VIDEO ANALYTICS ON THE MLK SMART CORRIDOR TESTBED

by

Jose Stovall

Mina Sartipi
Professor of Computer Science
(Chair)

Farah Kandah
Professor of Computer Science
(Committee Member)

Yu Liang
Professor of Computer Science
(Committee Member)

VIDEO ANALYTICS ON THE MLK SMART CORRIDOR TESTBED

by

Jose Stovall

A Thesis Submitted to the Faculty of the University of Tennesse at Chattanooga in Partial Fulfillment of the Requirements of the Degree of Master of Science: Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

May 2020

ABSTRACT

With the predicted boom of urban environment populations in the next 30 years, many new challenges in urban transportation will surface. In an effort to mitigate these, the Center for Urban Informatics and Progress (CUIP) has been introduced along with its testbed. One opportunity this testbed provides is the ability to utilize computer vision and video analytics to anonymously gather data on how citizens traverse the city. This thesis shall discuss an approach to real-time object tracking that serves as a basis for further analytics such as traffic flow data collection and near-miss detection. The proposed video analytics platform will aid citizens with their day-to-day commute through the corridor by deriving real-time data based on actual behavior seen in the citizens' commute. Furthermore, since the testbed is ever-expanding in both hardware and size the algorithms and software proposed in this thesis are designed to prioritize scalability.

ACKNOWLEDGMENTS

I would like to begin with thanking my family (all of them, including those that were chosen). Your support has motivated me to become the best individual that I can be, and without them I would be lost. I would like to extend my gratitude to Dr. Farah Kandah for his guidance and support throughout my graduate-level education. Additionally, I would like to thank Dr. Yu Liang for giving me support and motivation to continue my work and understand the underlying theory in otherwise black-boxed software implementations.

I would like to thank the team members of the lab when I originally joined in 2017, all of which helped me learn new concepts quicker than I ever have! Specifically, I would like to thank Dr. Zhen Hu, Rebekah Thompson, Jin Cho, Austin Harris, and Hector Suarez. Together, they helped keep me learning the many concepts I was so unfamiliar with at the time and supported me the entire time. I would also like to thank the current lab team for their support (both emotional and otherwise). These exceptional individuals have been there to help me understand concepts I was stuck on, give me a good laugh when I needed it, and help me up when I was down. Specifically, I'd like to thank Dr. Thanh Nam Doan, Peter Way, Jeremy Roland, Katie Rouse, Bennett Bowden, Yatri Patel, Alnour Alharin, and Sree Nukala.

I cannot extend enough thanks to thank Dr. Mina Sartipi, without whom I would have never come as far as I have. Through her, I have built confidence in my work and ability to develop cutting-edge algorithms that I would have never dreamt of doing otherwise. I am forever grateful for the opportunity she has given me, first at SCAL then at CUIP, and I cannot thank her enough for her major contribution to who I have become, both as an individual and as a professional.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Motivation for Thesis Applications

In the last decade, the population of Hamilton County, Tennessee has grown by 25.85% [1]. With this population growth comes challenges for the existing roadway infrastructures, such as increased roadway incidents. According to the Tennessee Department of Safety and Homeland Security [2], Hamilton County roadway incidents have had an increasing trend; in 2015 there were 12,956 roadway incidents, but in 2019 there were 14,101 roadway incidents. Furthermore, there were 95 incidents in 2015, however there were 107 incidents involving pedestrians in 2019 [2]. This may only be the beginning, as it has been predicted that two thirds of the world's population will live in urban environments by 2050 [3]. With this rapid urbanization, not only will roadway infrastructure challenges be exacerbated, but entirely new ones may arise [4]. This trend of increased incidents can be seen in the data and the expectation for greater challenges is cause for concern for the health and well-being of citizens.

To help mitigate these day-to-day safety concerns, a wide range of initiatives have been recently proposed, each defining and conceptualizing the range of sensors, IoT devices and infrastructure that is required to help overcome these challenges. These sensors can be used to gather data from citizens and their day-to-day actions, and this data can be used to help the citizens that create it. By gathering data from real events that occur on the testbed, it is possible to predict and analyze based on data from actual human behavior, instead of simulated data. This data can have practical applications that help citizens make decisions for their own safety and well-being while they commute through these sensor-equipped cities.

Unfortunately, there is no standardized method for augmenting a city with "smart" capabilities, so most solutions and design considerations must be designed from scratch. Additionally, there is no pre-existing collection of software specifically designed for data analytics in a city-scale environment. This poses a question of how to apply data analytics at scale, which is addressed by the video analytics discussed in this thesis. These video analytics are written by hand to allow us to take into consideration any requirements given by the environment of the testbed.

This thesis will discuss just three of the many methods of analytics in a smart city. These methods are all video-based and prioritize privacy and urgency by performing real-time analytics with no video storage. Amongst these video analytics are real-time scalable object tracking, real-time near-miss detection and real-time traffic flow analysis. The data from these real-time data analytics can be used to help citizens be aware of surroundings they cannot see in real-time so that their commute can be made safer for all who travel within the testbed.

The real-time scalable object tracking approach discussed in this thesis allows sensor-equipped cities (also referred to as smart cities) to perform real-time data analytics such as average traffic density and real-time object routes. The real-time object tracking implementation proposed in this thesis will discuss concerns for scalability so that the algorithms can grow with the city and equipment. Real-time object tracking also serves as a basis for further analytics such as real-time near-miss detection and traffic flow analysis. Near-miss detection will detect events where any two objects are at risk of collision should no further action be made, so this metric can be used to determine the driving safety of certain areas. Furthermore, traffic analysis makes it possible to perform real-time dynamic routing to prioritize fuel efficiency or travel time. This analysis also enables us to determine higher resolution traffic flow, such as which lanes are occupied and which streets are being turned on to the most.

Using these modern approaches of video analytics in a live urban environment paves the way for a connected city capable of helping the citizens within. In this environment,

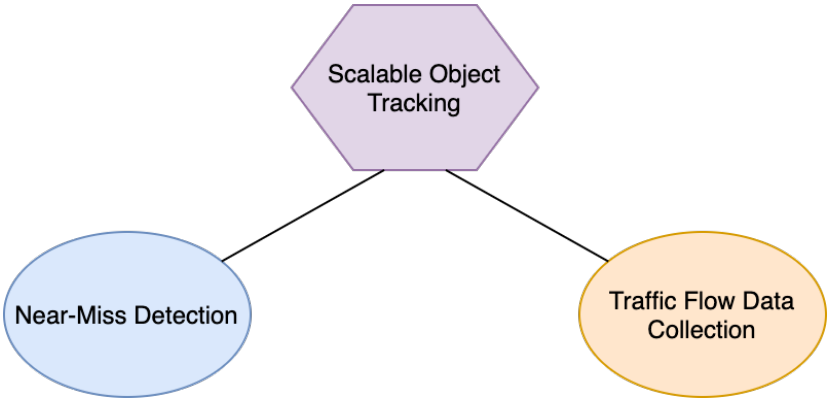


Figure 1.1 A relationship model displaying how Scalable Object Tracking (purple) is used for Near-Miss Detection (blue) and Traffic Flow Data Collection (orange).

video analytics can be used to prevent roadway incidents and decrease traffic congestion by increasing all citizens' awareness of their environment; by giving citizens a holistic view of the city's traffic activity they can more safely traverse it.

The remainder of this thesis consists of 5 chapters. Chapter 2 describes in thorough detail the testbed on which these video analytics are performed. Chapter 3 discusses the object tracking algorithm which enables the other forms analytics discussed in this thesis. Chapter 4 details the near-miss detection algorithm which has been built on top of the existing object tracking platform, which enables us to detect when two objects on the testbed are at risk of collision. Chapter 5 describes the traffic flow analytics that have also been built on top of the discussed object tracking platform. Lastly, Chapter 6 concludes this thesis with an overview of the work covered in this thesis and insight on future work that could be achieved.

CHAPTER 2

Background Information

This thesis will discuss many topics which are not considered common knowledge. Thus, this section of the thesis is dedicated to explaining the requisite background knowledge for the remainder of the thesis. The concepts discussed in this section include Computer Vision, Convolutional Neural Networks (a basis to most object detection algorithms), and Object Tracking.

2.1 Computer Vision

The entirety of computer vision can be considered overwhelmingly large, mostly due to its multi-faceted nature. Summarily, computer vision is a means of manipulating or analyzing imagery, be it a still image or a single frame in a video. The ability to modify and manipulate images programmatically enables developers to design algorithms for many tasks. Image manipulation includes simple tasks such as image resizing or flipping, but it also includes more complex tasks such as applying filters or removing noise. Image analysis using computer vision is often a more complex task which may require chains of image manipulation in order to create any understandable form of output [5, 2–6]. Computer vision analysis tasks include naive object detection, motion detection and even naive object tracking [6], which can be combined to use computer vision for automated vision manipulation (to be discussed throughout this work).

In order to understand how computer vision works, it is important to understand how images are represented by a computer. In the majority of computer vision implementations,

imagery (whether it is a still image or a single video frame) is represented as a matrix (grid). For gray-scale imagery, each item in the matrix represents a value between 0 and 255, where 0 represents black and 255 represents white (shown in Figure 2.1. In the case of colored imagery, the representing matrix is three dimensional, where the new third dimension represents the intensity values of their respective color (red, green, or blue). In this instance, a value of 0 represents no intensity of that color, and a value of 255 represents full intensity of that color. This has been visualized in Figure 2.2.

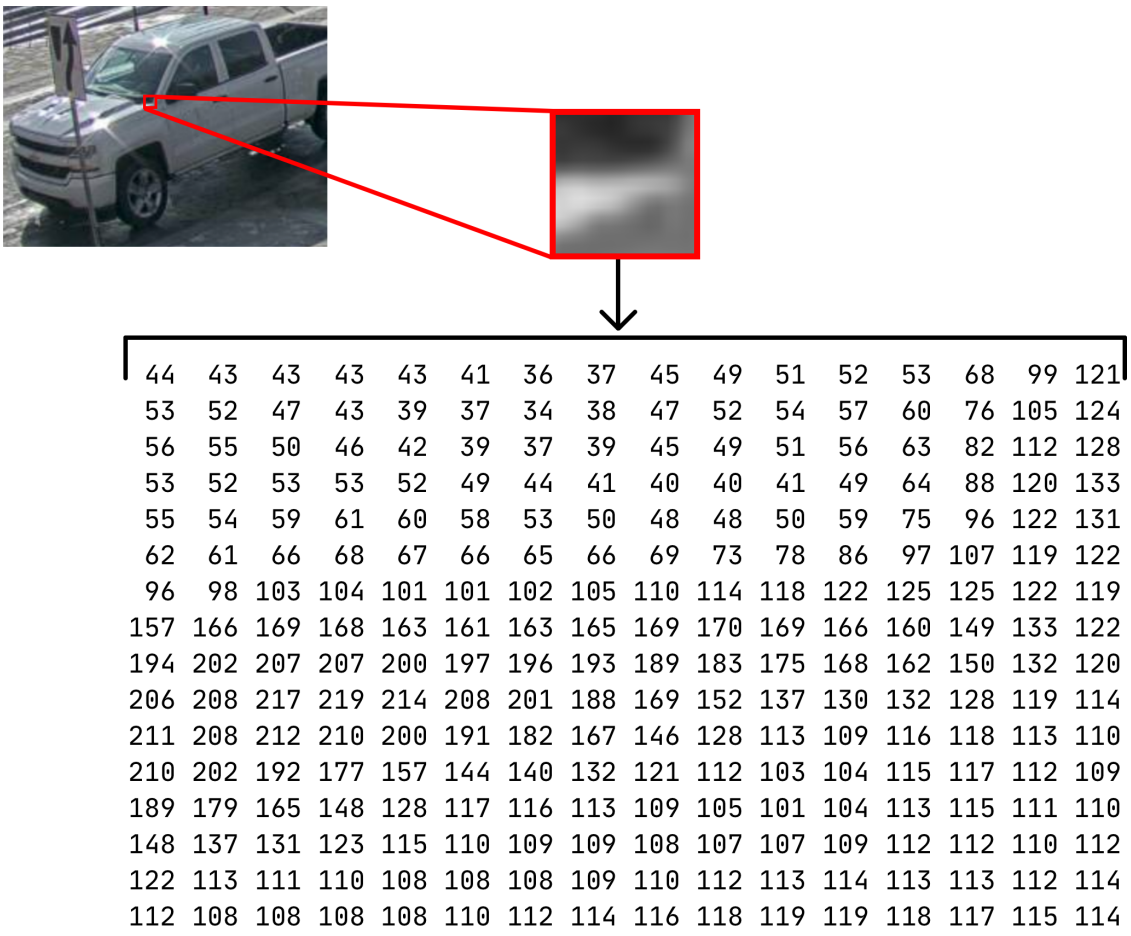


Figure 2.1 A 16 pixel by 16 pixel gray-scale sample, represented as a single matrix by averaging the three color values together. Each value in the matrix represents the intensity, from 0 (no intensity, black) to 255 (max intensity, white).

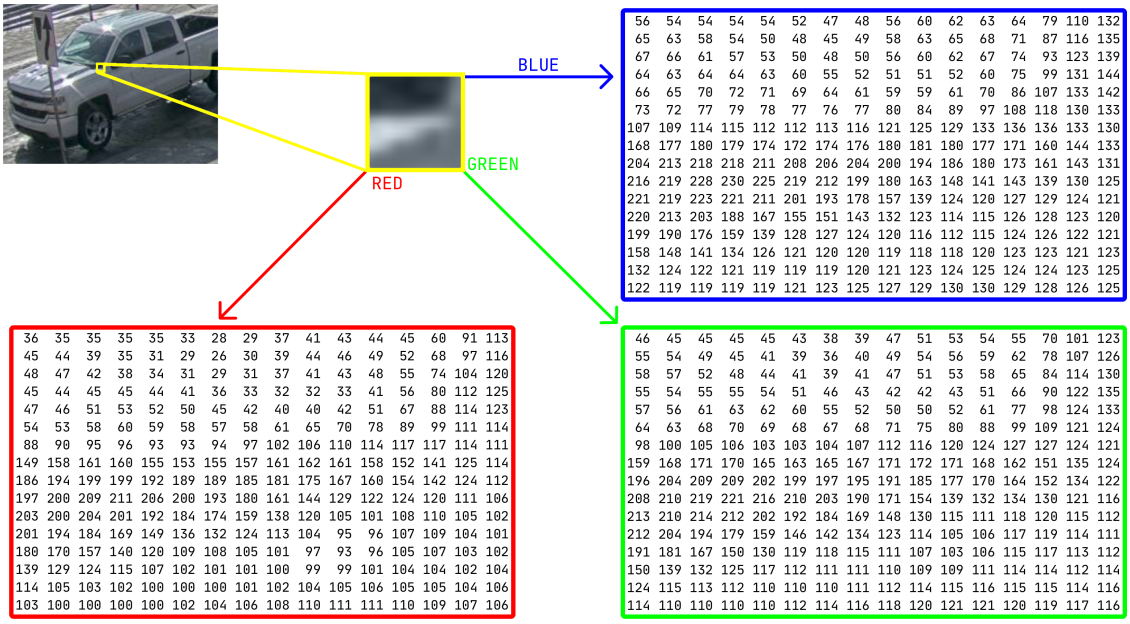


Figure 2.2 A 16 pixel by 16 pixel color sample, represented by three matrices, one for red, green and blue respectively. Each value in each matrix represents the intensity of that color value, from 0 (no intensity) to 255 (max intensity).

2.2 Convolutional Neural Networks

The task of automating the detection of objects has many applications from robotics and machinery [7] to real-time video surveillance streams [8]. Automated object detection is made possible by the underlying technology: a convolutional neural network, or CNN. A CNN is a type of Artificial Neural Network (ANN), in which a group of nodes (called neurons) are inter-connected to form a system, much like biological nervous systems. A nervous system is capable of responding to external stimuli, such as seeing an object, and identifying that object by following a path of interconnecting neurons. CNNs can be considered the computational adaptation to this same concept, wherein neurons have been implemented, and different inputs (e.g. stimuli like seeing an object) yield different outputs (e.g. identification). ANNs in general must be trained with sample, pre-labeled inputs in order to build the model which describes how the neurons within interconnect. During this process, neurons' components such as weight and bias are adjusted to control when and how that neuron is fired based on various inputs (illustrated in Figure 2.3). Once the CNN has been trained on a set of labeled

images, it can begin detecting objects that it was trained to detect [9].

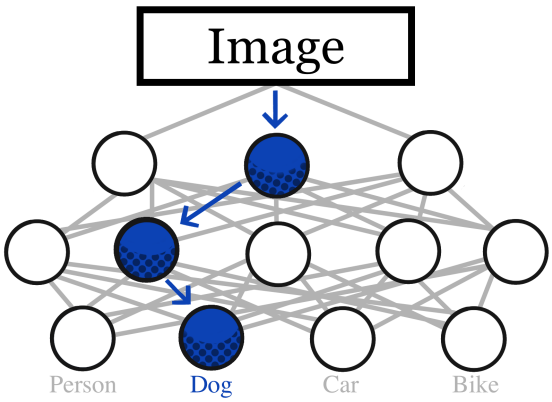


Figure 2.3 An illustrated sample arrangement of neurons, where the path in blue shows the path of fired neurons based on the given imagery.

Computer vision plays a critical role in the process of creating a CNN, as it is the intermediary process which converts what humans see (for example, a picture of a dog), and what the CNN can take for input (for example, the same picture of a dog represented by a matrix); computer vision is used to translate the human-interpretable input into a computer-interpretable input. The combination of computer vision and convolutional neural networks form the basis of a robust object detection pipeline, which yields substantially higher accuracy as well as reduced latency. While the focus of this thesis does not specifically include object detection using neural networks, object detection is a crucial component in many implementations of object tracking approaches introduced in this thesis. Without a vague understanding of how CNNs work, it may be more challenging to understand other components in this thesis such as the scalable architecture discussed in Chapter 4.

2.3 Object Tracking

While object detection has many applications, it differs greatly from object tracking. Object detection and object tracking sound like the same to most, but there is one critical difference: object detection only *detects* objects with no concept of that objects' permanence,

where as object tracking links the output from detecting objects and merges it with the concept of object permanence. For example, using object detection alone will be able to tell you that a vehicle is in the frame, and exactly where in the frame it is. Whether this vehicle returns in the following frames or is present in previous frames is unknown to the object detection model. Using object tracking adds this extra context; it is now possible to know that this is the same object from previous / next frames, which can be used in additional practical applications beyond detection alone [10]. There are a few implementations to object tracking, some correlating object across frames using a predictive filter [11], others use classification algorithms [12], and some use complex models for robust tracking [13]. Some such applications include determining the exact path of each object in a video, accurately counting objects without re-counting objects, and determining pixel-relative acceleration and velocity. Some of these applications will be discussed in further detail in this thesis.

As we discussed in the section dedicated to object detection, the output of most object detecting CNNs is a label and a defined bounding box which surrounds the detected object. This output is often a dependency for the object tracking algorithm implemented; some object tracking implementations do not perform the detection, they simply associate detections across frames using geometric and linear algebra methods. Other object tracking implementations have their own model dedicated solely to the tracking component, or are able to perform both object detection and object tracking [10].

CHAPTER 3

The Testbed Used

In our paper titled "MLK Smart Corridor: An Urban Testbed for Smart City Applications" [14], we introduce the MLK Smart Corridor as an urban testbed and platform for smart city development. The Center for Urban Informatics and Progress (CUIP) at the University of Tennessee at Chattanooga has designed and deployed this testbed with the help of the The Enterprise Center, Chattanooga Department of Transportation, and Chattanooga Electric Power Board (EPB). The testbed provides a real-world urban environment for testing and developing smart city infrastructure, transportation, and security applications in a real-world urban environment. The testbed facilitates experimentation, prototyping, and validation of new smart city and Connected Autonomous Vehicle (CAV) technologies. An illustration of the MLK Smart Corridor Testbed is shown in Figure 3.1.

3.1 Hardware and Infrastructure

The MLK Smart Corridor spans over a mile and a half of Downtown Chattanooga's Martin Luther King (MLK) Boulevard, as seen in Figure 3.2. Parallel to the University of Tennessee at Chattanooga, MLK Boulevard is one of downtown Chattanooga's busiest roadways [15]. It features ten signaled intersections, bike lanes, electric car charging stations, electric car / bike share stations, and roadside parking. Application specific enclosures deployed at intersections allow for easy and accessible hardware expansion. A wide array of sensors and communication devices are deployed at each intersection including IoT devices (listed in Table 3.1, communication devices (listed in Table 3.2), and edge compute nodes (listed in

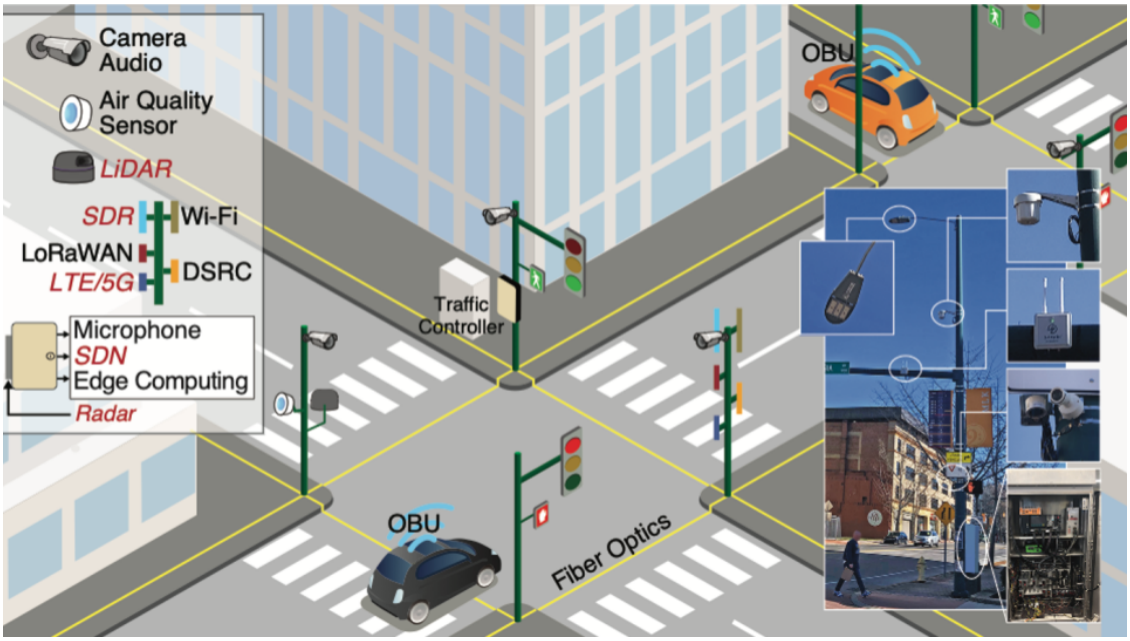


Figure 3.1 Graphical Representation of the MLK Smart Corridor's Testbed and its features.

Table 3.3) [16].

Each device was chosen to provide baseline functionality for users. The MLK Smart Corridor is available for research and industry partners. An online web portal allows users to submit projects and request resources; if a user would like to test vehicle-to-infrastructure (V2I) messaging standards compatibility, they are not expected to procure and install their own communications devices. In this case, the testbed provides DSRC Road-Side-Unit compatibility at each intersection that meets the United States Department of Transportation (USDOT) standards. These devices are integrated with Chattanooga's Department of Transportation Intelligent Transportation System (ITS).

Table 3.1 IoT Sensors on the MLK Smart Corridor Testbed

Device	**Description**
Purple Air PA-II-SD	Air sensor providing current air quality conditions
Axis P1448-LE	4K 30 FPS camera with variable FOV
Axis M2025-LE	1080P 30FPS camera with wide FOV
Sound Card + Preamp Mic	Mic connected to Raspberry Pi for audio analysis and processing
RP LiDAR	LiDAR sensor for bathymetric depth calculations and more
Banner Q240R RADAR	Narrow-beam Fixed RADAR sensor

Figure 3.2 All deployed intersections portrayed on OpenStreetMap. Each traffic signal icon represents an intersection at which sensors have been deployed.

Table 3.2 Networking and Wireless Communications on the MLK Smart Corridor Testbed

Device	**Description**
LimeSDR	Handles I/O of numerous radio signal varieties
Locomate DSRC	Dedicated short-range wireless communications
HackRF	One Programmable peripheral for software defined radio
Aruba AP 270	802.11AC Wireless Router with 2.4 and 5.0GHz frequencies
TPLink AD7200	802.11AD Wireless Router with 2.4, 5.0 and 60GHz frequencies
LoRa Gateway	Physical Layer IoT Wireless Networking Component

3.2 Data Architecture

Smart cities generate vast amounts of heterogeneous data. Low-latency transactions, high throughput, flexibility, scalability, and interoperability are all key design traits to take into consideration when designing a smart city's server infrastructure. Traditionally, applications may create direct data pipelines between systems, though at scale this is not an accepted architecture. A notable consideration is that managing and maintaining dedicated data

Table 3.3 Edge Computing Hardware on the MLK Smart Corridor Testbed

Device	Description
Nvidia Jetson TX2	Edge Computing node for GPU-intensive computation or processing
Raspberry Pi	Edge Computing node for simple processing or data transmission
Industrial Compute Node	Intel Core-i7-equipped machine capable of intensive work

pipelines becomes challenging at scale, but CUIP has equipped the testbed with a distributed event driven architecture that can address these design considerations and issues.

A software architecture for a testbed like this will be responsible for ingesting, analyzing, and storing all data generated by the testbed; this platform creates a central system where all data generated can be accessed from external systems. All systems and devices will consume data through this platform via integration tools and APIs, and its design eliminates the need for dedicated communication between individual systems on the testbed. New systems require some configuration, but no configuration is required on the central system. To ensure that their platform will provide a long-term solution for providing services, CUIP's data infrastructure utilizes an ingestion and integration system that supports horizontal scalability which provides the ability to scale up for new devices and systems that come online. The core of their software stack includes a cluster of brokers that make up the core of the infrastructure (shown in Figure 3.3).

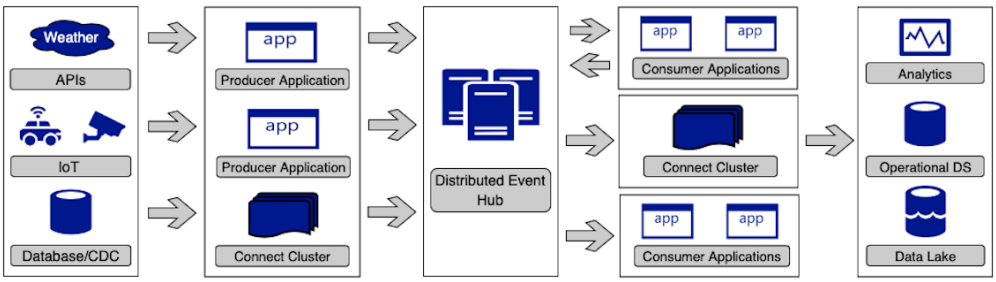


Figure 3.3 Smart city data integration platform architecture. Data from sources such as APIs, other databases or IoT sensors are submitted to the distributed event hub using producer applications or connect clusters. Then, the data contained in the event hub can be used for consumer applications which read from and write to the event hub. Additionally, connect clusters or consumer applications can also read from the event hub and can use that data freely.

A custom framework is used by systems generating data to push and pull data in and out

of the platform. This framework contains multiple features that in combination are designed to handle high velocity, large volume, and all varieties of data types. This plays a critical role in the work proposed in this thesis, as it produces approximately 200,000 messages each day using only ten cameras as video sources.

CHAPTER 4

Scalable Object Tracking

4.1 Introduction

In this chapter we introduce our approach to scalable object tracking, allowing us to anonymously detect and track objects as they pass through the MLK Smart Corridor testbed. This implementation makes it possible for near-miss events to be detected (discussed in Chapter 5) and traffic flow data to be collected (discussed in Chapter 6).

4.2 Goal of Scalable Object Tracking

Object tracking, unlike object detection alone, allows every detectable object to be treated as an event where the event begins when the object enters the camera's view and ends when the object leaves the camera's view. In order to bring this to the large-scale environment of the MLK Smart Corridor testbed, any solution designed must be capable of scaling as the testbed and city grow. Furthermore, since the testbed lies on a live urban environment, object tracking makes it possible to anonymously gather data from real trends derived from citizens. This data can be used to help the very citizens that created it, allowing for anonymized analytics to increase driver awareness in the event of a near-miss and mitigate traffic density.

4.3 Related Works

As discussed in Chapter 2, many object tracking implementations do not perform detection of objects, but instead associate detections or seed trackable points based off of detected

objects. As a result, this related works section will explore related object tracking methods as well as some of the more popular current object detection models. As object detection has been studied extensively, only a few of the most unique (be it accuracy, implementation or speed) will be discussed in this thesis.

4.3.1 Object Tracking Implementations

Object tracking poses a problem different from that of detection; in general, off-the-shelf object detection algorithms do not offer any means to maintain object persistence between the frames of a video. There are two common approaches within the subject of object tracking: single-object, and multi-object. Single-object tracking focuses on tracking a single object within the camera view, where multi-object tracking will track all given objects with the camera view. For the purposes of a smart city there will almost always be more than a single object within the camera's view, so this thesis has also been written to address a multiple-object environment (an environment in which multiple trackable objects are present at the same time).

The authors of "Robust Object Tracking with Online Multiple Instance Learning" [17] have proposed a Multiple Instance Learning (MIL) object tracker with online boosting (seen in Figure 4.1). Their proposed tracking system does not require object detection (though it *can* be augmented with it), as it only requires an initial bounding box which can be drawn by multiple sources (such as a human or a model). While this is particularly effective for less complex views, it is not an ideal solution for camera views with many objects entering and leaving the scene. Therefore, for the purpose of this thesis we have considered this tracker *only* in conjunction with a model, whose detections can be used to update the tracker's tracklets (also called "re-seeding"). Since new objects within the camera view can only be tracked once the tracker is provided with a new bounding box, a compromise must be made in how often to re-seed the tracker with model detections. One potential compromise would be to re-seed the tracker with model detections every other frame. This approach has been

found to result in a higher number of instances where an object's tracker instance is re-initialized. Another compromise would be to wait longer to re-seed the tracker with model detections (for example, every 10 frames). While this approach is a good compromise, it still allows for objects to transfer their trackers to other objects.

Figure 4.1 An Implementation of the MIL Object Tracker with online boosting on the MLK Smart Corridor. The yellow line indicates the point at which an object is counted, and the green indicates objects being tracked by the algorithm.

The authors of "Visual Object Tracking using Adaptive Correlation Filters" [18] introduce a correlation-filter based tracking algorithm which uses their proposed Minimum Output Sum of Squared Error (MOSSE) filter. This filter allows for stabilization of correlation filters upon initialization. This method is robust enough to maintain its performance through changes of multiple conditions such as scale, lighting, and more, all-the-while performing at over 600FPS. As it is a correlation-filter based tracker, it also suffers from the same issues and compromises that the work in [17] does.

The authors of "Simple Online and Realtime Tracking with a Deep Association Metric" [11] have proposed a "Simple Online Real-time Tracker". This tracker works by taking detections from some model, and associating a tracklet to these detections using Intersection

of Unions (IoU) and a Kalman Filter. The Kalman Filter is used to predict the next location of a tracklet, which is used with IoU and the current detections to associate a detection back to the tracklet. This tracker requires a model's detections *every frame*, as these detections are how the tracker performs its reassociation to tracklets. The tracker manages old tracklets by determining how recently the tracklet has been reassociated. If the tracklet has not been reassociated within some threshold of frames (user-configurable), it is deleted for memory conservation purposes. One shortcoming of this tracker is that it requires a model which is not provided in the paper. Additionally, the tracker's tracklets do not store some useful information such as a history of that tracklet's locations (and including timestamps would also prove useful), and obsolete tracklets (those tracklets which cannot be reassociated with detections) are simply deleted. This tracker is also heavily frame-rate dependent; the higher the frame-rate of the video and detections, the fewer instances of ID reassignment and higher the precision of the tracker.

The authors of "Multiple Object Tracking Using K-Shortest Paths Optimization" [12] have proposed a multiple object tracking solution which uses a K-Shortest-Path algorithm in conjunction with a linear equation to perform tracking. While this solution is highly accurate, it fails to perform with the accuracy and efficiency required to run the many cameras along the MLK Smart Corridor. Between the linear functions and the K-Shortest-Path algorithm, this object tracking model is enable to meet our 30FPS requirement, and would be problematic to scale with the smart city.

4.3.2 Object Detection Models

The authors of "You Only Look Once (YOLO)" [19–21] have introduced an incrementally improved real-time object detection model. The authors have proposed a Convolutional Neural Network (CNN), which solves the object detection problem by treating it similar to a regression problem. Their model provides both labels and bounding boxes, and comes in many variants. The original YOLOv1 is limited to only 20 different detectable objects, and

had two variants: YOLO and Fast YOLO. YOLO would perform at 45 FPS, and Fast YOLO would perform at 155 FPS. YOLOv2 is capable of detecting 80 different types of objects, and achieves over 40 FPS (depending on the variant). YOLOv3 brought about a series of smaller quality-of-life changes, such as utilizing the GPU for more tasks than before.

The authors of "Mask-RCNN" [22], an extension to Faster-RCNN [23], not only detects objects but also predicts each object's mask. Mask R-CNN outperforms all other existing single-model detectors (at the time the authors wrote their paper), and is designed for use beyond its original scope as an object detection model - such as human pose estimation. While Mask R-CNN is an excellent candidate for the detection component in a real-time scalable object tracking solution for smart cities, it cannot run at the requisite frame-rate. At a peak of 5 Frame Per Seconds (FPS), it is incapable of keeping up with the real-time environment that a smart city encompasses. The extra 25 frames that most cameras produce will be wasted, and such poor frame-rate may lead to issues with some object tracking algorithms (such as SORT [11]) and poses a risk of missing important event details; a near-miss may be undetectable at frame-rates so low.

The authors of "Objects as Points" [24] propose an implementation of object detection using points to determine detections, claiming that they are more efficient than axis-aligned bounding boxes. These points are found using keypoint estimation, and are located in the center of the object's bounding box. The additional information (such as the size, bounds, three-dimensional location, pose, and more) are found using regression. The authors claim that they have obtained the "best speed-accuracy trade-off" in their model implementation of this idea, CenterNet. Results from testing with the Pascal VOC 2007 test prove this to be true; CenterNET achieved as high as 142FPS average using CenterNet-Res101 (at a 512x512 input resolution) and still managed to achieve a 72.6 mean average precision (mAP) with an Intersection of Unions (IoU) threshold of 0.5 (mAP@0.5). This approach is capable of satisfying the needs of this thesis, and could be used as the detection model for the object detection pipeline.

4.4 Motivations and Contributions

While the combination of SORT [11] and an off-the-shelf object detection algorithm solves the challenge of tracking an object in real-time, it fails to address scalability concerns. We must design a scalable software architecture for tracking objects in real-time, allowing for new cameras to be added with ease to increase maintainability for the long term. As a part of this scalable workflow, we will need to submit trackers to a real-time publish/subscribe database (built around Apache Kafka) for graphing and analysis. In order to do so, SORT [11] will require modifications to optimize certain elements for this software architecture.

4.5 Methods

Our approach to scalable object tracking utilizes a few different critical components, to be discussed individually below. The first is our object detection model, which performs basic detection of objects. Following is discussionon our modified object tracking model, e-SORT, which is based off of the Simple Online Realtime Tracker (SORT) [11]. The last component to be discussed is the proposed scalable architecture, which includes discussion on the integration of various components including a detection and tracking model.

4.5.1 Enhanced SORT

Object tracking is one of many applications for video cameras in a smart city, and provides important data to the city's occupants. In most object detection algorithms, an image (be it a video frame or single image) is fed into the model and the detections are made with their corresponding bounding boxes and labels. While this addresses many challenges, a detection model fails to maintain object permanence in between frames. This poses the issue that object tracking must solve: how to maintain object permanence in between frames. Due to the simplicity of the problem object trackers are usually used in conjunction with object detection models (such as the aforementioned YOLO detection model) instead of developing

their own model to detect objects, and are only tasked to help correlate detections between frames to maintain object permanence. Unlike object detection alone, object tracking offers the ability to store all previous object locations, predict future ones, and calculate many variables and statistics based on an *individual* event. This can provide the city with helpful information on traffic trends, pedestrian density, jay-walking zones and more, allowing it to use this data to make the city more intelligent.

The proposed "Simple Online Real-time Tracker" described in [11] is an already highly optimal solution, but lacks many components that are required for our use-case on the MLK Smart Corridor Testbed. As discussed in Section 4.3.1, the original SORT implementation was observed to have several weaknesses such as its sub-optimal obsolete tracklet management, lack of historic location mapping and dependency on frame-rate for accuracy and performance. Due to the design of our scalable architecture combined with the desire to perform further analysis on tracked object data, many modifications to SORT [11] are required to resolve some of these shortcomings. These enhancements are the basis of our modified implementation of the SORT algorithm we call enhanced SORT, or e-SORT, and are discussed in further detail in the paragraphs to follow. e-SORT makes modifications to SORT which will address the many shortcomings and requirements for our implementation in a real-time, scalable environment.

4.5.1.1 Obsolete Tracklets

The original SORT algorithm, for the purposes of memory management, *deletes* obsolete tracklets upon calling the *update* function. Since our publish/subscribe database is not designed to handle the updating of an entry, it is required that the JavaScript Object Notation (JSON) submission contain *all* of the data for the tracklet, as opposed to submitting the data as it comes in. As a result, keeping the obsolete tracklets was a clear solution to this problem, as it would allow us to have all of the information at once and will not be updated (since the tracker is *obsolete* and won't be updated again within SORT). Therefore, mod-

ifications to the *update* function have been made to allow it to also return these obsolete tracklets. After enqueueing the tracklet to the Submission Queue, the obsolete tracklet is deleted. This retains the much needed memory management while meeting our needs for our real-time data infrastructure.

4.5.1.2 Object Labels

Each tracklet in SORT does not store its label - required to simplify the work-flow (as detailed in the Obsolete Tracklets portion above) - but this can be resolved with modifications to the original SORT algorithm. The *update* function now requires a *labels* argument, such that when the tracklet is reassociated to a detection, it can also be reassociated with the correct label. For further improvement, model detection labels may vary for the same tracklet. Instead of overwriting the label each time, a list of all assigned labels from the detection model are kept within the tracklet. When the tracklet's label is then requested using the *get_label* function, the label which is most frequently found is returned. This prevents any anomalies from the labelling process to be corrected in most instances.

4.5.1.3 History of Locations

Another trait required from SORT was a way for each tracklet to retain knowledge of "when" and "where" it was. This takes form as a Python *dict* whose keys are timestamps (in UTC) and whose values are a bounding box (of form x_1, y_1, x_2, y_2). A new key-value pair are appended to the tracklet's locations dict when *update* is called to the SORT instance. This is also the time at which the timestamp is created which delays the timestamp less than a second more than the actual event occurring, which for most use-cases in a smart city is negligible. This history allows for a user to subscribe to the publish/subscribe database later and perform later analysis and calculate properties such as velocity and acceleration. The tracklet's history of locations must be reduced before publishing to our real-time publish/-

subscribe database, as tracklets corresponding to still objects such as parked cars become overwhelmingly large for our data infrastructure, as that is not the use-case for which it was designed. This can be done by applying a threshold for the euclidean distances between two timestamp, as seen in Algorithm 1. This algorithm is important as it prevents a still car from creating *thousands* of entries to flood into the database, while retaining important details such as how long it stayed in that location. In testing, a 30 pixel threshold has been found to be most optimal within our environment.

```
Input: locations, threshold
for timestamp, bbox in locations do
    if distance(last_bbox, bbox) >= threshold then
        ret_val[timestamp] = bbox
        last_bbox = bbox
    end
end
return ret_val
```

Algorithm 1: Tracklet Location History Generalization Algorithm

4.5.2 Scalable Architecture

We propose a parallel architecture using multiprocessing, as seen in Figure 4.2. A blocking queue is used to transfer data between four different types of processes, and there is a queue dedicated for raw frames, processed frames and tracking results. Each of the queues' placements within the architecture can be seen in Figure 4.2. The four types of processes are:

4.5.2.1 Capture Processor

This architecture contains n Capture Processors (represented in blue in Figure 4.2, where n is the number of cameras in the Smart City. Each of these is a dedicated process for capturing video frames from one specific camera on the testbed. The Frame Processor is assigned at the program's start, allocating three Capture Processors to every one Frame

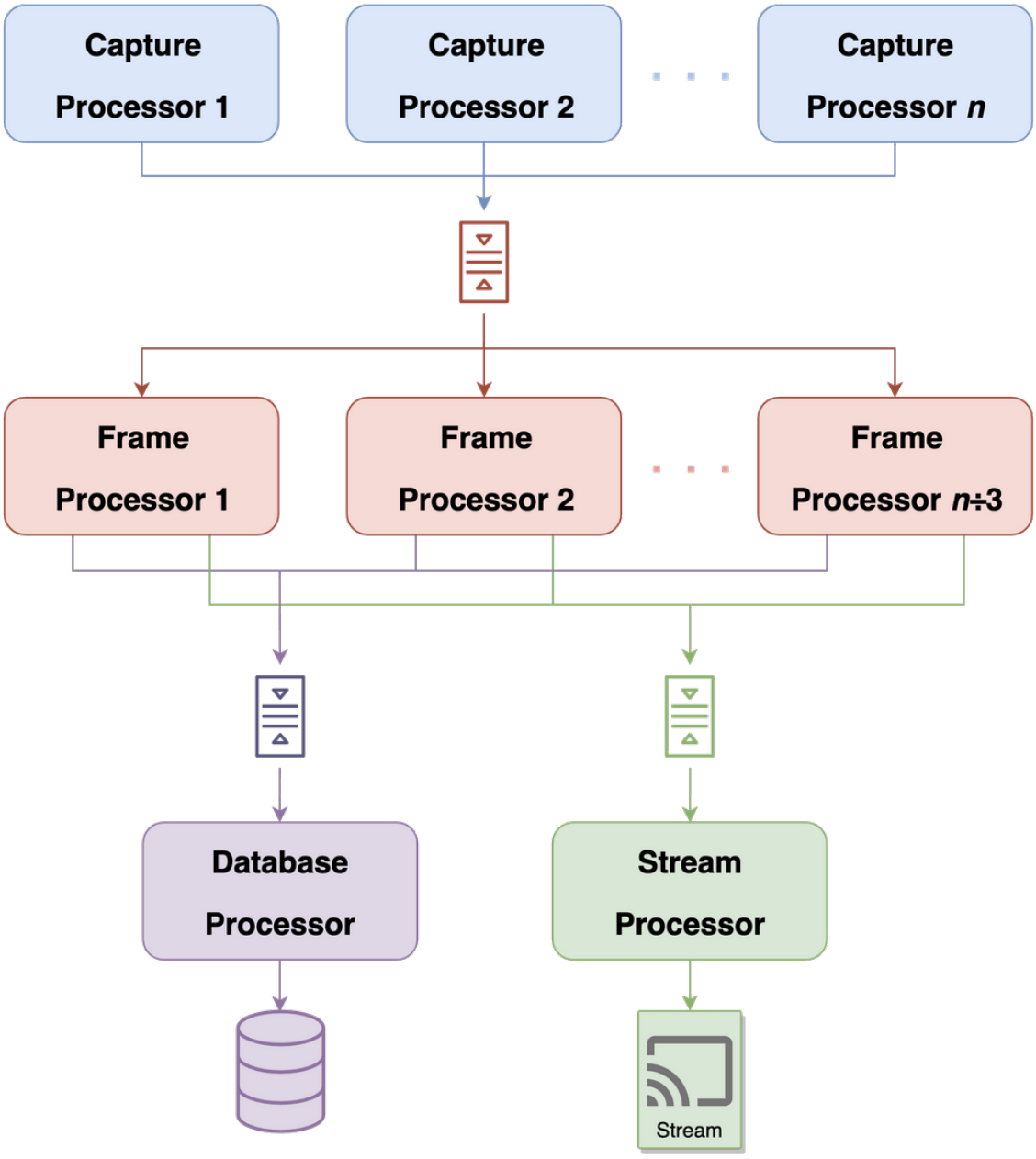


Figure 4.2 Architecture Overview, where n is the number of cameras in the Smart City. Blue indicates n number of Capture Processors, whose purpose is solely to capture video footage. Red indicates $n/3$ number of Frame Processors, whose purpose is to detect and track objects. Purple indicates the Database Processor, whose purpose is to form data from the Frame Processor into a JSON format and submit to our real-time publish/subscribe database. Green indicates the Stream Processor, whose purpose is to stream all n streams for demonstration purposes.

Processor.

4.5.2.2 Frame Processor

This architecture contains $n/3$ Frame Processors (represented in red in Figure 4.2), where n is the number of cameras in the Smart City. This number resulted from testing, where the Frame Processor was allocated another Capture Processor until the output frame-rate was below the input frame-rate. This distribution mechanism also serves as a rudimentary load balancing mechanism for the application. The Frame Processor is responsible for performing object detection and tracking. The results from detection are fed into the SORT instance which corresponds to the frame. Obsolete tracklets from the results of SORT are then fed into the Submission Queue. Following this, the original video frame is destroyed for privacy purposes and the frame from the Frame Processor. Within the Frame Processor, visualizations from tracking are drawn on the frame, which is to be fed into the Stream Queue for real-time streaming.

4.5.2.3 Submission Processor

This architecture contains 1 Submission Processor (represented in purple in Figure 4.2). The Submission Processor uses in the obsolete tracklets from the Submission Queue to create a JSON message containing critical information. This JSON message contains a UUID for the tracklet, the tracklet's label, the tracklet's hit count (the number of locations the tracklet was found within the frame), and the tracklet's history of locations. Once the JSON object is created, it is published to our real-time publish/subscribe database for further consumption. This data allows us to perform several useful analytics such as near-miss detection without sacrificing the privacy of the city's occupants.

4.5.2.4 Stream Processor

This architecture contains 1 Stream Processor (represented in green in Figure 4.2). The purpose of the Stream Processor is purely visual; it enables us to show the results of our tracking on a web interface, which has back-end authentication for privacy purposes. The Stream Processor takes frames from the Stream Queue and streams them using Flask, a flexible, Python-based web-server.

In order to communicate across the multiple types of processes discussed, blocking queues are used. These queues are placed between each type of processor so that each processor may communicate to the next. A queue is used to transfer video frames from the Capture Processor to the Frame Processor. After processing through the Frame Processor, the resulting frame is added to a queue to be streamed by the Stream Processor and the resulting data is added to a queue to be submitted by the Submission Processor. While inter-process communication via shared memory could yield better performance, this approach was not explored due to a shortage of time.

4.6 Results

To test e-SORT and the scalable architecture discussed in Chapter 4.5, we used the testbed located on Chattanooga's MLK Smart Corridor (located in Tennessee). The testbed is an open platform which permits research problems to be tested in a live urban environment. At the time of writing, the testbed contains 27 cameras across 9 major intersections. All IoT devices within the testbed have a 10-gigabit fiber backbone which supports real-time data transfers to and from each pole. The results described in this section come from real-time video stream from this testbed.

This approach has been deployed on two workstation computers equipped with eight-core hyper-threaded processors, each with 16 gigabytes of system memory and an Nvidia GTX 1080Ti. The application discussed in this chapter is capable of running five cameras on each workstation computer. This is mostly a CPU-bound issue resulting from video streaming,

as proven in testing where recorded GPU and memory utilization was low.

Determining the accuracy of our approach poses new challenges, as there are multiple metrics by which accuracy can be determined. This is made possible by the login-protected live streaming of our processed video streams, which allow an analyst to view what the algorithm is detecting. The method of accuracy determination used and discussed in this thesis was a process in which an analyst would count the number of times an event would occur by hand, then calculate the accuracy using total count of objects from the beginning and end of their session. The types of events counted for were multiple-ID instances, no ID instances, and mislabeled instances.

4.6.1 Multiple-ID Instances

Multiple-ID instances are cases in which one object gets multiple IDs, meaning its tracklet was lost too early. In observing the object tracking algorithm, on average for every 115 objects there would be five cases of ID reassignment. Considering that this count of objects is gathered from the visualization, the object count should actually be approximately five ID reassignments per 110 objects. This can be represented as a 95.45% ID retention rate or a 4.55% ID reassignment rate.

The importance of ID reassignment is beyond aesthetic; ID reassignment is a direct result of tracklet reassignment. This indicates that the tracklet prior to reassignment will be considered as obsolete before the object actually is, resulting in a single object represented by two tracklets. This poses a few issues, such as adding duplicate entries in the real-time publish/subscribe database. Since we submit the data from tracking upon retrieval of obsolete tracklets, one object will be represented by two events in the publish/subscribe database. This means that traffic counts may be higher (albeit only by a difference of 5-10 objects), but more importantly the future use of the dataset being created by our approach may be problematic, as two objects' paths may be shorter than they should be.

4.6.2 No ID Instances

No ID instances are cases in which an object is not given an ID at all, indicating that our detection algorithm did not *detect* the object to begin with. We have monitored these events and have found that we have a 2.978% likelihood for improperly tracking / detecting an object; that is to say, we have a 97.022% accuracy in this category. While the number of objects this was analyzed for was 6421 according to the visualization, this is technically incorrect. Considering that 6421 was the count for an average of 97.022% of all *actual* objects through the scene, then the actual count should be approximately 6612 objects.

4.6.3 Mislabeled Instances

Mislabeled instances: cases in which the label from our object detection algorithm failed to give the correct label for the object. We have gathered this metric for 8112 different objects, revealing that only 2.79% of the time is an object mislabeled; that is to say, we have a 97.21% accuracy in this category.

4.6.4 Real-time Visualizations

Visualizations serve a useful purpose in making large quantities of data easily digestible for most individuals. While there are many options to achieve this, two have already been implemented and receive regular use:

Graphing Dashboard: As a part of our server's software stack, we have deployed a graphic dashboard to read the data published to our real-time publish/subscribe database. This allows us to visualize the data easily, making some traffic trends quite clear. Figure 4.3 shows a sample for 10 cameras on Chattanooga Tennessee's up and coming smart city.

Overlay Generator: Written using the OpenCV library for Python, another visualization that uses the real-time tracking data is an Overlay Generator. This overlay generator uses an existing image of the camera view requested, and overlays paths of objects (retrieved

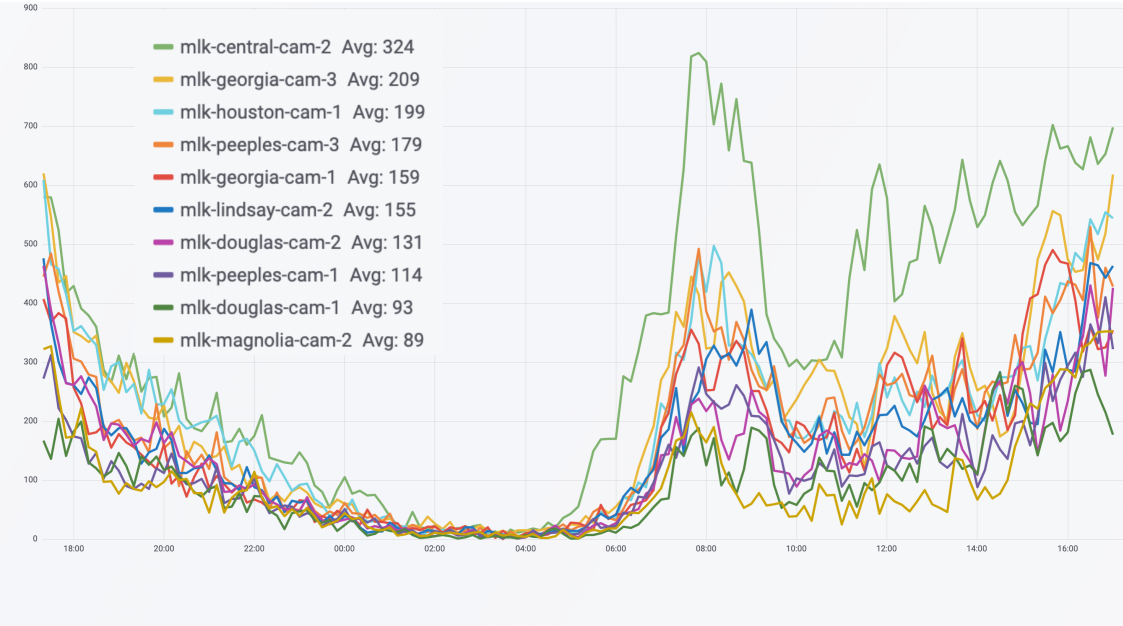


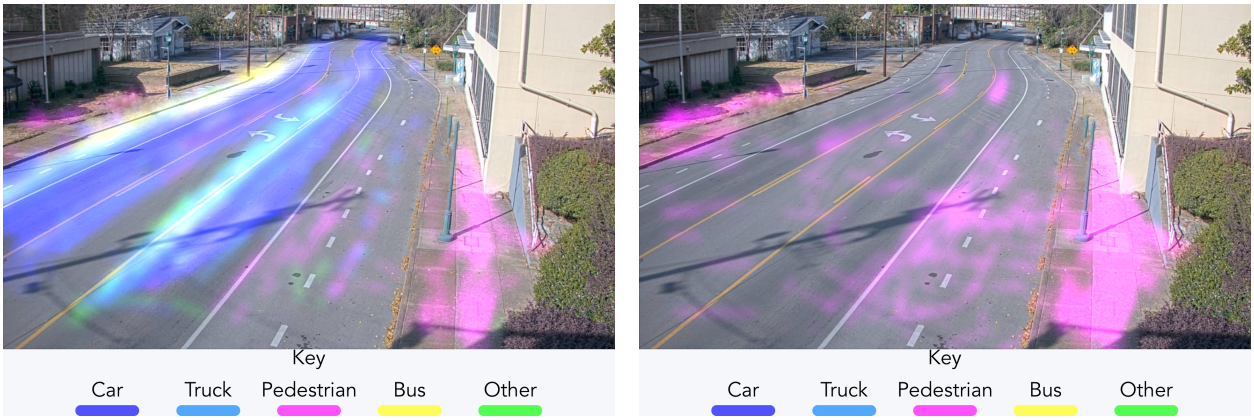
Figure 4.3 Graph of Counts of Objects from Tracking for 24 hours (aggregated into five-minute intervals), where each different color represents a different camera's object counts, e.g., mlk-central-cam-2 Avg: 251 indicates that an average of 251 objects have passed through that camera's view in a 10-minute interval.

from our real-time publish/subscribe database) in different colors for each object. This allows us to show off what the data looks like in an anonymous but still important way, and makes more visible. For example, this imagery makes it possible to see jay-walking trends, vehicles in the bike lanes and more, all of which are shown in Figure 4.4.

Utilizing only ten of the 27 cameras on the MLK Smart Corridor's testbed (due to hardware limitations), our proposed solution to scalable object tracking produces approximately 200,000 results per day. As the accuracy of our proposed solution shows improvement, this number will more accurately reflect a day's worth of traffic in the live urban environment we have performed tests on.

4.7 Summary

In this work, we have discussed our solution to object tracking which has been designed to scale with the size of a smart city. This approach allows for each detectable object



(a) Paths of all objects detected on the testbed.

(b) Paths of pedestrians detected on the testbed.

Figure 4.4 Heat map of the tracked objects from one camera. The figure on the right (filtered to only show pedestrian paths) visualizes jay-walking trends.

(determined solely by the object detection model used) to be treated as an event with unique attributes such as object label, path taken, timestamp of each location and more. With these metrics, it is possible to perform further analysis such as near-miss detection using velocity and trajectory prediction (discussed in Chapter 5) as well as analyze the overall flow of various types of traffic (discussed in Chapter 6)

CHAPTER 5

Near-Miss Detection

5.1 Introduction

In this chapter, we will discuss how we utilize data from the object tracking solution discussed in Chapter 4 for detecting near-miss events. To do so we utilize the stored mapping of historic locations to calculate velocity, which can be used to predict future locations and create potential collision paths.

5.2 Goal of Near-Miss Detection

A near-miss is considered any event in which two objects (often vehicles) are on a trajectory to collide with each other, but some last-minute action is taken to prevent the accident. The goal of successfully implementing a near-miss detection algorithm is two-fold: the first goal is to be able to keep metrics on the environment in which near-miss events occur, such as weather, location, and the objects' locations in reference to each other; the second goal is to create the foundations in which vehicles can receive near-miss alerts from the MLK Smart Corridor testbed itself, rather than from within their car. The advantages to having the testbed create the alert and open the communication to the vehicle is that the cameras within the testbed have a more holistic view of objects' surroundings than the objects themselves. With a near-miss detection solution in deployment, it is possible to build a platform for infrastructure-to-vehicle (I2V) communication for real-time near-miss notifications within vehicles.

5.3 Related Works

As other works regarding near-miss detection on a fixed-location monocular camera have not yet been written or published, this section will discuss other approaches to monocular near-miss detection in varying environments. The reason for specifying a *monocular* camera is that the approach discussed in this thesis utilizes a monocular camera as well. Unlike a monocular camera (a camera with only a single sensor), stereo cameras - those with two sensors at a fixed distance - have one major advantage that makes near-miss detection substantially more trivial: the fixed distance between each sensor in the camera. Since the distance between each camera is fixed, calculating the velocity and acceleration of an object is reduced to a detection task, wherein the distance travelled in inches or centimeters can be calculated by taking the time delta between the object's initial detection in each lens. While this approach is naive - as precision relies on the speed of the object detection model - it still vastly simplifies the problem as acceleration and velocity play a critical role in detecting near misses (discussed in further detail in Section 5.5).

The authors of "Video Analytics for the Detection of Near-Miss Incidents on Approach to Railway Level Crossings" [25] introduce an algorithm for detecting near-miss events at railway crossings. Their approach aims to using a train-mounted monocular camera to detect the presence of a vehicle stopped at a railway crossing. As this camera is mounted on a train, the authors exploit the fact that rail road tracks in their country are 1067 millimeters apart. Not only does this allow the authors to trivially convert pixels to meters, but it enables them to skew and distort the image to create an accurately-scaled birds-eye view of the scene even though the camera isn't mounted facing straight down. By creating this birds-eye view, the problem of near-miss detection is reduced from three dimensions (which simple video footage cannot perceive) to two dimensions. By reducing the dimensionality of the problem, the authors have created a much simpler task for themselves, making the near-miss detection a problem of algebra and geometry. Unfortunately, the testbed on which the algorithms discussed in this thesis have been deployed on has no constants; overall lane count changes, and a substantial margin of error can be found in the width of some lanes.

Without this advantage that the authors have, it becomes impossible to create an accurate birds-eye view of the street, thus preventing the reduction of dimensionality.

In the work described in "A Cost-Effective Framework for Automated Vehicle-Pedestrian Near-Miss Detection Through Onboard Monocular Vision" [26], the authors describe a near-miss detection algorithm which uses video footage from on-board cameras on vehicles. Their approach detects near-miss events between the vehicle recording and pedestrians within the camera frame. Their approach detects pedestrians using a Histogram of Gradients (HOG) [27], allowing the authors to detect pedestrians without the use of an object detection model. They use the KLT tracker to track their detected pedestrians across video frames in order to calculate relative speed using a delta in time and bounding box location. While this approach is most similar to the one discussed in this thesis, it has a few shortcomings. To begin, the authors' approach is not developed for real-time use, preventing the data from ever being usable for real-time I2V communication to prevent an accident from occurring. Additionally, the authors' approach uses a camera from inside the vehicle, forcing a myopic view for near-miss detection to occur within. The approach described also only works on pedestrians, meaning that their approach cannot work between vehicles or between that vehicle and other objects such as bicycles or motorbikes. While this isn't solely a limitation of using a Histogram of Gradients, it certainly may exceed the abilities of HOG-based detection when considering the ability to detect many types of many objects at once in a single frame.

The authors of "Drive Video Analysis for the Detection of Traffic Near-Miss Incidents" [28] propose a near-miss detection algorithm which is used to create a near-miss incident database (NIDB). This database, designed to be used to make driver assistance systems more robust, stores extracted video footage of a near-miss event occurring. Their implementation uses recorded video footage from in-vehicle mounted cameras, mounted inside of 100 taxis. These cameras were designed to begin recording in the event that the driver had to break urgently. Their cameras ran from 2006 to 2015, and in this range they gathered over 60,000 videos from these urgent breaking events, which were manually annotated and stored in the NIDB. Additionally, the authors have implemented a validation algorithm for detecting near-miss

using Trajectory-Pooled Deep-Convolutional Descriptors (TDD) [29] to take advantage of the temporal dimension added by using a TDD to calculate trajectory. Having the trajectory of an object allows the authors to determine if any trajectories cross. If the trajectories cross, then they can calculate Time 'til Collision (TTC) [30] to determine how long until the two objects may collide. With this, a threshold on TTC is applied to determine the severity of the near-miss (high or low-risk, depending on how much time until the collision may occur). One shortcoming of this approach is that the TDD used is not capable of real-time implementations. Similarly, the authors' approach uses pre-recorded in-vehicle footage, which allows them to use more complicated approaches as there are no real-time requirements. As such, this approach is also incapable of alerting drivers of the risk of near-miss using I2V because of its offline nature.

5.4 Motivations and Contributions

All of the near-miss detection methods that have been discussed so far have some critical flaw that keeps them from being adaptable into the scalable, real-time environment of the testbed. Fortunately, concepts from the work in [26] and [28]. From Chapter 4, this thesis has introduced a robust tracking solution similar to the work discussed in A Cost-Effective Framework for Automated Vehicle-Pedestrian Near-Miss Detection Through Onboard Monocular Vision [26]. Unlike their approach, the approach discussed in this thesis is able to track and detect *more* objects (such as other vehicles, busses, motorcycles and bicycles), as well as functions in real-time. Similarly, the work in [28] has implemented an approach that also tracks objects offline, as well introduces the concept of TTC [30]. The implementation of near-miss detection discussed in this thesis integrates concepts such as tracking-based velocity and acceleration calculation combined with trajectory prediction of objects. With the predicted trajectory, a threshold of TTC can be applied to this predicted trajectory to perform real-time, scalable near-miss detection that is built on top of the object tracking architecture discussed in Chapter 4.

5.5 Methods

5.5.1 Velocity and Trajectory

As discussed in Chapter 4, the implementation of scalable object tracking discussed in this thesis stores historic bounding box locations for each tracked object. Since these bounding box locations are mapped to a UNIX timestamp, it is possible to calculate velocity (in terms of x and y) using the change in time and bounding box. The formula for velocity is written as $velocity = \frac{|pos_{current} - pos_{previous}|}{time_{current} - time_{previous}}$. This formula yields a pixel-relative velocity for both x and y (seen in Figure 5.1).

Figure 5.1 Pixel-Relative Velocity drawn above a tracked bus (indicated by the green rectangle) on the testbed. Velocity is being calculated using the data from object tracking, using the delta in time and position for calculation.

The calculated velocities for x and y can be applied to the tracked object's current bounding box in order to predict a future location, thus predicting the trajectory of the object. Similar to velocity, trajectory prediction is also pixel relative, and is in terms of x and y. The predicted trajectory of an object is formulated as $traj_{pred} = pos_{curr} - (velocity \times \Delta time)$. An application of this algorithm is shown in Figure 5.2.

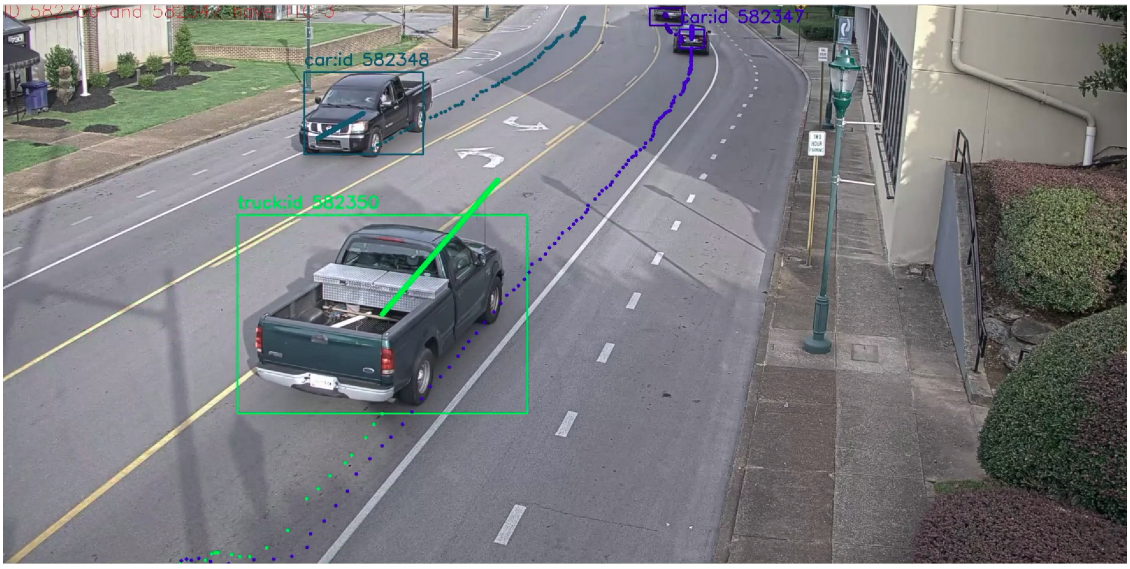


Figure 5.2 Trajectory prediction drawn alongside tracked objects on the testbed. The circles trailing each object represent previous locations of that object, and the colors are used to more easily match which paths correspond to which object. The line from the center of the object represents the predicted trajectory of that object.

5.5.2 Near-Miss Detection

Using the predicted trajectory of an object, we can compare the predicted trajectory of all other tracked objects that are present in the video frame. With this, we can calculate the TTC [30] between any of the objects whose trajectories cross. The algorithm for TTC begins by calculating the delta in the predicted trajectory of two tracklets. If the delta of the predicted trajectories is less than the delta if the known locations of the tracklets, we begin the TTC calculation algorithm defined in Algorithm 2.

Input: predictedDistance, knownDistance
if *predictedDistance* < *knownDistance* **then**
  ttc = 0
  while *knownDistance + (predDistance - knownDistance)* × *ttc* > *0* **do**
    ttc += 0.1
  end
  return ttc
end
return-1

Algorithm 2: The algorithm for calculating the TTC using a predicted distance and known distance as inputs, and returning a numerical value representing how long until two objects may collide, or -1 if they are not at risk of collision.

For these objects with intersecting trajectories, a threshold (similar to that in [28]) can be applied to determine when a near-miss event has occurred. The authors of the near-miss detection model in [28] have stated that they applied a threshold of $0.5 < TTC < 2.0$, but do not explain how they have derived these numbers. Since the authors fail to validate their TTC threshold, an experiment must be performed in order to validate it ourselves.

5.5.2.1 Data Usage

One concern regarding the data usage is how the resulting should be used immediately after creation; should detected near-miss events be sent to our real-time publish/subscribe data infrastructure, or should this data be used directly for real-time alerts? Due to the real-time nature of Apache Kafka (the basis of our real-time publish subscribe data infrastructure), the data is available to consumers (workers dedicated to reading data from Kafka) as soon as the data is pushed to Kafka by a producer [31]. Therefore, it could be said that as soon as the data is pushed, any EDGE-compute device deployed directly onto the testbed could be used to read this data in real-time as well. Once the EDGE node receives the message, it could then alert any potential victims of the near-miss before it is too late . The benefit of this method is that the inter-device communication is handled by the already implemented Kafka libraries and wrappers, so this would make the communication between the near-miss-detecting server and the EDGE compute node trivial. Conversely, while this is a perfectly practical approach, it could be argued that a custom implementation (such as a socket-based one) could be created to directly notify an EDGE-compute node of the near-miss event, thus reducing any latency as the messages travel through the network. This can ensure that the data reaches the potential victims first, and is stored after the notification has been made.

Detected near-miss events are sent to our real-time publish/subscribe database for later analysis and visualization. These messages are stored separately from the other messages from ordinary object tracking, but include the information required to cross-reference the

datasets in order to re-create the scenario in simulation for further analysis. To implement this behavior, we include the UUIDs of both objects when we submit the message to our data infrastructure. Doing so allows us to cross-reference the two datasets using UUID as jthe common key, thus allowing analysis on the objects' entire paths, object types, time, pixel-relative location and more (as discussed in Chapter 4.5.2).

5.5.2.2 Time Til Collision Threshold

Another concern to address deals with the TTC threshold applied - what does the threshold mean and how was it chosen? In this portion of this section, we discuss the interpretability of the TTC threshold and validate the threshold used in [28] as it pertains to our implementation. To begin, the TTC threshold defines the minimum and maximum acceptable threshold to be considered a near-miss. Should the TTC exceed the upper bounds of this threshold, then this generally means that two object's trajectories at low risk of collision [28]. Conversely, should the TTC fall below the lower bounds of threshold, this does not *necessarily* mean that a collision has occurred - this is only plausible (at best) if the TTC reaches zero or near zero - but it means that one of the two objects involved must take imminent action. In the authors' TTC threshold $0.5 < TTC < 2.0$, this means that one of the objects has less than 0.5 seconds to react and prevent a collision. Results from this test can be seen in Table 5.1.

Based on this information, the purpose of the threshold and its importance to the accuracy of the detection algorithm should be clear. Therefore, it seems necessary to address how the TTC thresholds are chosen. The authors of [28] do not discuss why they chose the threshold $0.5 < TTC < 2.0$, thus testing and validating this threshold is a necessary step for the purposes of this thesis. A testing approach was created, wherein the TTC between two tracked objects is compared against a range of thresholds. The lower and upper bounds are tested using increments of 0.5 between 0.5 and 5. By doing so we can compare a range of TTC thresholds and determine in which values the TTC was accepted. By applying this

Table 5.1 Results from TTC Threshold Testing

Threshold	**Percent Accepted**	**Threshold**	**Percent Accepted**
$0.5 < ttc < 1.0$	0%	$2.5 < ttc < 4.0$	23%
$0.5 < ttc < 1.5$	0%	$2.0 < ttc < 4.5$	27%
$0.5 < ttc < 2.0$	0%	$2.0 < ttc < 5.0$	27%
$1.0 < ttc < 1.5$	0%	$2.5 < ttc < 4.5$	27%
$1.0 < ttc < 2.0$	0%	$2.5 < ttc < 5.0$	27%
$1.5 < ttc < 2.0$	0%	$0.5 < ttc < 2.5$	64%
$2.0 < ttc < 2.5$	0%	$0.5 < ttc < 3.0$	64%
$2.0 < ttc < 3.0$	0%	$1.0 < ttc < 2.5$	64%
$2.5 < ttc < 3.0$	0%	$1.0 < ttc < 3.0$	64%
$3.0 < ttc < 3.5$	0%	$1.5 < ttc < 2.5$	64%
$3.0 < ttc < 4.0$	0%	$1.5 < ttc < 3.0$	64%
$3.5 < ttc < 4.0$	0%	$0.5 < ttc < 3.5$	88%
$4.0 < ttc < 4.5$	0%	$0.5 < ttc < 4.0$	88%
$4.0 < ttc < 5.0$	0%	$1.0 < ttc < 3.5$	88%
$4.5 < ttc < 4.5$	0%	$1.0 < ttc < 4.0$	88%
$4.5 < ttc < 5.0$	0%	$1.5 < ttc < 3.5$	88%
$3.0 < ttc < 4.5$	3%	$1.5 < ttc < 4.0$	88%
$3.0 < ttc < 5.0$	3%	$0.5 < ttc < 4.5$	91%
$3.5 < ttc < 4.5$	3%	$0.5 < ttc < 5.0$	91%
$3.5 < ttc < 5.0$	3%	$1.0 < ttc < 4.5$	91%
$2.0 < ttc < 3.5$	23%	$1.0 < ttc < 5.0$	91%
$2.0 < ttc < 4.0$	23%	$1.5 < ttc < 4.5$	91%
$2.5 < ttc < 3.5$	23%	$1.5 < ttc < 5.0$	91%

algorithm, we are able to modify the TTC threshold to determine which results in the most accurate boundary of time in which a tracked object must respond to avoid collision. The results from this experimentation is discussed in the next section.

5.6 Results

In order to perform tests on the TTC threshold, a dataset was created by performing near-miss detection on 6 hours of data and 90 potential near-miss candidates using the near-miss detection methods discussed in this chapter. In this dataset, we were able to determine some of the weakest and strongest candidates for intervals in which TTC can lie to be considered a near-miss. Astonishingly, the threshold $0.5 < TTC < 2.0$ provided by the authors of [28] rejected all 90 events as possible near-misses. This may represent a strict near-miss threshold,

allowing only the most last minute of events from being considered as likely near-miss events, which may in fact be accurate. Due to the environment and circumstances of the experiment, obtaining known good misses for repetitive use is both against CUIP's privacy policy, and highly difficult to accomplish from a technical perspective. Therefore, in the case of the experiment performed, this threshold was too strict to justify any candidates and near-misses. On the other end of the spectrum, a larger interval (such as $0.5 < TTC < 4.5$) allowed up to 91% of candidates to be considered a near-miss. When prioritizing safety-first, allowing more potential near-misses pass through the threshold is ideal; conversely, when conservation takes priority, the stricter thresholds such as the one suggested are more ideal.

In general, the interval between $0.5 < TTC < 2.5$ resulted in 64% of the near-miss candidates to be considered a near-miss. Overall, the threshold chosen for TTC is highly dependent on the use-case; an on-board unit (OBU) creating dozens of false-positive near-miss alerts would be frustrating to the user, but not receiving an alert at all due to a false-negative circumstance could prove both dangerous and impractical. For the purposes of this thesis and the long-term deployment of this technology, we will use the median threshold of $0.5 < TTC < 2.5$ which yields approximately 10 near-miss events per hour. Images resulting from our approach are shown in Figures 5.3 and 5.4.

5.7 Conclusions for Near-Miss Detection

In this chapter, we introduced the concept of determining near-miss events using the pre-existing software architecture discussed in Chapter 4. This work makes it possible for any monocular camera to determine near-miss events automatically. In order to detect a near-miss event, we use the historic location data stored in tracked objects to calculate pixel-relative velocity, which is used to calculate Time 'til Collision, or TTC. With this TTC value, a threshold can be applied to filter out only the events whose TTC is considered within the definition of a near-miss. To apply the best threshold, an experiment was created to determine that the threshold $0.5 < TTC < 2.5$ yielded a median number of near-miss

Figure 5.3 A near-miss event being detected between two vehicles at MLK and Peeples St., shown by the red line and number drawn near the top of the image.

Figure 5.4 A near-miss event being detected between two pedestrians at the intersection of MLK and Georgia Ave., portraying the success of the near-miss approach introduced in this chapter.

events at approximately 10 near-miss events hourly.

The approach discussed in this chapter allows us to notify travelers on the MLK Smart

Corridor Testbed of the event as it's happening. While our approach is naive in the sense that it does not take consideration for the third dimension (depth), it can still be used practically on many of the camera angles of the many cameras on the MLK smart corridor testbed. Where it is not practical or possible to ignore the depth component in near-miss detection, an approach similar to the work in [32, p.32-33] can be used to approximate depth. Using depth, we can continue to apply the same near-miss and TTC algorithms using this third dimension to greatly improve the accuracy and usability of the approach discussed in this chapter. This will be discussed further in Section 7.2.2.

CHAPTER 6

Traffic Flow Analysis

6.1 Introduction

This chapter will introduce our approach to traffic flow analysis, which uses the scalable object tracking implementation discussed in Chapter 4 to gain insight on what paths objects on the testbed take. This approach allows us to see details in citizens' commutes such as which lanes are being driven or which crosswalks are taken.

6.2 Goal of Traffic Flow Analysis

The ability to determine the routes taken by objects (such as pedestrians and vehicles) on the MLK Smart Corridor testbed has many practical uses such as real-time route optimization and robust traffic congestion data acquisition. The goal of implementing a traffic flow analytics platform is to be able to determine in real-time what actions travelers on the testbed choose to take. Not only would this make it possible to build a model for predicting traffic flow, but it allows us to show users the real-time congestion status of the intersections throughout the testbed. In the long-run, this enables us to build out route-optimization algorithms using this real-time data in order to optimize the travel time and fuel efficiency of vehicles in downtown Chattanooga Tennessee [33].

Having real-time analytics on the state of traffic in the testbed allows us to optimize traffic management in many ways [33], but automating this process without duplicating results can often be challenging. The goal of this chapter is to use the real-time object tracking implementation discussed in Chapter 4 to determine object routes. Data from

these routes can be used to help citizens with real-time information to help re-route them on the best path for fuel-efficiency or time. Additionally, this route data can be used to give a robust definition of traffic congestion, detailing exactly in which lanes the congestion lies. In order to accomplish this, a directed graph approach will be used to interpret the possible movements of pedestrians, cyclists and vehicles as they traverse the testbed. This graph can enforce "regular" traffic paths (such as ensuring that a vehicle isn't on a sidewalk), while also reporting anomalies such as these for safety assessment in the future.

6.3 Related Works

The authors of "CarWeb: A Traffic Data Collection Platform" [34] introduce *CarWeb*, their GPS-based implementation of a traffic data collection platform. Their paper, introduced in 2008, introduces a collection platform that utilizes the GPS of mobile phone devices within vehicles. This platform is said to be designed to provide real-time traffic information for route optimization [35] and intelligent transportation systems [36]. While their approach achieves their goal of creating a network which provides real-time traffic flow analytics and the concept was novel when it was introduced, this concept has been seen in many modern-day mobile mapping applications (such as Apple Maps and Google Maps), where traffic congestion is not crowd-sourced (such as Waze's implementation of traffic flow). Conversely, this implementation does not take into account those individuals without mobile devices or with GPS capabilities disabled. In these cases, these individuals are not recorded or taken into account when considering congestion and overall traffic flow.

The authors of "Microscopic Traffic Data Collection by Remote Sensing" [37] have developed an offline method for creating a dataset from traffic flow patterns in recorded footage. This data was collected by a digital camera inside of a helicopter, and was used with their software capable of detecting and tracking vehicles from video footage. While this paper was introduced in 2003, their approach to using traffic flow analytics through object tracking is an approach very similar to the one to be discussed in this thesis. Conversely, the

scale and practicality greatly differ; the utilization of a helicopter for offline imagery of the Dutch highway is highly impractical in both costs and benefit, especially since it only captures 210m of the highway. A dedicated internet-capable camera would allow for constant real-time analysis and greatly expand the scope that their algorithms could run. In their paper, the authors state that the algorithms are greatly influenced by weather and camera movement, both of which are greatly improved when using modern-day object detection and tracking approaches such as the ones discussed in this thesis. Lastly, the authors explicitly state that their approach only works on vehicles, which is a valid goal for a motorway though a shortcoming for our necessities in a live urban environment. Many of the shortcomings and impracticalities of this approach are addressed by the work discussed in this thesis.

The authors of "Video-Based Vehicle Detection and Classification System for Real-Time Traffic Data Collection Using Uncalibrated Video Cameras" [33] discuss their approach to collecting data on traffic patterns while foregoing the use of typical traffic sensors. The authors propose their cost-effective approach to create a video-based vehicle detection and classification system in order to effectively detect and count vehicles using pre-existing surveillance cameras. Their implementation resolves issues with negative effects from object occlusion and slight camera vibrations in order to detect vehicles with 97% accuracy. These issues are no longer present in modern-day implementations of object detection and object tracking, wherein occlusion is an expectation that is accounted for (seen in [11, 19–21, 38] and many more). While their approach allows them to gather vehicle count with reasonable reliability, the authors were unable to resolve their issues with headlight reflection, camera vibrations, and occlusion, adding additional inaccuracies to their traffic data collection approach.

6.4 Motivations and Contributions

While the authors' works discussed in Section 6.3 were unique for their given environments and limitations, none fit directly into the MLK Smart Corridor testbed. We want to

be able to integrate real-time traffic flow analysis with the already implemented real-time object tracking approach discussed in Chapter 4. This implementation should be able to detect anomalies in individuals' commute (such as driving on the sidewalk and jay-walking), while also making it possible to consume these metrics in real-time using our real-time data infrastructure. Lastly, this approach makes it possible to quickly and easily associate a tracked object with some action for events such as turning, crossing crosswalks, and more. This approach should run after the obsolete tracklets (discussed in Chapter 4) are gathered, avoiding any additional latency or overhead for the original object tracking and near-miss detection approach.

6.5 Methods

Determining the location of an object in terms of a region of interest (ROI) is a multi-stage process. This process begins with mapping polygon coordinates (pixel-relative) to ROIs; we must define a polygon that describes an area, such as the left turn lane. Next, we must apply these polygons to objects' mapping of historic bounding boxes to determine which section it is in. To help mitigate issues with depth, our approach should focus on a point at the bottom-center of the bounding box as it is most accurately represents the change in an object's position without being skewed by depth. This is necessary to most accurately describe the paths of objects, and is visually explained in Figure 6.1. There may be the possibility that the bottom-center point of the object lies on the intersection of two regions of interest, as the creation process of the ROI-representing polygons is approximate. In the event that the bottom-center point lies inside of two ROIs, the intersection of unions, or IOU, is taken between the polygons and the objects bounding box. Whichever intersection has the highest area will be chosen as the ROI for the object at that time. Lastly, the locations are applied to a directed graph to determine and validate the actions of the tracked objects, detect any anomalous movement (such as jay-walking or dangerous driving), and describe the action taken in plain terms.

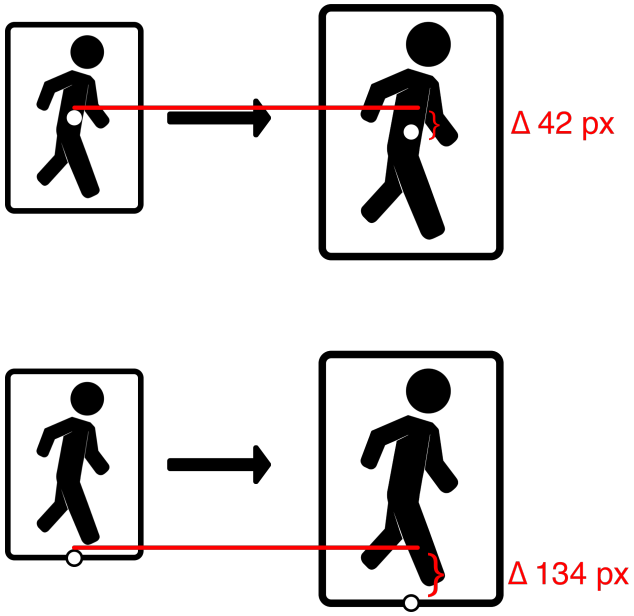


Figure 6.1 Top: Skewed delta in Y-value, showing a delta of only 42 pixels due to depth. Bottom: Actual delta in Y-value, showing a delta of 134 pixels. This difference is important as it indicates the actual movement of the object instead of the perceived movement produced by the camera's lack of depth.

6.5.1 ROI Labelling

In order to create polygonal representations of every ROI seen by cameras on the MLK Smart Corridor testbed, a tool named LabelMe [39] (based the original web-based annotation tool [40]) was used for its ability to annotate polygons, rather than rectangles. This advantage allows us to annotate every ROI across every camera deployed on the testbed, with a label corresponding to its purpose. During the labelling process, single-lane portions of Martin Luther King Boulevard were simply labeled "MLK-EB" or "MLK-WB" for east-bound or west-bound traffic respectively. Crosswalk labels describe the street they are crossing with their cardinal direction, and sidewalks are labelled with the assumption that pedestrians, like drivers, walk on the right side of the road. The LabelMe tool produces a JSON file (seen in Figure 6.4), organized by camera id similar to the other algorithms described in Chapter 4. The polygons were turned into objects using the Python library shapely [41], which handles the complexities of polygonal geometry for us. You can see some examples of the labelling process in Figures 6.2 and 6.3.

6.5.2 ROI Matching

Once the polygon-representations of ROIs-per-intersection have been created, it is then possible to apply these ROI representations in-code. To do so, the set of points created from the labelling process are used to create polygon instances using the Python library shapely [41]. Once these are created, we are able to use the library to determine if some point of form (x, y) lies within this polygon. As discussed in our methods, we use the bottom-center of the object's bounding box as this point to both avoid skewed y-values due to depth and to best determine what surface the object is on.

Also discussed in our methods is the chance that an object's bottom-center point lies inside of more than one ROI polygon. In the event that this happens, we take the IOU of the object's bounding box against the polygons (once again using geometry utilities provided by shapely [41]). Whichever polygon has the highest IOU with the object's bounding box is chosen. This algorithm can be seen in detail in Algorithm 3.

```
Input: collidedPolygons, boundingBox
highestIou = 0
highestPoly = None
for label, shape in collidedPolygons do
    iou = shape.intersection(boundingBox).area / shape.union(boundingBox).area
    if iou > highestIou then
        highestIou = iou highestPoly = shape
    end
end
return highestPoly
```

Algorithm 3: The algorithm for picking an ROI Polygon based on intersection of unions. Inputs include a list of the polygons which the bounding box collided with, and the bounding box itself. Outputs the polygon which the bounding box most collided with.

6.5.3 Action Definition

The last goal was to create easily understandable actions for objects using the implementations of traffic flow discussed in this chapter. These actions should be formatted similar to "took left turn", "crossed crosswalk", "continued straight", etc.. In order to accomplish this, a directed graph approach can be taken to gain multiple benefits. The first of these benefits

is that the directed graph can be used verify the path of the object by enforcing possible movements of the object. By defining the permissible path for each type of object tracked on the testbed, we can catch anomalous actions such as pedestrians walking in streets or vehicles driving on the sidewalk.

The second benefit of using a directed graph is that it can allow us to understand the path that was taken in mathematical context, upon which we can apply algorithms to determine the action taken. An example of the logic for determining the action of a pedestrian can be seen in Algorithm 4. In lines 2 through 3, we ensure that the iteration is still within the bounds of the graph. Lines 4 and 5 initialize temporary variables for easier understanding of index positions. Line 6 begins traversing a path in which the sidewalk is currently being taken. If it is, lines 7 through 12 verify if a crosswalk was recently traversed. If the currently traversed path isn't at a sidewalk, then lines 16 through 21 verify if a sidewalk was recently traversed assuming that a crosswalk is currently traversed. If in any of these cases these predictable paths are not adhered to (for example, starting or ending in a crosswalk), then this is reported as an anomalous action.

6.6 Results

In a real-time testing environment ROI matching have been proven successful. While this approach has many uses beyond the scope of traffic flow analysis, we have tested this approach on a real-time video stream from the MLK Smart Corridor's testbed. The success of the ROI matching process can be seen in Figure 6.5. In this figure we can see that the objects are properly correlated to the current lane they are in, and this data is recorded for later analysis such as the path generalization discussed in this section. While implementations for all objects traversing the testbed have not yet been established, the action generalization process has also proven successful in preliminary testing. Unfortunately, a use-case for this data has not yet been proposed or developed.

```
   Input: objectGraph
 1 for index, node in objectGraph do
 2 |   if index − 1 >= 0 then
 3 |   |   if index + 1 < len(objectGraph) then
 4 |   |   |   previous = objectGraph[index - 1]
 5 |   |   |   next = objectGraph[index + 1]
 6 |   |   |   if node.label == "sidewalk" then
 7 |   |   |   |   if previous.label == "crosswalk" then
 8 |   |   |   |   |   return "Crossed Crosswalk"
 9 |   |   |   |   end
10 |   |   |   |   if next.label == "crosswalk" then
11 |   |   |   |   |   return "Crossed Crosswalk"
12 |   |   |   |   end
13 |   |   |   end
14 |   |   |   else
15 |   |   |   |   if node.label == "crosswalk" then
16 |   |   |   |   |   if previous.label == "sidewalk" then
17 |   |   |   |   |   |   return "Crossed Crosswalk"
18 |   |   |   |   |   end
19 |   |   |   |   |   if next.label == "sidewalk" then
20 |   |   |   |   |   |   return "Crossed Crosswalk"
21 |   |   |   |   |   end
22 |   |   |   |   end
23 |   |   |   end
24 |   |   end
25 |   end
26 end
27 return "Continued Straight"
```

Algorithm 4: The algorithm for picking an action based on the graph made for the object's path.

6.7 Conclusions for Traffic Flow Analysis

Designing and implementing a traffic flow data collection algorithm which works in conjunction with the object tracking approach discussed in Chapter 4 requires some manual pre-labelling, but yields significantly helpful results for use in intelligent routing systems and real-time mapping applications. The approach discussed in this chapter allows us to accurately determine routes of vehicles and pedestrians, as well as catch any anomalies such as jay-walking and poor driving practices. The data produced by the methods discussed in this chapter allow for predictive modelling and forecasting based to further help the community in which the MLK Smart Corridor presides.

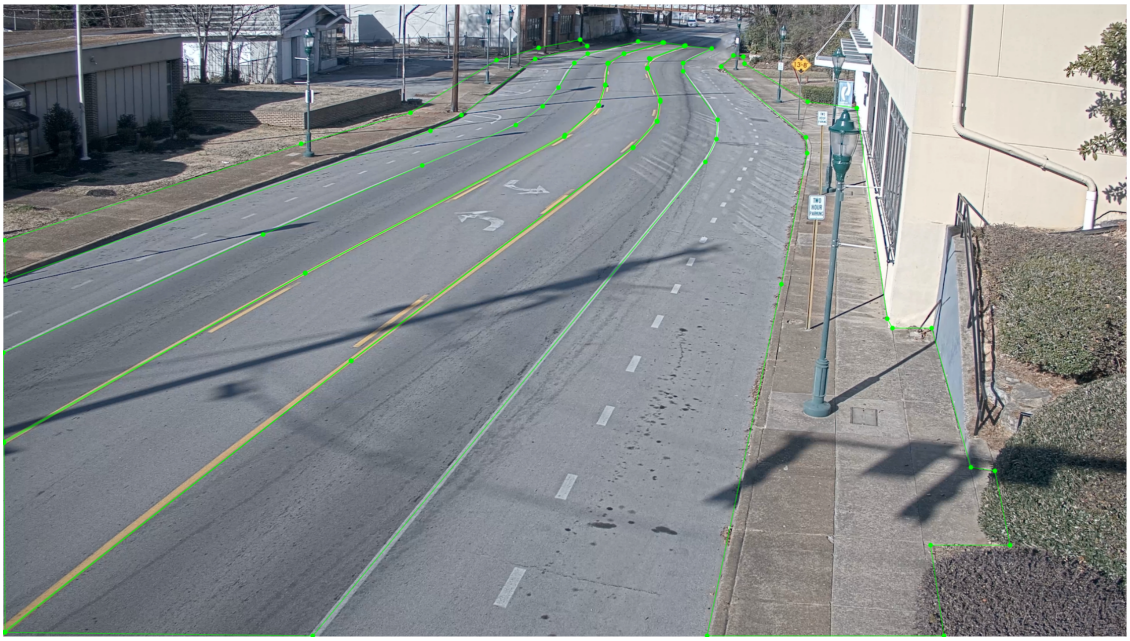


Figure 6.2 Polygonal ROI layout visualized at the intersection of MLK and Peeples. These are provided as a result of using LabelMe to hand label each segment of the street and sidewalk.

Figure 6.3 Polygonal ROI layout visualized at the intersection of MLK and Georgia.

```
{
  "label": "MLK-WB-LEFT-TURN",
  "line_color": null,
  "fill_color": null,
  "points": [
    [
      329,
      397
    ],
    [
      430,
      382
    ],
    [
      178,
      327
    ],
    [
      142,
      335
    ]
  ],
  "shape_type": "polygon"
},
```


Figure 6.4 Resulting JSON Document from Labelling the Intersection of MLK and Peeples, which makes it possible to read in the data from manual labelling into our script automatically.

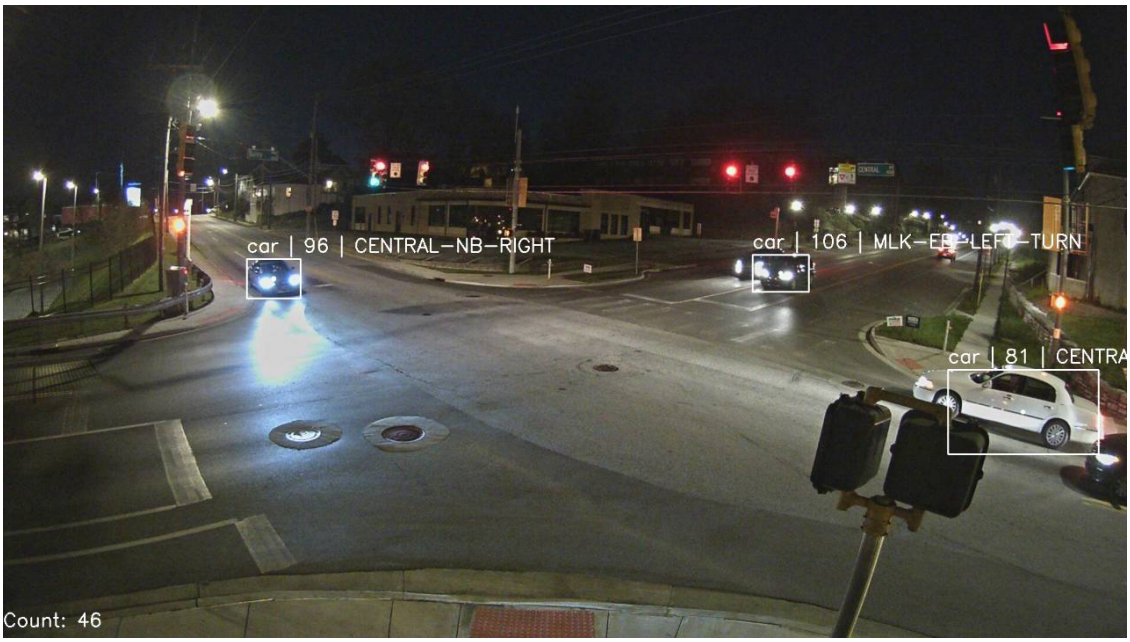


Figure 6.5 The ROI matching implementation live on the MLK Smart Corridor at Central Ave., showing each vehicle being both tracked and matched to a road segment from manual labelling.

CHAPTER 7

Conclusion

7.1 Closing Thoughts

This thesis discusses three implementations of video analytics using the testbed on the MLK Smart Corridor. We utilize the testbed's cameras to perform video analytics task at large-scale, and consider the city's growth to create video analytics applications with scalability in mind. All forms of video analytics discussed in this thesis prioritize real-time analytics and personal privacy to create fully anonymous solutions that can still aid citizens in their day-to-day commute throughout the corridor.

The basis of the video analytics tasks discussed in this thesis is scalable object tracking, discussed in Chapter 4. This implementation of tracking allows us to treat objects as events, wherein the object is able to be tracked for the entirety of its existence within the camera's viewing angle. Once the object leaves the viewing angle, we are able to submit the data from the tracked object to our real-time data infrastructure, wherein the data can pursue further processing or can be consumed by another software implementation.

This basis makes the remaining two video analytics possible. The near-miss detection approach discussed in this thesis utilizes the object tracking implementation of historic object locations to calculate a pixel-relative velocity for use in trajectory prediction. This predicted trajectory can be used to obtain the TTC between objects on the testbed. Should the TTC fall within the threshold, then a near-miss is detected and recorded.

Our object tracking basis also allows us to perform traffic flow analytics and data collection. With the data collection implementation, we are able to determine any anomalous activity such as jay-walking pedestrians or poor driving habits. Additionally, a directed

graph approach was used to translate the path taken by an object into a string which best describes the action overall. The approach used for traffic flow data collection can be used to build a predictive model, and analytics on this collected data can be used to determine traffic trends throughout the MLK Smart Corridor.

7.2 Future Work

Below we discuss tasks for future work as they relate to each variant of video analytics proposed in this thesis.

7.2.1 Scalable Object Tracking

There are a few methods which can be applied to improve the proposed solution for scalable object tracking, the first of which has to do with frame-rate. Due to the nature of how SORT [11] works, it is very frame-rate dependent; since it uses a Kalman Filter to predict the next location, a higher frame-rate will increase the accuracy of this prediction. The testbed on which this work is deployed is equipped with cameras which stream two types of video stream in parallel: H.264, and Motion Joint Photographic Experts Group (MJPEG). MJPEG is the most convenient solution to use in code as it has little-to-no overhead to utilize, but comes at the cost of low frame-rate (6 to 8 FPS) as a result of the camera itself. This is believed to be the main contributor to the re-assigned ID issue discussed in Chapter 4.6. Since the H.264 stream does maintain 30FPS with no issue it would is the ideal solution, but includes a decoding overhead which is not present when using the MJPEG stream. A solution has been implemented, but not yet deployed, which utilizes Compute Unified Device Architecture (CUDA) to accelerate the decoding process. This implementation is still not yet finalized nor has it been deployed on a dedicated compute device, so official metrics and reporting have not been reassessed using the improved frame-rate.

7.2.2 Near-Miss Detection

While this approach is naive in the sense that there is no consideration for the depth as the third dimension in the actual space, it allows us to calculate a pixel-relative velocity which has many practical approaches on its own. With some mathematics similar to the work in [32, p.32-33], depth can be approximated using average pixel-heights for detected objects at a known distance. Utilizing depth will greatly reduce the number of false-positives due to depth.

Figure 7.1 Screenshot of the new dashboard created using real-time data from the testbed.

7.2.3 Traffic Flow

The implementation of traffic flow data collection discussed in this thesis was implemented solely for pedestrians and vehicles. This approach has not yet been implemented to take into account bicycle lanes, as the ROI Labelling process did not include bicycle lanes. Re-performing the labelling process with bicycle lanes in consideration, it will be possible to implement bicycle traffic flow as well.

Additionally, the data from the traffic flow data collection process has not yet been utilized. There are a few possible use-cases for the traffic flow data, such as creating a predictive model using the historic traffic flow data. This can be merged with additional data such as weather to determine what routes are taken as a response to these conditions. This data can also used for real-time route optimization, avoiding intersections along the testbed where the traffic volume is above average. Lastly, this data can be integrated into the work-in-progress dashboard made for the MLK Smart Corridor testbed (shown in Figure 7.1), showing a real-time view of traffic volume at the intersections on the dashboard.

REFERENCES

[1] "Hamilton County, Tennessee Population 2020." [Online]. Available: https://worldpopulationreview.com/us-counties/tn/hamilton-county-population/

[2] "Tennessee County Crash Data." [Online]. Available: https://www.tn.gov/safety/stats/crashdata.html

[3] Department of Economic and Social Affairs, "World Urbanization Prospects," United Nations, 2018.

[4] M. d'Aquin, J. Davies, and E. Motta, "Smart Cities' Data: Challenges and Opportunities for Semantic Technologies," *IEEE Internet Computing*, vol. 19, no. 6, pp. 66–70, Nov 2015.

[5] G. Bradski and A. Kaehler, *Learning OpenCV, 1st Edition*, 1st ed. O'Reilly Media, Inc., 2008.

[6] P. N. Druzhkov, V. L. Erukhimov, N. Y. Zolotykh, E. A. Kozinov, V. D. Kustikova, I. B. Meerov, and A. N. Polovinkin, "New Object Detection Features in the OpenCV Library," *Pattern Recognition and Image Analysis*, vol. 21, no. 3, p. 384, sep 2011. [Online]. Available: https://doi.org/10.1134/S1054661811020271

[7] R. C. Luo and C. C. Lai, "Multisensor Fusion-Based Concurrent Environment Mapping and Moving Object Detection for Intelligent Service Robotics," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 8, pp. 4043–4051, Aug 2014.

[8] A. Raghunandan, Mohana, P. Raghav, and H. V. R. Aradhya, "Object Detection Algorithms for Video Surveillance Applications," in *2018 International Conference on Communication and Signal Processing (ICCSP)*, April 2018, pp. 0563–0568.

[9] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," 2015.

[10] A. Yilmaz, O. Javed, and M. Shah, "Object Tracking," *ACM Computing Surveys*, 2006.

[11] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," *CoRR*, vol. abs/1602.00763, 2016. [Online]. Available: http://arxiv.org/abs/1602.00763

[12] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua, "Multiple Object Tracking Using K-Shortest Paths Optimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 9, pp. 1806–1819, Sep. 2011.

[13] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *Proceedings - International Conference on Image Processing, ICIP*, 2018.

[14] A. Harris, J. Stovall, and M. Sartipi, "Mlk smart corridor: An urban testbed for smart city applications," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 3506–3511.

[15] "Busiest Streets in Chattanooga in Tennessee, United States of America." [Online]. Available: http://www.gomapper.com/travel/busiest-streets-in/chattanooga.html

[16] "CUIP." [Online]. Available: https://utccuip.com/

[17] B. Babenko, M. Yang, and S. Belongie, "Robust Object Tracking with Online Multiple Instance Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1619–1632, Aug. 2011.

[18] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2010, pp. 2544–2550.

[19] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *CoRR*, vol. abs/1506.0, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[20] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[21] ——, "YOLOv3: An Incremental Improvement," *arXiv*, 2018.

[22] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[23] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[24] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as Points," *CoRR*, vol. abs/1904.07850, 2019. [Online]. Available: http://arxiv.org/abs/1904.07850

[25] S. Aminmansour, F. Maire, and C. Wullems, "Video Analytics for the Detection of Near-Miss Incidents on Approach to Railway Level Crossings," in *2014 Joint Rail Conference, JRC 2014*, 2014.

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

[27] T. Surasak, I. Takahiro, C. H. Cheng, C. E. Wang, and P. Y. Sheng, "Histogram of oriented gradients for human detection in video," in *Proceedings of 2018 5th International Conference on Business and Industrial Research: Smart Technology for Next Generation of Information, Engineering, Business and Social Science, ICBIR 2018*, 2018.

[28] H. Kataoka, T. Suzuki, S. Oikawa, Y. Matsui, and Y. Satoh, "Drive Video Analysis for the Detection of Traffic Near-Miss Incidents," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2018.

[29] L. Wang, Y. Qiao, and X. Tang, "Action recognition with trajectory-pooled deep-convolutional descriptors," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2015.7299059

[30] Y. Matsui, M. Hitosugi, K. Takahashi, and T. Doi, "Situations of Car-to-Pedestrian Contact," *Traffic Injury Prevention*, 2013.

[31] K. M. M. Thein, "Apache kafka: Next generation distributed messaging system," *International Journal of Scientific Engineering and Technology Research*, vol. 3, no. 47, pp. 9478–9483, 2014.

[32] R. Thompson, "Data-fused urban mobility applications for smart cities," *Masters Theses and Doctoral Dissertations*, 2018. [Online]. Available: https://scholar.utc.edu/theses/571

[33] G. Zhang, R. P. Avery, and Y. Wang, "Video-based vehicle detection and classification system for real-time traffic data collection using uncalibrated video cameras," *Transportation Research Record*, 2007.

[34] C. H. Lo, W. C. Peng, C. W. Chen, T. Y. Lin, and C. S. Lin, "CarWeb: A traffic data collection platform," in *Proceedings - IEEE International Conference on Mobile Data Management*, 2008.

[35] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, "Adaptive fastest path computation on a road network: A traffic mining approach," in *33rd International Conference on Very Large Data Bases, VLDB 2007 - Conference Proceedings*, 2007.

[36] X. Li, J. Han, J. G. Lee, and H. Gonzalez, "Traffic density-based discovery of hot routes in road networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2007.

[37] S. P. Hoogendoorn, H. J. Van Zuylen, M. Schreuder, B. Gorte, and G. Vosselman, "Microscopic Traffic Data Collection by Remote Sensing," in *Transportation Research Record*, 2003.

[38] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Occlusion-Aware R-CNN: Detecting Pedestrians in a Crowd," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018.

[39] K. Wada, "labelme: Image Polygonal Annotation with Python," https://github.com/wkentaro/labelme, 2016.

[40] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "LabelMe: A database and web-based tool for image annotation," *International Journal of Computer Vision*, 2008.

[41] S. Gillies, A. Bierbaum, K. Lautaportti, and O. Tonnhofer, "Shapely," 2014.

VITA

Jose Stovall, born in Pensacola Florida, grew up in the Middle Tennessee area where he attended grade school. Upon graduating from LaVergne High School in 2011, he pursued an education in Digital Animation at Middle Tennessee State University, where he later found his passion for computer science. After taking a three-year hiatus from a college education due to a lack of funding, he returned to academia in 2015 at the University of Tennessee at Chattanooga. Upon finishing his undergraduate with a Bachelor's of Science in Computer Science with a focus in Software Systems, he continued his education at the university. Beginning his pursuit for even higher education in 2018, Jose has graduated from the University of Tennessee at Chattanooga with a Master's of Science in Computer Science with a focus of Data Science in May of 2020.