NumPy, Sklearn, OpenCV. DeepSort is used with YOLO for tracking multiple objects.

A.3 Cloud-related Software

Amazon SDK for DynamoDB – The AWS Java SDK for Amazon DynamoDB module holds the client classes that are used for communicating with Amazon DynamoDB Service. It requires an access key-secret key pair and a region for setup and uses that authorization for any request to the database.

DynamoDB – DynamoDB is a powerful, fully managed, low latency, NoSQL database service provided by Amazon AWS. It handles all the administrative burdens associated with operating and scaling a distributed database and offers encryption at rest. DynamoDB can assure dedicated throughput with predictable performance for any amount of request traffic. It has the following other features:

1. It automatically divides data and traffic over an adequate number of servers.
2. It provides on-demand backup capability.
3. DynamoDB allows automatic deletion of irrelevant and expired data from tables.
4. DynamoDB automatically spreads the data and traffic for tables over a sufficient number of servers to handle throughput and storage requirements, while maintaining consistent and fast performance.

Amazon EC2 – Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity and eliminates the need to procure hardware. Amazon EC2 can be configured based on storage, networking and security requirements.

Appendix B: Cloud Computing Providers

B.1 Amazon: AWS and AWS IoT

AWS, or Amazon Web Services, is the cloud offering from Amazon. The IoT offering is called AWS IoT and is meant to be a platform for creating complete Inter of Things solutions using only AWS and Amazon Technologies. Devices can connect to and communicate with applications running on the cloud over many different protocols (such as HTTP, MQTT, and WebSockets). Device-specific SDKs are available for different languages (such as Embedded C, JavaScript, and Python). It offers support for reliable bi- directional (device-to-cloud and vice versa) messaging using the concept of device shadows to enable communication when the device network connectivity is not reliable. It includes a declarative SQL-like rules engine that can be used to perform basic transformations of IoT data then reroute them to specific locations or endpoints such as a storage container (S3 bucket). It can also be used to trigger the AWS event-based programming platform called Lambda. It is also possible to redirect data streams into the Kinesis streams service which is meant to run real-time analytics on the streams. Finally, AWS includes an archival storage service (Glacier) and offers support for a wide range of hosted and self-managed database services. Hosting self-managed virtual machines is straightforward in the Amazon cloud. SDKs are offered for a wide range of languages and not just specific devices, which makes it easy to integrate many devices into the platform. Also, AWS is the only platform which offers edge-aware computing (covered below). In terms of commercial success and platform maturity, this is the most comprehensive solution in the market today.

B.2 Google Cloud and IoT Core

Google Cloud Platform is the cloud platform developed by Google. It integrates with the Cloud IoT Core service, and it aims to enable customers to connect easily and securely with many dispersed devices with the goal of managing and ingesting data from them. Cloud IoT Core, in combination with other services on Google Cloud platform, aim to provide a complete solution for “collecting, processing, analyzing, and visualizing IoT data in real time.” Google has also created an Android-based IoT OS known as Brillo. One of the key features of Brillo is Weave, a newly designed and implemented communication protocol. It is designed to allow IoT devices and controllers to talk to each other in a resource-constrained environment. Weave is designed to be a standard that provides discovery services and enables ability across IoT devices, mobility devices (phones or tablets), and the cloud. In summary, the Google cloud platform offers support for both live and archival storage for video, multiple database services for hosting tabular data, offers a streaming analytics service called dataflow, and allows for hosting self-managed virtual machines. It also offers traditional analytics and virtualization through a cloud data lab and data studio. Unfortunately, it offers little support for edge computing and therefore integrating an edge aware architecture may require more work on the part of a customer.

B.3 IBM Cloud and Watson IoT

Bluemix, IBM’s Cloud Platform as a Service, comes with an extensive catalog of services from IBM and third parties that may be used in applications. IBM’s IoT Platform is a fully managed cloud-hosted service. To set

up a traffic intersection monitoring application, various devices (e.g., sensors, cameras) are first connected to the IBM Cloud. The cloud-to-device and device-to-cloud communication mostly use the open lightweight MQTT messaging protocol. The data received from the sensors may be structured (traffic data) or unstructured (camera feed) and may be stored from entry-to-enterprise-level hybrid storage systems with flash drive support. IBM Spectrum Scale offers fast and simplified management of unstructured data, providing easy accessibility and non-disruptive scaling for massive volumes of file and object data. IBM Spectrum Scale allows global shared access to data with unified file (NFS/SMB), object (S3/Swift) and Hadoop (HDFS) support. Historical data may be stored using IBM Spectrum Archive to lower costs, while IBM Spectrum Protect enables reliable, efficient data protection and resiliency for software-defined, virtual, physical, and cloud environments. IBM DB2 Universal Database supports structured data types and queries in SQL. In summary, the IBM cloud platform offers support for storage but does not have the specialized storage services that may be used to reduce costs associated with storing vast amounts of data. It has support for multiple database services for hosting tabular data and offers a streaming analytics service called IBM streaming analytics. Hosting self-managed virtual machines is also possible. It has limited support for integrating a specific set of commercial IoT devices and integrating specific industrial IoT devices directly with the cloud may require more work by the customer.

B.4 Microsoft Azure

Microsoft has given the name Azure to its cloud offerings. IoT Hub is Microsoft Azure's managed IoT solution. IoT Hub supports AMQP, MQTT, and HTTP.

Data can be stored using Blob Storage (meant for offline processing) and archival or sent to Event Hubs for live processing. In summary, the Azure combines live and archival storage, it does support multiple database services for hosting tabular data, offers a streaming analytics service called Azure Stream Analytics, and allows for hosting self-managed virtual machines. They also offer analytics and virtualization through Power BI and other solutions. They are also starting to add support for edge-aware edge computing via Windows 10 IoT core.

B.5 Industry Comparison of Cloud Providers (Gartner)

According to the latest magic quadrant developed by Gartner for infrastructure-as-a-service (IaaS) (Figure 31), only Amazon Web Services and Microsoft are in the leader's quadrant. In fact, AWS is rated "the most mature, enterprise-ready provider, with the deepest capabilities for governing a large number of users and resources." Figure 32 shows a list of services from the different cloud service providers that will be useful for traffic applications.

Figure 31: *Magic quadrant for Cloud AI developer services. Source: Gartner (February 2020).*

Services	Microsoft Azure	Amazon Web Services	Google Cloud Platform	IBM BlueMix
COMPUTE				
Virtual servers	Virtual Machines	EC2	Compute Engine	Virtual Servers
STORAGE				
Object Storage	Blob Storage	S3	Cloud Storage	Swift
Shared File Storage	File Storage	Elastic File System		Manila
Archiving and Backup	Cool Blob Storage	Glacier and S3	Cloud Storage Nearline	Vault, Cold Vault
DATABASE				
Relational Database	SQL Database	RDS	Cloud SQL	IBM DB2
NoSQL Database	DocumentDB	DynamoDB	Cloud Datastore	IBM Cloudant
Data warehouse	SQL Data Warehouse	Redshift	Big Query	IBM dashDB
IoT				
Streaming Data	Event Hubs	Kinesis	Cloud Dataflow	IBM Streams
ANALYTICS & BIG DATA				
Visualization	Power BI	QuickSight	Cloud Datalab	Cognos
SECURITY & IDENTITY				
Authentication and Authorization	AzureAD RBAC Multi-factor Authentication	Identity and Access Management Multi-factor authentication	Google IAM Cloud Resource Manager Google Identity Toolkit Google Signin	IBM Support Assistant
Encryption	Key Vault BYOK	Key Encryption Service	Platform level encryption BYOK	IBM Security Key Lifecycle Manager
Security	Security Center	Inspector	Cloud Security Scanner	IBM Security

Figure 32: *List of services offered by the various cloud service providers.*