FINAL REPORT

Project D

August 2021

Evaluation of Advanced & Communication Technologies through Traffic Microsimulation

Dr. Pruthvi Manjunatha | University of Florida
Dr. Lily Elefteriadou | University of Florida
Dr. Michael P. Hunter | Georgia Institute of Technology
Xi Duan, MS | University of Florida
Dr. Clark Letter | HNTB
Somdut Roy, MS | Georgia Institute of Technology
Dr. Chelsea "Chip" White | Georgia Institute of Technology
Dr. Deborah Postma | Georgia Institute of Technology
Dr. Angshuman Guin | Georgia Institute of Technology

STRIDE | Southeastern Transportation Research, Innovation, Development and Education Center

UF | Transportation Institute UNIVERSITY *of* FLORIDA

TECHNICAL REPORT DOCUMENTATION PAGE

1. Report No. Project D	**2. Government Accession No.**	**3. Recipient's Catalog No.**
4. Title and Subtitle Evaluation of Advanced & Communication Technologies through Traffic Microsimulation		**5. Report Date** 8/2/2021
		6. Performing Organization Code
7. Author(s) Dr. Pruthvi Manjunatha, University of Florida; Dr. Lily Elefteriadou, University of Florida; Dr. Michael Hunter, Georgia Institute of Technology; Xi Duan, MS, University of Florida; Dr. Clark Letter, HTNB; Somdut Roy, MS, Georgia Institute of Technology; Dr. Chelsea "Chip" White, Georgia Institute of Technology; Dr. Deborah Postma, Georgia Institute of Technology; Dr. Angshuman Guin, Georgia Institute of Technology		**8. Performing Organization Report No.** STRIDE Project D
9. Performing Organization Name and Address **University of Florida**, College of Design, Construction & Planning, PO Box 115701, Gainesville, FL 32611 **Georgia Institute of Technology** School of Civil and Environmental Engineering 790 Atlantic Drive Atlanta, GA 30332-0355		**10. Work Unit No.**
		11. Contract or Grant No. Funding Agreement Number 69A3551747104
12. Sponsoring Agency Name and Address **University of Florida Transportation Institute** **Southeastern Transportation Research, Innovation, Development and Education Center (STRIDE)** 365 Weil Hall, P.O. Box 116580 Gainesville, FL 32611 **U.S Department of Transportation/Office of Research, Development & Tech** 1200 New Jersey Avenue, SE Washington, DC 20590 United States		**13. Type of Report and Period Covered** 4/23/2018 to 8/2/2021
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

Evaluation of VISSIM revealed that internal modeling of CAV has several limitations. For external modeling, two VISSIM interfaces are useful. The Component Object Model (COM)Application Programming Interface (API) is the superior approach for fetching data and modeling connectivity, whereas the External Driver Model (EDM) is a better tool for lateral and longitudinal control. Utilizing both the COM API and EDM overcomes the disadvantages of both, creating a more robust platform for CAV modeling. Based on this, a comprehensive simulation extension was developed to represent CAVs in VISSIM. CAVs were modeled and an isolated signalized intersection was simulated. The trajectory data from VISSIM were leveraged to estimate energy, fuel consumption, and greenhouse gas emissions using the Motor Vehicle Emission Simulator (MOVES) method. The results show that CAVs in the traffic stream result in net improvement in traffic operational measures (travel time and speed). CAV, the combination of the two technologies (i.e., autonomy and connectivity) yields better performance than each (CV and AV) on their own. However, emissions did not follow the same trend. While increasing AV penetration rates resulted in emissions reductions, increasing CV and CAV penetration rates resulted in higher emissions. A deeper analysis into the root cause for these trends showed that while the CV logic chosen for testing in the VISSIM simulation environment seeks to maximize the likelihood of vehicle arrival-on-green, the algorithm likely results in increased variations in second-by-second accelerations, leading to overall higher emissions. The results are based on a small and relatively simple network, and operations may be different for larger and more complex networks. In addition, the AV, CV, and CAV findings are limited to the connectivity and autonomy algorithms tested in this project. A more complex network with varying vehicle movement algorithms would allow for a more robust analysis.

17. Key Words Connected and Autonomous Vehicles (CAV), Microsimulation, VISSIM, Emissions, Modeling, MOVES		**18. Distribution Statement** No restrictions	
19. Security Classif. (of this report) N/A	**20. Security Classif. (of this page)** N/A	**21. No. of Pages** 132 Pages	**22. Price** N/A

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

Page Left Blank

DISCLAIMER

ACKNOWLEDGEMENT OF SPONSORSHIP AND STAKEHOLDERS

This work was sponsored by a contract from the Southeastern Transportation Research, Innovation, Development and Education Center (STRIDE), a Regional University Transportation Center sponsored by a grant from the U.S. Department of Transportation's University Transportation Centers Program.

Funding Agreement Number - 69A3551747104

LIST OF AUTHORS

Lead PI:

Pruthvi Manjunatha, PhD
University of Florida
pruthvim@ufl.edu
ORICD: 0000-0001-6596-268X

Co-PIs:

Lily Elefteriadou, PhD
University of Florida
elefter@ce.ufl.edu
ORCID: 0000-0003-4045-3365

Michael Hunter, PhD
Georgia Institute of Technology
michael.hunter@ce.gatech.edu
ORCID: 0000-0002-0307-9127

Additional Researchers:

Xi Duan, MS
University of Florida
xiduan@ufl.edu

Clark Letter, PhD
University of Florida (formerly)
cletter@HNTB.com

Somdut Roy, MS
Georgia Institute of Technology
somdut.roy@gatech.edu

Chelsea "Chip" White, PhD
Georgia Institute of Technology
chip.white@isye.gatech.edu

Deborah Postma, PhD
Georgia Institute of Technology
deborah@gatech.edu

Angshuman Guin, PhD
Georgia Institute of Technology
angshuman.guin@ce.gatech.edu

TABLE OF CONTENTS

DISCLAIMER iii
ACKNOWLEDGEMENT OF SPONSORSHIP AND STAKEHOLDERS iii
LIST OF AUTHORS iv
LIST OF FIGURES ix
LIST OF TABLES x
ABSTRACT xi
EXECUTIVE SUMMARY xii
1. INTRODUCTION 13
1.1 OBJECTIVES 14
1.2 SCOPE 14
2 EVALUATION OF VISSIM CAPABILITIES TO REPLICATE AUTONOMY AND CONNECTIVITY ... 15
2.1 Modeling CAVs 15
2.1.1 Modeling CAVs Internally 15
2.1.2 Modeling CAVs Externally 19
2.2 Functionality Evaluation of Two External Interfaces 20
2.2.1 COM API 20
2.2.2 External Driver Model (EDM) 23
2.2.3 Relationship between longitudinal control and lateral movement control 27
2.3 Summary 28
3 DEVELOPMENT OF AUTONOMOUS AND CONNECTED VEHICLE FUNCTIONALITY IN VISSIM 29
3.1 COM API data output function 30
3.1.1 Launch VISSIM and then load the network and layout files 31
3.1.2 Set the input parameters 31
3.1.3 Set the categories and frequency of output data 32
3.1.4 Find the connected vehicles within certain radius 32
3.2 EDM speed trajectory implementation 32
3.3 Summary 32

4 CAV MODELING AND SIMULATION 33
4.1 AV Model 34
4.1.1 Initial settings in VISSIM 34
4.1.2 AV Model implementation in COM API 36
4.1.3 Acceleration implementation through EDM 39
4.2 CV Model 39
4.2.1 Initial settings in VISSIM 40
4.2.2 CV Model implementation in COM API 42
4.2.3 Acceleration implementation through EDM 42
4.3 CAV Model 42
4.4 Simulations 45
4.4.1 Simulation network layout 47
4.4.2 Simulation scenarios 48
4.4.3 Simulation signal phasing and timing 49
4.5 Results 49
5 EMISSIONS MODELING 53
5.1 Introduction 53
5.2 Network and Data Information 53
5.2.1 AV and CV Logic implemented in VISSIM 54
5.3 Data Processing Methodology 55
5.3.1 Energy and Emission Estimation using MOVES 56
5.4 Comparative Analysis and Discussion 59
5.4.1 Travel-time Comparative Analysis 60
5.4.2 CO2 Emission (MOVES estimate) Comparative Analysis 62
5.5 Summary 68
6 CONCLUSION 70
7 RECOMMENDATIONS 71
REFERENCE LIST 73
APPENDICES 75
Appendix A: COM API Code 75
Appendix B: Speed and Delay Results by Scenario 79

Appendix C: COM API Code written in Python 3.0 82
Appendix D: EDM Source Code (drivermodel.dll) written in C++.. 95
Appendix E: Urban Freight Transport and Its Effects 103

LIST OF FIGURES

Figure 2-1 Wiedemann Car Following Logic (PTV AG, 2005) 16
Figure 2-2 Position over Time for Three Control Methods... 22
Figure 2-3 Speed over Time for Three Control Methods.. 22
Figure 3-1 Longitudinal Control Framework.. 30
Figure 4-1 Flowchart of AV Logic Implementation 35
Figure 4-2 Flowchart of CV Logic Implementation 41
Figure 4-3 Flowchart of CAV Logic Implementation 44
Figure 4-4 Network Layout 47
Figure 4-5 Network Layout in VISSIM 48
Figure 4-6 Average Network Speeds for Different Penetration Rates of CAVs 51
Figure 4-7 Average Network Delay for Different Penetration Rates of CAVs 52
Figure 5-1 Schematic Representation of EB-Through and NB-Through and their Respective Counters.. 54
Figure 5-2 Screenshot of Initial Lines of a Trajectory File from A VISSIM Run (AV Data: 40% penetration, v/c=0.7, random seed=1).. 55
Figure 5-3 Architecture for KPI Calculation and Route Allocation from Trajectory Data............ 56
Figure 5-4 Comparative Study: Travel-Time for EB-Through Route for v/c=0.7 Detected at (a) Counter-1, and (b) Counter-2 60
Figure 5-5 Comparative Study: Travel-Time for NB-Through Route for v/c=0.7 Detected at (a) Counter-1, and (b) Counter-2 61
Figure 5-6 Comparative Study: Travel-Time for EB-Through Route for v/c=0.9 Detected at (a) Counter-1, and (b) Counter-2 61
Figure 5-7 Comparative Study: Travel-Time for NB-Through Route for v/c=0.9 Detected at (a) Counter-1, and (b) Counter-2 62
Figure 5-8 Comparative Study: Emissions per Feet for EB-Through Route for v/c=0.7 Detected at (a) Counter-1, and (b) Counter-2 63
Figure 5-9 Comparative Study: Emissions per Feet for NB-Through Route for v/c=0.7 Detected at (a) Counter-1, and (b) Counter-2 64
Figure 5-10 Comparative Study: Emissions per Feet for EB-Through Route for v/c=0.9 Detected at (a) (left) Counter-1, and (b) (right) Counter-2 64
Figure 5-11 Comparative Study: Emissions per Feet for NB-Through Route for v/c=0.9 Counted at (a) (left) Counter-1, and (b) (right) Counter-2 65
Figure 5-12 Variations in Acceleration for EB-Through Vehicles with Increasing Penetration in CV for v/c=0.9, Simulation Random Seed=0.1 66
Figure 5-13 Variations in VSP-Bin Distribution for EB-Through Vehicles with Increasing Penetration in CV for v/c=0.9, Simulation Random Seed=0.1... 67
Figure 5-14 Variations in VSP-Bin Distribution for EB-Through Vehicles with Increasing Penetration in CAV for v/c=0.9, Random Seed=0.1... 68
Figure 5-15 Optimal Speed provided by VISSIM and Manually Calculated 70

LIST OF TABLES

Table 2-1 Wiedemann 99 Parameters (Woody, 2006) 17
Table 2-2 Lane Change Parameters (PTV- VISSIM, 2016) 18
Table 3-1 Retrieved Attributes and their Descriptions 31
Table 4-1 Simulation Scenarios 48
Table 4-2 Turning Movements (same for all approaches) 49
Table 4-3 Signal Timings 49
Table 5-1 MOVES VSP/STP Operating Mode Bins 58
Table 5-2 MOVES VSP Bin-Energy/Emission Lookup Table for 2017 Passenger Car 59

ABSTRACT

Proper evaluation of traffic operations integrating Connected and Autonomous Vehicles (CAVs) requires accurate representation of these emerging technologies within the context of microscopic simulation, allowing for detailed evaluation of their operational and environmental effects. To accomplish this, the objectives of this project were:

1. Evaluate the microscopic simulator VISSIM's ability to simulate CAVs
2. Develop a comprehensive simulation extension to represent CAVs in VISSIM
3. Integrate emissions modeling to calculate real-time energy and emission estimates
4. Assess traffic operational and environmental performance measures for various levels of CAVs.

Evaluation of VISSIM revealed that internal modeling of CAVs has several limitations. For external modeling, two VISSIM interfaces are useful. The Component Object Model (COM) Application Programming Interface (API) is the superior approach for fetching data and modeling connectivity, whereas the External Driver Model (EDM) is a better tool for lateral and longitudinal control. Utilizing both the COM API and EDM overcomes the disadvantages of both, creating a more robust platform for CAV modeling.

Based on this, a comprehensive simulation extension was developed to represent CAVs in VISSIM. CAVs were modeled and an isolated signalized intersection was simulated. The trajectory data from VISSIM were leveraged to estimate energy, fuel consumption, and greenhouse gas emissions using the Motor Vehicle Emission Simulator (MOVES) method.

The results show that CAVs in the traffic stream result in net improvement in traffic operational measures (travel time and speed). CAV, the combination of the two technologies (i.e. autonomy and connectivity) yields better performance than each (CV and AV) on their own. However, emissions did not follow the same trend. While increasing AV penetration rates resulted in emissions reductions, increasing CV and CAV penetration rates resulted in higher emissions.

A deeper analysis into the root cause for these trends showed that while the CV logic chosen for testing in the VISSIM simulation environment seeks to maximize the likelihood of vehicle arrival-on-green, the algorithm likely results in increased variations in second-by-second accelerations, leading to overall higher emissions.

The results are based on a small and relatively simple network, and operations may be different for larger and more complex networks. In addition, the AV, CV, and CAV findings are limited to the connectivity and autonomy algorithms tested in this project. A more complex network with varying vehicle movement algorithms would allow for a more robust analysis.

Keywords: Connected and Autonomous Vehicles (CAV), Microsimulation, VISSIM, Emissions Modeling, MOVES

EXECUTIVE SUMMARY

The goal of this project is to develop a robust microscopic simulation extension to allow the evaluation of traffic operational quality considering the presence of Connected and Autonomous Vehicles (CAVs).

The research team evaluated the capability of the microscopic simulator VISSIM (Version 10.0) to model CAVs. There are two external interfaces with powerful features available for CAV modeling in VISSIM: Component Object Model (COM) Application Programming Interface (API) and External Driver Model (EDM). CAV modeling was developed in VISSIM by leveraging the strengths of both interfaces: the research team used the COM API to access network elements and the EDM to maintain the longitudinal control of vehicles.

The trajectory data from VISSIM were used to estimate energy, fuel consumption, and greenhouse gas emissions. The calculations follow the Motor Vehicle Emission Simulator (MOVES) methods developed and mandated by the US Environmental Protection Agency (USEPA).

A four-legged isolated signalized intersection was modeled in VISSIM and a model by Talebpour and Mahmassani (2016) was used to replicate the Autonomous Vehicle (AV) logic. An Infrastructure- to-Vehicle (I2V) application (PTV, 2017) allowing the connected vehicles (CVs) to access signal timing information was used to replicate the information CVs receive. The CAV logic combines the AV logic and the CV logic, by replacing the VISSIM car-following model with the AV car-following model in the CV logic.

Several traffic scenarios were simulated using different demand and CAV penetration levels. The results show a net improvement in traffic operational measures when CAVs are in the traffic stream. However, emissions did not follow the same trend. While increasing AV penetration rates resulted in emissions reductions, increasing CV and CAV penetration rates resulted in higher emissions. A deeper analysis revealed the CV logic as the probable root cause.

The results are based on a small and relatively simple network, and thus may not be valid for larger and more complex networks. In addition, the AV, CV, and CAV findings are limited to the vehicle movement algorithms implemented. A more complex network with improved technology algorithms would allow for a more robust analysis.

The simulation extension developed in this project could be used to evaluate CAV strategies for a variety of networks and scenarios. Future work includes incorporating a CAV-based signal optimization algorithm developed by the University of Florida into the extension, enhancing the optimization to include emissions, and applying the tool developed to large-scale transportation challenges.

1. INTRODUCTION

Significant improvements in autonomous vehicle technologies as well as their connectivity and interaction with future generation traffic systems are expected to create a perfect storm in how vehicles would navigate through city roads and highways. Autonomous vehicles (AVs) can operate on their own and do not require a human driver. They can operate using a variety of sensors such as GPS, Lidar, Radar, and smart cameras, as well as terrain information.

Connected vehicles (CVs) can communicate with surrounding vehicles and infrastructure using wireless communications. The US DOT connected vehicle research program aims to enable wireless communications among vehicles, infrastructure, and personal communications devices (http://www.its.dot.gov/pilots/). The program is currently focusing on pilot deployments to implement existing research concepts and encourage further innovation. Initial connected vehicle technology deployments seek to warn drivers of impending dangers while the vehicle is controlled by a human driver; although, operations, sustainability, assess management, etc., CV applications are also being explored. AVs may also have connectivity capabilities; such vehicles are called connected/autonomous or automated vehicles (CAVs).

It is highly likely that soon both connected vehicles and CAVs will be operating side-by-side in large numbers in our nation's highways, along with conventional (CNV) vehicles. This creates many opportunities in improving surface transportation efficiency and safety. For example, the USDOT Multimodal Intelligent Traffic Signal Systems (MMITSS) initiative aims to provide a comprehensive traffic information framework to service all transportation modes, including general vehicles, transit, emergency vehicles, freight fleets, and pedestrians and bicyclists in a connected vehicle environment.

According to the National Transportation Operations Coalition, delays at traffic signals account for 5% to 10% of all traffic delay on major roadways and contributed an estimated 25% to the increase in total highway traffic delays during the past 20 years. Improvements in traffic signal timing have the potential to significantly benefit the transportation system. One source of delay at signals is inefficient green time utilization in response to fluctuating demand. Another source is driver reaction-related delays, including start-up delay. The use of autonomous and connected vehicle technology has the potential to reduce the impact of these two factors, through the use of their communication capability as well as the potential to fully control autonomous vehicle trajectories. Existing simulation tools are not able to accurately replicate the functionality of autonomous and connected vehicles. Therefore, the impact of various strategies and market penetrations on mobility and the environment cannot be accurately assessed.

1.1 OBJECTIVES

The goal of this project is to develop a robust microscopic simulation extension to allow the development and refinement of advanced transportation management strategies and evaluation procedures considering the presence of connected and autonomous vehicles. The research considers both mobility and environmental impacts. The goals of the project are accomplished using the VISSIM (10.0) microscopic simulator to replicate and evaluate these strategies. The specific objectives of the study are to:

1. Evaluate VISSIM's ability to simulate autonomous and connected vehicle technology.
2. Develop a comprehensive simulation extension to represent autonomous vehicles in the VISSIM simulation environment. This extension may represent autonomous vehicle behavior and produces exact vehicle trajectories. The module is to be designed to work as a plug-and-play solution across different computer and simulation networks.
3. Explore the Impact of AVs on delivery services.
4. Assess traffic operational quality (travel time, average speeds, queues, etc.) and environmental (emissions and energy –related) performance measures for a variety of demands and CAV market penetration levels at a signalized intersection.

1.2 SCOPE

In task 1 (section 2 in this document), evaluations are performed to examine the ability of VISSIM to simulate CAVs. The research team studied its capability in longitudinal and lateral movement control. In task 2 (section 3), we explore using the combination of COM API and EDM interfaces to maintain the longitudinal control of vehicles in a VISSIM simulation. In task 3 (section 4), we implement the framework proposed in task 2 with specific AV, CV, CAV-related algorithms. The team simulated several scenarios with different volume/capacity ratios (v/c ratio) under various AV, CV, and CAV market penetration levels and obtained the resulting operational performance at a signalized intersection. In task 4 (section 5) results from the emissions modeling using the MOVES model are presented. The last section provides conclusions and recommendations.

2 EVALUATION OF VISSIM CAPABILITIES TO REPLICATE AUTONOMY AND CONNECTIVITY

Traffic simulation of CAV creates a virtual environment to explore how CAVs will operate and impact traffic operations. VISSIM enables various ways for users to virtually simulate vehicles' automation from level 1 to level 5. Depending on the level of automation, users could conduct CAV simulation using VISSIM internal parameter settings or externally through an additional module and custom algorithm and code development.

In this section we evaluate the ability to simulate autonomy and connectivity using VISSIM. VISSIM provides options to model AV by changing driver parameters internally (e.g., headway time, standstill distance, following variation, see (Table 2-1) and externally with the COM API and EDM. The internal parameters change was evaluated first, followed by an evaluation of external interfaces. We specifically examined VISSIM's ability to modify vehicle trajectories by controlling the vehicle acceleration and steering of an autonomous vehicle.

2.1 Modeling CAVs

2.1.1 Modeling CAVs Internally

Internal parameters of the behavior models in VISSIM (10.0) can be adjusted to model AV-related features without any external or API programming. This can be achieved by changing VISSIM's default settings for parameters such as those related to car following, lane changing, and speed. This approach provides a full evaluation to assess changes in the selected parameters, but it is limited to modeling AVs with preset parameters and it cannot be used to model Vehicle-to-Vehicle (V2V) or Vehicle to Infrastructure (V2I) scenarios.

Car following

The car following model in VISSIM is based on research by (Fellendorf & Vortisch, 2001; Wiedemann & Reiter, 1991). The basic premise of the Wiedemann model states that a vehicle is in one of four states of car following: free, approaching, following, or braking. The following state changes when a threshold based on speed difference and distance difference between the lead and following vehicles are crossed. Figure 2-1 shows a graphical description of the Wiedemann car following model.

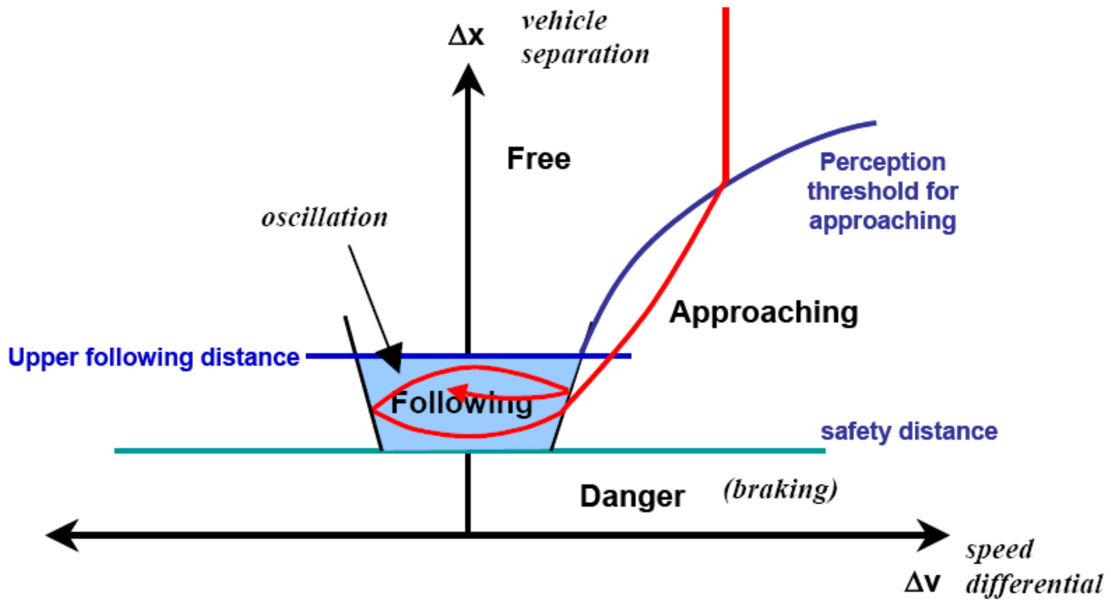


Figure 2-1 Wiedemann Car Following Logic (PTV AG, 2005)

The Wiedemann 99 car following model was developed to provide greater control of the car-following characteristics for freeway modeling in VISSIM (Wiedemann & Reiter, 1991). The Wiedemann 99 model consists of ten calibration parameters. Table 2-1 provides a description and the default values for each of the ‘CC’ parameters associated with the Wiedemann 99 model.

Table 2-1 Wiedemann 99 Parameters (Woody, 2006)

Parameters	**Description**	**Default Value**
Standstill distance (Wiedemann 99)	Desired distance between lead and following vehicle at v = 0 mph	4.92 ft
Headway Time (Wiedemann 99)	Desired time in seconds between lead and following vehicle	0.90 sec
Following Variation (Wiedemann 99)	Additional distance over safety distance that a vehicle requires	13.12 ft
Threshold for Entering 'Following' State:(Wiedemann 99)	Time in seconds before a vehicle starts to decelerate to reach safety distance (negative)	-8.00 sec
Negative 'Following' Threshold: (Wiedemann 99)	Defines negative speed difference during the following process. Low values result in a more sensitive driver reaction to the acceleration or deceleration of the preceding vehicle.	0.35 ft/s
Positive 'Following Threshold': (Wiedemann 99)	Defines positive speed difference during the following process. Low values result in a more sensitive driver reaction to the acceleration or deceleration of the preceding vehicle.	0.35 ft/s
Speed Dependency of Oscillation: (Wiedemann 99)	Influence of distance on speed oscillation while in the following process. If the value is 0, the speed oscillation is independent of the distance. Larger values lead to a greater speed oscillation with increasing distance.	11.44
Oscillation Acceleration: (Wiedemann 99)	Acceleration during the oscillation process	0.82 ft/s2
Look ahead distance. Observed vehicles	The number of observed vehicles or number of certain network objects affects how well vehicles in the link can predict other vehicles' movements and react accordingly. Higher value means vehicles can better react to multiple network objects in the network	4
Average standstill distance (feet) (Wiedemann 74)	Defines the average desired distance between two cars. Higher value means larger standstill distance and lower capacity	6.56 ft
Additive part of safety distance (Wiedemann 74)	Value used for the computation of the desired safety distance. Higher value means larger standstill distance and lower capacity	2.00 ft
Multiplic. Part of safety distance (Wiedemann 74)	Value used for the computation of the desired safety distance. Greater value equals greater distribution (standard deviation) of safety distance. Higher value means larger standstill distance and lower capacity	3.00 ft

Lane changing

There are several parameters to modify for lane change behavior, as shown in Table 2-2.

Table 2-2 Lane Change Parameters (PTV- VISSIM, 2016)

Parameters	Description	Default Value
Safety distance reduction factor	This factor is considered for each lane change. During the lane change, VISSIM reduces the safety distance to the value that results from the following multiplication: Original safety distance * safety distance reduction factor. The default value of 0.6 reduces the safety distance by 40%. Once a lane change is completed, the original safety distance is considered again.	0.6
Maximum deceleration - Own (ft/s2)	Upper bound of deceleration for own vehicle. Higher absolute value means more aggressive lane changing behaviors	-13.12 ft/s2
Advanced merging	If this option is selected, more vehicles can change lanes earlier, therefore capacity increases	Yes
Cooperative lane change	If this option is selected, trailing vehicles will make necessary lane change to facilitate the lane change of a leading vehicle	No

Changes in these internal parameters can be used to reflect simple first-order behaviors of AVs. For example, we can set lower standstill distances and lower lateral distances for AVs. However, more complex features such as vehicles' longitudinal and lateral movement control, vehicle-to-vehicle (V2V) communication, and vehicle-to-infrastructure (V2I) communication cannot be modeled through changes in internal parameters.

2.1.2 Modeling CAVs Externally

The following three external VISSIM interfaces can be used to enable modeling CAV behaviors:

i. COM Application Programming Interface (API);
ii. External Driver Model (EDM) using DriverModel.dll "dynamic link library" function; and
iii. DrivingSimulator.dll that connects with VISSIM simulation in real-time.

These three interfaces are defined below and described in more detail later in the report.

COM API

The COM script has access to nearly all data inside VISSIM, and can be made visible in a list window, enabling changes in driving behaviors and vehicle movements. The COM API cannot explicitly control the lateral movement of the CAVs because desired lane changes can only be made by VISSIM. The COM API does not depend on a certain programming language. COM objects can be developed in a wide range of programming and scripting languages. Certain simulation parameters such as acceleration and headway are only readable, whereas parameters such as desired speed and desired lane are both readable and writable.

External Driver Model (EDM)

The EDM is an external driver model compiled in C++ to substitute the VISSIM default driver model. VISSIM passes the current state of the vehicle and its surroundings to a dynamic link library file (drivermodel.dll), which then computes the 'reaction' of the vehicles based on this information. VISSIM allows some or all vehicles to be modeled with the user-defined drivermodel.dll which can specify all driving behaviors based on CAV logic.

DrivingSimulator.dll Interface

This interface allows testing of the interaction between externally-controlled vehicles or pedestrians in VISSIM (also known as "Human-in-the-Loop" simulation). All decisions and calculations of externally-controlled vehicles have to be specified, and are based on the reaction to 'other' vehicles in the simulation. Each vehicle moves through the VISSIM network based on simulator instructions, i.e. steering movements, acceleration. Other vehicles 'react' to what the external vehicle is doing.

2.2 Functionality Evaluation of Two External Interfaces

This evaluation is conducted to assess the capability of the two VISSIM external interfaces (COM API and EDM) to control CAV longitudinal and lateral movements. As discussed in the previous section, the DrivingSimulator.dll Interface requires a separate entity to control the vehicle (for example, a driving simulator) and therefore this approach is not examined further. Therefore, only the COM API and EDM are evaluated in the following sections.

2.2.1 COM API

The data contained in VISSIM can be accessed via the COM API. VISSIM version 4.0 and later allows for the data to be accessed via the COM API. The VISSIM COM API is automatically included when the software VISSIM is installed (but not in the Demo version). Starting with VISSIM Version 4.30, COM scripts can be called directly from the VISSIM main menu (PTV, 2017).

The COM API does not depend on a certain programming language. COM objects can be used in a wide range of programming and scripting languages, including VBA, VBS, Python, C, C++, C#, Delphi, and MATLAB. In this study, Python 3.0 is chosen as the programming language for COM API. There are two ways for users to access COM API:

1. Launch VISSIM – go to scripts menu – choose event-based scripts.
2. Launch Python – build connection between Python and VISSIM – Manipulation.

The second approach is more powerful as it enables the user to have full control of VISSIM through COM API. The first approach is useful if the user only requires partial functionality and faster operation. For this study we focus on the second approach.

Longitudinal movement control through COM API

There are three parameters of interest which can be altered in COM API to control vehicles' longitudinal movement: desired speed (the speed the vehicle/driver desires to travel at unless the acceleration is bound by vehicle dynamics), operating speed (instantaneous speed), and position (vehicle's current location, measured from the beginning of the link it is on).

The instantaneous acceleration cannot be manipulated directly through the COM API. Therefore, three methods of controlling vehicles' longitudinal movement by changing these parameters in COM API are tested and discussed below. To conduct the test, we specified a realistic vehicle trajectory (second-by-second acceleration, speed, and position for 60 seconds), which assumes constant acceleration within each time step. This is used to evaluate how VISSIM executes externally specified trajectories using different approaches. This trajectory is called “Specified Trajectory” in the evaluations described below.

- Change Desired Speed

Desired speed can be changed by using the command: *“SetAttValue('DesSpeed',Desired_speed_value)”.* It changes the vehicle’s desired speed, hence the vehicle will adjust its speed gradually to reach that desired speed. It leads to the smoothest realistic speed change as the vehicle will move in compliance with the VISSIM-defined acceleration function. However, it fails to reach the specified trajectory speed instantly, which results in a delay in the execution of the specified trajectory.

- Change Speed

Instant speed change can be implemented with the command *“SetAttValue('speed',speed_value)”*, which changes the vehicle speed instantly without considering the acceleration process.

- Change Position

Changing the position directly can be executed with the command *“MoveToLinkPosition”.* It ignores the speed and acceleration value and moves the vehicle directly to the assigned position. This enables perfect position match of a vehicle at each time step. However, it presumes that the acceleration and deceleration have already been checked and are realistic (i.e. it ignores the limitations of vehicle dynamics).

The results of using these three methods to force vehicles to follow the specified trajectory are shown in Figure 2-2 and Figure 2-3. Figure 2-2 has the y - axis as position, while Figure 2-3 has the y-axis as speed.

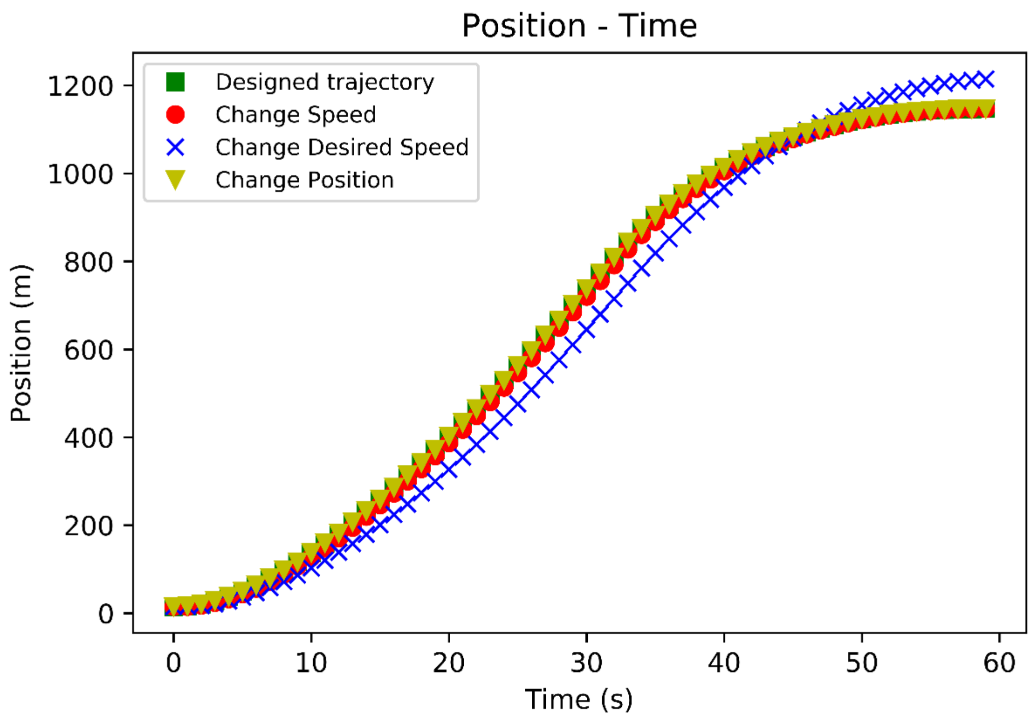


Figure 2-2 Position over Time for Three Control Methods

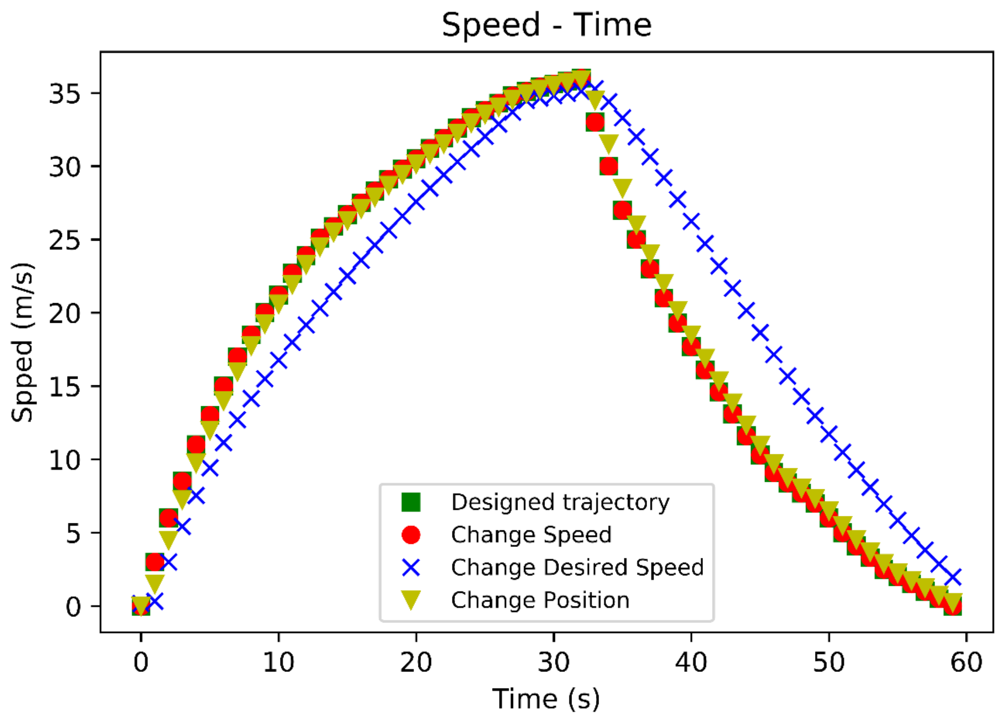


Figure 2-3 Speed over Time for Three Control Methods

As shown in Figure 2-2 , the “change position” curve overlaps with the “Specified Trajectory” curve, because the “change position” method forces the vehicle to follow the assigned trajectory perfectly. However, it ignores the acceleration and speed change process. In Figure 2-3, the “change speed” curve overlaps with the “Specified Trajectory” curve which indicates that vehicle control via the “change speed” method will perfectly follow the assigned trajectory. Similarly, to the “change position” method, this method does not consider the acceleration/deceleration implications.

As shown in Figure 2-2 and Figure 2-3, using the “change desired speed” method results in a lag between vehicles’ actual trajectory and the specified trajectory, i.e. the curve of “change desired speed” is located to the right of the specified trajectory curve). Using the “change speed” method, enables the vehicle to strictly follow the specified speed at every time step. It does not follow the specified position as it jumps to the specified speed at the beginning of each time step and omits the acceleration process. The “change position” method ensures the vehicle reaches the specified trajectory position at every time step, but it presumes that the acceleration and deceleration have already been checked and are realistic.

While the “change desired speed” method results in lag time issues, “change position” and “change speed” methods do not internally consider acceleration. Therefore, unless the trajectory has been developed through different means considering realistic acceleration/deceleration values, these methods may not be suitable for CAV modeling.

Lateral movement control through COM API

The COM API does not enable the direct control of vehicles’ lateral movement. The only option COM API provides is to set a desired lane and the vehicle will shift to the specified desired lane when it is able. The user could change the desired lane of a vehicle via a variable called “DesLane” during the simulation, but it may not necessarily lead to an instant lane change if VISSIM determines that the vehicle is not able to change lanes.

Data fetch through COM API

All data available on VISSIM is accessible through COM API, and the user has the option to fetch the data at the network level, i.e. the user could read all the data from the VISSIM network, and decide which data to use.

2.2.2 External Driver Model (EDM)

The EDM provides the option to replace the internal driving behavior by a fully-developed user-defined behavior for some or all vehicles in a simulation run.

During a simulation run, VISSIM calls the DLL code for each affected vehicle in each simulation time step to determine the behavior of the vehicle. VISSIM passes the current state of the vehicle and its surroundings to the DLL, and the DLL computes the acceleration / deceleration of the vehicle and the lateral behavior (mainly for lane changes) and passes these values back to VISSIM to be used in the current time step.

The EDM contains 3 functions which are called from VISSIM: `DriverModelSetValue`, `DriverModelGetValue` and `DriverModelExecuteCommand`.

`DriverModelSetValue` is used to pass a parameter from VISSIM to EDM, while `DriverModelGetValue` makes VISSIM retrieve a parameter from EDM. `DriverModelExecuteCommand` is used to execute commands for initializing the simulation, or creating, moving, and removing vehicles from the simulation (see below).

The currently available command constants are `DRIVER_COMMAND_INIT`, `DRIVER_COMMAND_CREATE_DRIVER`, `DRIVER_COMMAND_MOVE_DRIVER`, and `DRIVER_COMMAND_KILL_DRIVER`.

`DRIVER_COMMAND_INIT` is executed once when the simulation starts, and it is used to initialize the EDM.

Several basic parameters are passed to EDM via Set (e.g. time step, vehicle type) and several parameters are retrieved from EDM through Get (e.g. `DRIVER_DATA_WANTS_SUGGESTION` which determine if EDM needs a suggestion from VISSIM, `DRIVER_DATA_SIMPLE_LANECHANGE` that indicates if a simple lane change is undertaken).

`DRIVER_COMMAND_CREATE_DRIVER` is executed whenever a new vehicle enters the network in VISSIM. Several Set are called to provide the respective values to the EDM.

The `DRIVER_COMMAND_MOVE_DRIVER` is executed for every time step as long as the vehicle is still in the network. Multiple Set and Get commands are implemented for every execution of `DRIVER_COMMAND_MOVE_DRIVER`.

Lastly, when a vehicle in VISSIM leaves the network, the `DRIVER_COMMAND_KILL_DRIVER` is implemented, and several parameters pass to EDM via Set.

Longitudinal movement control through EDM

Since the vehicle acceleration can be set in EDM, the best way to implement specified vehicles' speed trajectory is by providing suitable values to "`get(DRIVER_DATA_DESIRED_ACCELERATION)`", which controls the vehicle's acceleration at each time step. The basic steps are as follows:

1. Construct a vehicle type in VISSIM, check the checkbox "Use external driver model" on the tab page "External Driver Model" and select a driver model DLL file and optionally a parameter file to be used.
2. Create a subdirectory DriverModelData\ in the directory of VISSIM.exe to avoid a warning message when a simulation run is started.
3. Set the Simulation Parameters "number of cores" (i.e. processing units) to be 1 in the VISSIM network, as the EDM needs to confirm that it supports multithreading. If it does not, the simulation run is canceled with an error message. (It is strongly suggested to set the number of cores to be one for simulation parameters; more detailed information is provided in EDM instructions (PTV VISSIM, 2016) (page 15).
4. Set the "`Get (DRIVER_DATA_USE_INTERNAL_MODEL)`" to be 0 in the DrivelModel.cpp file.
5. The vehicle's longitude movement then can be controlled by providing the relevant value to "`Get (DRIVER_DATA_DESIRED_ACCELERATION)`". It changes the vehicle's acceleration for every time step (since acceleration is continuously controlled at every time step, the desired speed is overridden). The vehicles' actual acceleration will be the communicated acceleration value but bounded by the VISSIM maximum acceleration functions.

The experiment was implemented and the result shows EDM matches perfectly with the designed longitudinal movement. The longitudinal movement for a time step can be calculated as $(v_{i-1} * t_s + \frac{1}{2} * a_{i-1} * t_s^2)$ while t_s is the time step length.

Lateral movement control through EDM

The EDM enables immediate lane change control. The EDM provides two methods to control the vehicles' lane change behaviors: simple lane change and full control lane change. Under the simple lane change mode, the user only needs to initiate the lane change ("when" to initiate it), then use VISSIM suggested lane change parameter values. However, under the full control lane change mode, the user needs to set/calculate the lane change-related parameters and provide these to the vehicles ("when" and "how" to conduct the lane change).

There are some import parameters for lane change control: `DRIVER_DATA_SIMPLE_LANECHANGE` is used to specify whether a simple lane change mode is used (set to 1 when using simple lane change mode, otherwise 0). `DRIVER_DATA_WANTS_SUGGESTION` indicates whether EDM wants suggestions on control parameter values from VISSIM (set to 1 if want suggestions from VISSIM, otherwise 0). `DRIVER_DATA_ACTIVE_LANE_CHANGE` is used to specify the direction of an active lane change (+1 = to the left, 0 = none, -1 = to the right). `DRIVER_DATA_DESIRED_LANE_ANGLE` is used to set the desired angle relative to the middle of the lane (positive = turning left).

1. *Simple Lane Change*: Under this mode, the user only needs to initiate a lane change for a vehicle. VISSIM assumes control of the lateral behavior of this vehicle while the lane change proceeds and informs the EDM when it is completed.
   - Set "`Get(DRIVER_DATA_SIMPLE_LANECHANGE)`" and "`Get(DRIVER_DATA_WANTS_SUGGESTION)`" to be 1.
   - Set "`Get(DRIVER_DATA_ACTIVE_LANE_CHANGE)`" to be either 1 (to left) or -1 (to right) to initiate a lane change.
   - Pass value from "`Set(DRIVER_DATA_DESIRED_LANE_ANGLE)`" `and` "`Set(DRIVER_DATA_REL_TARGET_LANE)`" `to` "`Get(DRIVER_DATA_DESIRED_LANE_ANGLE)`" `and` "`Get(DRIVER_DATA_REL_TARGET_LANE)`" respectively.

In this case, the vehicles will take a constant time to accomplish a lane change. Therefore, for a vehicle with higher speed, its second-by-second lane change angle will be smaller than that of vehicles with lower speed. A current lane change cannot be interrupted in the simple lane change mode. By default, the vehicles take 3 seconds to finish a lane change (this is the time it takes the middle of the vehicles' front end to reach the middle of the new lane), but extra time is needed for vehicles' middle rear bumper to reach the middle of the new lane

2. *Full Control Lane Change*: Users have full control of vehicles' lane change behaviors under this mode. Users are responsible for not only initiating a lane change, but also specify when the lane change is completed. Here, the lane change is executed by changing the "desired lane angle" parameter which allows the users to determine the time taken for lane change.
   - Set "`Get(DRIVER_DATA_SIMPLE_LANECHANGE)`" to be 0.
   - Set "`Get(DRIVER_DATA_ACTIVE_LANE_CHANGE)`", "`Get(DRIVER_DATA_REL_TARGET_LANE)`" and

"`Get(DRIVER_DATA_DESIRED_LANE_ANGLE)`" to the desired values to launch a lane change.

- Set "`Get(DRIVER_DATA_ACTIVE_LANE_CHANGE)`", "`Get(DRIVER_DATA_DESIRED_LANE_ANGLE)`" to be 0 to stop lateral movement.

When a lane change is launched, these three values should be set to non-zero values with the same sign. After the middle of the front end of the vehicle reaches the edge of the target lane, set "`Get(DRIVER_DATA_REL_TARGET_LANE)`" to be 0. When the whole width of the vehicle is in the new lane, set "`Get(DRIVER_DATA_ACTIVE_LANE_CHANGE)`" to be 0. When a lane change is accomplished and the vehicle is straight, set the "`Get(DRIVER_DATA_DESIRED_LANE_ANGLE)`" to be 0.

In testing these, these common errors were identified:

- When "`Get(DRIVER_DATA_ACTIVE_LANE_CHANGE)`" is non-zero, then a non-zero value with the same sign needs to be passed to "`Get(DRIVER_DATA_DESIRED_LANE_ANGLE)`".
- The value passed to "`Get(DRIVER_DATA_DESIRED_LANE_ANGLE)`" should be large enough especially when the speed is low.

Data fetch through EDM

The purpose of data fetch in EDM is to provide the vehicle's current status (acceleration, speed, position, etc.) and its surroundings (distance to the signal ahead, headway between its leading vehicle, etc.) from VISSIM to EDM. The user is only provided the data related to each individual vehicle, hence data fetch through EDM is at an individual vehicle level. Therefore, other data may not be available for users, e.g. the vehicles along another approach at an intersection.

2.2.3 Relationship between longitudinal control and lateral movement control

The longitudinal movement is controlled by "`Get(DRIVER_DATA_DESIRED_ACCELERATION)`" exclusively. Changing the vehicle's lateral movement has no influence on the vehicle's longitudinal movement. Besides, the speed value retrieved from VISSIM considers the longitudinal movement only. Therefore, longitudinal movement for a time step can be calculated as $(v_{i-1} * t + \frac{1}{2} * a_{i-1} * t^2)$ where t is the time step length.

The lateral movement is controlled by "`Get(DRIVER_DATA_DESIRED_LANE_ANGLE)`". It has no impact on longitudinal movement. However, the higher longitudinal speed will enable a wider range setting for "`Get(DRIVER_DATA_DESIRED_LANE_ANGLE)`".

In the simple lane change mode, VISSIM assumes constant time (3s) for vehicles to finish a lane change. For vehicles with lower speed, the lane change angle will be set higher at every time step to enable it to accomplish the lane change within the same time. Under fully lateral control mode, the vehicles will adjust their lane change angle to follow users' command.

Hence, it can be concluded that the lateral movements do not have any impact on the longitudinal movement in VISSIM. However, the longitudinal speeds have an impact on lane angle and hence the lateral movement.

2.3 Summary

VISSIM provides multiple ways for users to simulate CAV operations. AVs can be modeled internally In VISSIM by pre-setting certain behavioral parameters to match AV behavior. However, connectivity cannot be modeled internally.

The external interfaces COM API and EDM offer more powerful features. There are two main benefits to using COM: first, it provides better scalability and flexibility, e.g., the developer could write a function in COM enabling the CV to receive information from the infrastructure or other CVs that meets certain conditions (e.g., all vehicles within a certain radius). In contrast, the EDM can access the information of several leading and following vehicles only. Also, the COM API is easier to use, while the EDM logic and programming environment (C ++) are harder to master.

The COM API cannot control the acceleration (longitudinal movement) and lane changing (lateral movement) directly. It provides limited opportunities to control a vehicle through setting certain behavioral parameters in VISSIM. This method does not produce realistic acceleration. Unless the trajectory has been developed through different means ensuring realistic acceleration, this method should not be used.

The EDM is capable of both longitudinal and lateral movement control. Acceleration (longitudinal movement) and lane changing (lateral movement) can be modeled directly.

In the next section, we explore using the combination of COM API and EDM interfaces to maintain the longitudinal control of vehicles in VISSIM. We leverage the COM API's ability to access network elements for modeling connectivity and we use EDM to implement specified trajectories.

3 DEVELOPMENT OF AUTONOMOUS AND CONNECTED VEHICLE FUNCTIONALITY IN VISSIM

As concluded in the previous section, while the COM API can fetch all data available on VISSIM lists, it cannot directly control vehicles' longitudinal and lateral movement. The EDM is capable of both longitudinal and lateral control but it has limitations related to data fetching. Therefore, it is preferable to use selected features of each interface.

In this section we explore using the combination of COM API and EDM interfaces to maintain the longitudinal control of vehicles in VISSIM simulation. We leverage the COM API's ability to access network elements for modeling connectivity and we use EDM to implement the specified accelerations.

A data fetching function is developed in COM API to acquire the required data from VISSIM (e.g. the signal timing and phasing, vehicles speed, acceleration, and positions). The data fetching function then feeds that data into a developed COM API CV/AV logic, which generates the acceleration for AV/CV (specified acceleration) for the next time step. This can be based on autonomous driving logic for AVs or recommended trajectories for CVs. Then the COM API outputs the specified accelerations into a txt file, which is read and implemented in VISSIM by the EDM in the next time step (Figure 3-1).

There are two main benefits to using COM: first, it provides better scalability and flexibility, e.g., the developer could write a function in COM enabling the CV to receive information from the infrastructure or other CVs that meets certain conditions (e.g., all vehicles within a certain radius). In contrast, the EDM can access the information of several leading and following vehicles only. Also, the COM API is easier to use, while the EDM logic and programming environment (C ++) are harder to master.

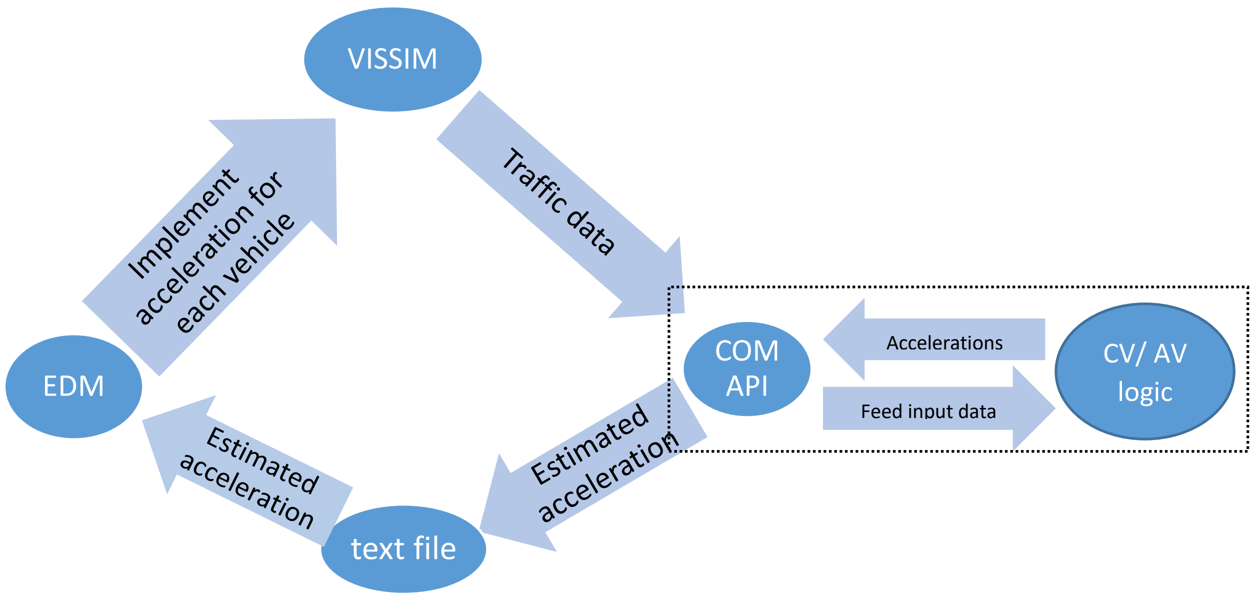


Figure 3-1 Longitudinal Control Framework

3.1 COM API data output function

As indicated earlier, the COM API has access to all the data available in VISSIM. Example data types are shown in VISSIM drop-down menu: “Lists – Results– Vehicles in Network”. The data includes the status and characteristics of vehicles that are running in the network. Table 3-1 shows some of the attributes critical to this effort that may be retrieved from VISSIM through the COM API from this list. These attributes are useful for calculating CAV trajectories. Other useful data include signalization and links’ attributes.

Table 3-1 Retrieved Attributes and their Descriptions (PTV VISSIM, 2016)

Attributes	Descriptions
No	Unique vehicle number
VehType	Vehicle type
Length	Vehicle length
Lane	Number of lane on which vehicles is used
Pos	Distance on the link from the beginning of the link or connectors
Speed	Speed at the end of time step
DesSpeed	Desired speed
Acceleration	Acceleration during the time step
Headway	Distance to the preceding vehicle before the time step
SpeedDiff	Relative to the preceding vehicle in the time step
SimSec	Simulation time (sec)
Other user defined attributes	Users could define their own attribute, which get from other build in attributes.

The process of data fetching is summarized below.

3.1.1 Launch VISSIM and then load the network and layout files

First, Python needs to call COM API and launch VISSIM with the command:

```
"import win32com.client as com", "Vissim=
com.Dispatch ("VISSIM.VISSIM.1000")"
```

Here “1000” is used for VISSIM version 10.

The network and layout file will be loaded using the command:

```
VISSIM.LoadNet(Filename, flag_read_additionally)

VISSIM.LoadLayout(Filename)
```

The directory of network and layout files need to match those on the computer.

3.1.2 Set the input parameters

The user has the option to set input parameters (e.g. vehicle flow rate, vehicle composition) through COM API or in VISSIM directly. The example code in Appendix A shows how to set vehicle flow rate (veh /hr), vehicle composition (CNVs as type “100” and connected vehicles (CAV) as type “630”), desired speed, and relative flow (relative share of each vehicle type in the vehicle composition).

3.1.3 Set the categories and frequency of output data

The vehicle- related data from VISSIM can be accessed with:

```
“Vissim.Net.Vehicles.GetMultipleAttributes
(“Attribute of selection”)”
```

Users can go to the VISSIM lists and find the corresponding attributes names. In our example code (Appendix A), vehicle ID, vehicle type, vehicle length, vehicle’s front bumper’s coordinate, vehicle’s rear bump’s coordinate, acceleration, and the distance from preceding vehicle are selected as the output. Those data are output in the .csv format at a selected frequency. In addition, there is another output file in .txt format, the specified accelerations file, in which only the vehicle ID and their corresponding acceleration are written. That file is then read by EDM.

3.1.4 Find the connected vehicles within certain radius

We have also developed a function (Appendix A) to acquire the information of surrounding CAVs from a CAV within a certain radius. The radius may be calculated based on the coordinates of a vehicle’s front bumper or rear bumper.

3.2 EDM speed trajectory implementation

As discussed in section 2, EDM is suitable for longitudinal and lateral control. In this study we have developed an innovative longitudinal control framework. In each time step, instead of performing the car-following model calculations within EDM we use COM to collect all needed information and calculate the next acceleration value for the given vehicle(s). COM writes the accelerations to a txt file. In the EDM, a function has been created to read the text file and find the specified acceleration corresponding to a subject vehicle CAV ID. The EDM reads and then implements the specified acceleration (Figure 3-1).

3.3 Summary

In this section, a longitudinal movement control framework was developed, which uses both the COM API and the EDM. The COM API retrieves the necessary data and feeds these to the AV/CV logic. It then outputs the specified accelerations obtained from the AV/CV logic to a txt file. The EDM then reads the txt file and assigns the acceleration to the corresponding vehicle ID.

In the next section, we incorporate existing AV and CV algorithms for an isolated intersection into VISSIM. We evaluate the effectiveness of the simulation extensions to control vehicle trajectories and to represent CAVs. We evaluate the control algorithm by testing various scenarios varying the demand and market penetration of vehicles.

4 CAV MODELING AND SIMULATION

A vehicle longitudinal control framework was developed and described in the previous section, which makes use of the selected features of COM API and EDM. In this framework (Figure 3-1), the following actions occur during each step:

1. A COM-based data fetching function accesses the required data (vehicle's ID, type, position, headway, presence of lead vehicle or signal downstream, signal state etc.) in the VISSIM simulation during each time step.
2. The COM API provides the data to the CV/AV logic, which generates an acceleration for the CAV.
3. The COM API outputs the resulting acceleration into a .txt file, which is read and implemented in VISSIM by EDM.
   As discussed in previous sections, EDM is more suitable for longitudinal and lateral control. In COM, a function is created to find the estimated acceleration corresponding to a CV/AV. In the EDM, a function is written to read the estimated acceleration corresponding to a CV/AV from the specific acceleration text file. The vehicle's longitude movement is then controlled by EDM by providing this estimated acceleration value to VISSIM through the command:

   ```
   "Get (DRIVER_DATA_DESIRED_ACCELERATION)".
   ```

   The vehicle's acceleration is updated every time step. The implemented acceleration is the communicated acceleration value but bounded by the VISSIM's maximum acceleration function for the given vehicle model.
4. Utilizing both the COM API and EDM overcomes the disadvantages of both, creating a more robust platform of CAV modeling.

In this section, an AV model and a CV model are selected from the literature and implemented with the longitudinal control framework developed in the previous section.

Three simulation networks are built with a mix of:

i. AV and CNV
ii. CV and CNV
iii. CAV and CNV

In the following sections, the AV, CV, and CAV models simulated are discussed along with the relevant simulation details and results.

4.1 AV Model

In this implementation, a car-following model developed by (Talebpour & Mahmassani, 2016) is used to replicate the AV logic. The AV logic uses a modified version of the Intelligent Driver Model (IDM) with parameters based on (Van Arem, Van Driel, & Visser, 2006) to represent the response of an AV. The model uses three regimes: car following (when a lead vehicle is present), deceleration mode (when the vehicle needs to slow down for a static object such as a red signal), and free driving mode.

When making right or left turns in VISSIM, vehicles must pass through simulation objects called "connectors". Gap acceptance models are required while using connectors making turns. We have not developed a gap acceptance model, hence the driver behavior control is given back temporarily to VISSIM when the AVs need to make a turn.

The following sub-sections discuss these aspects of AV movement logic implementation:

- Initial settings in VISSIM to enable AV modeling
- AV movement model implementation through COM API (Figure 4-1, right part)
- Real-time acceleration implementation through EDM (Figure 4-1, left part).

4.1.1 Initial settings in VISSIM

In VISSIM, an important user-defined attribute needs to be set: "`DistanceToSigHead`" is a "Vehicles in Network" object which shows the distance from the vehicle to the signal downstream on the current lane. This value becomes zero once the vehicle passes through the signal head. This helps associate vehicles to the downstream signal.

The following equation determines this parameter:

"IF(([LANE\MIN:SIGHEADS\POS]=0) | ([POS]>[LANE\MIN:SIGHEADS\POS]),0, [LANE\MIN:SIGHEADS\POS]-[POS])".

VISSIM 10 does not provide a separate vehicle type for AVs. The "AV" category needs to be defined under the "vehicle types" tab in VISSIM. For this vehicle type, under the car following behavior option, the EDM with "DriverModel.dll" file needs to be selected. This ".dll" file is the mechanism through which estimated accelerations are communicated to VISSIM.

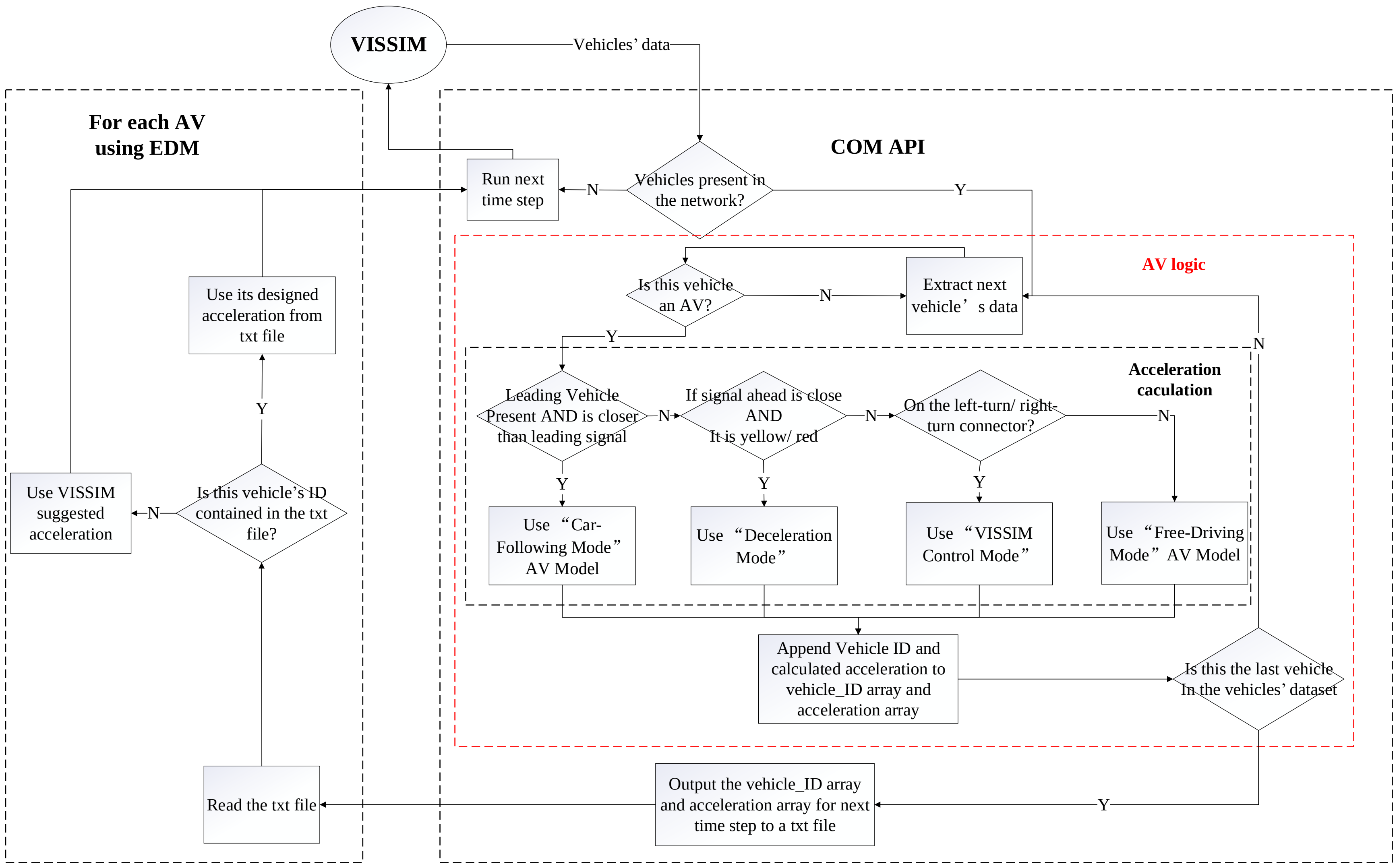

VISSIM
Vehicles' data
For each AV using EDM
COM API
Run next time step
N
Vehicles present in the network?
Y
AV logic
Is this vehicle an AV?
N
Extract next vehicle's data
Y
Use its designed acceleration from txt file
Y
Acceleration caculation
Leading Vehicle Present AND is closer than leading signal
N
If signal ahead is close AND It is yellow/ red
N
On the left-turn/ right-turn connector?
N
N
Y
Y
Y
Use "Car-Following Mode" AV Model
Use "Deceleration Mode"
Use "VISSIM Control Mode"
Use "Free-Driving Mode" AV Model
Use VISSIM suggested acceleration
N
Is this vehicle's ID contained in the txt file?
Append Vehicle ID and calculated acceleration to vehicle_ID array and acceleration array
Is this the last vehicle In the vehicles' dataset
Read the txt file
Output the vehicle_ID array and acceleration array for next time step to a txt file
Y

Figure 4-1 Flowchart of AV Logic Implementation

4.1.2 AV Model implementation in COM API

During each time step, the COM API reads traffic data from VISSIM. The traffic data are extracted with the command:

```
"VISSIM.Net.Vehicles.GetMultipleAttributes(('No',
'VehType', 'length', 'Acceleration', 'Speed', 'Pos',
'Hdwy', 'LeadTargNo', 'LeadTargType', 'DistanceToSigHead',
'SignalState','Lane')) "
```

Where,

"No"	is the vehicle's ID
"VehType"	is the vehicle type
"Pos"	is the vehicle's current position from the start point of its current lane (in meters)
"Hdwy"	is the headway distance from its leading vehicle (in meters)
"LeadTargNo"	is the ID of leading target (it could be ID of signal heads, vehicles, etc).
"LeadTargType"	is the leading target type (if it is a vehicle, signal head etc.)
"SignalState"	is the state of leading signal (green, yellow, red).

If the vehicle type is an AV, then based on the car following regime that the vehicle data fits into, COM API adopts one of four modes to calculate the acceleration of each AV for the next time step. For each time step and for each vehicle ID, the calculated acceleration is stored in a ".txt" file. The following sub-sections detail each mode.

Car-following (Talebpour & Mahmassani, 2016)

The "car-following mode" is executed if the closest leading target is a vehicle. The acceleration calculation for a vehicle mainly considers two elements: safety constraints and vehicle movement.

- The following equation calculates the distance the AV should keep from the leading vehicle if the leading vehicle decelerates at its maximum deceleration level:

$$\Delta x_n = (x_{n-1} - x_n - l_{n-1}) + v_n \tau + \frac{v_{n-1}^2}{2a_{n-1}^{decc}} \tag{1}$$

Where:
Subscript n and n-1 represent the AV and its leader

x_n	is the location of vehicle n

l_n	is the length of vehicle n
v_n	is the speed of vehicle n
τ	is the reaction time of vehicle n
a_n^{decc}	is the maximum deceleration of vehicle n
v_{limit}	is the speed limit of the road

In this study, τ = 0.1s, $a_n^{decc} = -6m/s^2$, v_{limit}= 20 mph

- The following equation provides the safe space headway, considering the minimum of safe distance and sensor detection range (a distance of 300 m was used). It assumes that there is a vehicle at a complete stop outside of the sensors' detection range, which cannot be detected by the sensors at the time of decision-making:

$$\Delta x = \min\{Sensor\ Detection\ range, \Delta x_n\} \tag{2}$$

- This equation calculates the maximum safe speed (v_{max}) considering both the safe distance and maximum deceleration:

$$v_{max} = \min(\sqrt{-2a_n^{decc}\Delta x}, v_{limit}) \tag{3}$$

- This equation shows the movement model that considers the acceleration of the leading vehicle, the speed difference between the AV and the leading vehicle, and the difference between the real headway and reference headway S_{ref}:

$$a_n^d(t) = k_a a_{n-1}(t-\tau) + k_v\big(v_{n-1}(t-\tau) - v_n(t-\tau)\big) + k_d\big(S_n(t-\tau) - S_{ref}\big) \tag{4}$$

Where:

S_n	is the spacing
S_{ref}	is the minimum of: Minimum distance (S_{min}), Following distance based on the reaction time (S_{system}), or Safe following distance (S_{safe}).
a_n^d	is the acceleration of vehicle i
k_a, k_v , and k_d	are model parameters.

Based on the recommendations of Van Arem et al. (2006), $k_a = 1.0$, $k_v = 0.58$, $k_d = 0.1$. In this study, the minimum distance (S_{min}) is set at 2.0 m.

The following equations calculate $S_{system}$and S_{safe}:

$$S_{safe} = \frac{v_{n-1}^2}{2}\left(\frac{1}{a_n^{decc}} - \frac{1}{a_{n-1}^{decc}}\right) \quad (5)$$

$$S_{system} = v_n \tau \quad (6)$$

$$S_{ref} = \min\left(S_{safe}, S_{system}, S_{min}\right) \quad (7)$$

$$a_n(t) = \min(a_n^d(t), k\big(v_{max} - v_n(t), a_{comf} \big) \quad (8)$$

Where,

k	is a model parameter
a_{comf}	is the comfortable acceleration level

In this study, $k = 1.0$, and a_{comf} is assumed to be 2.5 m/s^2.

Deceleration mode

The deceleration mode is executed when the closest object to a vehicle is a signal head with a red or yellow indication, and the vehicle is approaching it.

- The following equation calculates the distance to stop S_d, i.e., the distance required for the vehicle to come to a complete stop if it decelerates at a comfortable deceleration a_{comf}^{decc}:

$$S_d = \frac{1}{2}\frac{V_n(t)^2}{a_{comf}^{decc}} \quad (9)$$

Where,

S_d	is the acceleration distance,
a_{comf}^{decc}	is the comfortable deceleration rate assumed to be $-3.5 m/s^2$. $a_n(t)$ if the distance to signal is less than distance to stop S_d.

VISSIM control mode

When neither of the two aforementioned situations occur (i.e., the vehicle is not influenced by a leading vehicle nor by a signal) and the vehicle is making a right or left turn, it has to pass through a simulation object called a "connector".

Gap acceptance models are required while using connectors making turns. We have not developed a gap acceptance model, hence the driver behavior control is given back temporarily to VISSIM when the AVs need to make a turn. In this case, the vehicle will use the VISSIM suggested acceleration for the next time step and until it exits the connector.

Free driving mode

When the vehicle is not impacted by a leading vehicle, a leading signal, or turning, then the "Free driving" mode is implemented. In this case the vehicle will aim to maintain or reach the desired speed. Acceleration is estimated as:

$$a_n(t) = k(v_{desired} - v_n(t)) \tag{10}$$

Where k is the desired change rate, assumed to be 1 is used in this study.

This acceleration value is subject to VISSIM's vehicle dynamics model and its constraints.

4.1.3 Acceleration implementation through EDM

During every time step, the VISSIM suggested acceleration is stored in the variable "desired_acceleration", through this command:

```
"case DRIVER_DATA_DESIRED_ACCELERATION :
desired_acceleration = double_value;"
```

A function is written to check whether a vehicle ID is contained in the first column of the acceleration ".txt" file. If yes, then the vehicle will use the acceleration from the .txt file. If no, then it will use the VISSIM suggested acceleration (Figure 4-1 left part).

4.2 CV Model

The same longitudinal control framework used for AVs is used in the implementation of CV logic. An Infrastructure-to-Vehicle (I2V) application (PTV, 2017) allowing the CVs to access signal timing information is used to replicate the CV logic. The CV logic (as developed and recommended by VISSIM) seeks to maximize the likelihood of arrival-on-green by changing a vehicle's speeds within certain bounds.

It must be noted here that the "CVs" in VISSIM application example always follow the advice while a true CV would make a choice depending on the driver. This could be modeled by using a factor for driver compliance. In this implementation however, VISSIM logic is used "as-is".

In this case, the CV's acceleration is constrained by both the CV model and the VISSIM default car following model. During each time step, the CV receives Signal Phasing and Timing (SPaT) information. The speed of the CV for the next time step is calculated based on the signal status as follows:

1. If the signal ahead is green: The CV first compares its desired speed with the minimum speed required to arrive at the intersection before this green phase ends (`minSpeedForGreenEnd`)

    1.1 If the desired speed is greater than "`minSpeedForGreenEnd`", the vehicle will maintain its current desired speed.

    1.2 If the desired speed is less than "`minSpeedForGreenEnd`", the vehicle compares the desired speed with the maximum speed to arrive at the intersection after the next green start (`maxSpeedForGreenStart`). The desired speed is chosen as the lower one between the current desired and "`maxSpeedForGreenStart`".

2. If the signal ahead is red the same logic as 1.2 is used. The acceleration is then calculated based on the difference between the current speed and the desired speed.

The CV model implementation is discussed in the following sub-sections:

- Initial settings in VISSIM to enable CV modeling
- CV model implementation through COM Application Programming Interface

  (Figure 4-2, right part)
- Real-time acceleration implementation through EDM (Figure 4-2, left part)

4.2.1 Initial settings in VISSIM

In addition to the settings in AV implementation, the following user-defined attributes are created:

- "`GreenEnd`" is an input attribute which defines the cycle second a particular signal group switches from green to amber.
- "`GreenStart`" is an input attribute which defines the cycle second a particular signal group switches to green.
- "`TimeUntilNextGreen`" calculates the time until the next green phase starts (this is applicable only for pre-timed control), considering the cycle time and the `GreenStart` & `GreenEnd`.
- "`TimeUntilNextRed`" calculates the time until the next red phase starts, considering the `GreenStart` & `GreenEnd`.

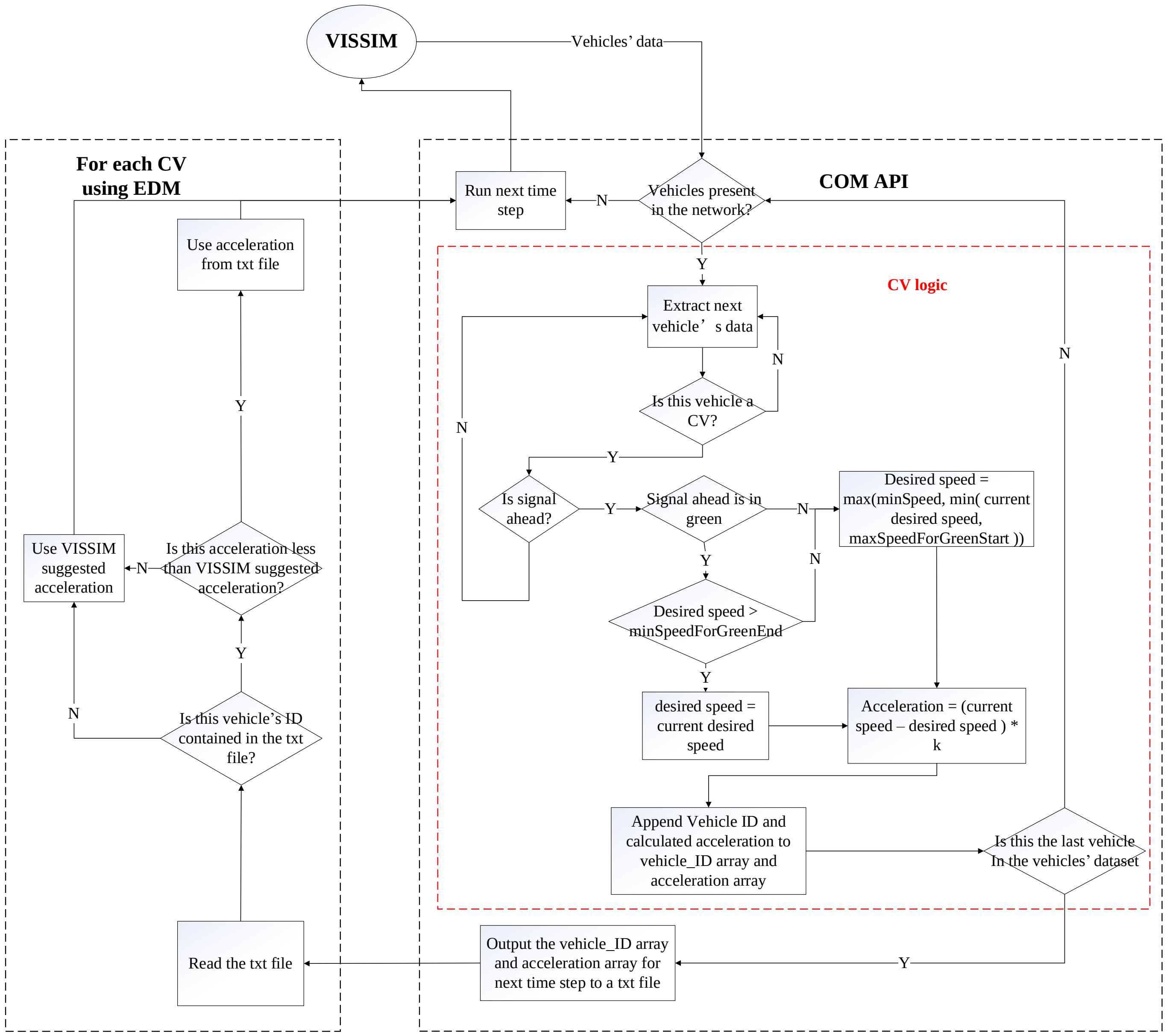

VISSIM
Vehicles' data
For each CV using EDM
COM API
CV logic
Run next time step
Vehicles present in the network?
Extract next vehicle's data
Is this vehicle a CV?
Is signal ahead?
Signal ahead is in green
Desired speed = max(minSpeed, min(current desired speed, maxSpeedForGreenStart))
Desired speed > minSpeedForGreenEnd
desired speed = current desired speed
Acceleration = (current speed – desired speed) * k
Append Vehicle ID and calculated acceleration to vehicle_ID array and acceleration array
Is this the last vehicle In the vehicles' dataset
Output the vehicle_ID array and acceleration array for next time step to a txt file
Read the txt file
Is this vehicle's ID contained in the txt file?
Is this acceleration less than VISSIM suggested acceleration?
Use VISSIM suggested acceleration
Use acceleration from txt file
Y
N

Figure 4-2 Flowchart of CV Logic Implementation

- "`SpeedMaxForGreenStart`" calculates the maximum speed to arrive at the next green start. If the vehicle drives faster, it would arrive at the signal before the next green time.
- "`SpeedMinForGreenEnd`" calculates the minimum speed to arrive before the next green end. If the vehicle drives slower, it would not make it in the current/next green time.

4.2.2 CV Model implementation in COM API

The model implementation considers the following rules:

- If the signal ahead is in green, the COM API first calculates if it is possible to arrive at the intersection within the current green phase, by comparing `SpeedMinForGreenEnd` with desired speed ($v_{desired}$).
- If $v_{desired} >$ `SpeedMinForGreenEnd`, then $v_{desired}$ will remain the same and the vehicle could pass through the intersection during the current green phase at the current desired speed.
- If $v_{desired} <$ `SpeedMinForGreenEnd`, or the signal ahead is red, then COM API will compare $v_{desired}$ with `SpeedMaxForGreenStart`, to check if the vehicle needs to decelerate to arrive after the next green phase start.
- $v_{desired}$ = min ($v_{desired}$, `SpeedMaxForGreenStart`).
- After $v_{desired}$ for each vehicle is determined, equation (10) is used to calculate its acceleration.

4.2.3 Acceleration implementation through EDM

The acceleration implementation through EDM for CVs is the same as for AVs, with one difference (Figure 4-2, left part). When the EDM finds the vehicle ID in the acceleration ".txt" file, it compares the estimated acceleration to the VISSIM suggested acceleration, and uses the lower one (i.e. the vehicle is restricted by both the VISSIM car-following model and the CV logic).

4.3 CAV Model

The CAV logic combines the AV logic and the CV logic, by replacing the VISSIM car-following model with the AV car-following model in the CV logic. The CAV acceleration is then constrained by both the AV car-following model and the CV model.

Note that the two red boxes in Figure 4-3 which are the AV logic and the CV logic, are the same as the red boxes in Figure 4-1 and Figure 4-2 respectively. The "Merge vehicle ID array and acceleration array" process is used to make CAVs follow the lower acceleration value of the acceleration outputs from the AV logic and the CV logic. The following rules apply:

- *Initial settings in VISSIM:* VISSIM settings of CAVs are exactly the same as for CVs, except that the user should create a vehicle type for CAV and select EDM to control it.

- *CAV model implementation in COM API:* The acceleration calculation combines the AV logic and the CV logic ($a_n(t) = \min(a_n(t)_{av}, a_n(t)_{cv})$)

  The CAV uses the AV model for the car-following behavior unless a slower speed (or acceleration) would allow for the vehicle to arrive on green, the boundary condition set by the CV algorithm.

- *Acceleration implementation through EDM:* This is the same as for CVs.

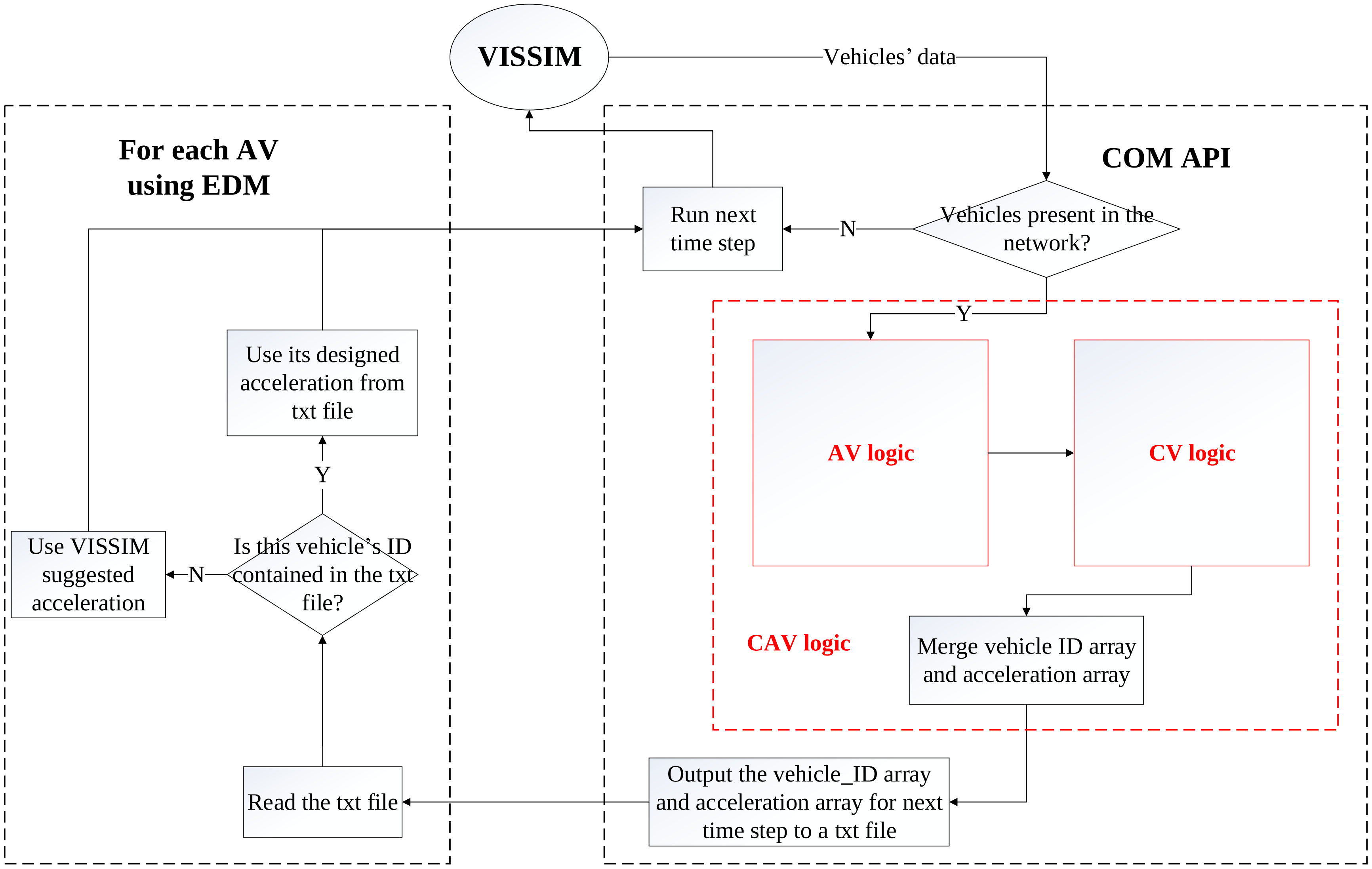

VISSIM
Vehicles' data
For each AV using EDM
COM API
Run next time step
N
Vehicles present in the network?
Y
Use its designed acceleration from txt file
AV logic
CV logic
Y
Use VISSIM suggested acceleration
N
Is this vehicle's ID contained in the txt file?
CAV logic
Merge vehicle ID array and acceleration array
Read the txt file
Output the vehicle_ID array and acceleration array for next time step to a txt file

Figure 4-3 Flowchart of CAV Logic Implementation

4.4 Simulations

The AV and CV models described above were implemented through the COM API and several simulations were run with different combinations of leader and follower vehicles in VISSIM Version 10.0. These vehicles were inserted at different times during the signal cycle to demonstrate and check the implementation of the logic.

Figure 4-4 shows the results of one such test with the trajectories of a pair of vehicles approaching the stop bar of a signalized intersection. The upstream end of the road segment is 500 ft from the stop bar. When the lead vehicle enters the detection range there are 10 seconds of green remaining.

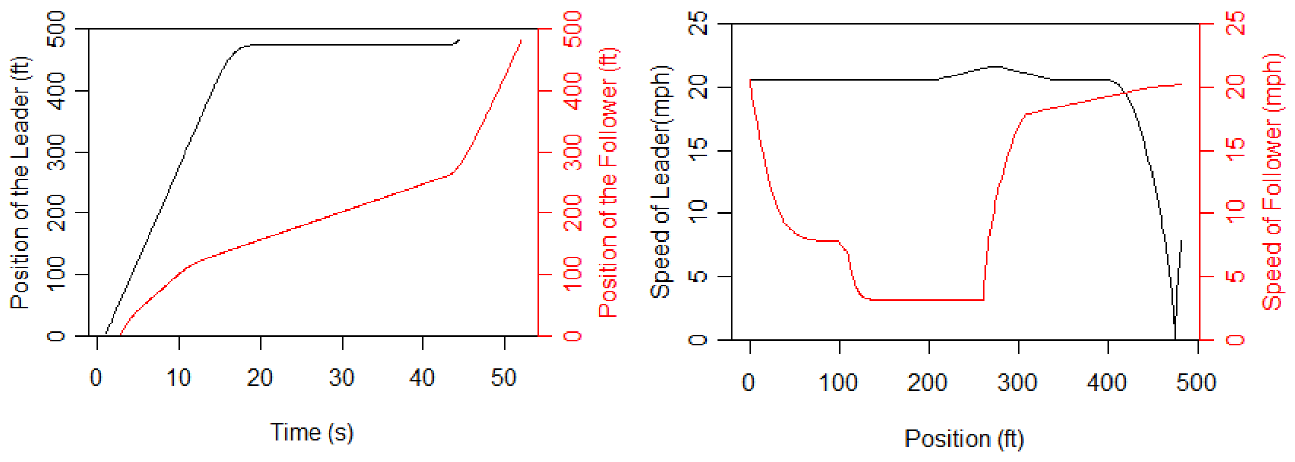
In Figure 4-4 (a) the lead vehicle is a CNV and the follower a CV. The CNV maintains its speed at the speed limit (20 mph) until it approaches the stop bar, and then it decelerates to a complete stop. The CV (the follower) receives the information that the green phase ends soon and it does not have time to cross the stop bar. Hence it slows down and reaches a lower speed, cruising until the next green, and accelerating later to cross the intersection.

In Figure 4-4 (b) the lead vehicle is a CV and the follower a CNV. The leader CV decelerates, cruises and then accelerates to cross the intersection during the next green. The CNV does not have this information and aims to achieve its desired speed, exhibiting oscillation when following the CV which maintains a lower cruising speed.

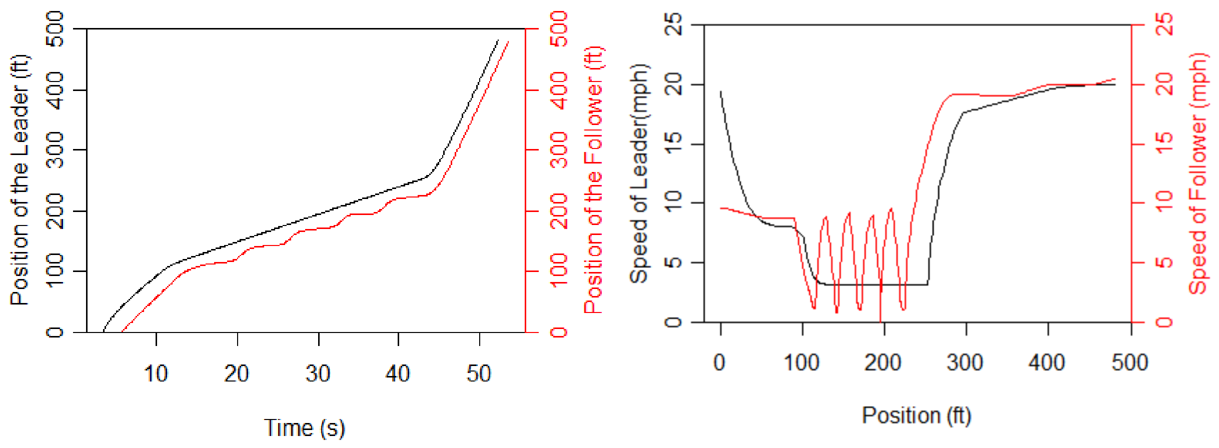
In Figure 4-4 (c) both vehicles are CVs. Therefore, they both exhibit the same behavior, and they decelerate to a lower cruising speed before accelerating to cross the intersection during the next green. In this scenario, the following CV does not exhibit any oscillations since both leader and follower are CVs that follow the same trajectory pattern.

The research team conducted several similar tests with vehicle arrivals throughout the cycle, to ensure the individual vehicles behaved as expected. Next, we simulated a four-leg isolated signalized intersection. The research team replicated the signalized intersection of Gale Lemerand Drive and Stadium Road in Gainesville, Florida.

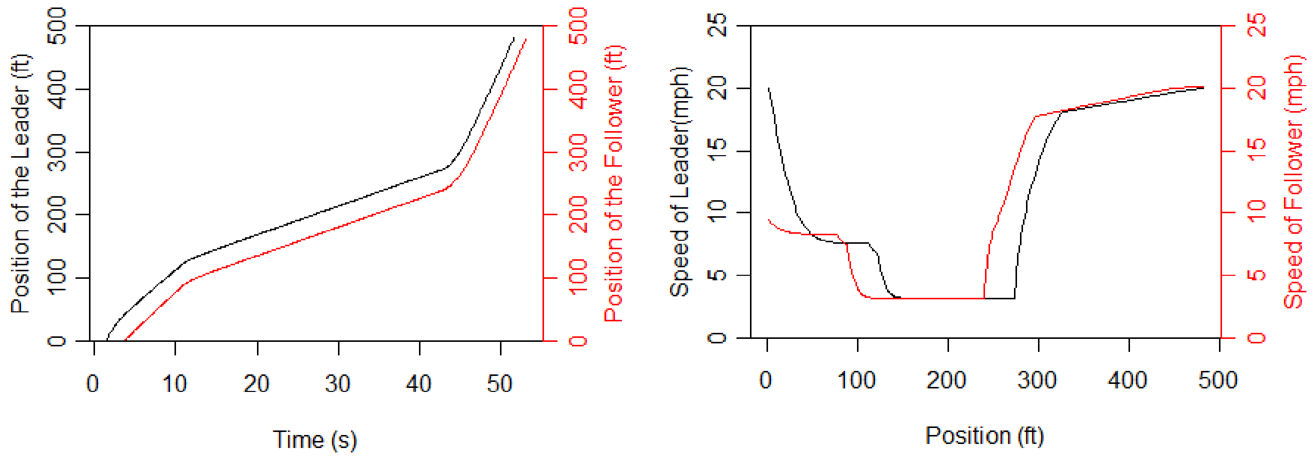
The following sections describe the network layout, simulation scenarios and results.



(a) Leader- CNV, Follower-CV



(b) Leader- CV, Follower-CNV



(c) Leader- CV, Follower-CV

Figure 4-4 Trajectories of Leader and Follower Vehicles

4.4.1 Simulation network layout

Figure 4-4 provides the study intersection layout. The speed limit is 20 mph for all approaches. Each approach has two lanes with an exclusive left turn bay and a shared lane for through and right turning traffic. Figure 4-5 provides the network modeled in VISSIM. The width of lanes and the length of the left turn bays were measured using Google maps.

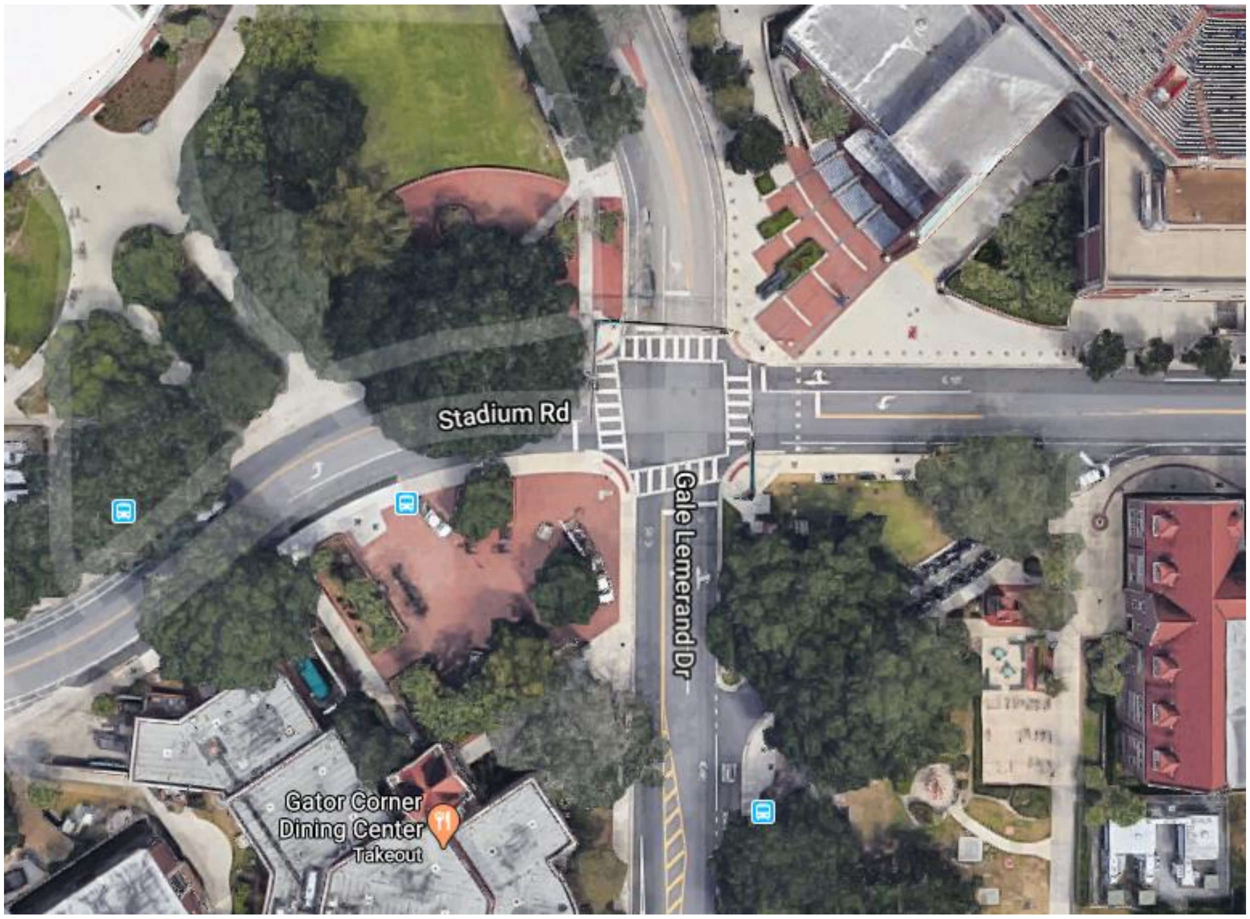


Figure 4-4 Network Layout

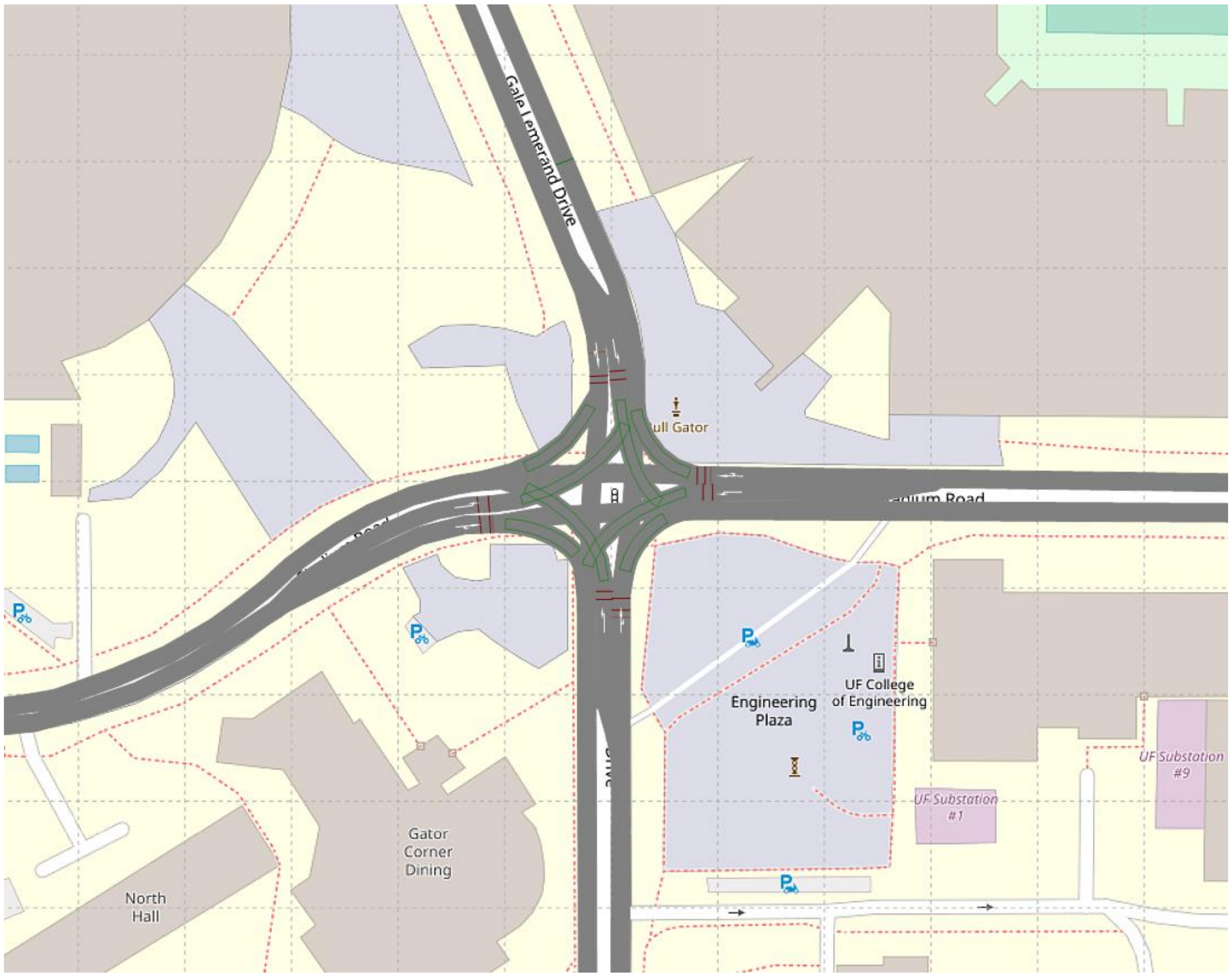


Figure 4-5 Network Layout in VISSIM

4.4.2 Simulation scenarios

To understand the impact of CAVs on traffic operations, different penetration rates of AV, CV, or CAV under different v/c ratios were replicated. Eighteen scenarios were designed with different combinations of v/c ratio and penetration rate for AV, CV, and CAV each (Table 4-1).

Table 4-1 Simulation Scenarios

Penetration Rate → **v/c ↓**	**0**	**0.2**	**0.4**	**0.6**	**0.8**	**1.0**
0.7						
0.85						
0.9						

4.4.3 Simulation signal phasing and timing

Table 4-2 shows the turning movement flows used in the simulation. For simplicity, the total volume of vehicles and the ratio of turning movements from each approach is kept the same. The ratio is set at 60:30:10 for through, right-turns and left-turns, respectively.

Table 4-2 Turning Movements (same for all approaches)

v/c	**Flow rate (veh / h)**			
	L	**T**	**R**	**Total**
0.9	61	365	182	608
0.85	57	344	172	573
0.7	47	284	142	473

Based on the field data collection, the simulated intersection uses two-phase operation with permitted left-turns. Optimal cycle length times were calculated using the HCM6 and rounded to the nearest second, with a min allowable cycle of 60 seconds. The signal phasing and timings used are shown in Table 4-3. As shown, the green and amber durations are the same for both phases.

Table 4-3 Signal Timings

v/c	C_{opt} (s)	**Cycle length used (s)**	**Green time for each phase (s)**	**Amber time for each phase (s)**
0.9	110	110	52	3
0.85	73.33333	74	34	3
0.7	36.66667	60	27	3

4.5 Results

As shown in Table 4-1, 18 scenarios were developed to replicate various market penetrations for each vehicle type (AV, CV and CAV). Each of these 54 scenarios (18*3) were simulated 5 times with different random seeds. The same 5 random seeds were used across all scenarios to ensure the same traffic arrival pattern for different vehicle types (AV, CV, CAV).

Average travel speed (mph) and average delay (s/veh) for the entire network were obtained for each scenario, and these are shown in Figure 4-5 and Figure 4-6 (detailed results are provided in Appendix B). The following are concluded from the simulation runs:

- Increasing penetration rates of CAVs resulted in reductions in delay. Scenarios with CAVs showed the highest delay reduction compared to AVs and CVs alone (Figure 4-6).
- Scenarios with AVs result in better performance at lower traffic volumes (when v/c is 0.7 and 0.85) than the respective scenarios with CVs, as indicated by higher average speed and lower average delay (Figure 4-5 and Figure 4-6). However, the scenarios with CVs show better network-wide performance than those with AVs at higher demands (i.e., for v/c = 0.9).
- The improvements in delay could be attributed to three factors; The reduction in start-up lost time, acceleration of CVs according to the information received and reduction in driver reaction time.
- The results show that a combination of the two technologies (i.e. autonomy and connectivity) yields better performance than each (CV and AV) on their own.

While the simulations show improvement in traffic operational performance, it is also important to consider the environmental impact. In the next section we use the simulation outputs to run an emissions model and evaluate these for each scenario.

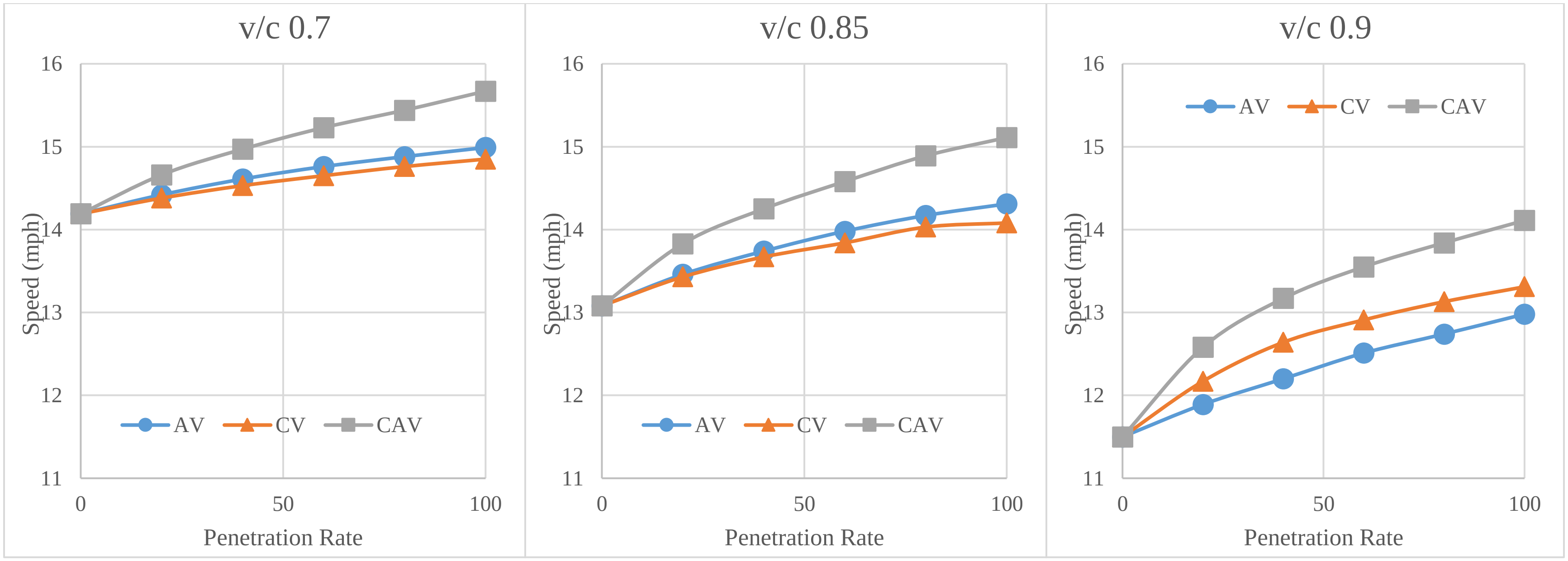


Figure 4-6 Average Network Speeds for Different Penetration Rates of CAVs

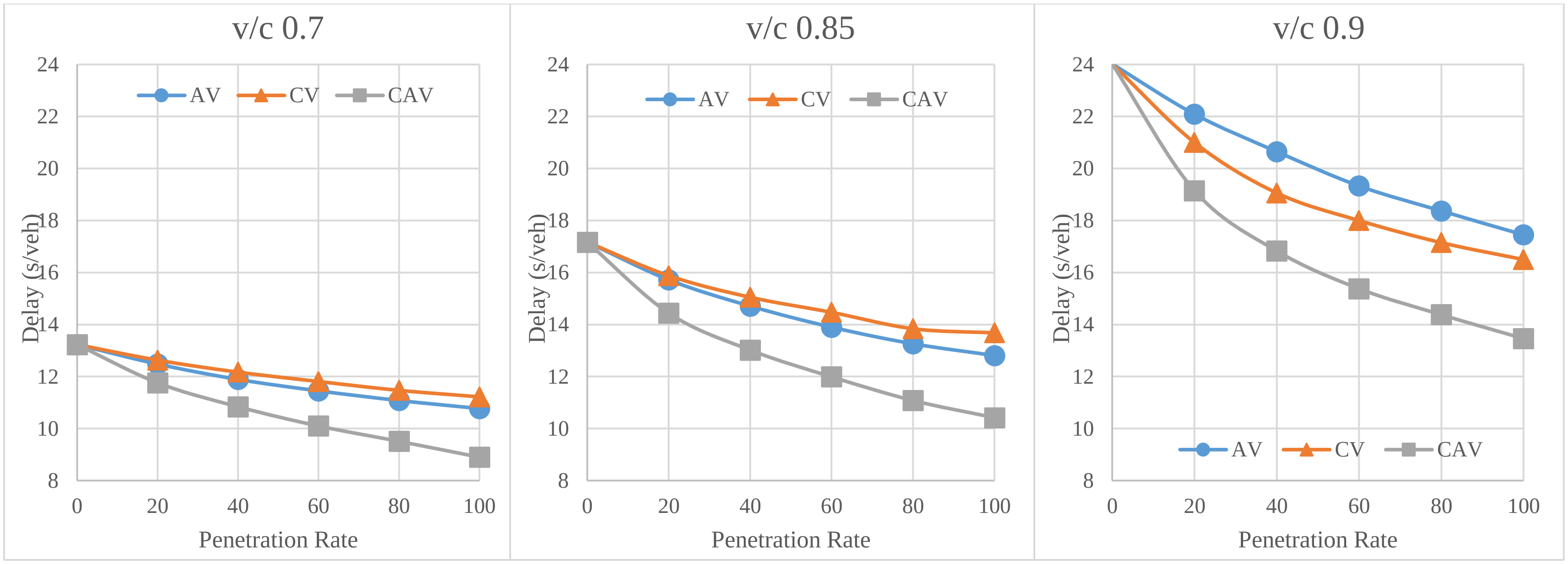


Figure 4-7 Average Network Delay for Different Penetration Rates of CAVs

5 EMISSIONS MODELING

5.1 Introduction

In this section, we further explore the VISSIM CAV implementation discussed thus far in this report and some of the analysis that may be performed on the real-time data streams. The following sections of this chapter provide a brief insight into the network and the trajectory data produced by VISSIM as well as the methodology for processing the trajectory data to calculate several Key Performance Indices (KPIs). Two of the KPIs, CO2 emissions and travel-time, are discussed in this report. Finally, the methodology is applied to several CAV scenarios, with varying demand levels and penetrations rates, providing a comparative analysis of the calculated KPIs.

5.2 Network and Data Information

In this implementation, the four-leg isolated signalized intersection at Stadium Rd. @ Gale Lemerand Dr., FL, is modelled in VISSIM. The network extends 1,405ft along the East-West direction (Stadium Rd) and 1,087ft along the North-South direction (Gale Lemerand Drive). To evaluate the effect of CAV technologies on network environmental metrics, several traffic scenarios are run: demand levels of v/c=0.7 and v/c=0.9, and technology penetration rates of 0% to 100%, at 20% increments.

Trajectory data, i.e., vehicular position over time, is collected for all vehicles during each simulation run and is utilized as the fundamental data for the KPI calculations. For the analysis, the roadway is divided into sections, with each section start and end point labeled as a counter. Figure 5-1 schematically shows the counter placement for different chosen routes in the network. The location of the counters for measuring and comparing KPIs are chosen such that they satisfy the following criteria:

(1) Allow for KPI calculations to reflect operations on both the mainline and side street roadways. In this analysis Stadium Road Eastbound (EB) and Gale Lemerand Dr Northbound (NB) are studied.
(2) Limit initial analysis to through vehicle movements. Turn movements are not included in the current KPIs.
(3) Capture the entire queue lengths. Thus, starting counters are placed at least 300 feet upstream from the stop-bars in all directions. Based on a review of the simulation runs, 300 feet was found sufficient for the given scenarios.
(4) Reflect the result of queuing at the stop bar and the effects of acceleration after the stop bar. Thus, segment end point counters are placed at the stop-bar and at points sufficiently downstream of the stop bar for vehicles to have reached their desired travel speed.

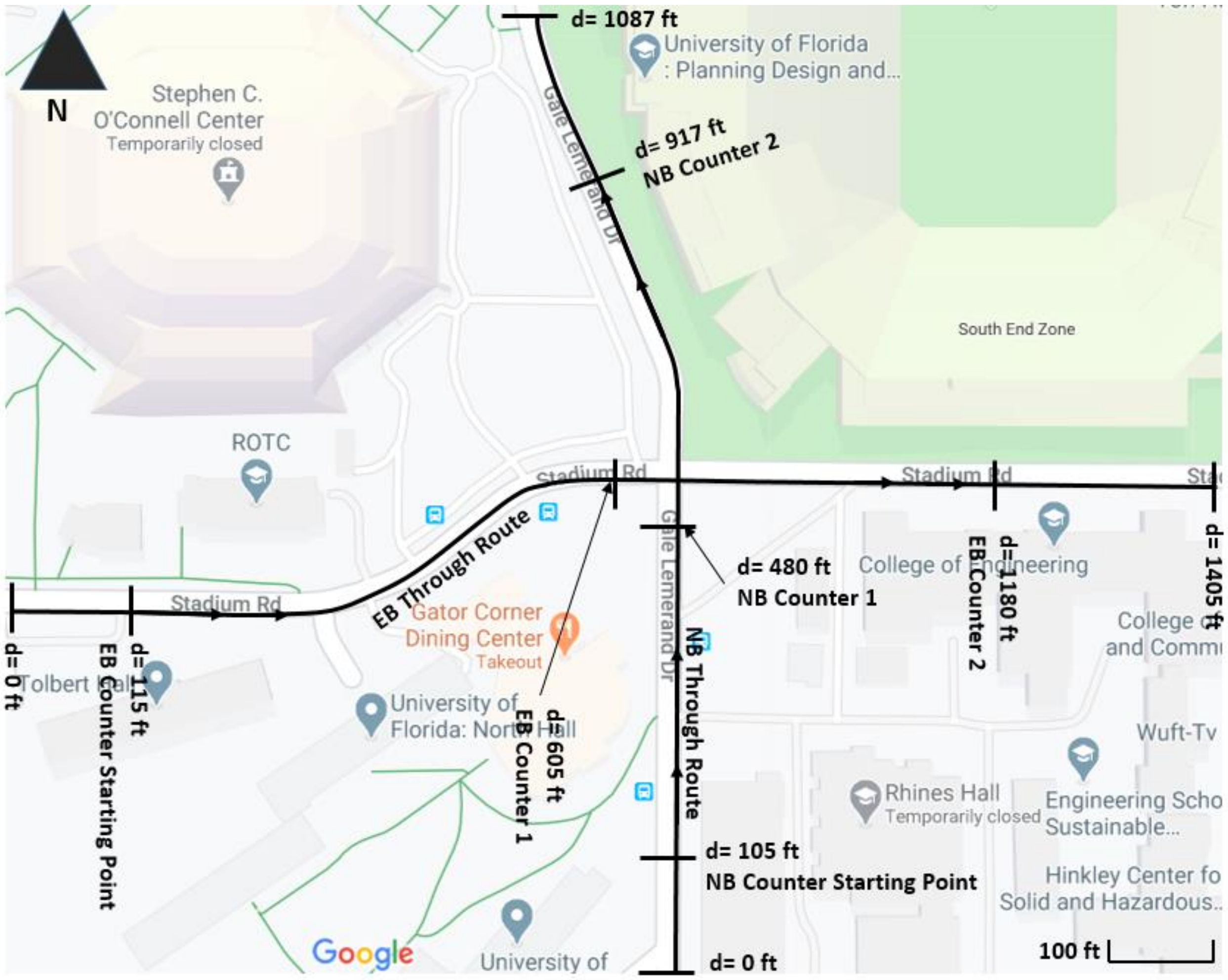
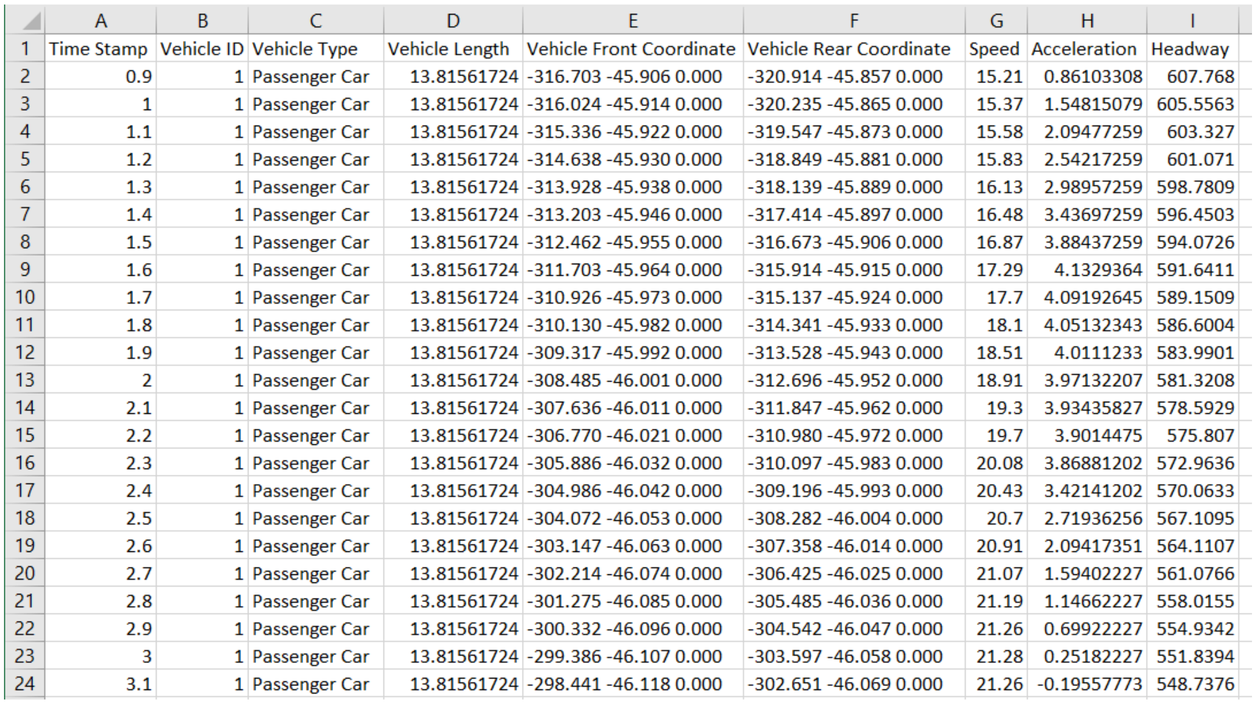


Figure 5-1 Schematic Representation of EB-Through and NB-Through and their Respective Counters

5.2.1 AV and CV Logic implemented in VISSIM

As seen earlier, a model from (Talebpour & Mahmassani, 2016) was used to replicate Autonomous Vehicle (AV) logic. This logic assumes that an AV will have information about all vehicles within its sensor range. The AV logic uses a modified version of the Intelligent Driver Model (IDM) with the parameters being set based on (Van Arem et al., 2006) to represent the response of an AV. An Infrastructure to Vehicle (I2V) application (Evanson, 2017) allowing the CVs to access signal timing information was used to replicate CV logic. The CV logic seeks to maximize the likelihood of arrival-on-green by changing a vehicle’s speeds within certain bounds. Both AV and CV scenarios were implemented separately at different levels of penetration, as well as a CAV scenario which implements both the AV and CV logic. For each penetration rate and demand level, a one-hour VISSIM run is executed. Trajectory data are extracted at 10 Hz for all vehicles present in the network. The VISSIM trajectory file contains the number ID, length, speed, acceleration, front-coordinates, rear-coordinates, and headway for every vehicle, every 0.1 simulation second, for the entire simulation hour. Figure 5-2 shows a screenshot of the initial part of one such trajectory file. Five VISSIM runs were replicated for each demand-penetration scenario by changing random seeds.



	A	B	C	D	E	F	G	H	I
1	Time Stamp	Vehicle ID	Vehicle Type	Vehicle Length	Vehicle Front Coordinate	Vehicle Rear Coordinate	Speed	Acceleration	Headway
2	0.9	1	Passenger Car	13.81561724	-316.703 -45.906 0.000	-320.914 -45.857 0.000	15.21	0.86103308	607.768
3	1	1	Passenger Car	13.81561724	-316.024 -45.914 0.000	-320.235 -45.865 0.000	15.37	1.54815079	605.5563
4	1.1	1	Passenger Car	13.81561724	-315.336 -45.922 0.000	-319.547 -45.873 0.000	15.58	2.09477259	603.327
5	1.2	1	Passenger Car	13.81561724	-314.638 -45.930 0.000	-318.849 -45.881 0.000	15.83	2.54217259	601.071
6	1.3	1	Passenger Car	13.81561724	-313.928 -45.938 0.000	-318.139 -45.889 0.000	16.13	2.98957259	598.7809
7	1.4	1	Passenger Car	13.81561724	-313.203 -45.946 0.000	-317.414 -45.897 0.000	16.48	3.43697259	596.4503
8	1.5	1	Passenger Car	13.81561724	-312.462 -45.955 0.000	-316.673 -45.906 0.000	16.87	3.88437259	594.0726
9	1.6	1	Passenger Car	13.81561724	-311.703 -45.964 0.000	-315.914 -45.915 0.000	17.29	4.1329364	591.6411
10	1.7	1	Passenger Car	13.81561724	-310.926 -45.973 0.000	-315.137 -45.924 0.000	17.7	4.09192645	589.1509
11	1.8	1	Passenger Car	13.81561724	-310.130 -45.982 0.000	-314.341 -45.933 0.000	18.1	4.05132343	586.6004
12	1.9	1	Passenger Car	13.81561724	-309.317 -45.992 0.000	-313.528 -45.943 0.000	18.51	4.0111233	583.9901
13	2	1	Passenger Car	13.81561724	-308.485 -46.001 0.000	-312.696 -45.952 0.000	18.91	3.97132207	581.3208
14	2.1	1	Passenger Car	13.81561724	-307.636 -46.011 0.000	-311.847 -45.962 0.000	19.3	3.93435827	578.5929
15	2.2	1	Passenger Car	13.81561724	-306.770 -46.021 0.000	-310.980 -45.972 0.000	19.7	3.9014475	575.807
16	2.3	1	Passenger Car	13.81561724	-305.886 -46.032 0.000	-310.097 -45.983 0.000	20.08	3.86881202	572.9636
17	2.4	1	Passenger Car	13.81561724	-304.986 -46.042 0.000	-309.196 -45.993 0.000	20.43	3.42141202	570.0633
18	2.5	1	Passenger Car	13.81561724	-304.072 -46.053 0.000	-308.282 -46.004 0.000	20.7	2.71936256	567.1095
19	2.6	1	Passenger Car	13.81561724	-303.147 -46.063 0.000	-307.358 -46.014 0.000	20.91	2.09417351	564.1107
20	2.7	1	Passenger Car	13.81561724	-302.214 -46.074 0.000	-306.425 -46.025 0.000	21.07	1.59402227	561.0766
21	2.8	1	Passenger Car	13.81561724	-301.275 -46.085 0.000	-305.485 -46.036 0.000	21.19	1.14662227	558.0155
22	2.9	1	Passenger Car	13.81561724	-300.332 -46.096 0.000	-304.542 -46.047 0.000	21.26	0.69922227	554.9342
23	3	1	Passenger Car	13.81561724	-299.386 -46.107 0.000	-303.597 -46.058 0.000	21.28	0.25182227	551.8394
24	3.1	1	Passenger Car	13.81561724	-298.441 -46.118 0.000	-302.651 -46.069 0.000	21.26	-0.19557773	548.7376

Figure 5-2 Screenshot of Initial Lines of a Trajectory File from A VISSIM Run (AV Data: 40% penetration, v/c=0.7, random seed=1)

5.3 Data Processing Methodology

Comparative KPI analysis across different scenarios was performed on two quantities, travel-time, and total vehicular Carbon Dioxide (CO2) emissions, which is estimated using EPA's Motor Vehicle Emission Simulator (MOVES) model for the 2017 fleet mix. The EPA's MOVES Model (Agency U.S.E.P, n.d.) estimates vehicle energy consumption and emissions given the vehicle's speed and acceleration. Additional details on the methodology of using MOVES emissions estimator model in this analysis, are provided later. The MOVES model is applied at a 1 Hz rate, thus the 10 Hz trajectory data is aggregated to 1 Hz. For this project median speed for every ten records of each vehicle is used, condensing the 10 Hz data to 1 Hz (i.e., second-by-second). Other methods could be selected for aggregating to a 1 Hz rate, such as mean speed or every tenth data point; however, some initial comparisons showed minimal difference. Median speed was selected as it provides some smoothing of the 10 Hz data while still allowing for fast runtimes, which is critical for real-time applications as discussed elsewhere (Saroj, Roy, Guin, Hunter, & Fujimoto, 2019). The individual vehicle acceleration is derived from the difference in consecutive median speeds in the condensed trajectory data. The second-by-second speed-acceleration data are entered into MOVES to estimate emissions each second for each vehicle. For this analysis, the second-by-second vehicle front-coordinates are utilized to select vehicles that complete the entirety of the EB-Through or NB-Through routes. For each direction, the total number of seconds for a vehicle to travel between counters is recorded, which represents the vehicle travel time. In a similar manner, the total vehicle emissions for a trip was calculated by adding the emission data for every second of the trip between each pair of counters. The calculation of travel-time and emissions was implemented using Python 3.7 (Python, 2018); (Numpy, 2020). Figure 5-3 depicts the overall architecture for the analysis procedure.

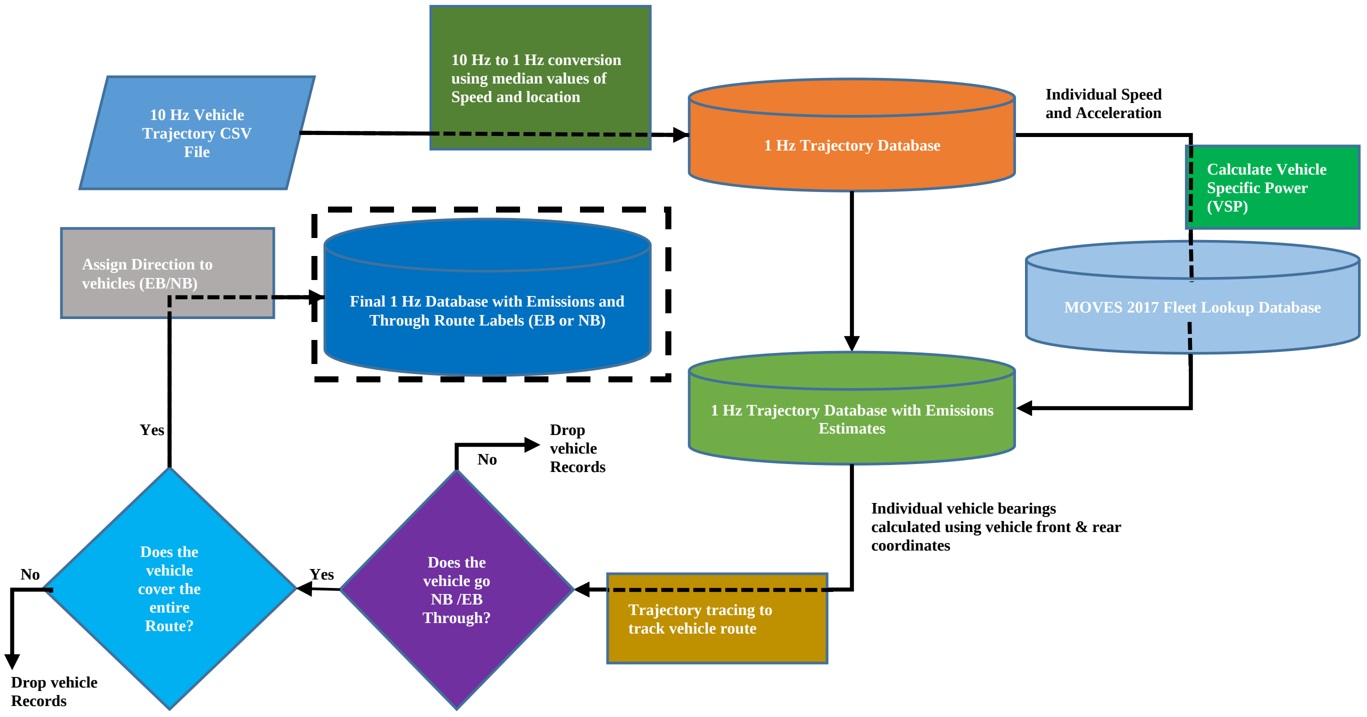

10 Hz Vehicle Trajectory CSV File
10 Hz to 1 Hz conversion using median values of Speed and location
1 Hz Trajectory Database
Individual Speed and Acceleration
Calculate Vehicle Specific Power (VSP)
MOVES 2017 Fleet Lookup Database
1 Hz Trajectory Database with Emissions Estimates
Assign Direction to vehicles (EB/NB)
Final 1 Hz Database with Emissions and Through Route Labels (EB or NB)
Individual vehicle bearings calculated using vehicle front & rear coordinates
Trajectory tracing to track vehicle route
Does the vehicle go NB /EB Through?
No
Drop vehicle Records
Yes
Does the vehicle cover the entire Route?
Yes
No
Drop vehicle Records

Figure 5-3 Architecture for KPI Calculation and Route Allocation from Trajectory Data

Before proceeding with the analysis and the results discussion, we briefly describe the emission estimation technique using MOVES.

5.3.1 Energy and Emission Estimation using MOVES

The initially condensed 1 Hz trajectory data are processed to estimate emissions using MOVES. The MOVES process of estimating vehicular energy and emissions is developed and mandated by the US Environmental Protection Agency (USEPA) (Guensler et al., 2017). At every one-second time-step an individual vehicle's speed and acceleration is used to calculate Vehicle Specific Power (VSP) for each vehicle. The speed, acceleration, and VSP is then used to identify the operating mode bin. A combination of the bin-number and vehicle type is then utilized to estimate the energy consumption and emissions (CO_2, NOX, etc.). Aggregation across vehicles and time-steps is used to obtain the hourly rates. The individual elements of this procedure are described in more detail in the following subsections.

5.3.1.1 Vehicle Specific Power (VSP) Calculation:

VSP (or STP) is a function of vehicle mass, dynamics parameters, speed, acceleration, road grade (if available), and gravitational acceleration.

$$VSP/STP = \left(\frac{A}{M}\right)v + \left(\frac{B}{M}\right)v^2 + \left(\frac{C}{M}\right)v^3 + \left(\frac{m}{M}\right)(acc + g * \sin\theta)v \quad (1)$$

Where:

$$VSP = vehicle\ specific\ power\ (\frac{kW}{tonne}, power\ to\ weight\ ratio),$$
$$STP = scaled\ tractive\ power\ (kW/tonne)$$
$$v = second\text{-}by\text{-}second\ velocity, m/sec$$
$$acc = second\text{-}by\text{-}second\ acceleration, \frac{m}{sec^2}$$
$$g = graviational\ acceleration (9.81\ m/sec^2)$$
$$\theta = road\ grade (radians\ or\ degrees, as\ required\ by\ the\ sin\ calculation\ algorithm)$$
$$m = vehicle\ mass\ (tonnes)$$
$$A = rolling\ resistance\ (kW\text{-}sec/m)$$
$$B = rotating\ resistance\ (kW\text{-}sec^2/m^2)$$
$$C = aeodynamic\ drag\ (kW\text{-}sec^3/m^3)$$
$$M = fixed\ mass\ factor\ for\ the\ source\ type\ (tonnes)$$

A, B, C, M, and m are predetermined constants that vary depending on the type of vehicle (Passenger Car, Single Unit, Bus, Trailer Truck, etc.) (Zhai, Frey, & Rouphail, 2008).

5.3.1.2 VSP Operating Mode Bin using Speed, Acceleration, and VSP:

A combination of VSP, speed, and acceleration are used to determine the operating mode for each second of vehicle operation. Each operating mode corresponds to an ID (VSP bin) which then directs to a lookup table that provides an estimate of energy consumption or vehicular emissions. The VSP operating mode bins are assigned as given in Table 5-1 below (Guensler et al., 2017).

Table 5-1 MOVES VSP/STP Operating Mode Bins

Operating Mode ID	**Operating Mode Description**	**Vehicle Specific Power (VSP)**	**Vehicle Speed**	**Vehicle Acceleration**
		(KW/tonne)	**(v_t, mph)**	**(a, mph/sec)**
0	Deceleration/Braking			$a_t \leq -2.0$ OR ($a_t < -1.0$ AND $a_{t-1} < -1.0$ AND $a_{t-2} < -1.0$)
1	Idle		$-1.0 \leq v_t < 1.0$	Any
11	Coast	$VSP_t < 0$	$0 \leq v_t < 25$	Any
12	Cruise/Acceleration	$0 \leq VSP_t < 3$	$0 \leq v_t < 25$	Any
13	Cruise/Acceleration	$3 \leq VSP_t < 6$	$0 \leq v_t < 25$	Any
14	Cruise/Acceleration	$6 \leq VSP_t < 9$	$0 \leq v_t < 25$	Any
15	Cruise/Acceleration	$9 \leq VSP_t < 12$	$0 \leq v_t < 25$	Any
16	Cruise/Acceleration	$12 \leq VSP_t$	$0 \leq v_t < 25$	Any
21	Coast	$VSP_t < 0$	$25 \leq v_t < 50$	Any
22	Cruise/Acceleration	$0 \leq VSP_t < 3$	$25 \leq v_t < 50$	Any
23	Cruise/Acceleration	$3 \leq VSP_t < 6$	$25 \leq v_t < 50$	Any
24	Cruise/Acceleration	$6 \leq VSP_t < 9$	$25 \leq v_t < 50$	Any
25	Cruise/Acceleration	$9 \leq VSP_t < 12$	$25 \leq v_t < 50$	Any
27	Cruise/Acceleration	$12 \leq VSP_t < 18$	$25 \leq v_t < 50$	Any
28	Cruise/Acceleration	$18 \leq VSP_t < 24$	$25 \leq v_t < 50$	Any
29	Cruise/Acceleration	$24 \leq VSP_t < 30$	$25 \leq v_t < 50$	Any
30	Cruise/Acceleration	$30 \leq VSP_t$	$25 \leq v_t < 50$	Any
33	Cruise/Acceleration	$VSP_t < 6$	$50 \leq v_t$	Any
35	Cruise/Acceleration	$6 \leq VSP_t < 12$	$50 \leq v_t$	Any
37	Cruise/Acceleration	$12 \leq VSP_t < 18$	$50 \leq v_t$	Any
38	Cruise/Acceleration	$18 \leq VSP_t < 24$	$50 \leq v_t$	Any
39	Cruise/Acceleration	$24 \leq VSP_t < 30$	$50 \leq v_t$	Any
40	Cruise/Acceleration	$30 \leq VSP_t$	$50 \leq v_t$	Any

5.3.1.3 Use Energy/Emissions Lookup for Estimate:

Based on the VSP bin and the year and classification of the vehicle, MOVES provides an estimate of energy and emissions. As an example, Table 5-2 below (Guensler et al., 2017) provides the lookup table for energy and emissions estimates, for the 2017 Passenger Car fleet, as by (Guensler et al., 2017) at Georgia Tech as part of the Moves Matrix effort. The table values are hourly; therefore, to obtain energy/emissions for 1 second of operation, the table value is divided by 3600. For this analysis, all vehicles were passenger cars. The 2017 vehicle fleet mix is utilized for estimating emissions.

Table 5-2 MOVES VSP Bin-Energy/Emission Lookup Table for 2017 Passenger Car

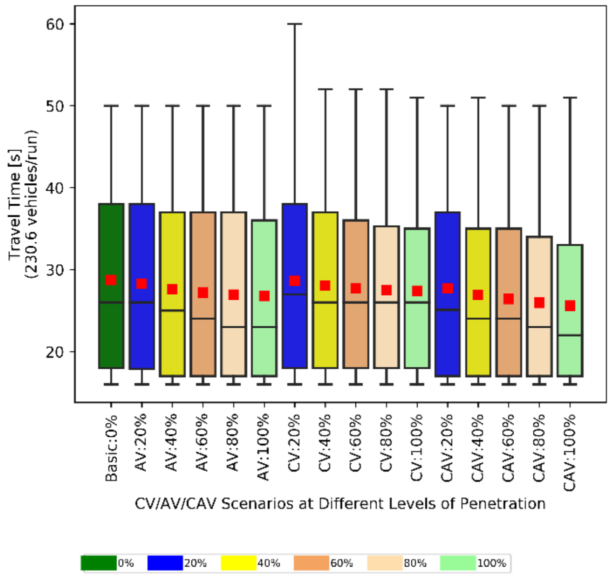
VSP Bin	Energy [kJ/hr]	CO2 [g/hr]	HC [g/hr]	CO [g/hr]	NOX [g/hr]	VOC [g/hr]	PM10 [g/hr]	PM2.5 [g/hr]
0	38883.4	2795.14	0.056143	1.18597	0.031367	0.0273302	0.0312485	0.0276559
1	36154.1	2598.94	0.013884	0.211493	0.0304802	0.0067588	0.0269716	0.0238707
11	56203	4040.16	0.040471	5.19213	0.0508617	0.0197014	0.0267616	0.0236848
12	76539.5	5502.05	0.031012	8.47656	0.0776475	0.0150966	0.0281616	0.0249238
13	104739	7529.16	0.058614	7.81524	0.181771	0.0285334	0.0398926	0.0353062
14	131682	9465.99	0.07973	11.2135	0.320917	0.0388126	0.0395628	0.0350143
15	156582	11256	0.11115	16.2606	0.56877	0.0541079	0.0380609	0.033685
16	187952	13510.9	0.177545	27.4365	1.18617	0.0864292	0.0991837	0.0877805
21	76113.9	5471.46	0.060581	6.76729	0.100583	0.029491	0.0510926	0.0452185
22	85229.3	6126.72	0.055637	8.96628	0.163372	0.0270841	0.0681786	0.0603401
23	102924	7398.67	0.059982	11.5309	0.246983	0.0291994	0.0492257	0.0435662
24	131428	9447.74	0.114459	16.8562	0.416271	0.0557189	0.0541987	0.0479675
25	174998	12579.7	0.114031	19.1302	0.58379	0.0555102	0.0674152	0.0596645
27	229015	16462.8	0.180204	28.7293	0.919244	0.0877231	0.108152	0.0957173
28	308695	22190.6	1.21865	110.987	4.80478	0.59324	0.236312	0.209143
29	422913	30401.2	2.16363	235.053	8.43589	1.05326	1.13558	1.00502
30	531087	38177.3	3.57204	825.556	11.0986	1.73887	1.6822	1.48879
33	106213	7635.11	0.058195	5.08099	0.214375	0.0283295	0.0704087	0.0623138
35	168360	12102.6	0.080815	8.67589	0.591387	0.0393406	0.104319	0.092325
37	218360	15696.8	0.103557	12.7708	0.826399	0.0504116	0.080005	0.0708068
38	284731	20467.9	0.822918	101.727	4.07977	0.400597	0.195906	0.173382
39	379256	27262.9	1.1949	107.339	6.07319	0.581679	0.412559	0.365127
40	483460	34753.6	1.56228	315.464	7.6491	0.760519	0.476596	0.421802

5.4 Comparative Analysis and Discussion

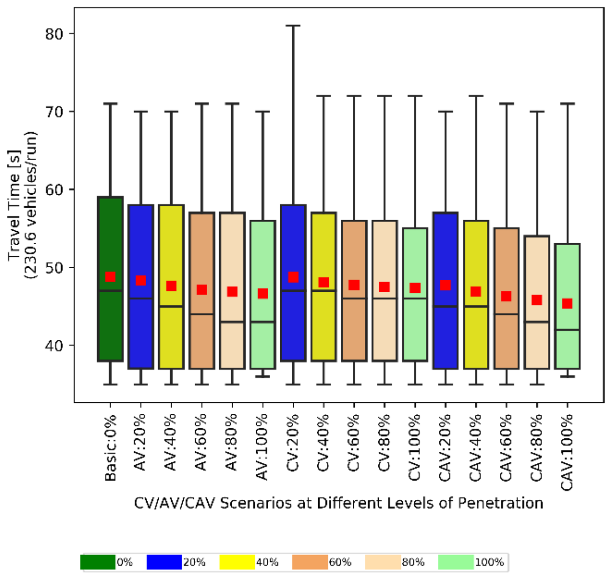
As stated, two KPIs are determined in the current analysis: (1) Travel-time and (2) Total MOVES Estimated CO2 Emissions. These are determined for EB-Through (main-street) and NB-Through (side-street) vehicles from the starting counter to the stop bar and from the starting counter to a point downstream of the intersection (as depicted in Figure 5-1). Vehicles that enter the model in the first 600 simulation seconds were not included in the analysis, to allow for a simulation warm-up period. Therefore, for all aspects of analysis in this section, only vehicle trips starting at t=600s or later and ending at or before t=3600s are considered. For all the boxplots that are presented in the sub-sections that follow (Figure 5-4 to Figure 5-11), the red square dots represent the mean of the given quantity. As a brief recap on boxplots in general, the top and bottom of the solid rectangular box represent the 75th percentile and 25th percentile, respectively, and the black line splitting the solid portion of the box into two is the median.

5.4.1 Travel-time Comparative Analysis

Travel-time is studied across penetration and demand levels, aggregated over five replicate runs. For the EB through traffic with the v/c=0.7 scenario, there were approximately 1,150 vehicles that traversed the entire EB-Through route, between t=600s and t=3,600s. Figure 5-4 (a) & (b) reflect the travel-time from the starting counter to the stop-bar (counter 1) and from the starting counter to a point downstream of the intersection (counter 2), respectively (counter locations are shown in Figure 5-1). Both plots show a trend of decreasing mean, median, and 75th percentile travel-time with increases in AV, CV, and CAV penetration levels. The trend is more prominent in the AV and CAV cases than in the CV cases. The same trends are consistently observed for NB-Through vehicles as well (Figure 5-5 (a) & (b)).

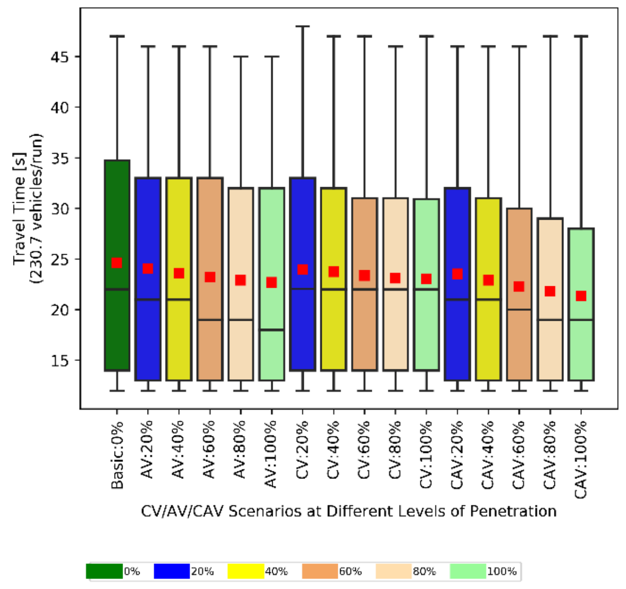


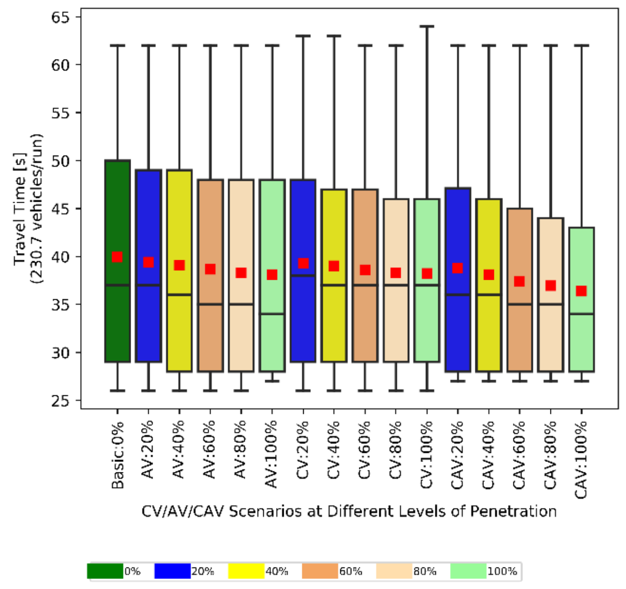
(a)



(b)

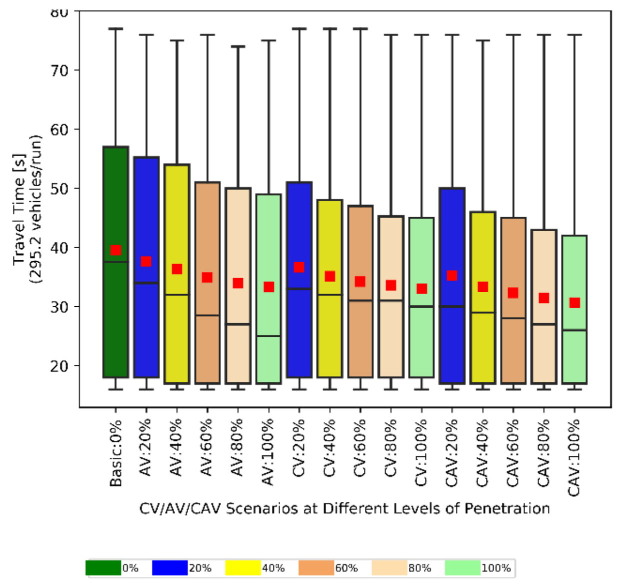
Figure 5-4 Comparative Study: Travel-Time for EB-Through Route for v/c=0.7 Detected at (a) Counter-1, and (b) Counter-2

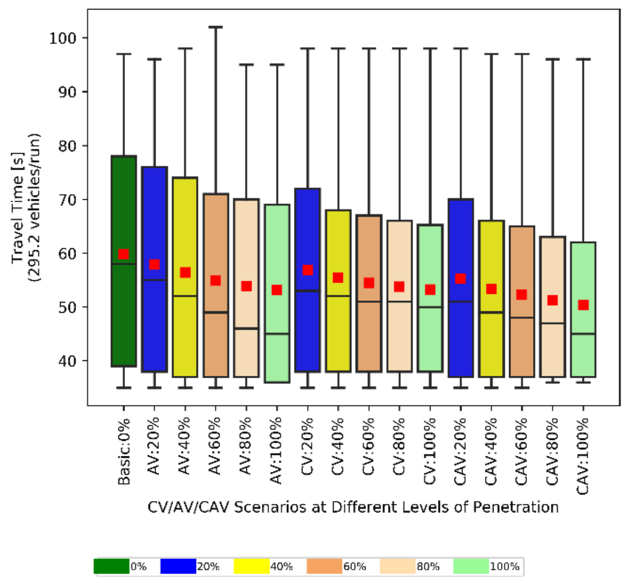




(a) **(b)**

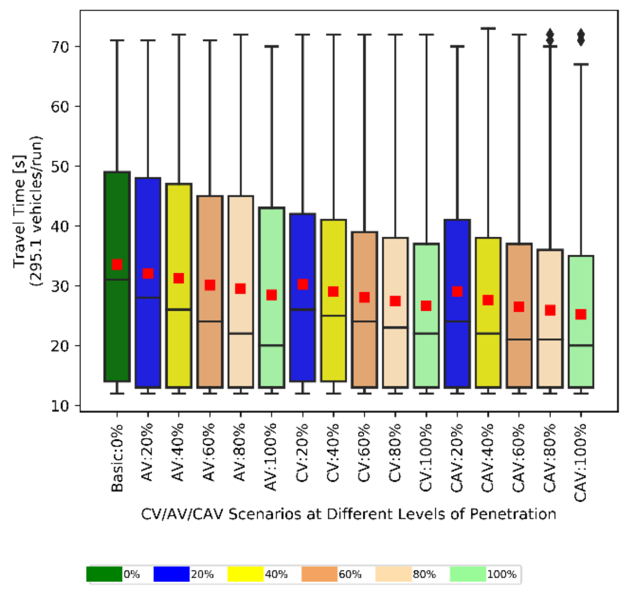
Figure 5-5 Comparative Study: Travel-Time for NB-Through Route for v/c=0.7 Detected at (a) Counter-1, and (b) Counter-2



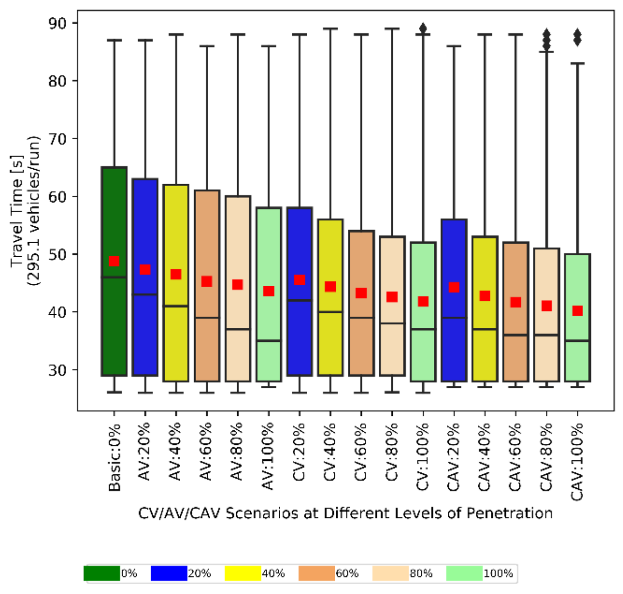


(a) **(b)**

Figure 5-6 Comparative Study: Travel-Time for EB-Through Route for v/c=0.9 Detected at (a) Counter-1, and (b) Counter-2



(a)



(b)

Figure 5-7 Comparative Study: Travel-Time for NB-Through Route for v/c=0.9 Detected at (a) Counter-1, and (b) Counter-2

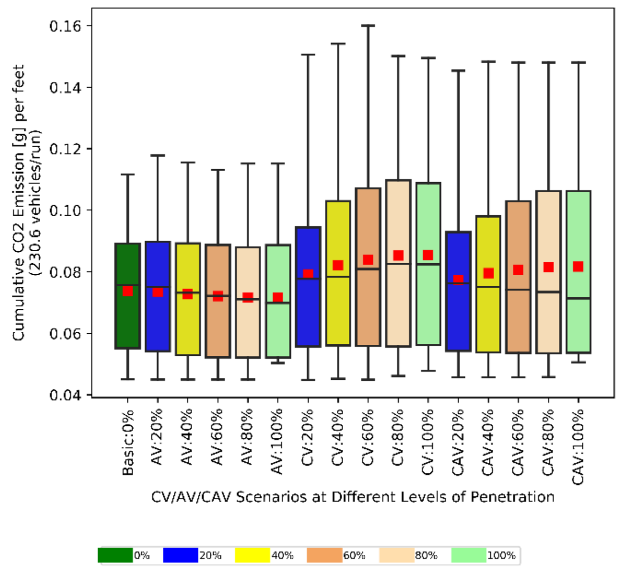
When the analysis is performed for the higher demand level scenario with v/c=0.9, the routes consistently processed between 1,470 and 1,480 vehicles, across the five replicate runs. As can be seen in Figure 5-6 (a) & (b) and Figure 5-7 (a) & (b), the trend of reduction in mean travel-time with increases in AV, CV, and CAV penetration is observed to become more pronounced in the v/c=0.9 scenario. Also, the travel-time Inter-Quartile Range (IQR), depicted by the vertical length of the boxes in the plots, shows greater reductions with increasing penetration rates. In summary, it can be concluded that increasing AV, CV, and CAV penetration rates leads to overall improvement in travel time, and the effect is more prominent under higher demand levels for the modeled intersection.

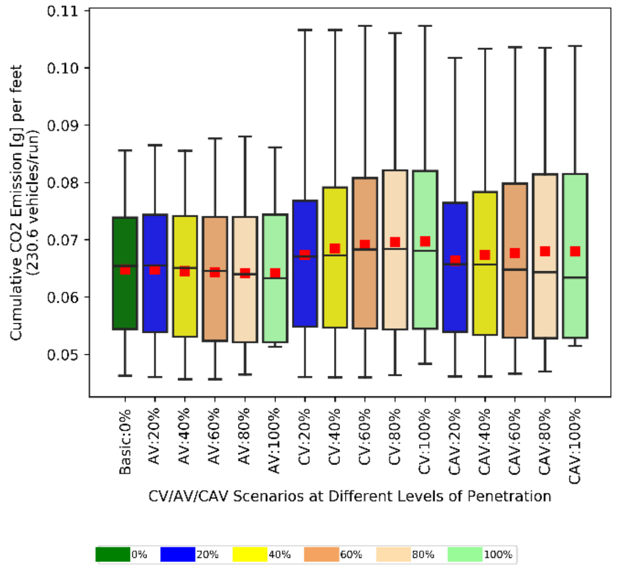
It is noted that these values differ slightly from those reported in the previous section as the travel times are recorded only over the zones set to allow for emissions calculations and only through vehicles are included in this analysis.

5.4.2 CO2 Emission (MOVES estimate) Comparative Analysis

Similar analysis is performed for the MOVES estimated CO_2 emissions. To allow for a comparison between routes and scenarios, CO_2 emissions is expressed as emissions per unit distance (in grams per feet) for each individual vehicle. The same roadway sections are used as in the previous subsection.

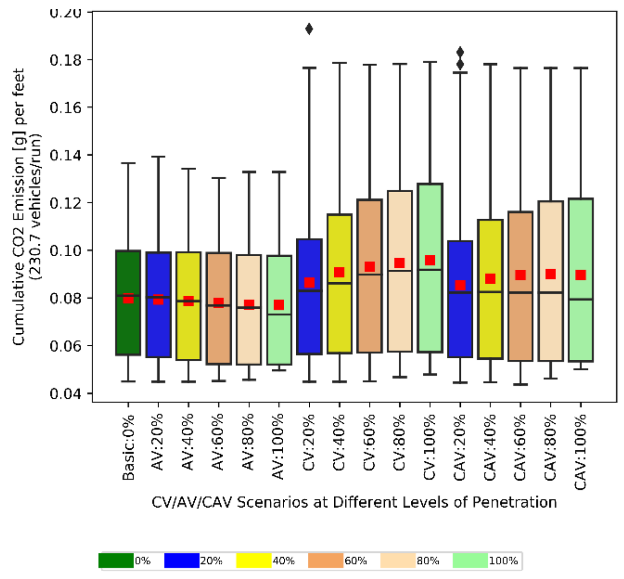
As seen in Figure 5-8 (a) & (b) below, for EB-Through vehicles, the mean and median emissions show a slight reduction with increasing AV penetration. However, mean, and median emissions show an increase with increasing CV penetration, while CAV increasing penetration rate has an increasing average but decreasing median. The emissions IQR also increases for CV and CAV. The same trends are observed for NB-Through vehicles, as shown in Figure 5-9(a) & (b) below.

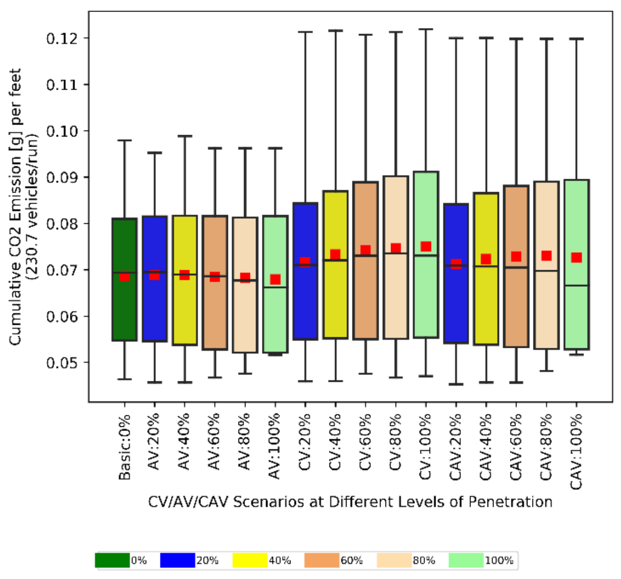




(a) **(b)**

Figure 5-8 Comparative Study: Emissions per Feet for EB-Through Route for v/c=0.7 Detected at (a) Counter-1, and (b) Counter-2

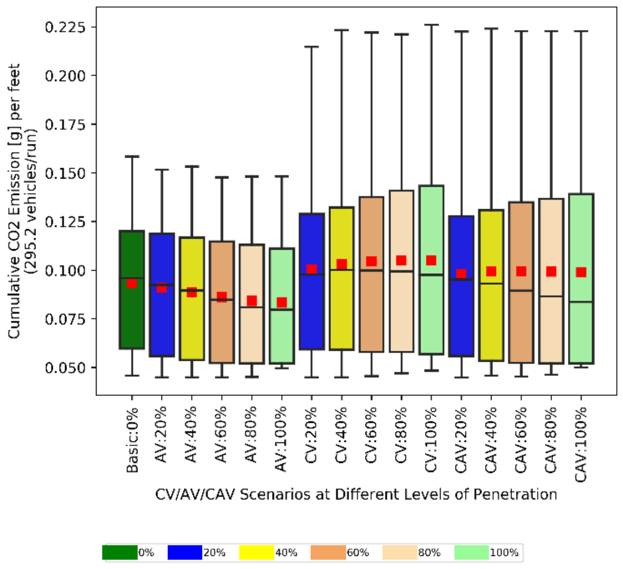


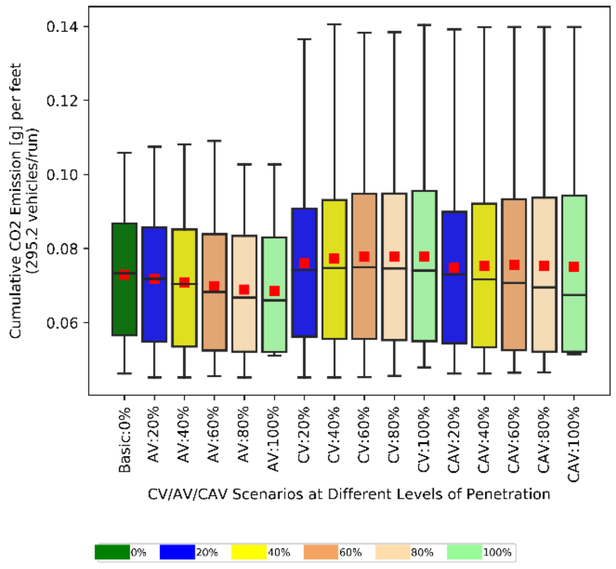


(a) **(b)**

Figure 5-9 Comparative Study: Emissions per Feet for NB-Through Route for v/c=0.7 Detected at (a) Counter-1, and (b) Counter-2

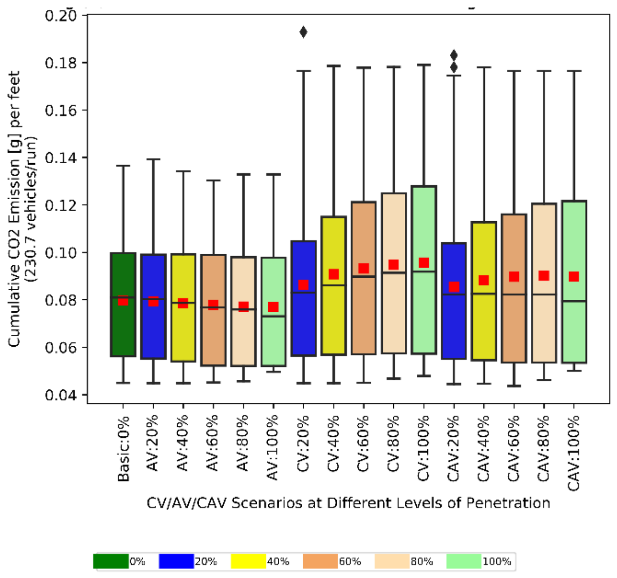
Emission trends are further studied for the higher demand level (v/c=0.9). As seen in Figure 5-10 (a) & (b) and Figure 5-11 (a) & (b), emissions follow similar trends as with the lower demand level (v/c=0.7), although the increases are less pronounced.

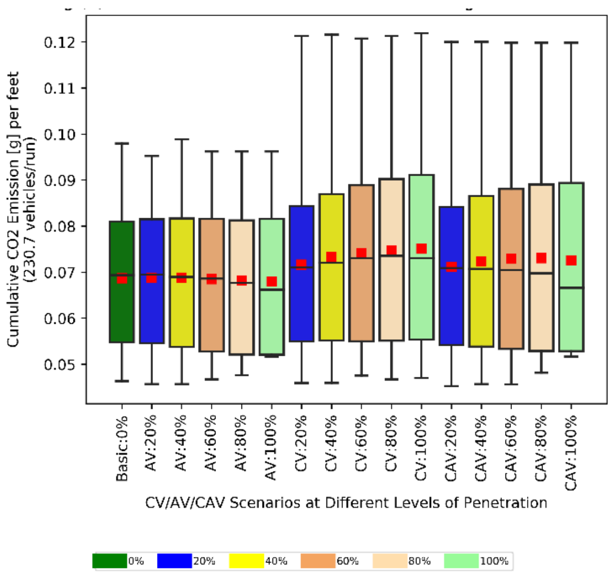




(a) **(b)**

Figure 5-10 Comparative Study: Emissions per Feet for EB-Through Route for v/c=0.9 Detected at (a) (left) Counter-1, and (b) (right) Counter-2





(a) **(b)**

Figure 5-11 Comparative Study: Emissions per Feet for NB-Through Route for v/c=0.9 Counted at (a) (left) Counter-1, and (b) (right) Counter-2

Throughout these analyses it is also seen that the AV had the lowest emissions for each penetration level, with the CV having the highest and the CAV falling between the two. It is also seen throughout that the CV and CAV scenarios consistently have higher emissions than the base case, which has no connected or autonomous technology.

Based on the travel-time studies in the previous subsection, it would appear, vehicles should have more efficient route traversal with increasing AV, CV, and CAV penetration, which would imply less overall stopping (either in duration, number of stops, or both). Hence, it should intuitively follow that vehicles overall should have lower emissions with increasing penetration rates. However, this was not seen for CV and CAV. A discussion of this counter-intuitive result is given in the next section.

5.4.2.1 Emission Analysis: Individual Speed Trajectories

The preceding emissions analysis were at an aggregate level. To find the root cause for increasing emissions with increasing CV and CAV penetration, individual vehicle trajectories need to be considered. Figure 5-12 (a, b, c, and d) depict the variation of acceleration with time for three individual vehicles that had mean emission values closest to the mean emissions for their corresponding scenarios. These acceleration profiles are shown for CV penetration rates of 0% (Figure 5-12 (a)), 20% (Figure 5-12(b)), 60% (Figure 5-12(c)), and 100% (Figure 5-12(d)), for v/c=0.9, and simulation random seed=1, on the EB route. The plots show increasing second-by-second fluctuations in accelerations, with increases in CV penetration. It is hypothesized that, in the implemented CV logic, the vehicles are undergoing more instances of acceleration and decelerations by undertaking frequent adjustments to their speed to maximize the likelihood of arrival-on-green. This increasing number of changes in acceleration results in higher emissions.

However, in the real world there are certain limitations on how fast an acceleration change can be propagated through the drivetrain as well as limitations on the frequency and magnitude of acceleration reversals that are acceptable for driver comfort. It is therefore expected that for implementation success a CV algorithm would need to smooth out the accelerations.

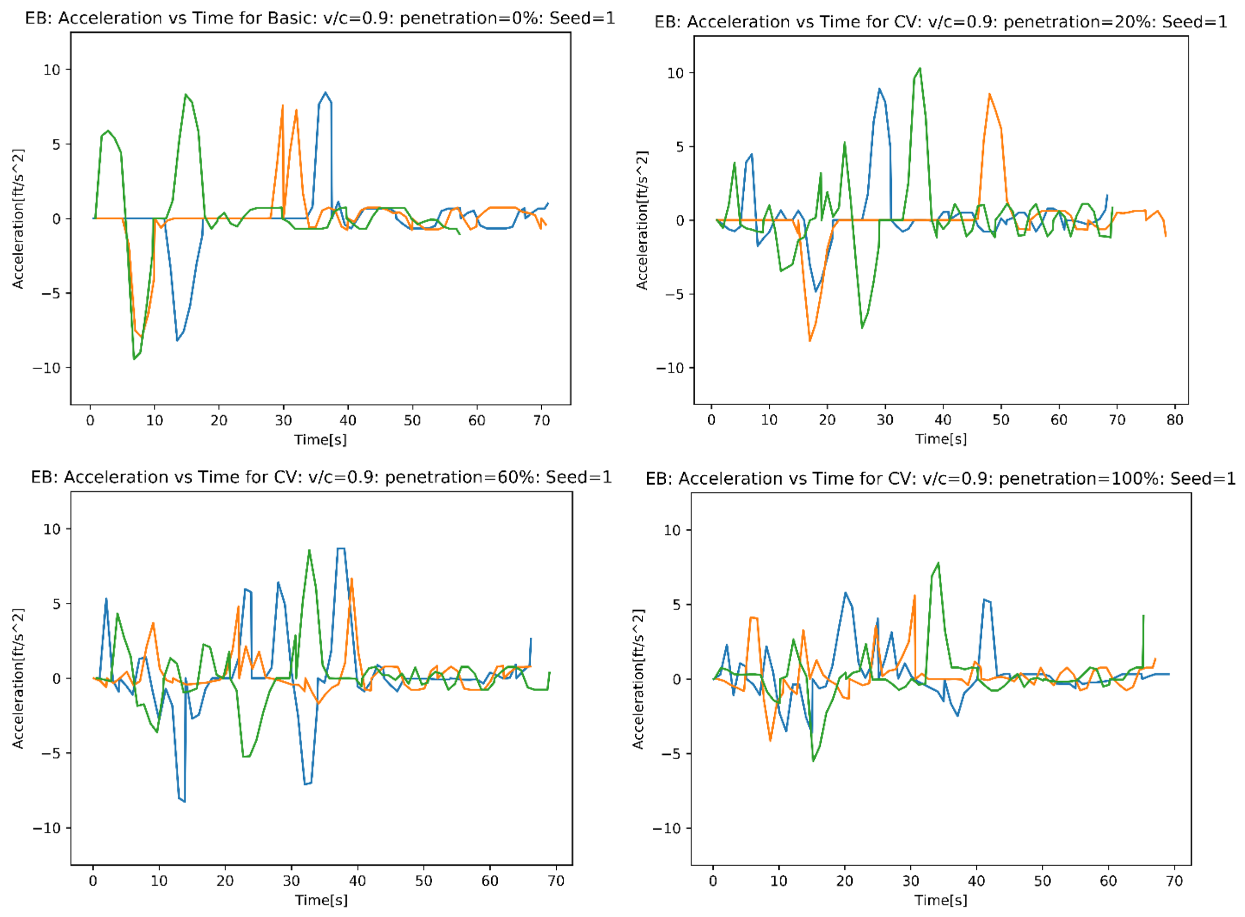


Figure 5-12 Variations in Acceleration for EB-Through Vehicles with Increasing Penetration in CV for v/c=0.9, Simulation Random Seed=0.1

(a) 0% Penetration (top left), (b) 20% Penetration (top right), (c) 60% Penetration (bottom left), and (d) 100% Penetration (bottom right).

5.4.2.2 Emission Analysis: VSP Operating Mode Bin Histogram

The next part of the analysis took a closer look at the “turbulence” in acceleration that was revealed in the acceleration-time plots of individual vehicles. The trends in assignment of vehicle trajectory data to VSP bins as it relates to increases in CV penetration are studied. Figure 5-13 (a, b, c, and d) below show the VSP Bin density histograms for CV penetration rates of 0%, 20%, 60% and 100%, respectively, for v/c=0.9, simulation random seed=1, on the EB route. Since the vehicle speed for the network never exceeded 25 mph, no data-points were observed beyond VSP-Bin 16 (see Table 5-1 for bin definitions). By comparing the plots, it becomes evident that with increasing CV penetration, more data-points appear in Bin-12 (Cruising/Acceleration) and fewer data-points fall in bins 0 (Deceleration), 1 (Idle), and 11 (Coasting). This shows that the CV algorithm has achieved its stated goal; that is, the reduction in idle time shows less time stopped. However, there is an increase in data-points in Bin 12, which has a higher emissions value than Bins 0, 1, and 11, resulting in more overall emissions. Thus, in the current CV logic implemented in VISSIM, for this testbed, with greater CV penetration, vehicles have a lower tendency of being in decelerating/idle/coasting mode and a higher tendency for cruising with mild acceleration, resulting in higher overall emissions. Similar trends are observed for CAV cases as reflected in Figure 5-14 (a, b, c, d).

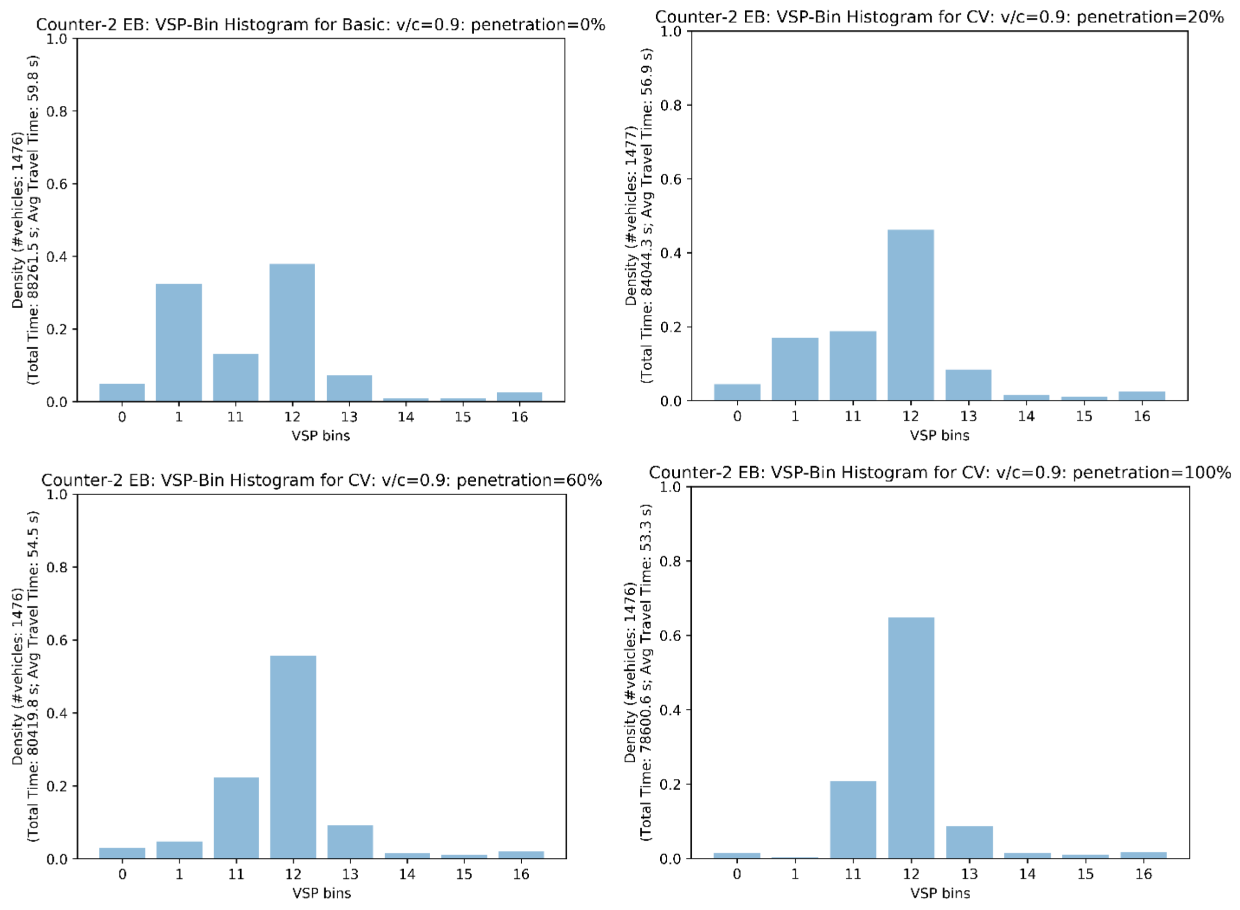


Figure 5-13 Variations in VSP-Bin Distribution for EB-Through Vehicles with Increasing Penetration in CV for v/c=0.9, Simulation Random Seed=0.1

(a) 0% Penetration (top left), (b) 20% Penetration (top right), (c) 60% Penetration (bottom left), and (d) 100% Penetration (bottom right).

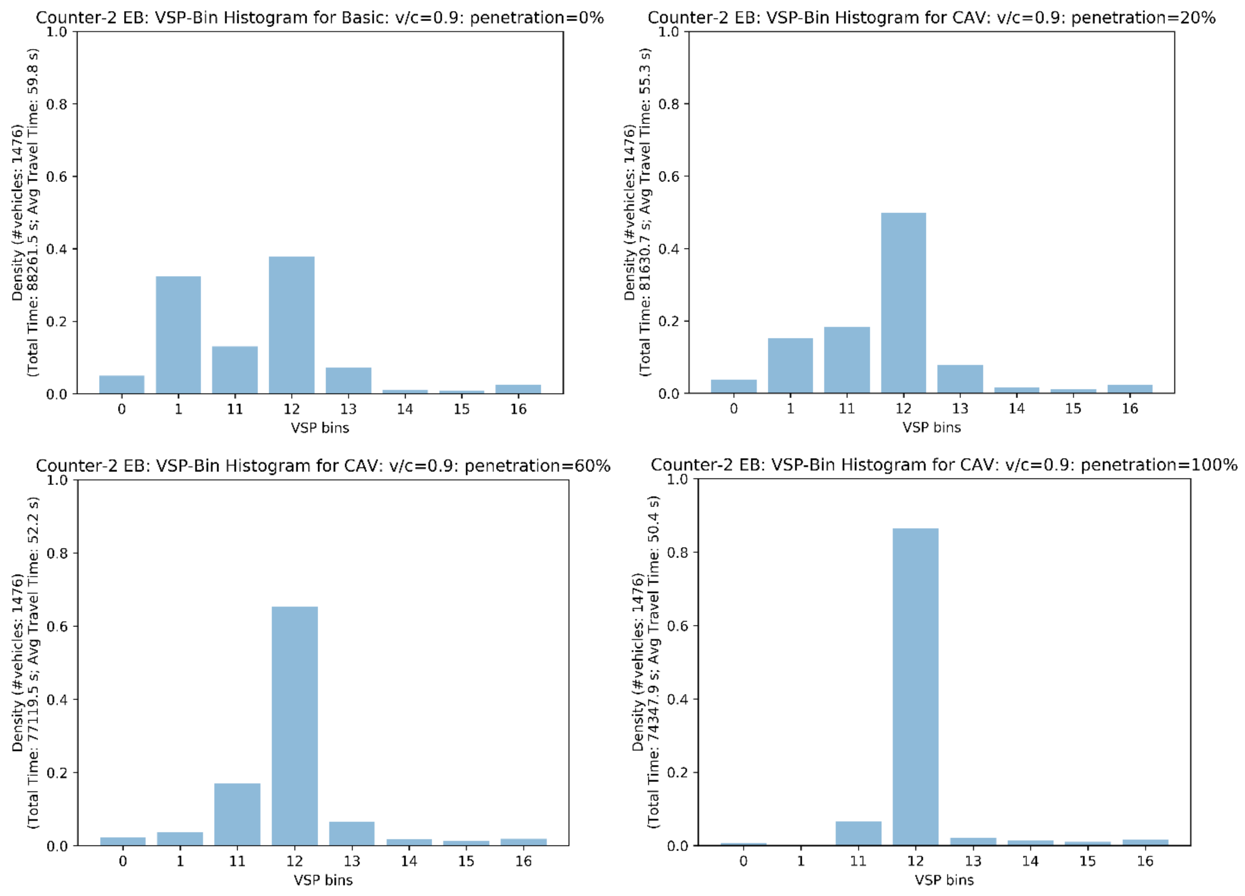


Figure 5-14 Variations in VSP-Bin Distribution for EB-Through Vehicles with Increasing Penetration in CAV for v/c=0.9, Random Seed=0.1

(a) 0% Penetration (top left), (b) 20% Penetration (top right), (c) 60% Penetration (bottom left), (d) 100% Penetration (bottom right).

5.5 Cause of higher emission with higher penetration levels of CV

A deeper analysis into the root cause for these trends showed that while the CV logic chosen for testing in the VISSIM simulation environment seeks to maximize the likelihood of vehicle arrival-on-green, the algorithm likely results in increased variations in second-by-second accelerations, leading to overall higher emissions.

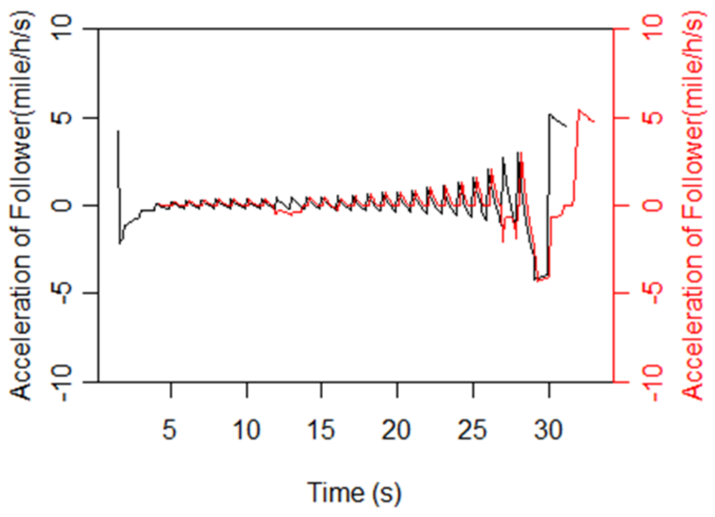
Examining the trajectories for scenarios with higher penetration levels of CV, we found the following reasons these produce higher emissions:

i. *Conventional Vehicle (CNV) following behavior*
When the lead vehicle is a CV and the follower a CNV, the CNV does not have the same information as the CV. Therefore, instead of slowing down to a lower cruising speed, as the CV does (see Figure 4-4 (b)), the CNV aims to achieve its desired speed and enters a sharp oscillation pattern around the lower cruising

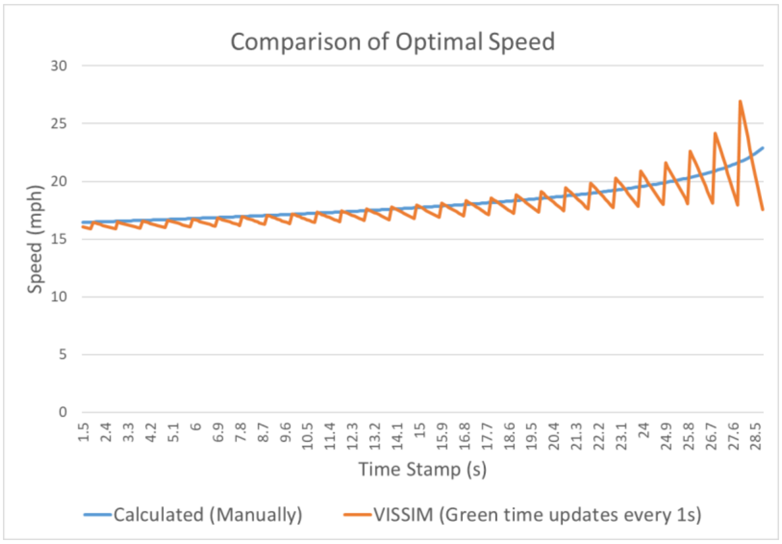
speed of the leader CV. These oscillations likely contribute to increased emissions.

ii. *VISSIM's CV movement logic*

When either the leader or the follower are a CV which enters the communication range when the signal is red, VISSIM's CV logic utilizes a parameter called "SpeedMaxForGreenStart", which is the maximum speed to arrive at the next green start (details of these calculations are provided in sections 4.2.1 and 4.2.2). The calculation of this parameter depends on the green time remaining. While our simulation runs at 0.1s frequency, the COM API updates this parameter (internally) every 1s by default (regardless of simulation frequency set). This leads to an approximation of "green time remaining" to the nearest second. This results in "spikes" every half a second (due to rounding) and results in the oscillation pattern shown in Figure 5-15 (a). This occurs only when the CV arrives during the red signal. The oscillation can be alleviated by calculating the "SpeedMaxForGreenStart" parameter manually according to the chosen simulation frequency (Figure 5-15 (b)). However, for the purposes of demonstration we chose to use VISSIM's CV code as is, and this may be one of the contributing factors to increased emissions with CVs.



(a) Acceleration of Leader- CV and Follower-CV with 30s red remaining (VISSIM's CV logic)



(b) Comparison of VISSIM CV logic and manually estimated speed

Figure 5-15 Optimal Speed provided by VISSIM and Manually Calculated

iii. Artifact of binning
Finally, it is possible that a portion of the results may be an artifact of the binning process used in the MOVES approach. The emissions may be being overestimated if the vehicle operations are occurring at the boundaries of bins 1, 11, and 12, e.g., a minor change in vehicle operations is shifting the emissions calculation from bin 1 to bin 12. Future efforts will explore this potential impact.

6 CONCLUSIONS

This project evaluated the capability of VISSIM to model CAVs and concluded that internal modeling provides limited access to vehicle/driver behavior parameters and cannot model connectivity. Externally, COM API and EDM have powerful features to enable CAV modeling.

While COM API has access to all VISSIM data and is helpful in modeling connectivity, it cannot provide direct and accurate longitudinal and lateral movement control. The EDM enables full control of both longitudinal and lateral movements but with limited accessibility to VISSIM data. Hence, this project developed the ability to simulate CAVs in VISSIM by using COM API to access network elements and EDM to maintain the longitudinal control of vehicles.

The functionality and results of this new procedure is demonstrated by simulating CAVs at a four-legged isolated signalized intersection using VISSIM. A model developed by Talebpour and Mahmassani (2016) was used to replicate the AV logic. The AV logic uses a modified version of the Intelligent Driver Model (IDM) with parameters set based on Van Arem et al. (2006) to represent the response of an AV. VISSIM's I2V application (PTV, 2017) allowing the CVs to access signal timing information was used to replicate the CV logic. This CV logic seeks to maximize the likelihood of arrival-on-green by changing a vehicle's speed within certain parameters.

Several traffic scenarios were simulated (demand levels of v/c=0.7, v/c=0.85 and v/c=0.9, and CV, AV and CAV market penetration rates of 0% to 100%, at 20% increments (each scenario was tested using the same five random seeds). The results show net improvement in traffic operational measures (travel time and speed). CAV, the combination of the two technologies (i.e., autonomy and connectivity) yields better performance than each of the technologies (CV and AV) on their own.

However, emissions did not follow the same trend. While increasing AV penetration rates resulted in emissions reductions, increasing CV and CAV penetration rates resulted in higher emissions. A deeper analysis into the root cause for these trends showed that while VISSIM's CV logic seeks to maximize the likelihood of vehicle arrival-on-green, the algorithm likely results in oscillation of the second-by-second speeds leading to overall higher emissions.

7 RECOMMENDATIONS

The project developed a framework for evaluating AV, CV, and CAV technology in VISSIM. This framework can be used for any model that simulates vehicle movement based on these technologies. While important insights were drawn from this study, they are based on a small highway network. Testing of this framework on a larger arterial network and on freeways would provide additional information, particularly related to network-wide effects from AV, CV, and CAV. In addition, this project focused on specific AV, CV, and CAV algorithms that replicate vehicle movement. Additional analysis should be conducted to evaluate new algorithms as these become available.

In future work, we plan to incorporate into the simulation extension an intersection optimization algorithm for CAVs, developed by the University of Florida. Further, the emissions could be incorporated along with the existing delay- based objectives into the problem formulation of the optimization.

While this report is focused on the integration of CV, AV, and CAV algorithms into a simulation framework, the research team has also considered the application of such a tool to both small- and large-scale transportation challenges. For instance, in sections 4 and 5 the ability of this platform to determine the impact of these technologies on operational and environmental performance is demonstrated for a small facility. Appendix E starts the discussion on a much larger topic, presenting the case for reimaging the entire parcel delivery system within this new technology paradigm. Future efforts will seek to implement and test the proposed delivery system in the developed simulation platform.

Finally, while important insights were drawn from this study the simplicity and limited size of the network provides a challenge in attempting to generalize the findings. In addition, the AV, CV, and CAV findings are limited to the algorithms implemented. A more complex network with improved technology algorithms would allow for a more robust analysis. To apply the tools developed from this project and draw accurate conclusions for a real dataset, calibration and validation would also be needed.

REFERENCE LIST

1.Agency U.S.E.P. (n.d.). MOVES and Other Mobile Source Emissions Models. Retrieved from MOVES and Other Mobile Source Emissions Models

2. Evanson, A. (2017). CONNECTED AUTONOMOUS VEHICLE (CAV) SIMULATION USING PTV VISSIM. *Winter Simulation Conference*, 4220–4227.

3. Fellendorf, M., & Vortisch, P. (2001). Validation of the Microscopic Traffic Flow Model VISSIM in Different Real-World Situations. *Transportation Research Board 80th Annual Meeting*, (January 2001), 1–9. Retrieved from Validation of the Microscopic Traffic Flow Model VISSIM in Different Real-World Situations

4. Guensler, R., Liu, H., Xu, Y., Akanser, A., Kim, D., Hunter, M. P., & Rodgers, M. O. (2017). Energy consumption and emissions modeling of individual vehicles. *Transportation Research Record*, *2627*, 93–102.Energy consumption and emissions modeling of individual vehicles.

5. Numpy. (2020). Numpy Reference. Retrieved from Numpy Referencel

6. PTV- VISSIM. (2016). *WDOT VISSIM Guidance*. 9–12. Retrieved from WDOT VISSIM Guidance. 9–12

7. PTV. (2017). *Connected Autonomous Vehicles Context / Overview*. 1–44.

8. PTV AG. (2005). *VISSIM 4.1 Manual*. Karlrushe, Germany.

9. PTV VISSIM. (2016). *PTV VISSIM 10, Introduction to the COM API*. Germany.

10. Python. (2018). Python 3.7.0. Retrieved from Python 3.7.0.

11. Saroj, A., Roy, S., Guin, A., Hunter, M., & Fujimoto, R. (2019). Smart city real-time data-driven transportation simulation. *Proceedings - Winter Simulation Conference*, *2018-Decem*(RenewAtlantaBond), 857–868. Smart city real-time data-driven transportation simulation

12. Talebpour, A., & Mahmassani, H. S. (2016). Influence of connected and autonomous vehicles on traffic flow stability and throughput. *Transportation Research Part C: Emerging Technologies*, *71*, 143–163. Influence of connected and autonomous vehicles on traffic flow stability and throughput

13. Van Arem, B., Van Driel, C. J. G., & Visser, R. (2006). The impact of cooperative adaptive cruise control on traffic-flow characteristics. *IEEE Transactions on Intelligent*

Transportation Systems, *7*(4), 429–436. The impact of cooperative adaptive cruise control on traffic-flow characteristics

14. Wiedemann, & Reiter. (1991). Microscopic Traffic Simulation: The Simulation System Mission,. In *PTV America, Inc*.

15. Woody. (2006). *Calibrating freeway simulation models in Vissim*. University of Washington.

16. Zhai, H., Frey, H. C., & Rouphail, N. M. (2008). A vehicle-specific power approach to speed- and facility-specific emissions estimates for diesel transit buses. *Environmental Science and Technology*, *42*(21), 7985–7991. A vehicle-specific power approach to speed- and facility-specific emissions estimates for diesel transit buses

APPENDICES

Appendix A: COM API Code

```
import win32com.client as com
import os
import time
import pandas as pd

# determine the type of data to be output
# define the the column names of the output csv file
out_DataType_Traj=['Time Stamp','Vehicle ID', 'Vehicle Type', 'Vehicle
Length','Vehicle Front Coordinate','Vehicle Rear
Coordinate','Speed','Acceleration','Headway']
out_DataType_Signal=['Time Stamp','Signal head
ID','Color','Latitude','Longitude','X','Y']
# the dictionary to change the default output data from VISSIM to required
output data
table_vehicle_type = dict({'100':"Passenger Car",'200':"Passenger
Truck",'300':"Transit Bus",'400':"Tram",'630':"Passenger Car"})
table_signal_color = dict({'RED':0, 'AMBER':1, "GREEN":2})
# variables name to fetch the data from VISSIM
get_DataType_traj=('No', 'VehType','Length',
'CoordFront','CoordRear','Speed','Acceleration','Hdwy')
get_DataType_signal=('No','SigState')
# // specify the VISSIM version here// 1000 means VISSIM 10
VISSIM= com.Dispatch("VISSIM.VISSIM.1000")
#User input---------------------------------------------------
# //input your fold directory which contains the VISSIM file here//
Path_of_COM_Basic_Commands_network = "C:\\Users\\xiduan\\OneDrive -
University of Florida\\UF\\research\\2.VISSIM AV&CV\\network"
# //input your network file and layout file name here//
Network_file='network2.inpx'
Layout_file='network2.layx'
# // write your output fold directory here//
Path_output_file = "C:\\Users\\xiduan\\OneDrive - University of
Florida\\UF\\research\\2.VISSIM AV&CV\\Document\\Task1\\"
# //input your simulation duration here//
simulation_duration=3600
# Load a VISSIM Network:
Filename                 =
os.path.join(Path_of_COM_Basic_Commands_network,Network_file)
flag_read_additionally  = False # you can read network(elements)
additionally, in this case set "flag_read_additionally" to true
VISSIM.LoadNet(Filename, flag_read_additionally)
# Load a Layout:
Filename = os.path.join(Path_of_COM_Basic_Commands_network, Layout_file)
VISSIM.LoadLayout(Filename)

# Set vehicle input:
# // change the vehicles input here//
```

```
VI_number   = 1 # VI = Vehicle Input
new_volume  = 1600 # vehicles per hour
VISSIM.Net.VehicleInputs.ItemByKey(VI_number).SetAttValue('Volume(1)',
new_volume)

# Set vehicle composition:
# //set the vehicles composition here//
Veh_composition_number = 1
Rel_Flows =
VISSIM.Net.VehicleCompositions.ItemByKey(Veh_composition_number).VehCompRelFl
ows.GetAll()
# here the type 630 is the connected vehicles,type 100 is the normal vehicles
Rel_Flows[0].SetAttValue('VehType',        100) # Changing the vehicle type
Rel_Flows[1].SetAttValue('VehType',        630) # Changing the vehicle type
Rel_Flows[0].SetAttValue('DesSpeedDistr',   70) # Changing the desired speed
distribution
Rel_Flows[1].SetAttValue('DesSpeedDistr',   70) # Changing the desired speed
distribution
Rel_Flows[0].SetAttValue('RelFlow',        50) # Changing the relative flow
Rel_Flows[1].SetAttValue('RelFlow',        50) # Changing the relative flow
of the 2nd Relative Flow.

## function to calculate the distance
# a, b are two list contain coordinate like [x,y,z]
def cal_dis(coord1,coord2):
    return ((float(coord1[0])-float(coord2[0]))**2+(float(coord1[1])-
float(coord2[1]))**2)**0.5

## function to find the vehicles ID within certain range/radiums
# Num is the vehicles ID and Radiums is the range
def Vehicle_within(Num, Radiums,add_data):
    current_coord=add_data.loc[add_data['No']==Num,'CoordFront'][0]
    current_coord=current_coord.split()
    vehicle_list=[]
    No_=add_data.loc[add_data['No']!=Num, 'No']
    Coor_=[i.split() for i in add_data.loc[add_data['No']!=Num,
'CoordFront']]
    look_table=dict(zip(No_,Coor_))
    for i in No_:
        if cal_dis(look_table[i],current_coord)<=Radiums:
            vehicle_list.append(i)
    return vehicle_list




#---------------------   for the simualtion
time_step=0
#  time sleep time
time_sleep=0
#create the dataframe for output
dataSet_traj=pd.DataFrame(columns=out_DataType_Traj)
```

```
dataSet_signal=pd.DataFrame(columns=out_DataType_Signal)
# initiate the simulation parameters
time_step=0
time_sleep=0
simulation_duration=3600
## Read the signal coordinate data:
signal_input = pd.read_excel(Path_output_file+"Signal data input.xlsx")
# get the CVs(type 630) data
while time_step<simulation_duration:
    all_veh_attributes =
VISSIM.Net.Vehicles.GetMultipleAttributes((get_DataType_traj))
    select_vehicles=[veh for veh in all_veh_attributes if veh[1]=='630']
# define and output the signal file here:
    add_data_signal = pd.DataFrame(out_DataType_Signal)
    add_data_signal=pd.DataFrame([i for i in
VISSIM.Net.SignalHeads.GetMultipleAttributes(get_DataType_signal)])
    add_data_signal.insert(0,'Time Stamp',time_step)
    add_data_signal=pd.concat([add_data_signal,
signal_input.loc[:,['latitude','longitude','x','y']]],1)
    add_data_signal.columns = out_DataType_Signal
#data conversion: including the unit and type
    add_data_signal.loc[:,'Color'] =
add_data_signal.loc[:,'Color'].replace(table_signal_color)
    add_data_signal.loc[:,'Time Stamp'] = add_data_signal.loc[:,'Time
Stamp']/10
#output
    dataSet_signal=dataSet_signal.append(add_data_signal,ignore_index = True)
    dataSet_signal=dataSet_signal[out_DataType_Signal]
    dataSet_signal.to_csv(Path_output_file+'Signal_data.csv',  index=False )
#output trajectory data here
# check if have the CVs in the network
    if not select_vehicles:
        time_step+=1
        time.sleep(time_sleep)
        VISSIM.Simulation.RunSingleStep()
    else:
# collect and output the trajectory data
        add_data_traj = pd.DataFrame(select_vehicles)
        add_data_traj.insert(0,'Time Stamp',time_step)
        add_data_traj.columns=(out_DataType_Traj)
#data conversion: including the unit and type
        add_data_traj.loc[:,'Vehicle Type']=add_data_traj.loc[:,'Vehicle
Type'].replace(table_vehicle_type)
        add_data_traj.loc[:,'Time Stamp']=add_data_traj.loc[:,'Time
Stamp']/10
        add_data_traj.loc[:,'Acceleration']=
add_data_traj.loc[:,'Acceleration']/1.46667 #feet/second /s  to mile/h/ s
        dataSet_traj=dataSet_traj.append(add_data_traj,ignore_index = True)
        dataSet_traj=dataSet_traj[out_DataType_Traj]
        dataSet_traj.to_csv(Path_output_file+'Trajectory_data.csv',
index=False )
# run the simulation
```

```
        VISSIM.Simulation.RunSingleStep()
        time_step+=1
        # Method #5: Accessing all attributes directly using
"GetMultipleAttributes" (even more faster)
        all_veh_attributes = VISSIM.Net.Vehicles.GetMultipleAttributes(('No',
'VehType', 'acceleration', 'Speed', 'DistanceToSigHead'))
        for cnt in range(len(all_veh_attributes)):
            print ('%s  |  %s  |  %.2f  |  %.2f  |  %s' %
(all_veh_attributes[cnt][0], all_veh_attributes[cnt][1],
all_veh_attributes[cnt][2], all_veh_attributes[cnt][3],
all_veh_attributes[cnt][4])) # only display the 2nd column)
        time.sleep(time_sleep)
```

Code to retrieve vehicles within specified radius

```
## function to calculate the distance
# a, b are two list contain coordinate like [x,y,z]
def cal_dis(coord1,coord2):
    return ((float(coord1[0])-float(coord2[0]))**2+(float(coord1[1])-
float(coord2[1]))**2)**0.5

## function to find the vehicles ID within certain range/radiums
# Num is the vehicles ID and Radiums is the range
def Vehicle_within(Num, Radiums,add_data):
    current_coord=add_data.loc[add_data['No']==Num,'CoordFront'][0]
    current_coord=current_coord.split()
    vehicle_list=[]
    No_=add_data.loc[add_data['No']!=Num, 'No']
    Coor_=[i.split() for i in add_data.loc[add_data['No']!=Num,
'CoordFront']]
    look_table=dict(zip(No_,Coor_))
    for i in No_:
        if cal_dis(look_table[i],current_coord)<=Radiums:
            vehicle_list.append(i)
    return vehicle_list
```

Appendix B: Speed and Delay Results by Scenario

Average speed (mile/h) for AV simulation

v/c	penetration rate	runs	mean	SD	min	max
0.7	0	5	14.19	0.14	14.01	14.38
	20	5	14.42	0.12	14.23	14.53
	40	5	14.61	0.12	14.41	14.71
	60	5	14.76	0.14	14.52	14.88
	80	5	14.88	0.15	14.64	15.04
	100	5	14.99	0.14	14.78	15.18
0.85	0	5	13.08	0.06	13.00	13.17
	20	5	13.46	0.11	13.33	13.64
	40	5	13.74	0.10	13.64	13.89
	60	5	13.98	0.09	13.89	14.11
	80	5	14.17	0.11	14.08	14.34
	100	5	14.31	0.13	14.19	14.51
0.9	0	5	11.50	0.12	11.33	11.67
	20	5	11.89	0.11	11.71	12.01
	40	5	12.20	0.08	12.07	12.30
	60	5	12.51	0.07	12.43	12.58
	80	5	12.74	0.05	12.70	12.82
	100	5	12.98	0.04	12.94	13.04

Average speed (mile/h) for CV simulation

v/c	penetration rate	runs	mean	SD	min	max
0.7	0	5	14.19	0.14	14.01	14.38
	20	5	14.38	0.13	14.18	14.51
	40	5	14.53	0.10	14.36	14.62
	60	5	14.65	0.10	14.50	14.75
	80	5	14.76	0.10	14.61	14.85
	100	5	14.85	0.10	14.69	14.97
0.85	0	5	13.08	0.06	13.00	13.17
	20	5	13.43	0.08	13.35	13.53
	40	5	13.67	0.07	13.60	13.76
	60	5	13.84	0.10	13.72	13.97
	80	5	14.03	0.09	13.92	14.15
	100	5	14.08	0.10	13.95	14.22
0.9	0	5	11.50	0.12	11.33	11.67
	20	5	12.17	0.11	12.01	12.30
	40	5	12.64	0.07	12.53	12.72
	60	5	12.91	0.07	12.80	12.95

v/c	penetration rate	runs	mean	SD	min	max
	80	5	13.13	0.09	12.99	13.23
	100	5	13.31	0.10	13.17	13.39

Average speed (mile/h) for CAV simulation

v/c	penetration rate	runs	mean	SD	min	max
0.7	**0**	5	14.19	0.14	14.01	14.38
	20	5	14.66	0.08	14.53	14.73
	40	5	14.97	0.10	14.82	15.10
	60	5	15.23	0.09	15.08	15.34
	80	5	15.44	0.11	15.27	15.55
	100	5	15.67	0.11	15.51	15.80
0.85	**0**	5	13.08	0.06	13.00	13.17
	20	5	13.83	0.12	13.68	13.99
	40	5	14.25	0.10	14.12	14.38
	60	5	14.58	0.09	14.51	14.74
	80	5	14.89	0.11	14.79	15.07
	100	5	15.11	0.14	14.98	15.32
0.9	**0**	5	11.50	0.12	11.33	11.67
	20	5	12.58	0.06	12.52	12.64
	40	5	13.17	0.07	13.08	13.26
	60	5	13.55	0.06	13.48	13.64
	80	5	13.84	0.06	13.77	13.92
	100	5	14.11	0.05	14.06	14.18

Average delay (s/veh) for AV simulation

v/c	penetration rate	runs	mean	SD	min	max
0.7	**0**	5	13.23	0.44	12.62	13.74
	20	5	12.48	0.37	12.14	13.07
	40	5	11.89	0.37	11.52	12.49
	60	5	11.45	0.42	11.12	12.15
	80	5	11.08	0.42	10.67	11.77
	100	5	10.77	0.39	10.29	11.37
0.85	**0**	5	17.17	0.21	16.93	17.44
	20	5	15.72	0.36	15.18	16.16
	40	5	14.71	0.31	14.28	15.04
	60	5	13.90	0.26	13.53	14.19
	80	5	13.26	0.30	12.80	13.57
	100	5	12.81	0.37	12.28	13.21
0.9	**0**	5	24.02	0.63	23.20	24.94
	20	5	22.09	0.53	21.59	22.97
	40	5	20.65	0.42	20.20	21.31
	60	5	19.33	0.31	18.96	19.74

	80	5	18.37	0.26	17.98	18.63
	100	5	17.45	0.20	17.13	17.67

Average delay (s/veh) for CV simulation

v/c	penetration rate	runs	mean	SD	min	max
0.7	**0**	5	13.23	0.44	12.62	13.74
	20	5	12.63	0.43	12.21	13.24
	40	5	12.17	0.32	11.82	12.66
	60	5	11.81	0.32	11.44	12.24
	80	5	11.47	0.30	11.21	11.90
	100	5	11.22	0.31	10.85	11.65
0.85	**0**	5	17.17	0.21	16.93	17.44
	20	5	15.88	0.34	15.41	16.24
	40	5	15.05	0.28	14.61	15.30
	60	5	14.47	0.35	14.08	14.89
	80	5	13.84	0.30	13.47	14.22
	100	5	13.68	0.37	13.26	14.09
0.9	**0**	5	24.02	0.63	23.20	24.94
	20	5	21.00	0.46	20.35	21.59
	40	5	19.06	0.40	18.65	19.70
	60	5	18.00	0.41	17.75	18.73
	80	5	17.15	0.49	16.76	17.97
	100	5	16.50	0.50	16.16	17.28

Average delay (s/veh) for CAV simulation

v/c	penetration rate	runs	mean	SD	min	max
0.7	**0**	5	13.23	0.44	12.62	13.74
	20	5	11.76	0.24	11.48	12.11
	40	5	10.84	0.30	10.40	11.24
	60	5	10.10	0.27	9.75	10.50
	80	5	9.51	0.28	9.18	9.94
	100	5	8.90	0.26	8.60	9.29
0.85	**0**	5	17.17	0.21	16.93	17.44
	20	5	14.44	0.39	13.99	14.93
	40	5	13.02	0.29	12.69	13.44
	60	5	11.99	0.25	11.58	12.22
	80	5	11.08	0.29	10.61	11.36
	100	5	10.42	0.35	9.91	10.77
0.9	**0**	5	24.02	0.63	23.20	24.94
	20	5	19.15	0.26	18.87	19.42
	40	5	16.83	0.29	16.44	17.27
	60	5	15.38	0.23	15.08	15.71

	80	5	14.38	0.19	14.14	14.60
	100	5	13.46	0.16	13.26	13.68

Appendix C: COM API Code written in Python 3.0

```
import win32com.client as com
import os
from data_io import msgManager
import time
import pandas as pd
import auModel
import argparse
#--
# os.chdir(r"C:\Users\essie-adm-luan\Downloads\CAV-simulation-xixi\CAV-simulation-
xixi");
# LC: saves you the effort of updating the working directory
currentDir = os.path.abspath(os.path.dirname(__file__))
os.chdir(currentDir)
#---simulation scenarios
# 1 0.8 0.6 0.4 0.2
# 1 0.8 0.6 0.4 0.2
# 0.8 0.6 0.4 0.2
Random_Seed = 887


# vc ratio 0.9,0.85,0.7
vc = [0.85, 0.7]
#penetration 0.0001 0.2 0.4 0.6 0.8 1.0
penetration = [ 0.2, 0.4, 0.6, 0.8, 1.0]


for kk0 in vc:
    for kk in penetration:
        print(kk)
        vc_ratio = kk0
        penetration_rate = kk
        network_folder = "network3 - vc" + " " + str(vc_ratio) + " " +
str(int(penetration_rate*100)) + "%"
        #network_folder = "network - vc 0.9 80% - basic"
        #--------------------------------------------------------------------------
-------------------------
        parser = argparse.ArgumentParser()

        # directory setting
        parser.add_argument('--path_output_file', default =
os.path.join(currentDir,"output"), help = "the directory of output file, the default
is the current working directory ")
        parser.add_argument('--path_of_network', default = os.path.join(currentDir,
network_folder), help = "the directory of network file, including .inpx and .layx")

        # simulation parameters setting
        parser.add_argument('--version_VISSIM', default = "VISSIM.VISSIM.1000" , help
= "1000 means VISSIM 10")
        parser.add_argument('--network_file', default = "network2.inpx" , help = "the
name of VISSIM network file")
        parser.add_argument('--layout_file', default = "network2.layx" , help = "the
name of VISSIM layout file")
```

```
        parser.add_argument('--duration', default = 36000 ,type = int, help =
"simulation duration, unit in sim second")
        parser.add_argument('--time_sleep', default = 0 ,type = float, help = "the
sleep time between each time step of simulation")
        # simulation traffic input setting
        parser.add_argument('--sb_volume', default = 1500 ,type = int, help = "traffic
volume of southbound, vehicles / hr")
        parser.add_argument('--nb_volume', default = 1500 ,type = int, help = "traffic
volume of northbound, vehicles / hr")
        parser.add_argument('--wb_volume', default = 1500 ,type = int, help = "traffic
volume of westbound, vehicles / hr")
        parser.add_argument('--eb_volume', default = 1500 ,type = int, help = "traffic
volume of eastbound, vehicles / hr")
        parser.add_argument('--CAV_rel_flow', default = 0.2 ,type = float, help = "the
percentage of CAV volume")
        parser.add_argument('--Desired_speed', default = 32 ,type = float, help = "the
desired speed km/h")
        parser.add_argument('--minSpeed', default = 5 ,type = float, help = "the
minimal speed for AV km/h")


        # traffic model setting
        parser.add_argument('--max_dec', default = -5.0 ,type = float, help = "the
maximum deceleration, m / s^2")
        parser.add_argument('--comf_dec', default = -3.5 ,type = float, help = "the
comfortable deceleration, m / s^2")
        parser.add_argument('--max_acc', default = 3 ,type = float, help = "the
maximum acceleration, m / s^2")
        parser.add_argument('--comf_acc', default = 2.5 ,type = float, help = "the
comfortable acceleration, m / s^2")
        parser.add_argument('--desired_speed_change_rate', default = 1 ,type = float,
help = "used to define how driver would change to desired speed")
        simulation = parser.parse_args()



        left_lane = ['10012-1','10015-1','10011-1','10008-1','10003-1','10006-
1','10010-1','10001-1']



        # Loads the message manager that will send and receive the messages
        msgCfgFile = os.path.join(currentDir, "data_io_config",
"Local_RIO_VISSIM.yml")
        messageManager = msgManager(msgCfgFile)

        sendMessage = False # this is to avoid breaking the code for now.
        f = open(simulation.path_output_file + "\\Trajectory.txt", "w")
        f.close()



        # determine the type of data to be output ------------------------------------
--
        # define the the column names of the output csv file

        out_DataType_Traj = ['Time Stamp','Vehicle ID', 'Vehicle Type', 'Vehicle
Length','Vehicle Front Coordinate','Vehicle Rear
Coordinate','Speed','Acceleration','Headway','Lane']
        # data type for signal output
        out_DataType_Signal = ['Time Stamp','Signal head
ID','Color','Latitude','Longitude','X','Y']
```

```
        # data type for trajectory output
        out_DataType_Traj_em = ['Time Stamp','Vehicle ID', 'Vehicle Type', 'Vehicle 
Length','Vehicle Front Coordinate','Vehicle Rear 
Coordinate','Speed','Acceleration','Headway']
        # data type for CV (RIO) output
        out_DataType_CVModel = ["No", "Pos", "Speed","pos_rms", "speed_rms", 
"veh_type", "veh_length", "max_accel", "max_decel"]
        # the dictionary to change the default output data from VISSIM to required 
output data
        table_vehicle_type = dict({'100':"Passenger Car",'200':"Passenger 
Truck",'300':"Transit Bus",'400':"Tram",'630':"Passenger Car"})
        table_signal_color = dict({'RED':0, 'AMBER':1, "GREEN":2})
        # variables name to fetch the data from VISSIM
        get_DataType_traj=('No', 'VehType','Length', 
'CoordFront','CoordRear','Speed','Acceleration','Hdwy', 'Lane' )
        get_DataType_signal=('No','SigState')
        # // specify the VISSIM version here// 1000 means VISSIM 10
        VISSIM= com.Dispatch(simulation.version_VISSIM)

        #// input the lanes number and their corresponding start point's coordinate
        '''
        Lane 4: Eastbound,  (-181, -22)
        Lane 3: Northbound  (-117, -141)
        Lane 1: Westbound   (60, -1)
        Lane 2: Southbound  (-165, 131)

        '''
        eb_coor = (-181, -22)
        nb_coor = (-117, -141)
        wb_coor = (60, -1)
        sb_coor = (165, 131)
        eb_lane = 3
        nb_lane = 4
        wb_lane =1
        sb_lane = 2
        lane_to_cor = {eb_lane : eb_coor, nb_lane :  nb_coor, wb_lane: wb_coor, 
sb_lane : sb_coor }
        store_speed = pd.DataFrame(columns = ["Vehicle ID","x speed", "y speed"])
        vehicle_type_dict = {"Passenger Car" : 0}

        #------------------------------------------------------------
        ## Load a VISSIM Network:
        Filename                =
os.path.join(simulation.path_of_network,simulation.network_file)
        flag_read_additionally  = False # you can read network(elements) additionally, 
in this case set "flag_read_additionally" to true
        VISSIM.LoadNet(Filename, flag_read_additionally)
        ## Load a Layout:
        Filename = os.path.join(simulation.path_of_network,simulation.layout_file)
        VISSIM.LoadLayout(Filename)
        CAV = '650'
        TRA_car = '100'
        # Set vehicle input:
        # // change the vehicles input here//

#===============================================================================
        # Desired_speed = 20
        # cav_flow_rate = simulation.CAV_rel_flow
        # VISSIM.Net.VehicleInputs.ItemByKey(1).SetAttValue('Volume(1)', 
simulation.eb_volume)
        # VISSIM.Net.VehicleInputs.ItemByKey(2).SetAttValue('Volume(1)', 
simulation.nb_volume)
```

```
        # VISSIM.Net.VehicleInputs.ItemByKey(3).SetAttValue('Volume(1)',
simulation.wb_volume)
        # VISSIM.Net.VehicleInputs.ItemByKey(4).SetAttValue('Volume(1)',
simulation.sb_volume)
        # # Set vehicle composition:

#===============================================================================
          # //set the vehicles composition here//
        VISSIM.Simulation.SetAttValue('RandSeed', Random_Seed)
        Veh_composition_number = 1
        Rel_Flows =
VISSIM.Net.VehicleCompositions.ItemByKey(Veh_composition_number).VehCompRelFlows.GetAl
l()
          # here the type 630 is the connected vehicles,type 100 is the normal
vehicles
        conventional_flow =  (1- penetration_rate) * 100 if penetration_rate != 1 else
0.001
        Rel_Flows[0].SetAttValue('RelFlow',        conventional_flow) # Changing the
relative flow
        Rel_Flows[1].SetAttValue('RelFlow',        penetration_rate *100) # Changing
the relative flow of the 2nd Relative Flow.

        #

#===============================================================================
        ## function to calculate the distance
        # a, b are two list contain coordinate like [x,y,z]
        def cal_dis(coord1,coord2):
            return ((float(coord1[0])-float(coord2[0]))**2+(float(coord1[1])-
float(coord2[1]))**2)**0.5

        ## function to find the vehicles ID within certain range/radiums
        # Num is the vehicles ID and Radiums is the range
        def Vehicle_within(Num, Radiums,add_data):
            current_coord=add_data.loc[add_data['No']==Num,'CoordFront'][0]
            current_coord=current_coord.split()
            vehicle_list=[]
            No_=add_data.loc[add_data['No']!=Num, 'No']
            Coor_=[i.split() for i in add_data.loc[add_data['No']!=Num, 'CoordFront']]
            look_table=dict(zip(No_,Coor_))
            for i in No_:
                if cal_dis(look_table[i],current_coord)<=Radiums:
                    vehicle_list.append(i)
            return vehicle_list

        def leading(lead_vehicle_id,data_set, start ):
            current_index = start
            while current_index >= 0:
                item = data_set[current_index]
                if item[0] == lead_vehicle_id:
                    return [item[2],item[3],item[4]]
                current_index -= 1
            return False




        ##---------------------------------------------------   CV coding part

        def toList(NestedTuple):
            # function to convert a nested tuple to a nested list
            return list(map(toList, NestedTuple)) if isinstance(NestedTuple, (list,
tuple)) else NestedTuple
```

```
        def Init():
            # Initialisation.
            # add global variables
            global minSpeed
            global vehTypesEquipped
            global vehsAttributes
            global vehsAttNames
            # read the minimum Speed from the script UDA
            minSpeed = simulation.minSpeed

            vehsAttributes = []
            vehsAttNames = []

            # read which vehicle types are able to receive the signal information and
being able to adjust their speed.
            vehTypesAttributes = VISSIM.Net.VehicleTypes.GetMultipleAttributes(['No',
'ReceiveSignalInformation'])
            vehTypesEquipped = [str(x[0]) for x in vehTypesAttributes if x[1] == True]
# list of vehicle types which are able to adjust their speed, e.g. [102, 103]

        def OptimalSpeedMin(minSpeed, desSpeed):
            # A minimum speed is required to arrive during the current green.
            if minSpeed < desSpeed: # check if the desired speed is higher then the
minimum speed
                # keep desired speed because it is faster => the vehicle will arrive
at the signal within the current green
                optimalSpeed = desSpeed
            else:
                optimalSpeed = -1 # no optimal speed in case the desired speed is
larger or equal the required minimum speed
            return optimalSpeed

        def OptimalSpeedMax(maxSpeed, desSpeed):
            # The vehicle should not drive above the maximum speed in order to arrive
just when the next green starts.
            if maxSpeed > desSpeed: # check if the maximum speed is higher then the
desired speed
                # keep desired speed because the desired speed to lower than the
maximum speed => the vehicle will arrive after the signal turned green
                optimalSpeed = desSpeed
            else:
                optimalSpeed = maxSpeed # optimal speed for arriving at the next green
            return optimalSpeed
        # equivalent to # return min(maxSpeed, desSpeed)

        def GetVISSIMDataVehicles():
            # this function reads vehicle attributes from PTV VISSIM
            global vehsAttributes
            global vehsAttNames
            vehsAttributesNames = ['No', 'VehType', 'Lane', 'DesSpeed', 'OrgDesSpeed',
'DistanceToSigHead', 'SpeedMaxForGreenStart', 'SpeedMinForGreenEnd','Speed']
            vehsAttributes =
toList(VISSIM.Net.Vehicles.GetMultipleAttributes(vehsAttributesNames))

            # create dictionary for the attribute names read from PTV VISSIM:
            vehsAttNames = {}
            cnt = 0
            for att in vehsAttributesNames:
                vehsAttNames.update({att: cnt})
                cnt += 1
        def ChangeSpeed():
            diffSpeed = 2 # keep speed a little smaller so that vehicle arrive shortly
before the signal head.
            GetVISSIMDataVehicles() # read vehicle attributes from PTV VISSIM to
global variable "vehsAttributes"
            acc_cv = []
            ID_cv = []
            if len(vehsAttributes) > 0: # if there are any vehicles in the network
                for vehAttributes in vehsAttributes: # loop over all vehicles in the
network
                    if vehAttributes[vehsAttNames['VehType']] in vehTypesEquipped: #
check if vehicle is able to receive signal information
                        if vehAttributes[vehsAttNames['Lane']] in left_lane:
                            continue
                        # set easier variables of the current vehicle:
                        DesSpeed = vehAttributes[vehsAttNames['DesSpeed']]
                        OrgDesSpeed = vehAttributes[vehsAttNames['OrgDesSpeed']]
                        DistanceToSigHead =
vehAttributes[vehsAttNames['DistanceToSigHead']]
                        SpeedMaxForGreenStart =
vehAttributes[vehsAttNames['SpeedMaxForGreenStart']] # Maximum speed to arrive at the
next green start. If the vehicle drives faster it would arrive at the signal before
the next green time.
                        SpeedMinForGreenEnd =
vehAttributes[vehsAttNames['SpeedMinForGreenEnd']] # Minimum speed to arrive at the
next green end. If the vehicle drives slower, it would not make it in the current /
next green time.

                        if OrgDesSpeed == None: # if the original desired speed has
not yet saved, save it to the UDA "OrgDesSpeed"
                            OrgDesSpeed = DesSpeed
                            vehAttributes[vehsAttNames['OrgDesSpeed']] = DesSpeed #
OrgDesSpeed = DesSpeed;  save original desired speed

                        # if the vehicle does not have a upcoming signal: set original
desired speed
                        if DistanceToSigHead <= 0:
                            vehAttributes[vehsAttNames['DesSpeed']] = OrgDesSpeed #
DesSpeed = OrgDesSpeed
                            continue # jump to next vehicle

                        #---------------------------------------
                        #  Decide about the optimal speed      |
                        #---------------------------------------
                        if SpeedMinForGreenEnd > SpeedMaxForGreenStart:
                            # The minimum speed for arriving before the next green end
is higher than the maximum speed to arriving after the next green start. This is only
possible in case the next signal is green.
                                 # > there is green ahead!
                            optimalSpeed= OptimalSpeedMin(SpeedMinForGreenEnd,
OrgDesSpeed)
                            if optimalSpeed == -1:
             # check if no optimal speed in case the desired speed is larger or equal
the required minimum speed
                                optimalSpeed= OptimalSpeedMax(SpeedMaxForGreenStart,
OrgDesSpeed)
                        else:
                                 # There is red light ahead!
                                 # Use maximum speed:
                            optimalSpeed = max(min(SpeedMaxForGreenStart, OrgDesSpeed)
- diffSpeed, minSpeed)
```

```
                        vehAttributes[vehsAttNames['DesSpeed']] = optimalSpeed # set
optimal speed to the vehicles attributes
                        acc_cv.append((optimalSpeed -
vehAttributes[vehsAttNames['Speed']] ) / 3.6 * simulation.desired_speed_change_rate)
                        ID_cv.append(vehAttributes[vehsAttNames['No']])
                #-------------------------------------------------------------------------
-------
                #  After iterating though all vehicles, update the speeds in PTV
VISSIM     |
                #-------------------------------------------------------------------------
-------

            return ID_cv, acc_cv



        ##----------------------  setting for the simualtion
        start_time = time.time()
        traj_file = pd.DataFrame(columns = out_DataType_Traj_em)
        traj_file.to_csv(simulation.path_output_file + "\\" + network_folder
+"Trajectory_data.csv",  index = None,  mode='w')
        signal_file = pd.DataFrame(columns = out_DataType_Signal)
        signal_file.to_csv(simulation.path_output_file + "\\" + network_folder +
"Signal_data.csv",  index = None,  mode='w')
        log_col = ['time','No', 'VehType', 'pos','acceleration',' speed','leading
pos', 'leading speed', 'leading acc','leading
length','headway','Lane','design_acceleration', 'state']
        log_file = pd.DataFrame(columns = log_col)
        log_file.to_csv(simulation.path_output_file + "\\" + network_folder +
"log.csv",  index = None,  mode='w')
        #create the dataframe for output
        CV_data = pd.DataFrame(columns = out_DataType_CVModel)


        # initiate the simulation parameters
        Init()
        # initial the cv
        time_step=0
        pre_data_traj = pd.DataFrame(columns = out_DataType_Traj)
        speed_vehicle = pd.DataFrame(columns = ["Vehicle ID", "Split Speed"])
        ## Read the signal coordinate data:
        inputpath = simulation.path_output_file + "\\Signal_data.txt"
        signal_input = pd.read_table(inputpath)
        buffer =  2
        #------------------------------------- start simulation
        #  Random_Seed = 42
        #  VISSIM.Simulation.SetAttValue('RandSeed', Random_Seed)
        #


        def merge_array(ID_1, ACC_1, ID_2, ACC_2):
            if len(ID_1) == 0:
                return ID_2, ACC_2
            if len(ID_2) == 0:
                return ID_1, ACC_1
            dict1 = {ID_1[i]:ACC_1[i] for i in range(len(ID_1))}
            for i in range(len(ID_2)):
                key = ID_2[i]
                if key in dict1:
                    dict1[key] = min(ACC_2[i], dict1[key])
                else:
                    dict1[key] = ACC_2[i]
            return list(dict1.keys()), list(dict1.values())
        VISSIM.Simulation.RunSingleStep()
        while time_step <= simulation.duration:
            all_veh_attributes =
VISSIM.Net.Vehicles.GetMultipleAttributes((get_DataType_traj))
        # select_vehicles,add_data_traj, dataSet_traj  are for emission model purpose
            all_veh_attributes=[veh for veh in all_veh_attributes]
        # define and output the signal file here:
            add_data_signal = pd.DataFrame(out_DataType_Signal)
            add_data_signal = pd.DataFrame([i for i in
VISSIM.Net.SignalHeads.GetMultipleAttributes(get_DataType_signal)])
            add_data_signal.insert(0,'Time Stamp',time_step)
            add_data_signal = pd.concat([add_data_signal,
signal_input.loc[:,['latitude','longitude','x','y']]],1)
            add_data_signal.columns = out_DataType_Signal
        #data conversion: including the unit and type
            add_data_signal.loc[:,'Color'] =
add_data_signal.loc[:,'Color'].replace(table_signal_color)
            add_data_signal.loc[:,'Time Stamp'] = add_data_signal.loc[:,'Time
Stamp']/10
        #output
            add_data_signal.to_csv(simulation.path_output_file + "\\" + network_folder
+ 'Signal_data.csv',   header = None,  index = None,  mode='a' )
        #output trajectory data here
        # check if have the CVs in the network
            vehicle_ID = []
            vehicle_acc = []
            if not all_veh_attributes:
                print("no vehicles")
            else:
        # collect and output the trajectory data
                vehicle_ID2, vehicle_acc2 = ChangeSpeed()
                add_data_traj = pd.DataFrame(all_veh_attributes)
                add_data_traj.insert(0,'Time Stamp',time_step)
                add_data_traj.columns=(out_DataType_Traj)
        # log file initilization:
                log_file = pd.DataFrame(columns = log_col)
        #data conversion: including the unit and type
                add_data_traj.loc[:,'Vehicle Type']=add_data_traj.loc[:,'Vehicle
Type'].replace(table_vehicle_type)
                add_data_traj.loc[:,'Time Stamp']=add_data_traj.loc[:,'Time Stamp']/10
                add_data_traj.loc[:,'Speed']=add_data_traj.loc[:,'Speed'] / 1.6 # km/h
to mile/h
                add_data_traj.loc[:,'Acceleration']=
add_data_traj.loc[:,'Acceleration'] * 2.237 #meter/second /s  to mile/h/ s
                add_data_traj.loc[:,'Vehicle Length']= add_data_traj.loc[:,'Vehicle
Length'] * 3.28084 #meter  to feet
                add_data_traj.loc[:,'Headway']= add_data_traj.loc[:,'Headway'] *
3.28084 #meter  to feet
        # calculate the lateral and longitudinal speed


#================================================================================
        #         speed_cal_curr = add_data_traj.loc[:,["Vehicle ID" , "Vehicle Front
Coordinate", "Lane"]]
        #         speed_cal_prev = pre_data_traj.loc[:,["Vehicle ID" , "Vehicle Front
Coordinate", "Lane"]]
        #         speed_cal_curr["x_coor_curr"] = [float(speed_cal_curr.loc[:,
"Vehicle Front Coordinate"][i].split()[0]) for i in range(speed_cal_curr.shape[0]) ]
        #         speed_cal_curr["y_coor_curr"] = [float(speed_cal_curr.loc[:,
"Vehicle Front Coordinate"][i].split()[1]) for i in range(speed_cal_curr.shape[0]) ]
```

```
        #         speed_cal_prev["x_coor_prev"] = [float(speed_cal_prev.loc[:,
"Vehicle Front Coordinate"][i].split()[0]) for i in range(speed_cal_prev.shape[0]) ]
        #         speed_cal_prev["y_coor_prev"] = [float(speed_cal_prev.loc[:,
"Vehicle Front Coordinate"][i].split()[1]) for i in range(speed_cal_prev.shape[0]) ]
        #         speed_data = pd.merge(speed_cal_curr,speed_cal_prev,   on = "Vehicle
ID", how = "left")
        # # check if there are any vehicles just join the network, which does not have
prev_coor values
        #         if (speed_data.loc[:, "x_coor_prev"].isnull()).any() :
        #             for i in speed_data.loc[speed_data.loc[:,
"x_coor_prev"].isnull()].index:
        #                  lane_num = int(speed_data.loc[i, "Lane_x"].split('-')[0])
        #                  speed_data.loc[i, "x_coor_prev"] = lane_to_cor[lane_num][0]
        #                  speed_data.loc[i, "y_coor_prev"] = lane_to_cor[lane_num][1]
        #
        #         speed_x = speed_data.loc[:, "x_coor_curr"] - speed_data.loc[:,
"x_coor_prev"]
        #         speed_y = speed_data.loc[:, "y_coor_curr"] - speed_data.loc[:,
"y_coor_prev"]
        #         speed_coor = [[speed_x[i], speed_y[i]] for i in range(len(speed_x))]
        #         add_data_traj["speed_coor"] = speed_coor
        #         CV_data = add_data_traj.loc[:, ["Vehicle ID", "Vehicle Front
Coordinate","speed_coor", "nan", "nan", "Vehicle Type", "Vehicle Length"] ]
        #         CV_data["max_acc"] = simulation.max_acc
        #         CV_data["max_decce"] = simulation.max_dec
        #         CV_data["Vehicle Front Coordinate"] = [ i.split()[0:2] for i in
(CV_data.loc[:, "Vehicle Front Coordinate"])]
        #         CV_data.replace(vehicle_type_dict, inplace = True)
        #         pre_data_traj = add_data_traj

#================================================================================

        #--------------------------------------------------------------------------------
-----
                add_data_traj = add_data_traj[out_DataType_Traj_em]
                add_data_traj.to_csv(simulation.path_output_file + "\\" +
network_folder + 'Trajectory_data.csv',  header = None,  index = None,  mode='a' )

        # run the simulation

        # Method #5: Accessing all attributes directly using "GetMultipleAttributes"
(even more faster)
        # all_veh_attributes are for printing purpose
                all_veh_attributes = VISSIM.Net.Vehicles.GetMultipleAttributes(('No',
'VehType', 'acceleration', 'Speed', 'DistanceToSigHead'))
                for cnt in range(len(all_veh_attributes)):
                    print ('%.f | %s  |  %s  |  %.2f  |  %.2f  |  %s' %
(time_step,all_veh_attributes[cnt][0], all_veh_attributes[cnt][1],
all_veh_attributes[cnt][2], all_veh_attributes[cnt][3], all_veh_attributes[cnt][4])) #
only display the 2nd column)
        ## data_vehicles are for AV/CV modeling purpose
                data_vehicles = VISSIM.Net.Vehicles.GetMultipleAttributes(('No',
'VehType', 'length','Acceleration','Speed', 'Pos',
'Hdwy','LeadTargNo','LeadTargType','DistanceToSigHead', 'SignalState','Lane'))
                dist_stop = (simulation.Desired_speed / 3.6) **2 / 2 / -
(simulation.comf_dec)
        #data_vehicles: 0- No, 1-vehicle type, 2-length, 3-acceleration, 4-speed, 5-
position, 6-headway, 7-lead No, 8- lead type, 9-distance to signal, 10- signal state,
11- lane
                for index in range(len(data_vehicles)):
        # it is only used for CAV
                    vehicle = data_vehicles[index]
                    if (vehicle[1] != CAV):
                        continue
        # There are three mode for vehicles, free flow, signal controlled, and car
following mode
        # since the AV logic does not consider signal, we designed a signal controlled
mode that vehicles are going to decelerate at comfortable deceleration rate when get
close enough to signal head.
                    dist_safe = (vehicle[4] / 3.6) **2 / 2 /  -(simulation.comf_dec)

                    if (vehicle[8] == 'VEHICLE' and leading(vehicle[7],data_vehicles,
index)):
                        x_n = vehicle[5]
                        v_n = vehicle[4]
                        l_n_1, a_n_1, v_n_1 = leading(vehicle[7],data_vehicles,index)
                        x_n_1 = x_n + vehicle[6]
                        ak = auModel.AV_Model( x_n, v_n / 3.6, x_n_1, v_n_1 /3.6,
a_n_1, l_n_1).cal_acc()
                        ak = min(ak, (simulation.Desired_speed/  3.6 - v_n / 3.6) *
simulation.desired_speed_change_rate)

#===============================================================================
        #                 if v_n_1 < 5 and vehicle[6] < dist_safe + buffer:
        #                     ak = simulation.comf_dec
        #                 if vehicle[9] < dist_safe + buffer and (vehicle[10] =='RED'
or vehicle[10] =='AMBER'):
        #                     ak = simulation.comf_dec

#===============================================================================
                        record = [[time_step, vehicle[0],vehicle[1], x_n,vehicle[3],
v_n / 3.6, x_n_1, v_n_1 /3.6, a_n_1, l_n_1,x_n_1 - x_n,vehicle[11], ak, 'car
following']]
                        vehicle_ID.append(record[0][1])
                        vehicle_acc.append(record[0][12])
                    elif vehicle[9] < dist_safe + buffer and (vehicle[10] =='RED' or
vehicle[10] =='AMBER'):
                        ak = simulation.comf_dec;
                        record = [[time_step, vehicle[0], vehicle[1],
vehicle[5],vehicle[3], vehicle[4] / 3.6,vehicle[5] + vehicle[9] , 0 , 0 ,0,vehicle[9],
vehicle[11], ak, 'signal']]
                        vehicle_ID.append(record[0][1])
                        vehicle_acc.append(record[0][12])
                    elif not vehicle[11] in left_lane:
                        ak = min(simulation.comf_acc, (simulation.Desired_speed  -
vehicle[4]) / 3.6 *simulation.desired_speed_change_rate )
                        record = [[time_step, vehicle[0],vehicle[1],
vehicle[5],vehicle[3], vehicle[4] / 3.6,0 , 0 ,0, 0 ,0 ,vehicle[11], ak, 'free']]
                        vehicle_ID.append(record[0][1])
                        vehicle_acc.append(record[0][12])
                    else:
                        record = [[time_step, vehicle[0],vehicle[1],
vehicle[5],vehicle[3], vehicle[4] / 3.6,0 , 0 ,0, 0 ,0 ,vehicle[11], 999, 'VISSIM']]

                    log_file = log_file.append(pd.DataFrame(record,  columns =
log_col),ignore_index=True)

                ## update the traj file, here plus 0.01
                vehicle_ID, vehicle_acc = merge_array(vehicle_ID, vehicle_acc,
vehicle_ID2, vehicle_acc2)
                traj = pd.DataFrame({'a':vehicle_ID,'b':vehicle_acc})
                traj.to_csv(simulation.path_output_file + "\\Trajectory.txt",
header=None, index = None, sep=' ', mode='w')
                log_file.to_csv(simulation.path_output_file + "\\" + network_folder +
"log.csv", header = None,  index = None,  mode='a')
```

```
                """
                LC: Xi,the function bellow will need a dataFrame containing the
following
                headers. Make sure that the headers names match. If you add extra
headers it does not matter.


                         'No',
                         'pos', # List [UTM easting (float), UTM northing (float)]
                         'vel', # List [UTM easting dot (float), UTM northing dot
(float)]
                         'pos_rms', # List [UtM easting (float), UTM northing (float)]
                         'vel_rms', # List [UTM easting dot (float), UTM northing dot
(float)]
                         'veh_type', # 0 or 1 -- 0 for CNV, 1 for CAV
                         'veh_len', # float
                         'max_accel', # float
                         'max_decel', # float
                         ])
                """
                if sendMessage:
        #           print('hi')
                    if len(CV_data) > 0:
                        messageManager.send(CV_data)
        #                print(time_step)
        #                if time_step == 10:
        #                    print(time_step)
        #                    import pdb;pdb.set_trace()
            time_step += 1
            time.sleep(simulation.time_sleep)
            VISSIM.Simulation.RunSingleStep()

        '''
        Signal control part

        SC_number = 1z
        SG_number = 1
        SignalController = VISSIM.Net.SignalControllers.ItemByKey(SC_number)
        SignalGroup = SignalController.SGs.ItemByKey(SG_number)
        SignalGroup.SetAttValue("SigState", "GREEN")
        SignalGroup.SetAttValue("ContrByCOM", False)

        '''
        VISSIM.Simulation.Stop()

        print(time.time() - start_time)

        Filename = os.path.join(simulation.path_of_network, simulation.network_file)
        VISSIM.SaveNetAs(Filename)
        Filename = os.path.join(simulation.path_of_network, simulation.layout_file)
        VISSIM.SaveLayout(Filename)
        VISSIM = None
```

AV Model

```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov  5 11:17:55 2019
@author: xiduan
"""

# -*- coding: utf-8 -*-
"""
Created on Mon Nov  4 20:21:47 2019
@author: xiduan
"""
import numpy as np
class AV_Model ():

    def __init__(self, x_n, v_n, x_n_1, v_n_1, a_n_1, l_n_1, d_n = -6, d_n_1= -6, k = 
1, k_a = 1, k_v = 0.58, k_d = 0.1,de_x = 300, acc_max = 3 ,tau = 0.1 ):
        '''
        vehicle n is the subject, vehicle n-1 is the leading vehicle
       :param x_n
          location of vehicle n
       :param v_n
          the speed of vehicles n
       :param x_n_1
           the location of vehicles n-1
       :param v_n_1
           the speed of vehicles n-1
       :param a_n_1
           the acceleration of vehicles n-1
       :param l_n_1
           the length of vehicles n-1
       :param d_n
          the maximum deceleration of vehicles n
       :param d_n_1
          the maximum deceleration of vehicles n-1
       :param k default: 1.0
         model parameter
       :param k_a default: 1.0
         model parameter
       :param k_v default: 0.58
          model parameter
       :param k_d default: 0.1
          model parameter
       :param de_x
          the maximum detection distance of autonomous vehicle
       :param acc_max
          the maximum acceleration of vehicle n
       :param tau
          the reaction time of AV
        '''
        self.tau = tau
        self.x_n = x_n
        self.v_n = v_n
        self.x_n_1 = x_n_1
        self.v_n_1 = v_n_1
        self.a_n_1 = a_n_1
        self.l_n_1 = l_n_1
        self.d_n = d_n
        self.d_n_1 = d_n_1
        self.k = k
        self.k_a = k_a
        self.k_v = k_v
        self.k_d = k_d
        self.de_x = de_x
        self.acc_max = acc_max

    def cal_acc(self):

        s_safe = (self.v_n_1)**2 / 2* (1 / self.d_n -  1 / self.d_n_1)

        s_system =  self.v_n *  self.tau
        s_min = 1.5 + self.l_n_1
        s_ref = np.max([s_system,s_safe,s_min])

        acc_t  =  self.k_a *  self.a_n_1  +  self.k_v * ( self.v_n_1 - self.v_n) +
self.k_d * ((self.x_n_1 - self.x_n)  - s_ref)
        # delta_x is the space
        delta_x = np.min([(self.x_n_1 - self.x_n - self.l_n_1) + self.v_n * self.tau -
self.v_n_1**2 /2 / self.d_n_1,self.de_x])
        # v_max is the maximum safe speed
        v_max = (-2*self.d_n_1 * delta_x)**0.5

        acc = np.min([acc_t, self.k * (v_max - self.v_n),self.acc_max])



        return acc
```

Appendix D: EDM Source Code (drivermodel.dll) written in C++

```
/*==========================================================================*/
/*  DriverModel.cpp                                  DLL Module for VISSIM  */
/*                                                                          */
/*  Interface module for external driver models.                            */
/*  Dummy version that does nothing (uses VISSIM's internal model).         */
/*                                                                          */
/*  Version of 2018-09-13                                   XI DUAN
*/
/*==========================================================================*/

#include "DriverModel.h"
#include <fstream>  //std:ifstream
#include <iostream>
#include <string>
#include <sstream>
#include <algorithm>
using namespace std;

/*==========================================================================*/
// # 1 user working directory

/*Trajectory.txt, format as vehicles_ID, Vehicle_acceleration */
char inputPath[] = "P:\\STRIDE on VISSIM CAV\\code\\output_AV_Saif\\Trajectory.txt";
char outputPath[] = "P:\\STRIDE on VISSIM CAV\\code\\output_AV_Saif\\";

// # 2 initial parameters
double time_step = 0;

long    vehicle_color = RGB(0, 0, 0);

// # 3 output parameters
// note: the parameters with preceding Veh_ is output
long         VehID;
//the input_ID and input_acceleration will read from txt from python, **note: the size
of the array
long   input_ID[1000];
double  input_acceleration[1000];
double veh_speed = 0;
double  desired_velocity     = 20;
long    turning_indicator    = 0;
long         lane = 0;
double laneWidth;
double veh_lat_po = 0;
double time = 0;
// the value sent by VISSIM

double  veh_acceleration = 0;
double veh_lane_angle = 0;
long veh_target_lane = 0;
long veh_active_lane = 0;
// index_num and size_num are used to assign the acceleration to corresponding vehicle
ID
int index_num = 0;
int size_num = 0;
// #3 control parameters, this block is not used
// these are arrays used for latitude and longitude trajectory control
/*double  desired_acceleration[] = { 3,3,3,3,3,3,3,3,1.5,   1.2,   1.5,   1.2,   1.2,
       0.8,   0.8,   0.8,   0.8,   0.8,   0.7,   0.7,   0.7,   0.7,   0.7,   0.7,   0.5,
       0.5,   0.5,   0.3,   0.3,   0.2,   0.2,   0.2,   -3,    -3,    -3,    -2,    -2,
       -2,     -1.7,  -1.6,  -1.6,  -1.5,  -1.5,  -1.5,  -1,     -0.5,  -0.5,  -0.2,  -0.1,
       1,      1,     1,     1.2,   1.2,   1.2,   1.2,   -1,     -1,    0,     0
};
double  desired_lane_angle[] = { 1,     1,     1,     1,     1,     1,     1,     1,
       -1,     -1,    -1,    -1,    -1,    -1,    -1,    -1,    1,     1,     1,     1,
       1,      1,     1,     1,     -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
       -1,     -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
       -1,     -1,    -1,    -1,    1,     1,     1,     1,     1,     1,     1,     1,
       1,      1,     1,     1
};
long    active_lane_change[] = { 1,     1,     1,     1,     1,     1,     1,     1,
       -1,     -1,    -1,    -1,    -1,    -1,    -1,    -1,    1,     1,     1,     1,
       1,      1,     1,     1,     -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
       -1,     -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
       -1,     -1,    -1,    -1,    1,     1,     1,     1,     1,     1,     1,     1,
       1,      1,     1,     1
};
long    rel_target_lane[] = { 1, 1,     1,     1,     1,     1,     1,     1,     -1,
       -1,     -1,    -1,    -1,    -1,    -1,    -1,    1,     1,     1,     1,     1,
       1,      1,     1,     -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
       -1,     -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,    -1,
       -1,     -1,    -1,    1,     1,     1,     1,     1,     1,     1,     1,     1,
       1,      1,     1
};*/
int     curren_time = -1;
double  lane_angle =1;
long    active_lane = 0;
long    target_lane = 0;
double desired_acceleration = 0;
// define the input file of acceleration input
ifstream myfile;


// Pointers used for the opening of ".txt" files.
// file_rm used for general log
// file_lt_in show the value pass from VISSIM
// file_lt_out shows the value used in the dll
FILE *file_rm = NULL;
FILE *file_lt_in = NULL;
FILE *file_lt_out = NULL;

// The function is used to return the index of key in the array, find the acceleration
for the corresponding ID
int find(long arr[], int n, long key)
{
       int index = -999;

       for (int i = 0; i < n; i++){
              if (arr[i] == key){
                     index = i;
                     break;
              }
       }
       return index;
}

/*==========================================================================*/

BOOL APIENTRY DllMain (HANDLE  hModule,
                       DWORD   ul_reason_for_call,
                       LPVOID  lpReserved)
{
  switch (ul_reason_for_call) {
      case DLL_PROCESS_ATTACH:
      case DLL_THREAD_ATTACH:
      case DLL_THREAD_DETACH:
      case DLL_PROCESS_DETACH:
         break;
  }
  return TRUE;
}

/*==========================================================================*/

DRIVERMODEL_API  int  DriverModelSetValue (long   type,
                                           long   index1,
                                           long   index2,
                                           long   long_value,
                                           double double_value,
                                           char   *string_value)
{
  /* Sets the value of a data object of type <type>, selected by <index1> */
  /* and possibly <index2>, to <long_value>, <double_value> or            */
  /* <*string_value> (object and value selection depending on <type>).    */
  /* Return value is 1 on success, otherwise 0.                           */

  switch (type) {
    case DRIVER_DATA_PATH                    :

             //will need to update file path
             strcat(outputPath, "AV_Log.txt");
             //strcat(outputPath, "lateral_in.txt");
             //strcat(outputPath, "lateral_out.txt");
             fopen_s(&file_rm, outputPath, "wt");
             //fopen_s(&file_lt_in, "C:\\Users\\xiduan\\OneDrive - University of
Florida\\UF\\research\\2.VISSIM AV&CV\\Dll code\\lateral_in.txt", "wt");
             //fopen_s(&file_lt_out, "C:\\Users\\xiduan\\OneDrive - University of
Florida\\UF\\research\\2.VISSIM AV&CV\\Dll code\\lateral_out.txt", "wt");
             //Statement to write data to the text file. To change type of data
output, change the character after %: s -string; d - long; f - double
             if (file_rm != NULL) { fprintf_s(file_rm, "101 DRIVER_DATA_PATH: \t%s,
\n", string_value); }
    case DRIVER_DATA_TIMESTEP                :
             time_step = double_value;
    case DRIVER_DATA_TIME                    :
             time = double_value;
      return 1;
    case DRIVER_DATA_USE_UDA                 :
      return 0; /* doesn't use any UDAs */
          /* must return 1 for desired values of index1 if UDA values are to be
sent from/to VISSIM */
    case DRIVER_DATA_VEH_ID                  :

       // store the VehID information here
             VehID = long_value;
             // read the trajectory from the txt file defined before




             return 1;

    case DRIVER_DATA_VEH_LANE                :
             // get the current lane number from VISSIM
             lane = long_value;
             return 1;
    case DRIVER_DATA_VEH_ODOMETER           :
    case DRIVER_DATA_VEH_LANE_ANGLE         :
             veh_lane_angle = double_value;
             return 1;
    case DRIVER_DATA_VEH_LATERAL_POSITION   :
             // get the current lateral position from VISSIM
             veh_lat_po = double_value;
             return 1;
    case DRIVER_DATA_VEH_VELOCITY           :
             // get the speed
             veh_speed = double_value;
             return 1;
    case DRIVER_DATA_VEH_ACCELERATION       :
             // get the acceleration
             veh_acceleration = double_value;
             return 1;
    case DRIVER_DATA_VEH_LENGTH             :
    case DRIVER_DATA_VEH_WIDTH              :
    case DRIVER_DATA_VEH_WEIGHT             :
    case DRIVER_DATA_VEH_MAX_ACCELERATION   :
      return 1;
    case DRIVER_DATA_VEH_TURNING_INDICATOR  :
      turning_indicator = long_value;
      return 1;
    case DRIVER_DATA_VEH_CATEGORY           :
    case DRIVER_DATA_VEH_PREFERRED_REL_LANE :
    case DRIVER_DATA_VEH_USE_PREFERRED_LANE :
      return 1;
    case DRIVER_DATA_VEH_DESIRED_VELOCITY   :
      desired_velocity = double_value;
      return 1;
    case DRIVER_DATA_VEH_X_COORDINATE       :
    case DRIVER_DATA_VEH_Y_COORDINATE       :
    case DRIVER_DATA_VEH_Z_COORDINATE       :
    case DRIVER_DATA_VEH_REAR_X_COORDINATE  :
    case DRIVER_DATA_VEH_REAR_Y_COORDINATE  :
    case DRIVER_DATA_VEH_REAR_Z_COORDINATE  :
    case DRIVER_DATA_VEH_TYPE               :
      return 1;
    case DRIVER_DATA_VEH_COLOR              :
      vehicle_color = long_value;
      return 1;
    case DRIVER_DATA_VEH_CURRENT_LINK       :
      return 0; /* (To avoid getting sent lots of DRIVER_DATA_VEH_NEXT_LINKS messages)
*/
                /* Must return 1 if these messages are to be sent from VISSIM!
*/
    case DRIVER_DATA_VEH_NEXT_LINKS         :
    case DRIVER_DATA_VEH_ACTIVE_LANE_CHANGE :
             veh_active_lane = long_value;
             return 1;
    case DRIVER_DATA_VEH_REL_TARGET_LANE    :
             veh_target_lane = long_value;
             return 1;
    case DRIVER_DATA_VEH_INTAC_STATE        :
    case DRIVER_DATA_VEH_INTAC_TARGET_TYPE  :
    case DRIVER_DATA_VEH_INTAC_TARGET_ID    :
    case DRIVER_DATA_VEH_INTAC_HEADWAY      :
    case DRIVER_DATA_VEH_UDA                :
    case DRIVER_DATA_NVEH_ID                :
    case DRIVER_DATA_NVEH_LANE_ANGLE        :
    case DRIVER_DATA_NVEH_LATERAL_POSITION  :
    case DRIVER_DATA_NVEH_DISTANCE          :
```

```
    case DRIVER_DATA_NVEH_REL_VELOCITY      :
    case DRIVER_DATA_NVEH_ACCELERATION      :
    case DRIVER_DATA_NVEH_LENGTH            :
    case DRIVER_DATA_NVEH_WIDTH             :
    case DRIVER_DATA_NVEH_WEIGHT            :
    case DRIVER_DATA_NVEH_TURNING_INDICATOR :
    case DRIVER_DATA_NVEH_CATEGORY          :
    case DRIVER_DATA_NVEH_LANE_CHANGE       :
    case DRIVER_DATA_NVEH_TYPE              :
    case DRIVER_DATA_NVEH_UDA               :
    case DRIVER_DATA_NVEH_X_COORDINATE      :
    case DRIVER_DATA_NVEH_Y_COORDINATE      :
    case DRIVER_DATA_NVEH_Z_COORDINATE      :
    case DRIVER_DATA_NVEH_REAR_X_COORDINATE :
    case DRIVER_DATA_NVEH_REAR_Y_COORDINATE :
    case DRIVER_DATA_NVEH_REAR_Z_COORDINATE :
    case DRIVER_DATA_NO_OF_LANES            :
    case DRIVER_DATA_LANE_WIDTH             :
    case DRIVER_DATA_LANE_END_DISTANCE      :
    case DRIVER_DATA_CURRENT_LANE_POLY_N    :
    case DRIVER_DATA_CURRENT_LANE_POLY_X    :
    case DRIVER_DATA_CURRENT_LANE_POLY_Y    :
    case DRIVER_DATA_CURRENT_LANE_POLY_Z    :
    case DRIVER_DATA_RADIUS                 :
    case DRIVER_DATA_MIN_RADIUS             :
    case DRIVER_DATA_DIST_TO_MIN_RADIUS     :
    case DRIVER_DATA_SLOPE                  :
    case DRIVER_DATA_SLOPE_AHEAD            :
    case DRIVER_DATA_SIGNAL_DISTANCE        :
    case DRIVER_DATA_SIGNAL_STATE           :
    case DRIVER_DATA_SIGNAL_STATE_START     :
    case DRIVER_DATA_SPEED_LIMIT_DISTANCE   :
    case DRIVER_DATA_SPEED_LIMIT_VALUE      :
      return 1;
    case DRIVER_DATA_DESIRED_ACCELERATION :
             desired_acceleration = double_value;
             return 1;
    case DRIVER_DATA_DESIRED_LANE_ANGLE :
             lane_angle = double_value;
      return 1;
    case DRIVER_DATA_ACTIVE_LANE_CHANGE :
     // get the active lane change value from VISSIM
             active_lane = long_value;
      return 1;
    case DRIVER_DATA_REL_TARGET_LANE :
             target_lane = long_value;
      return 1;
    default :
      return 0;
  }
}

/*--------------------------------------------------------------------------*/

DRIVERMODEL_API  int  DriverModelGetValue (long   type,
                                           long   index1,
                                           long   index2,
                                           long   *long_value,
                                           double *double_value,
                                           char   **string_value)
{
  /* Gets the value of a data object of type <type>, selected by <index1> */
  /* and possibly <index2>, and writes that value to <*double_value>,     */
  /* <*float_value> or <**string_value> (object and value selection       */
  /* depending on <type>).                                                */
  /* Return value is 1 on success, otherwise 0.                           */

  switch (type) {
    case DRIVER_DATA_STATUS :
      *long_value = 0;
      return 1;
    case DRIVER_DATA_VEH_TURNING_INDICATOR :
      *long_value = turning_indicator;
      return 1;
    case DRIVER_DATA_VEH_DESIRED_VELOCITY   :

              /*if (find(input_ID, size_num, VehID) == -999) {
                     *double_value = 2;
              }
              else {


                     index_num = find(input_ID, size_num, VehID);
                     *double_value = 10;
              } */
              *double_value = desired_velocity;

      return 1;
    case DRIVER_DATA_VEH_COLOR :
      *long_value = vehicle_color;
      return 1;
    case DRIVER_DATA_VEH_UDA :
      return 0; /* doesn't set any UDA values */
    case DRIVER_DATA_WANTS_SUGGESTION :

              *long_value = 1;
      return 1;
    case DRIVER_DATA_DESIRED_ACCELERATION :

              // also need to change the directory
              myfile.open(inputPath);

              /*read the trajectory.txt file here, store the ID its corresponding
acceleration as array*/
              for (int i = 0; !myfile.eof(); i++) {

                     myfile >> input_ID[i] >> input_acceleration[i];
                     size_num = i;
              }
              myfile.close();

              /*if the Vehicle ID is not in the trajectory.txt, pass the default value
from VISSIM or constant 0.3 to it.*/
              if (find(input_ID, size_num, VehID) == -999) {
                     *double_value = desired_acceleration;
              } else {
                     /* if we could find corresponding vehicle, pass the designed
acceleration to the correspoing vehicle.*/
               index_num = find(input_ID, size_num, VehID);
              *double_value = input_acceleration[index_num];
              }

              return 1;
    case DRIVER_DATA_DESIRED_LANE_ANGLE :
     // *double_value = desired_lane_angle[curren_time];
       //*double_value = lane_angle;
```

```
                    *double_value = lane_angle;


                return 1;
      case DRIVER_DATA_ACTIVE_LANE_CHANGE :
        //*long_value = active_lane_change[curren_time];
                //*long_value =1;
                /*if (veh_lane_num == 2 && veh_lat_po == 0.4) {
                        *long_value = 0;
                }
                else {
                        *long_value = 1;
                } */
                *long_value = active_lane;
                /*if (curren_time == 10) {
                        *long_value = 1;
                }
                else if (curren_time == 80) {
                        *long_value = -1;
                }
                */
        return 1;
      case DRIVER_DATA_REL_TARGET_LANE :
      // *long_value = rel_target_lane[curren_time];

                /*if (curren_time == 10) {
                        *long_value = 1;
                }
                else if (curren_time == 80) {
                        *long_value = -1;
                }
                */
                *long_value = target_lane;
                return 1;
      case DRIVER_DATA_SIMPLE_LANECHANGE :
        *long_value = 1;
        return 1;
      case DRIVER_DATA_USE_INTERNAL_MODEL:
        *long_value = 0; /* must be set to 0 if external model is to be applied */
        return 1;
      case DRIVER_DATA_WANTS_ALL_NVEHS:
        *long_value = 0; /* must be set to 1 if data for more than 2 nearby vehicles per
lane and upstream/downstream is to be passed from VISSIM */
        return 1;
      case DRIVER_DATA_ALLOW_MULTITHREADING:
        *long_value = 0; /* must be set to 1 to allow a simulation run to be started
with multiple cores used in the simulation parameters */
        return 1;
      default:
        return 0;
    }
}

/*==========================================================================*/

DRIVERMODEL_API  int  DriverModelExecuteCommand (long number)
{
  /* Executes the command <number> if that is available in the driver */
  /* module. Return value is 1 on success, otherwise 0.               */
  switch (number) {
    case DRIVER_COMMAND_INIT :
             if (file_rm != NULL) { fprintf_s(file_rm, "**Initialization:\t%d
time_step\t%.2f time: %.1f\n:", VehID,time_step,time); }
             if (file_rm != NULL) { fprintf_s(file_rm, "Time: %.1f\tVehicle ID:
%d\tLane: %d\tSpeed: %.4f\tAccel: %.4f\tLC_angle %0.3f\n", time, VehID, lane,
veh_speed, veh_acceleration, veh_lane_angle); }


             return 1;
    case DRIVER_COMMAND_CREATE_DRIVER :
             if(file_rm != NULL) { fprintf_s(file_rm, "*******************Create
Driver ID:\t%d time: %.1f\n", VehID,time); }
             curren_time = -1;

      return 1;
    case DRIVER_COMMAND_KILL_DRIVER :
             if (file_rm != NULL) { fprintf_s(file_rm, "*******************Kill Driver
ID:\t%d \n", VehID); }
             return 1;
    case DRIVER_COMMAND_MOVE_DRIVER :
             myfile.open(inputPath);

             for (int i = 0; !myfile.eof(); i++) {

                    myfile >>input_ID[i] >> input_acceleration[i];
                    size_num = i;
             }
             myfile.close();

             if (file_rm != NULL) { fprintf_s(file_rm, "ID: %d\t acceleration: %f.3\t
speed: %d\t\n", VehID, veh_acceleration,size_num); }
             if (file_lt_out != NULL) { fprintf_s(file_lt_out, "acc1: %.3f \t acc2:
%.3f\t acc3: %.3f\tacc4: %.3f\tacc5: %.3f\tacc6: %.3f\n", input_acceleration[0],
input_acceleration[1], input_acceleration[2], input_acceleration[3],
input_acceleration[4], input_acceleration[5]); }
             if (file_lt_in != NULL) { fprintf_s(file_lt_in, "ID 1 %d\t ID 2 %d\tID 3
%d\tID 4 %d\tID 5 %d\tID 6 %d\t\n",input_ID[0], input_ID[1], input_ID[2], input_ID[3],
input_ID[4], input_ID[5]) ; }
             curren_time += 1;
             return 1;
    default :
      return 0;
  }
}




/*==========================================================================*/
/*  End of DriverModel.cpp                                                  */
/*==========================================================================*/
```

Appendix E: Urban Freight Transport and Its Effects

Introduction and Background Information

When considering emerging technology, and specifically AVs, the ability to test truck fleets on fully controlled access facilities (i.e. freeways) has seen great attention and is likely one of the first business cases with a high chance of both technical and economic success. However, there is also vital need for local, last-mile delivery innovations, especially with the expectation of fast delivery paired with warehouses located outside the city. The last-mile delivery remains a key issue to be solved. This literature review examines impacts of delivery trucks, Urban Freight Transport (UFT), on congestion, air pollution, and more. This is followed by a discussion on mobile access hubs, an innovative solution to the growing need to improve efficiency of urban freight deliveries that likely fits well with the emerging technology paradigm.

Transportation is the top source of greenhouse gases in the United States, and freight trucking contributes 23 percent of these emissions.[1] With rising urban populations and e-commerce growth, the proportion of these emissions attributed to delivery freight can reach as high as 40% of a city’s transportation carbon dioxide emissions and as high as 50% of air pollutants. [2] While recognizing these negatives it is also acknowledged that goods are essential to city residents and businesses, and the demand for delivery is increasing along with the growth in city population and e-commerce. Fulfillment times – the length of time from the placement of the order to its delivery – are getting shorter since delivery companies are getting more efficient and customers are preferring increasingly faster fulfillment. Faster fulfillment times tend to increase the number of single-parcel deliveries direct to customers and thus increase more vehicles on the road.[7] As a result, there is a major shift from business-to-business (B2B)

[1] Popovich, Nadja and Denise Lu. “The Most Detailed Map of Auto Emissions in America”. *New York Times*. 10 Oct. 2019.

[2] Lindeman, Tracey. ”Can ‘nests’ and eco bikes reduce the environmental impact of parcel delivery in cities?” *The Guardian.* 4 Nov. 2019. Accessed 24 Nov. 2019.

shipping to business-to-customer (B2C) shipping. Therefore, delivery companies such as UPS and FedEx must implement innovative and creative delivery solutions to remain competitive, efficiently, and sustainably. Considering this need, we will discuss the impact of UFT and the current and future freight delivery innovations that attempt to address these impacts.

E-Commerce Growth and Fast Delivery

In theory, e-commerce can be good for the environment since it can reduce the number of vehicular trips to the store by outsourcing these trips to a single pickup and delivery (P&D) vehicle.[3] However, these potential environmental benefits disappear as fulfillment times get shorter. The growth of e-commerce paired with two-day, next-day, same-day, and shorter delivery expectations has caused the proliferation of P&D vehicles into busy city centers, often during rush-hour.[2] Since 2010, there has been a 15 percent e-commerce growth year-over-year.[4] At the same time, consumers' expectations of how fast their package should be delivered is also rising. Amazon is a major reason for these fast delivery expectations, and the expectations are not going away. It is worth noting that 71% of millennials are Amazon Prime members and live in urban areas.[5] Because of delivery-time deadlines, P&D vehicles sometimes leave the distribution center half-empty,[2] which leads directly to decreased efficiency. If unchanged, this situation will only get worse, as the proportion of the world's population living in cities, and thus ordering packages to be delivered in cities, is expected to grow to from 55% to 68% by 2050.[6] Furthermore, urbanization is often associated with an increase in the average affluence in a city, and those with more disposable income tend to order more online goods. [7]

[3] Pandey, Erica. "The climate stakes of speedy delivery." *Axios*. 21 Jun. 2019. Accessed 10 Dec. 2019.

[4] "The Final 50 feet Urban Goods Delivery System." *Univ. of Washington Supply Chain Transport & Logistics Center.* Seattle Department of Transportation.

[5] Patel, Ketul. "Why urban fulfillment centers?: How retailers are adapting to connected consumers." *Deloitte Perspectives*. 2019. Accessed 25 Nov. 2019. https://www2.deloitte.com/us/en/pages/consulting/articles/why-urban-fulfillment-centers-retail.html.

[6] Malladi, Satya, et al. "Stochastic Fleet Optimization: Evaluating Electromobility in Urban Logistics." 3 Oct. 2019.

[7] UPS The Road to Sustainable Urban Logistics: A 2017 UPS/GreenBiz Research Study. *GreenBiz*. 2017.

Consequently, if the people moving to a city are more affluent, the number of packages ordered will increase even more.

Environmental Effects of UFT

Households now receive more shipments than businesses; thus, P&D vehicles are increasingly driving on neighborhood streets, increasing congestion everywhere. [8] Most P&D vehicles worldwide in urban areas make at least 100 brief stops a day, contributing to congestion, air pollution, and expensive traffic delays.[2] For example, in part because of the increase in UFT, cars in the busiest parts of Manhattan move at 23 mph, 7 miles slower than in 2000.[8] P&D vehicles contribute to congestion, and so reducing stem time and distance (the time and distance between the distribution center and the area in which the P&D vehicles makes stops for pickup and/or delivery), vehicle miles travelled (VMT), and altering routing and the sequence of stops may help alleviate the negative effects that P&D vehicles have on pollution.

Additionally, straining the P&D process is the transport of perishable goods, such as produce and refrigerated and frozen food. It is expected that these goods will be delivered even more expeditiously than other goods, e.g., same day or within hours.[8] Perishables often require temperature controlled transport and have special handling requirements, making it difficult to transport perishables (e.g., temperature controlled food) and non-perishables (e.g., consumer electronics) in the same vehicle. This fact increases single-parcel delivery, further decreasing the ability for delivery agencies to package and deliver efficiently. Ironically, people love the convenience of getting their packages quickly, but they do not like the congestion and air pollution that accompanies deliveries. As an indication of the rapid growth of fast fulfillment parcel P&D, it is reported that in some New York City neighborhoods, Amazon's boxes are stacked and sorted on the sidewalk, essentially "using public space as their private warehouse."[8] We remark that such use of public space is quite common in the mega-cities throughout China and Southeast Asia.

[8] Haag, Matthew, and Winnie Hu. "11.5 Million Packages a Day: The Internet Brings Chaos to N.Y. Streets". *New York Times.* 27 Oct. 2019. Accessed 7 Nov. 2019.

Further contributing to the environmental effects of the increased inefficiency caused by fast deliveries is the warehouse space and packing materials needed for the deliveries. As of June 2019, about 225 million square feet of warehouse space was under construction in the U.S., and many of these warehouses are being built outside the city center.[3] All of this warehouse space needs light, heat, and air conditioning, which all costs money and adds to our environmental impact. The packing material that goes into delivery boxes is additionally a major driver of the global plastics crisis.[3]

As the demand for package deliveries and particularly the demand for fast deliveries grows, it is essential that delivery agencies work to implement economically feasible, efficient, and environmentally sustainable solutions to alleviate the air pollution caused by P&D as much as possible. With options such as parcel hubs, self-driving trucks, delivery lockers, autonomous satellite vehicles, and more, there is potential to change the scope of deliveries in the United States and its impact on congestion and pollution.

Freight Effects on the Roadway

In a study on air pollution related to freight, it was determined that trucking is the most harmful mode of goods transport relative to air, rail, and ocean shipping.[9] The US Freight Transportation Forecast predicts 27 percent increase in P&D between 2016 and 2020[9]; therefore, many city agencies are looking into various methods to decrease related congestion. For instance, New York City's transportation commissioner said that the city is experimenting with enforcement and creative curb management.[8] Growing freight demand increases recurring congestion at bottlenecks between freight hubs caused by converging traffic at highway intersections and railroad junctions.[10]

[9] Organization for Economic Co-operation and Development. 1997. http://www.oecd.org/environment/envtrade/2386636.pdf

[10] "Freight and Congestion." *FHWA*. Freight and Management Operations. 1 Feb. 2017. Accessed 7 Nov. 2019. https://ops.fhwa.dot.gov/freight/freight_analysis/freight_story/congestion.htm

The Federal Highway Association attributes 947,000 hours of vehicle delay to delivery trucks parked curbside in dense urban areas where office and store buildings lack off-street loading; limitations on delivery times exacerbate this issue.[11]

UPS, FedEx, and Amazon: The Need for Change and Innovative Delivery Solutions

UPS recognizes that the current methods for delivery packages is inadequate to serve current and future needs of P&D, considering the impact on the environment and the growing demand for P&D services. Not only is the demand for P&D growing, but the demand for fast deliveries is growing. As curb-space becomes in higher demand by P&D vehicles, rideshare vehicles, and more, UPS and FedEx P&D vehicles are parking wherever they can find somewhere to stop. As a consequence of double parking on streets, blocking bus and bike lanes[8], UPS and FedEx are being fined. In New York City alone, FedEx incurred $14.9 million, and UPS $33.8 million, in parking violation fines in 2018.[12]

UPS Recognizes the Need for Improvement in Deliveries

In UPS's 2018 Progress Report, the CEO's message continually mentions how the interrelated topics of e-commerce, urbanization, and climate change are greatly shifting both markets and everyday life. [13] Thus, since 2009, UPS has invested more than $1 billion in alternative ways to mitigate the congestion and pollution emitted by their P&D vehicles in dense urban areas.[13] UPS has set an "ambitious" goal to reduce the absolute greenhouse gas emissions of its global ground operations by 12% by 2025. In order to accomplish this goal, UPS is exploring alternative fuel and advanced technology vehicles and infrastructure globally and is collaborating with cities to create innovative last-mile delivery solutions.[13]

---

[11] Urban, Angela. "With online shopping on the rise, cities look to address congestion impacts of deliveries." *Mobility Lab.* 13 Apr. 2017. Accessed 19 Nov. 2019.

https://mobilitylab.org/2017/04/13/role-of-deliveries-in-congestion/

[12] Baker, Linda. "UPS hit with $33.8 million in NYC parking fines; FedEx, $14.9 million." *Freight Waves.* 22 Mar. 2019. Accessed 21 Nov. 2019.

[13] UPS Creating Our Tomorrow, Sustainably: 2018 Corporate Sustainability Progress Report. *UPS*. 2018.

In its sustainability report, UPS reiterates the importance of finding innovative solutions to change the scope of delivering packages:

> **Crowded streets, with no parking available and poor curbside access to buildings have become iconic images of urban living. At the same time, city goals to promote greater quality of life, and reduce air pollution and congestion often are accompanied by regulations focused on personal vehicle use, without regard on the impact on commercial vehicles. … When combined effectively, consolidated shipments on environmentally-focused vehicles using data-driven routing can provide a step forward**.[7]

Because UPS recognizes the need for delivery innovations, they employ various methods, such as using delivery lockers, eBike fleets, alternative fuel vehicles, additive manufacturing, and more.[7] The company also has 17 warehouse projects underway, totaling more than 5 million square feet, to increase capacity and efficiency for business to business and business to consumer growth.[4]

The Role of Regulation

Government regulation can both positively and negatively affect last mile delivery logistics. In Santiago, Chile, the city has dedicated parking spots for freight vehicles during certain hours, helping to lessen congestion and improve efficiency. [14] Carbon taxes also have the potential to positively affect the environmental effects of freight transport. A carbon tax could encourage companies to be more efficient in their vehicle routing and stop sequencing. Alternatively, government regulation in Latin America has been counter-productive. In many places here, the government has imposed access restrictions for commercial vehicles based on vehicle type. However, these restrictions have led to an increase in the number of vehicles because companies split the load into smaller vehicles which are allowed in the city.[14] Done right, government regulation can help foster more efficient deliveries. Curb Flow, discussed later in this report, is one example of this.

Cities Do Not Plan for Freight Growth

[14] Brown, Eric. "E-commerce spurs innovation in last mile." MIT News. 4 Sept. 2018. Accessed 10 Dec. 2019. http://news.mit.edu/2018/mit-e-commerce-spurs-innovations-last-mile-logistics-0904

While cities are focusing on developing strategies to move people more efficiently and safely, they have not similarly focused on deliveries and delivery logistics, which drive the economy of a city.[7] The Director of the Urban Freight Lab at the University of Washington, Anne Goodchild succinctly states the issue, "Most cities do not have a freight plan. They have a transportation plan, a bike master plan, a transit master plan. But freight has not been something cities have been planning for."[7] Considering the projected growth of worldwide urban populations paired with the growth in e-commerce, cities may need to step in and plan for freight growth.

Urban Freight Delivery Innovations

One of the biggest challenges in urban environment deliveries is the last mile of the delivery, which are complicated by increasing gridlock in cities. Additionally, in the city, shipments are "much smaller and more fragmented than in regional transport."[14] There are more trucks on the street making more stops, and consumer e-commerce increases the chance of delivery failure. Since urban characteristics vary city to city, a delivery option's effectiveness is relative to a specific city. Various technologies which are in use and may eventually operate at scale are discussed in this section.

Data and GPS Route Planning

Although companies now have access to vast amounts of potentially relevant data, it is typically the case that a small subset of these data is the most useful. These data are transactions, delivery records, and customer information. By combining these data properly, a company can "generate a lot of insight into how demand is structured, how customers behave, and how to adapt delivery systems to better serve customer needs."[14] Route planners are often separate from the daily realities of P&D drivers, leading to mistaken assumptions. For instance, route planners often assume that drivers can park the P&D vehicle in front of a customer's house, even when this is not possible. Movement data through GPS tracking can help to extract local driver knowledge. By linking movement and transactional data, route planners can determine where the P&D vehicle was parked and which customers were served from that stop, enabling the route planners to develop more sophisticated, realistic routes and stop sequences for delivery drivers.

Delivery Lockers

Delivery lockers, which can be accessed with a key card or cell phone, centralizes the P&D process. These lockers are in public spaces, such as retail stores, on university campuses, and subway terminals. Apartment buildings and businesses also increasingly have package rooms with P&D lockers.

In the future, we can envision a combination of a delivery locker and an autonomous vehicle that can drive itself to a specific location in a neighborhood, college campus, etc. and park for an allotted amount of time, creating a centralized place for people to pick up and drop off their packages.[7]

Reserving Curb Space

As the curbside becomes more valuable with rideshare vehicles, delivery vehicles and on-demand restaurant deliveries, cities are working to find ways to better control and utilize the curbside. To reduce double-parking, a San Francisco based startup, called curbFlow, has an app which allows delivery drivers to reserve curb space.

curbFlow is a mobility company that coordinates commercial pickup and drop-off activities at the curbside in real time[15], acting as "air traffic control for the curbside."[16] The company launched its first market in Washington, DC on August 1, 2019. Commercial operators participating include UPS, Grubhub, and DoorDash. [17] On November 18, 2019, curbFlow launched its second market in Columbus, Ohio.[18] curbFlow provides loading zones to commercial and private vehicles (e.g. online food delivery) making deliveries. Drivers can check in on arrival using the curbFlow app or can reserve space up to 30 minutes in advance.

[15] DDOT, curbFlow Research Project Finds High Demand for Pickup, Dropoff Zones. District Department of Transportation. 13 Nov. 2019. Accessed 11 Dec. 2019. https://ddot.dc.gov/release/ddot-curbflow-research-project-finds-high-demand-pickup-dropoff-zones.
[16] "curbFlow: The Solution." Accessed 11 Dec. 2019. Curbflow.com.
[17] "curbFlow's Curb Management Program in Washington, DC Results in Safer, More Productive Streets." 13 Nov. 2019. Accessed 11 Dec. 2019. https://www.curbflow.com/dcfindings.
[18] "Press release: City of Columbus, curbFlow Kick Off Innovative Curbside Management Program." 18 Nov. 2019. Accessed 11 Dec. 2019.

The idea is that by using curbFlow, cities can “1) drastically reduce double parking, 2) provide safer streets for pedestrians, bicyclists, and drivers, 3) reduce carbon emissions from operators circling for space, 4) maximize productivity of one of their most valuable assets: curb space.”[16] Similarly, drivers can “1) be more efficient by driving down delivery times, 2) improve driver morale and retention with better delivery experience, 3)prioritize safety and compliance by getting out of the right-of-way, 4) reduce fines and citations.”[16]

DC tested the app for three months (Aug-Oct 2019) by making nine temporary delivery zones out of parking spaces. Researchers collected data from 6,350 commercial drivers representing more than 900 companies.[16] The project was regarded with success and found high-demand for pickup and dropoff zones with a decrease in double-parking incidents and illegal U-turns by 64%.[19] The test was well received, as 85 percent of users rated it as 9 or 10 as something they would recommend. The temporary delivery zones were restored to parking spaces after the program ended.

The idea of curbFlow is not technologically innovative, but seems to work despite, or perhaps because, it is not. It fosters city coordination with delivery drivers, dynamically using the curb-space for deliveries throughout the day, instead of using the space for parking for private vehicles. In this way, the curb-space, a public amenity, is serving more people. Giving dedicated curb-space to delivery drivers not only helps those immediately involved, the delivery drivers, but also positively affects cyclists (with decreased double-parking, which is often in bike lanes) and moving traffic (double-parking often blocks a lane of traffic).

Unmanned Deliveries (AVs and Drones)

Autonomous vehicles (AVs) and drone deliveries will become more feasible as their respective technologies advance. AVs are especially relevant when delivering perishable goods. The growth of perishable goods deliveries, which are often expected to be delivered within the hour and kept hot or cold, is inhibited by the cost of such deliveries. Whereas most packages can be

[19] Shaver, Katherine. Study: “Allowing delivery drivers to reserve curb space reduces double parking.” *The Washington Post.* 17 Nov. 2019. Accessed 21 Nov. 2019.

efficiently transported by a UPS P&D vehicle with dozens of other packages, perishable goods cannot because of their temperature and fast fulfillment constraints.

In Houston, Nuro, an autonomous vehicle delivery company, has been testing its fleet of robotically piloted vehicles for several months in 2019, delivering groceries to homes and restaurants.[20] The vehicles' sensors map the city as they drive around the city. Currently, two "operators" ride in the car and have to "remain in a state of near-constant focus for hours at a time," in order to monitor the vehicles' turns, braking, acceleration, and more.[20] This is just one of many companies that are trying to win the race to deploy truly autonomous delivery vehicles.

The Limitations of Unmanned Deliveries

With autonomous taxis, routes can be coordinated to pick up new passengers near the drop off point, so that the vehicle is rarely empty. However, with autonomous delivery vehicles, the AV will have to go back to the distribution center empty after it delivers packages, at least under one of the current models.[14] Perhaps delivery AVs will prove to be more efficient as smart locker vessels.

In order for drones to be a contender for efficient delivery, there needs to be the proper infrastructure, such as a landing area very near or on houses. Additionally, coordinating efficient and safe routes of thousands of delivery drones would be complicated and would contribute to noise pollution.[14]

However, combining the powers of AVs and drones could be useful. Delivery AVs could drive through the city and, without stopping, launch drones from the AV close to their destination. This would reduce the number of drones and the distance they fly, minimizing complications while also speeding up AVs, as the AVs would no longer have to make stops or even park.[14] This idea would not work for larger packages, though.

Electric Cargo Bikes

[20] Holley, Peter. "The future of autonomous delivery may be unfolding in an unlikely space: Suburban Houston." *Houston Chronicle.* 13 Nov. 2019. Accessed 16 Nov. 2019.

Electric-assist bikes (eBikes) have been deployed by delivery companies. Their battery-powered electric motors afford versatility and sustainability. eBikes, combined with human power, can cover long distances while carrying substantial loads. eBikes' size relative to traditional P&D vehicles make it so that they have less impact on bike lanes and lane blockages. eBikes at UPS work in harmony with a centralized container which can be brought into a city during non-peak hours.[7]

One city that is piloting eBike fleets is Montreal. The city has created several miniature distribution centers, dubbed "hubs", within the city where eBikes come to pick up and then deliver packages. Berlin, Germany, and Oslo, Norway, are two cities that already employ eBike programs.[2] eBike deliveries help address the last-mile delivery problem, which has been exacerbated by the proliferation of same-day and next-day delivery. P&D with eBikes optimizes square footage for maximum efficiency[2] while also being environmentally friendly.

Alternative Fuel Vehicles

A fully loaded, zero-emission P&D vehicle can reduce the environmental impact of inefficient and gas-powered P&D services. In cities, electric vehicles offer a huge potential for emission reduction since they are not as constrained by their range as they are in suburban areas.[7] Compounding this issue, however, is that more parcels are delivered to homes than to businesses. In sprawling cities, the distances vehicles will have to travel may still be great.

In Car Delivery Services

Volvo and other automobile manufacturers are experimenting with in-car delivery services.[7] A smartphone app can be assigned a one-time digital key to access the customer's trunk. This service could be especially convenient in garages and car parks.

Additive Manufacturing

3D printing enables goods to be "printed" closer to a customer's location. With additive manufacturing, only the raw materials, called feedstocks, that can be used to manufacture many different products need to be supplied.[7] This manufacturing innovation fosters greater flexibility and efficiency in providing certain goods in an urban area.

As of January 2, 2018, Amazon holds a patent for a retailing system that can take custom orders for 3D printed items.[21] Amazon is attempting to bring similar advantages of 3D printing to manufacturing projects to satisfy the needs of the everyday customer. Amazon gave an example of a household handle breaking off, and a customer placing an order, locating a digital CAD model of the replacement part in an online library.[21] This part would then be printed from one of the 3D printing sites and sent to the customer. By doing so, Amazon minimizes the amount of space they must use in warehouses, increasing efficiency.

Park and Walk

Sometimes it is more efficient for deliverers to park their truck and then use a handcart to deliver packages than to drive the trunk along the entirety of the road. Washington, DC's Foggy Bottom neighborhood is one example where deliverers park their and simply walk the package deliveries.[7]

UPS Depot-to-Door System

UPS announced on November 11, 2017 that they were testing a "depot-to-door" delivery system in a partnership with the city of London. The dubbed "Low Impact City Logistics project" was aimed to reduce traffic congestion and emissions associated with urban package delivery by using a power-assisted delivery trailer. UPS describes how the system works:

> **Packages will be loaded onto pay load boxes at the depot and delivered by a single trailer to a central hub located within a busy urban area. The boxes are distributed from the hub via power-assisted trailers. The packages are then delivered to homes and businesses by bicycle or on foot. The pay load boxes are moved by electric assisted trailers which feature patented net-neutral technology, which means the weight of the parcels – up to 200 kilograms - is not felt by the handler. This allows for increased last mile deliveries in a sustainable manner. The trial will feature bike trailers making deliveries in and around Camden during November and December.**[22]

[21] Cawley, David. "Amazon awarded patent for innovative new on-demand 3D printing retail service." 3ders.org. 3 Jan. 2018. Accessed 9 Dec. 2019. https://www.3ders.org/articles/20180103-amazon-awarded-patent-for-innovative-new-on-demand-3d-printing-retail-service.html.

[22] "Innovative 'depot-to-door' system reduces traffic congestion and carbon emissions." *UPS Pressroom.* London. 21 Nov. 2017. Accessed 19 Nov. 2019

The bike trailer used in this process is a product by the development firm Fernhay. It is a "net neutral" technology that prevents the rider from feeling the weight of the trailer so that anyone, regardless of their fitness level, will be able to make deliveries using the system. Furthermore, this project includes optimization algorithms through a "GPS tracker fitted within the trailer allowing for continuous improvement in route speed and efficiency."[22] This city partnership and others is an attempt by UPS to develop innovations in response to the increase in demand for package deliveries.

Conclusion and the Need for Future Research

Urban populations, urban affluence, e-commerce demand, and fast delivery expectations, are all projected to rise. With the current rate of e-commerce and UFT, we are already seeing worldwide negative effects on the environment and the inability of cities to properly handle the new roadway capacity pressures UFT places on them. Fast delivery expectations increase the inefficiencies of package deliveries, minimizing the potential positive environmental effects of e-commerce. While cities typically have an economic master plan, or a transportation master plan, they do not have a freight plan. There can be unintended negative effects of government regulation, but there is also the potential for a large positive impact. What is clear is that the current UFT delivery system cannot effectively and efficiently function as it currently stands.

There is a vital need for local, last-mile delivery innovations, especially with the expectation of fast delivery paired with warehouses located outside the city. UFT delivery innovations and ideas are being piloted in many places. Those discussed in this report, such as delivery lockers, unmanned deliveries, electric cargo bikes, additive manufacturing, and curb-space management, all contribute to help the last-mile issue in some way. A combination of the innovations discussed may produce the best results. However, there still lacks a solution that largely solves the last-mile delivery issue. More research and pilot-programs are necessary. Therefore, the next section discusses mobile access hubs as an innovative solution to the growing need to improve efficiency of urban freight deliveries.

---

https://pressroom.ups.com/pressroom/ContentDetailsViewer.page?ConceptType=PressReleases&id=1511883753076-442.

Mobile Access Hubs for Freight Pickup and Delivery

Problem Description and Background

As described in the literature review, urban logistics is growing in importance for several reasons. World population is increasing, the percentage of people living in urban areas is increasing world-wide, and new business models (fast fulfillment; the Amazon effect) are particularly accelerating the growth of urban freight pickup and delivery (P&D). Urban transportation infrastructure growth is not able to keep up with these demographic and business model changes, and as a result, urban congestion and pollution are increasing. Although building out of congestion and pollution has not been effective and is unlikely to be effective in the future, managing out of congestion and pollution has potential. The objective of this study is to investigate the congestion, pollution, and cost implications of a freight network design and management innovation: the deployment of mobile storage units, which we refer to as mobile access hubs, for urban P&D. This is considered with an eye toward AV and an understanding that such a system becomes increasingly feasible given AV and other emerging technologies

The P&D System

An urban P&D system network is composed of non-overlapping geographical areas, or *zones*, that contain clusters of customer locations. A single P&D courier provides P&D service for each zone. The courier drives a motorbike or van, uses a bicycle or handcart, and/or walks through his (or her) zone, delivering and picking up packages. After package delivery and pickup, the courier then travels back to a local hub (LH) to drop off the packages that have been picked up, load his vehicle with packages to deliver, and once loaded, drives back to his zone to begin anew the process of delivering and picking up packages. The courier may travel back and forth between the LH and his zone on a (road worthy) motorbike or in a van, both with small storage capacity, necessitating several trips per day to and from the local hub.

The LH is a transshipment location that provides P&D sorting and dispatch services to a substantial number (30 to 40) of zones in its local urban area, sends packages picked up in its zones to other local hubs in the city and to locations outside the city, and receives packages from other local hubs in the city and other locations outside of the city that are to be delivered to the zones in its local area.

We refer to the time the courier is traveling between the LH and his zone 'stem time' and the distance traveled 'stem distance'. Stem time is costly and unproductive since no pickups or deliveries occur during stem time. However, courier time is expended, travel expenses are incurred, air pollution is generated, and another vehicle is added to possibly congested roads. Private sector and public sector objectives are in alignment to reduce stem times during P&D as much as possible.

One solution for reducing stem times is to provide small storage units, called *access hubs*, that provide service to a small number (1 to 4) of zones. Access hubs receive packages from the LH that are to be delivered to the zones served by the access hub and receive packages from these zones to be transshipped through the LH. At most only minor sorting is done at an access hub. Packages are transported between access hubs and the LHs by a road worthy shuttle, a van, or small truck that typically has a greater carrying capacity than the vehicles used for transport by the couriers between the LH and the zone. A courier's stem time is reduced and often eliminated if his zone is provided service by an access hub since the distance from the zone to the access hub is considerably shorter (sometimes zero) than the distance from the zone to the LH. Since the carrying capacity of the vehicle moving packages between the access hub and the LH is larger than the carrying capacity of vehicles used by the couriers, total stem time and distance are reduced, and courier productivity and customer service level are increased.

There are two types of access hubs, fixed and mobile. Fixed access hubs cannot be easily relocated whereas mobile access hubs require a relatively modest amount of time and effort for relocation. Examples of a fixed access hub include a storage shed, a store backroom, and a smart locker bank, and these may be attended or unattended. Depending on the location (e.g. public vs. private space), attended or unattended, and who has access (single user or multi-party user), various levels of security can be used to secure access to the stored packages.

A mobile access hub (MAH) is a mobile storage facility that can be motorized (e.g., truck or van) or towed (e.g., trailer) and can be easily repositioned in an urban environment, assuming space availability. The figures below illustrate a UPS MAH in Munich and a TNT (recently acquired by FedEx) MAH in Brussels, both temporarily located in reserved parking spaces. In both cases, a courier picks up packages for delivery from the MAH and delivers packages picked up in his zone to the MAH, using a motorized tricycle. A shuttle vehicle, or the MAH itself if motorized, may be used to move packages between the MAH and the LH.

Manufacturing capacity can also be relocatable and, also true with storage capacity, it is easier to relocate some forms of capacity than others. 3D printers and bioreactors (any manufactured device or system that supports a biologically active environment) are often small enough to be moved with a handcart or hand jack and a van. Referring to either manufacturing or storage capacity that is relocatable as a *module*, we remark that how often modules would be relocated and the lead time for the relocation would in reality be situation and module dependent. For 3D printers, bioreactors, and smart locker modules (another form of potentially mobile storage capacity), relocation decisions might be made monthly or quarterly, and relocation may take a day or two. Location may be event driven. For example, there may be a surge of package shipment demand in specific zones due to a special holiday or entertainment event or in response to a production disruption, requiring the rapid (within hours) deployment of a van, package car, or straight truck. Relocation decisions for container-sized modular production units for pharmaceutical intermediates might be made quarterly or annually and require several weeks of lead time if the module is to be moved across a national border. However, if the relocatable module is a smart locker or 3D printer on a truck or trailer (GeekWire, 2018; Verlinde et al., 2014), then relocation decisions might be made once or more a week, even daily, and require less than an hour or two of lead time.

We now evaluate the relative merits of three options for zones served by a LH, the no access hub option, the fixed access hub option, and the MAH option:

- The no access hub option generates the most stem time and requires the least capital expenditure by not having either a fixed access hub or MAH, both of which require capital investment and accrue expenses. The no access hub option is reasonable to consider for zones close to the local hub (i.e., those having short stem times), zones where fixed storage facilities do not exist or are expensive, or zones where there is no available and reasonably priced reserved parking or loading zone for a MAH.
- The fixed access hub option is reasonable to consider when stem times are long, demand for storage is significant for most of the week, fixed storage capacity is available and affordable, and reserved parking is unavailable or expensive.
- The MAH option is reasonable to consider when stem times are long, demand for storage may be significant for only a portion of the week (e.g., demand is high on Monday and Tuesday mornings and otherwise modest or negligible), demand at other zones in the urban area is negatively correlated (e.g., there are other zones where demand peaks at times of the week other than Monday and Tuesday mornings and is low on Monday and Tuesday mornings), reserved parking is available and affordable, and fixed storage space is expensive or unavailable. In general, loading zones or reserved parking is easier to secure than is a fixed access hub.

The following two examples illustrate these three options.

Example 1: The intent of this example is to illustrate the value of an access hub (either fixed or mobile), relative to the no access hub option. Assume there is a single LH, a single zone Z, and that there is a single class of service that places no deadlines throughout the day for pickups or deliveries. For the no access hub option, the courier's day begins at the LH by loading his P&D vehicle to full capacity with packages to deliver to customers in Z. We assume:

- Loading takes 15 minutes.

- The trip from LH to Z takes 15 minutes. We assume that the vehicle transporting the packages is a road worthy motorbike or van, either of which is a source of air pollution.
- Once in Z, deliveries and pickups take 1 hour. The deliveries and pickups may be accomplished by the vehicle transporting the packages from LH or by other, less polluting means, such as walking with a handcart.
- The return trip from Z to LH takes 15 minutes.
- Unloading the packages and minor sorting once the courier has arrived at LH takes 15 minutes.

Thus, a cycle takes 2 hours. During an 8-hour day, 4 cycles are possible, with 2 hours of stem time. We assume that air pollution generated during stem time is high, relative to air pollution generated during P&D in Z, and the P&D vehicle contributes to congestion during stem time but not during P&D in Z.

Assume an access hub exists (mobile or fixed), packages are loaded at LH and transported to Z prior to the beginning of the courier's workday, either by the MAH or by a shuttle to the fixed access hub. We assume this transport provides enough packages for a full day of courier P&D. At the end of the courier's workday, the packages picked up during the day in Z are transported back to LH. Thus, only one round-trip between LH and Z is required by either the MAH or by the shuttle transporting packages for a fixed access hub. A courier's cycle now takes 1.5 hours, and 5 and 1/3rd cycles are possible with no hours of stem time for the courier. Hence, the courier's productivity has increased 33% and the total amount of air pollution generated by the courier is less than the total amount of air pollution generated by the courier when no access hub exists. The MAH or the shuttle supplying the fixed access hub generates 30 minutes of air pollution at a relatively high rate. However, total air pollution generated with an access hub is likely to be considerably less than the total air pollution generated without an access hub. Further, there is a 75% reduction in vehicle stem time, reducing the traffic congestion contribution, relative to the case where no access hub exists.

Example 2: The intent of this example is to illustrate the potential value of a MAH, compared to a fixed access hub. We continue Example 1 having a single LH but providing service to two zones, Z1 and Z2, on different days of the week. Assume P&D demand at Z1 is high on Monday and Tuesday and negligible the rest of the week and P&D demand at Z2 is negligible on Monday and Tuesday and high for the rest of the week. There are two options: (1) place a fixed access hub in both zones for a total of two fixed access hubs, (2) deploy a single MAH to Z1 Monday and Tuesday and to Z2 the rest of the week. Although a MAH may cost more per unit volume of storage space than a fixed access hub, the cost per unit volume of storage space of a single MAH is unlikely to be twice that of a fixed access hub. Thus, we get the benefit of access hubs described in Example 1 at less cost and more flexibility to the company by deploying a single MAH, rather than two fixed access hubs.

The above examples are highly stylized for illustrative purposes and to some extent represent best case scenarios. In reality, a company would offer multiple classes of service with deadlines for delivery in the zones (e.g., expedited packages guaranteed to be delivered the next day before 9:30am) or delivery to the LH for the departure of straight trucks or vans to other local hubs and linehaul vehicles outside of the urban area. The transport of packages between the LH and the zones would have to take into consideration delivery deadlines and departures from the LH to other destinations. These considerations are highly likely to increase the number of shuttle trips between the zones and the LH. The examples do not take into consideration vehicle capacity limitations and freight handling risk (the more a package holding a fragile object is handled, the higher the likelihood of (i) breakage, (ii) the need to file an insurance claim, and (iii) a disgruntled customer), which motivates a company to minimize the number of times a package is handled. Also, negative demand correlations are highly unlikely to be as clear cut as assumed in the second example. On the other hand, a shuttle probably would be deployed to make 'milk runs' involving multiple stops at multiple access hubs, which could introduce additional operational efficiencies with ancillary reductions in congestion and pollution.

In summary, determining how many access hubs to have, which zones should have them, which of these should be serviced by mobile or fixed access hubs is a complicated dynamic network design task. How to deploy the MAHs throughout the day and week, and how many of the MAHs should be motorized and how many should be trailers are highly dependent on zone P&D real-time P&D demand data and their correlations, reserved parking and fixed access hub availability and cost, and a myriad of other costs. Further, managing such a network is highly data-driven and time sensitive, and hence operating the network is a complex management task. Such freight distribution networks are Next Generation networks. These networks are data-driven and controlled in real-time, and the level of their performance improvement is a function of the quality of the real-time data and the analytics that convert these data into real-time network management decisions. Adoption by industry is driven by the need to remain competitive. The examples have indicated the potential for impact of such networks on firm productivity and on urban traffic congestion and air pollution and hence indicate how the public sector can incentivize socially positive corporate behavior by helping to accelerate the development and adoption of such networks.

It is also clear that while replacing today's current P&D system with AVs may reduce labor costs it has a high potential to increase congestion, environmental impacts, and other system inefficiencies. Emerging technologies must not be considered in isolation, or simply as replacements of activities currently accomplished through non-AV means. To avoid the many potential negative impacts feared by the introduction of AV it is necessary to reimage entire systems, such as P&D.

Quantitative Analysis

In this section, we present an approach for analyzing the impact of affordable, reserved parking (or loading zones) availability on congestion and pollution reduction. We begin by making three comments. First, the more reserved parking is available and affordable, the more likely MAHs will be used. Second, as we assumed in the examples, the cost per unit of time for a unit of area storage space is likely to be lower for a fixed access hub than for a MAH. Hence, for zones that are generally busy, a fixed access hub might be the best option, rather than any alternative. Third, as can be inferred from the discussion in Example 2, negative correlations in the demand for P&D for zones help to identify where a single MAH can provide service to multiple zones by intelligent, demand-driven repositioning. The more zones having negatively correlated demand, the fewer MAHs are needed. However, do such negative correlations typically exist in an urban area? If so, then as these negative correlations increase, the likelihood increases that mobile hubs could support efficient package P&D and hence have positive impact on congestion and pollution. To the best of our knowledge, data are limited in this regard. However, we are aware of an unpublished package demand correlation study where approximately 25% of the zones in a large urban area have negatively correlated demand, supporting the claim that the number of MAHs needed to provide service to zones in an urban area may be considerably smaller than the number of zones, especially if a MAH can provide service to multiple (presumably adjacent) zones.

The first comment raises the question: what level of public sector intervention would elicit a private sector response that would have a material impact on reducing congestion and pollution? This question in turn raises two questions:

1. For an urban area, what is the potential for congestion and pollution reduction using mobile and/or fixed access hubs, where the no access hub option is the baseline, and how can this reduction be affected by public sector policy interventions?
2. How much does freight delivery contribute to urban pollution and congestion and how much of this is due to P&D?

The second question has been addressed in the literature search, which indicates P&D is a major contributor to urban pollution and congestion. With regard to the first question, we thus far have argued that MAHs can help to reduce urban congestion and pollution and have postulated that the public sector can accelerate the development and adoption of MAHs by providing or incentivizing affordable reserved parking space for them and have noted in the literature review the significant impact of curbFlow in a three month study in San Francisco. Our argument thus far has been qualitative. We now present a first step in quantitatively determining the impact of MAHs.

Parking for private automobiles can be either short term (e.g., at commuter parking lots that do not permit overnight parking) or long term (e.g., long term airport parking). At this point, it is not clear if reserved parking for MAHs will be short term, long term, or a mixture of both. However, the analytic and computational implications are different and significant if reserved parking is short term or long term. If the MAHs are motorized, then they may return to the local hub in the evening to unload and remain at the LH overnight and be reloaded with packages that are to be delivered to their zones for next day's P&D operations. If this is the case, then it is sufficient to consider a model of MAH deployment that only involves a time horizon of a single day. Such a model would not have to consider the strategic deployment of the MAH for the next day, later in the week, or over the weekend and hence would require relatively less complex analysis.

If parking for periods longer than a day is permitted, and this may be preferred for MAHs that are trailers like the UPS and TNT MAHs pictured above, then we would want the model to take into consideration multi-day strategic deployment. We now consider the long-term parking problem and will treat the short-term parking problem as a special case.

Model Formulation

Assume there are $\boldsymbol{Z}$ zones that a MAH may be deployed to for at least one day during a 7-day week. We assume that P&D demand for the zones is periodic with periodicity 7; e.g., the demand for Monday is the demand for all Mondays. Assume $a(z,t) = 1\ (= 0)$ if a MAH is deployed (not deployed) at zone z on day t. Throughout day t, the positioning of the MAHs is described by $\{a(z,t)\}$, which we consider the state of the P&D distribution system. We assume decisions for next day MAH deployment is made with full knowledge of how the MAHs are currently deployed. Additionally, we assume redeployment during the day is not feasible; a more granular (e.g., hour to hour or morning and afternoon) model would allow intra-day redeployment.

In the evening of day t, the following events occur. If the MAH serves as its own shuttle, the MAH returns to the LH to deposit packages picked up through day t that are destined for other LHs within the urban area or destinations outside of the urban area. Throughout the night, packages are loaded into the MAH that are to be delivered to the zone that it will service on day $t + 1$. We remark that the MAH may be assigned to a zone on day $t + 1$ that is different from the zone it was assigned to on day t, as illustrated in Example 2. Then early on day $t + 1$, the MAH travels to its assigned zone to support P&D services.

If the MAH is a trailer (i.e., cannot relocate by itself) and it is to service a zone on day $t+1$ different from its assigned zone on day t, then the MAH must be repositioned, which we assume accrues a repositioning cost C. For notational simplicity, we initially assume that the repositioning cost is independent of the zones on days t and $t+1$. An actual application may require a more detailed repositioning model and analysis of repositioning cost.

Let $c_1(z,t)$ $\left(c_0(z,t)\right)$ and be the expected P&D cost to be accrued during day t, given zone z has (does not have) a MAH that provides it service. The precise determination of $c_1(z,t)$ and $c_0(z,t)$ is non-trivial and would be application specific. Then, the expected total P&D cost accrued during day t, given the state of the P&D system on day $t-1$ is $\{a(z,t-1)\}$ and the state on day t is $\{a(z,t)\}$, is

$$\sum_{z}\left[a(z,t)c_1(z,t)\;+\;\left(1-a(z,t)\right)c_0(z,t)\;+\;C|a(z,t-1)-a(z,t)|/2\right].$$

We remark that it is likely that $c_1(z,t) \leq c_0(z,t)$ for zones far from the LH and $c_1(z,t) \geq c_0(z,t)$ for zones close to the LH, and we will assume $\boldsymbol{Z}$ is less than or equal to the number of zones that satisfy $c_1(z,t) \leq c_0(z,t)$ for any t. The optimization problem then becomes

$$\min\sum_{t}\beta^t\sum_{z}\left[a(z,t)c_1(z,t)\;+\;\left(1-a(z,t)\right)c_0(z,t)\;+\;C|a(z,t-1)-a(z,t)|/2\right]$$

subject to

$a(z,t) \in \{0,1\}$ for all z and t

$\sum_z a(z,t) = \boldsymbol{Z}$ for all t

for discount factor β such that $0 \leq \beta < 1$.

This optimization problem is an infinite horizon assignment problem (a form of combinatorial optimization problem) that can be analyzed as a periodic dynamic program having the optimality equations $v = H_t v, t = 1, \ldots, 7$, where:

$$[H_t v](a) = \min_{a'} \{[a'(z)c_1(z,t) + \left(1 - a'(z)\right)c_0(z,t) + C|a(z) - a'(z)|/2] + \beta v(a', t + 1)\},$$

and when $t = 7, t + 1 = 1$ (modulo 7). Straightforward extensions of results in (Puterman, 1994) guarantee that there exists a unique set of real-valued functions ($v_t^*, t = 1, \ldots, 7$) such that $v_t^* = H_t(v_t^* + 1), t = 1, \ldots, 6, v_7^* = H_7(v_1^*)$ and that these fixed points are the minimum expected total discounted cost over the infinite horizon of starting at day t. Also, the policy that causes the minimum to be attained is an optimal policy, where a policy is a mapping from the state at day $t - 1$ to the state at day t (modulo 7) and these 7 policies are periodic (e.g., a policy that is optimal for Monday is optimal for all Mondays). It is also true that for any bounded function $v_{0,7}$ and sequence of functions $\{v_{t,n}\}$, where $v_{7,n+1} = H_1(v_{1,n}), v_{t-1,n} = H_t(v_{t,n}), t = 2, \ldots, 7$, then in the limit at n goes to infinity, $||v_t^* - v_{n,t}||$ goes to zero, where $|| \cdot ||$ is the sup norm.

We remark that given v, the single period optimization problem $[H_t v](a)$, a single successive approximations iteration, is an assignment problem (an integer program with binary variables) and can be solved with standard optimization software. We remark that the number of operations per successive approximations step is 2 to the power $\boldsymbol{Z}^2$.

If overnight parking is prohibited and/or all of the MAHs are motorized and are required to return to the LH every evening, then repositioning cost is automatically accrued and there is no need to take into consideration the cost of repositioning the MAHs when making the daily repositioning decision. Under these circumstances, the repositioning cost can be subsumed into $c_1(z,t)$ and $c_0(z,t)$ and we can set $C = 0$. Then the above optimization problem becomes a much simpler single period assignment problem, identical in complexity and hence computational effort to one step of successive approximations (with $v = 0$). This problem can be solved with the following simple (and optimal) heuristic: on day t, send MAHs to the zones having the $\boldsymbol{Z}$ largest values of $c_0(z,t) - c_1(z,t)$, where we have assumed above that this difference is always positive (otherwise, park some of the MAHs and/or reduce $\boldsymbol{Z}$).

MAHs Serving Multiple Zones

Thus far, we have assumed that a zone is served by either zero or one MAH, an assumption that we will continue to make. Also, thus far, we have assumed that a MAH serves only one zone per day. Recalling Example 1 and assuming there is a MAH serving two geographically adjacent zones, relative to the no MAH case for both zones, the same increase of benefits are accrued for both zones as outlined in Example 1 at half the capital costs of a MAH. Thus, congestion and pollution reduction per zone remains the same and capital expenditures are reduced by 50%. If the MAH serves two zones that are geographically separated, presumably by a short distance, then at least one of the two couriers would have a modest amount of stem time between the MAH and his zone, modestly increasing his contribution to congestion and pollution. This discussion illustrates the value of having access hubs serve multiple (2-4) zones and having the zones served to be as geographically co-located as possible.

Related Unpublished Studies

We are aware of two yet unpublished studies on topics related to storage capacity mobility that indicate the potential of relocatable (storage and production) capacity. The first study is part of an on-going 2-year simulation study of freight flows in a large urban area (Shenzhen, PRC) for a large package express company. An overview description of the study is presented in (Faugère, et al., 2018). Part of this study evaluates the value of access hubs (both fixed and mobile), takes into consideration multiple levels of service, but assumes no access to long term (overnight) reserved parking. Preliminary yet unpublished results indicate that MAHs serving one to two zones can reduce the stem distance traveled by 11% to 60%. We remark that a correlation analysis of demand at the zone level in the urban area, also as yet unpublished, shows that there are substantial (25%) negative correlations among pairs of zones, indicating that opportunities exist for a single MAH to provide service to multiple zones over the course of a week, which would reduce the capital requirements for (and hence barriers to) implementing MAHs. However, it is unknown at this point if this percentage of negatively correlated pairs of zones is typical of domestic major metropolitan areas.

The second study (Malladi, et al, 2019) is concerned with relocatable manufacturing capacity and shows that production capacity mobility plus transshipment in a 5-location manufacturing network can improve systems performance by as much as 41% relative to the case where manufacturing capacity cannot be relocated, and transshipment is not permitted. Further, this study shows that production capacity mobility, assuming transshipment is not permitted, can yield as much as 10% more savings compared to when transshipment is permitted but production capacity is immobile. Both studies indicate the significant potential value of mobile capacity, a design feature that we anticipate will be key facet of many future supply chain designs.

Faugère, L., Malladi, S., White, C., Montreuil, B., Smart Locker Based Access Hub Network Capacity Deployment in Hyperconnected Parcel Logistics. Proceedings of 5th International Physical Internet Conference, 2018

Malladi, S., Erera, A., White, C., "A Dynamic Mobile Production Capacity and Inventory Control Problem", IISE Transactions, accepted for publication, November 2019 A Dynamic Mobile Production Capacity and Inventory Control Problem

Conclusions and Future Research

The curbFlow study in San Francisco, our simple analysis of stylized situations in Examples 1 and 2, and the two unpublished studies indicate that use of MAH's, more generally access hubs, in urban areas is a P&D network design innovation with significant potential for reducing congestion, pollution, and private sector costs. It is clear that such approaches must be considered as the transportation system evolves to AV and other emerging technologies.

Further, we have taken a first step in the development of an optimization problem that is tractable under many realistic circumstances (especially for the case where MAH overnight parking is not permitted) and can be useful in supporting an in-depth study of the impact of MAHs. A barrier to the private sector adoption of this cost saving network design innovation would be the lack of affordable access to reserved parking for the MAHs, a barrier that the public sector could play a role in removing or reducing. A future research topic would be an in-depth study of what role(s) the public sector might play to accelerate the use of MAHs in urban areas for congestion and pollution reduction.

REFERENCES

- Verlinde, S., Macharis, C., Milan, L., & Kin, B. (2014). Does a mobile depot make urban deliveries faster, more sustainable and more economically viable: results of a pilot test in Brussels. *Transportation Research Procedia*, *4*, 361-373.
- Marujo, L. G., Goes, G. V., D'Agosto, M. A., Ferreira, A. F., Winkenbach, M., & Bandeira, R. A. (2018). Assessing the sustainability of mobile depots: The case of urban freight distribution in Rio de Janeiro. *Transportation Research Part D: Transport and Environment*, *62*, 256-267.
- Arvidsson, N., & Pazirandeh, A. (2017). An ex ante evaluation of mobile depots in cities: A sustainability perspective. *International Journal of Sustainable Transportation*, *11*(8), 623-632.
- M. L. Puterman, “Markov decision processes: Discrete stochastic dynamic programming,” p. 672, 1994.
- GeekWire (2018). Amazon finally wins a patent for 3-D printing on demand, for pickup or delivery. URL: Geekwire website.com/2018/amazon-gets-patent-3-d-printing-demand-pickup-delivery/.