RESEARCH REPORT

Institute of Transportation Studies | UNIVERSITY OF CALIFORNIA

Streamlining Connected Automated Vehicle Test Data Collection and Evaluation in the Hardware-in-the-Loop Environment

Zhe Fu, Graduate Student Researcher, Department of Civil & Environmental Engineering
Hao Liu, Ph.D., Assistant Research Engineer, Partners for Advanced Transportation Technology
Xiao-Yun Lu, Ph.D. Research Engineer, Partners for Advanced Transportation Technology
University of California, Berkeley

December 2020

Berkeley Institute of Transportation Studies

Report No.: UC-ITS-2020-23 | DOI: 10.7922/G2PC30PD

Technical Report Documentation Page

1. Report No. UC-ITS-2020-23	**2. Government Accession No.** N/A	**3. Recipient's Catalog No.** N/A
4. Title and Subtitle Streamlining Connected Automated Vehicle Test Data Collection and Evaluation in the Hardware-in-the-Loop Environment		**5. Report Date** December 2020 **6. Performing Organization Code** ITS Berkeley
7. Author(s) Zhe Fu, orcid.org/0000-0003-4478-3978; Hao Liu, Ph.D. orcid.org/0000-0001-5585-6576; Xiao-Yun Lu, Ph.D. orcid.org/0000-0001-6491-3990		**8. Performing Organization Report No.** N/A
9. Performing Organization Name and Address Institute of Transportation Studies, Berkeley 109 McLaughlin Hall, MC1720 Berkeley, CA 94720-1720		**10. Work Unit No.** N/A **11. Contract or Grant No.** UC-ITS-2020-23
12. Sponsoring Agency Name and Address The University of California Institute of Transportation Studies www.ucits.org		**13. Type of Report and Period Covered** Final Report (July 2019 – June 2020) **14. Sponsoring Agency Code** UC ITS
15. Supplementary Notes DOI:10.7922/G2PC30PD		

16. Abstract

Quality data collection, processing, and analysis are foundational to good research, policy making and regulation development. With the rapid development of Connected Automated Vehicles (CAV) technologies, it is urgent for both researchers and policy makers to obtain and evaluate good quality CAV data to better understand CAV impacts. CAV hardware-in-the-loop (HIL) tests can expedite CAV performance evaluation and system implementation. This research aims at equipping an existing HIL test tool with data management functions. To this end, a database instance on MySQL has been integrated with an existing HIL test tool. The improved HIL test tool can greatly streamline CAV data collection and quality so that it is beneficial for performance analysis. A detailed comparison and selection of available database tools, database instance design and implementation have been performed to help other California institutes develop and improve their own systems. A user-friendly test tool setup guide and a specific user guide have been provided to enable potential users to easily get started using the data management functions. In addition, two example CAV tests are presented to demonstrate the detailed data collection and performance evaluation procedure. Those examples can serve as a guide to assist users in applying the HIL test tool in their own CAV tests.

17. Key Words Connected vehicles, automated vehicles, data collection, hardware in loop simulation, databases, data management, data quality		**18. Distribution Statement** No restrictions.	
19. Security Classification (of this report) Unclassified	**20. Security Classification (of this page)** Unclassified	**21. No. of Pages** 56	**22. Price** N/A
Form Dot F 1700.7 (8-72)		Reproduction of completed page authorized	

About the UC Institute of Transportation Studies

The University of California Institute of Transportation Studies (UC ITS) is a network of faculty, research and administrative staff, and students dedicated to advancing the state of the art in transportation engineering, planning, and policy for the people of California. Established by the Legislature in 1947, ITS has branches at UC Berkeley, UC Davis, UC Irvine, and UCLA.

Acknowledgments

This study was made possible through funding received by the University of California Institute of Transportation Studies from the State of California through the Public Transportation Account and the Road Repair and Accountability Act of 2017 (Senate Bill 1). The authors would like to thank the State of California for its support of university-based research, and especially for the funding received for this project.

Disclaimer

Institute of Transportation Studies | UNIVERSITY OF CALIFORNIA

Streamlining Connected Automated Vehicle Test Data Collection and Evaluation in the Hardware-in-the-Loop Environment

Zhe Fu, Graduate Student Researcher, Department of Civil & Environmental Engineering
Hao Liu, Ph.D., Assistant Research Engineer, Partners for Advanced Transportation Technology
Xiao-Yun Lu, Ph.D. Research Engineer, Partners for Advanced Transportation Technology
University of California, Berkeley

December 2020

Berkeley Institute of Transportation Studies

Report No.: UC-ITS-2020-23 | DOI: 10.7922/G2PC30PD

Table of Contents

Table of Contents

List of Tables

List of Figures

Executive Summary

Executive Summary

Testing the performance of Connected Automated Vehicles (CAV) in various road facilities and traffic scenarios is a critical step towards successful deployment of this advanced technology. Large scale field CAV tests are difficult to conduct due to resource limitations, safety constraints and high expense. Simulation-based analysis often adopts simplified models of the real-world system, thus limiting the explanatory capability of the analysis. The hardware-in-the-loop (HIL) experiment is an alternative method to conducting field tests or simulations. However, developing a HIL test requires comprehensive knowledge of vehicle dynamics control, traffic flow modeling, machine learning, and communication. Few teams have researchers from all those fields and it may be too time-consuming to establish a HIL test environment every time a test is proposed.

This research developed a proper HIL test tool to integrate the existing CAV systems, communication algorithms and traffic flow models and streamline their combined applications. The HIL test tool will not only coordinate the execution of and manage the data flow from different test systems, but also offer database functions to store raw data sets collected from the test systems and generate performance metrics based on the raw data. We developed data management functions for a prototype HIL test tool constructed in a previous project[1] so that the functionality of the existing HIL tool was substantially improved. For the data management tool development, we first performed a detailed comparison and selection of available database tools, database instance design, and database implementation for the existing HIL tool. We chose the most popular relational database, MySQL, as the database server for the test tool. We developed an Entity-Relationship Diagram and its corresponding attribute variables tables for a clean and clear database instance design. The presented design and implementation procedures of the proposed data management functions are not only beneficial for the existing HIL tool, but also can be easily extended to other CAV test systems.

This report also provides a step-by-step setup procedure including a tutorial for connecting MySQL to Visual Studio, which supports the HIL test tool, and the proposed data management functions. In addition, a user guide for the HIL test tool has been developed. This user-friendly guide allows researchers to easily use the HIL tool so that they can concentrate on developing and evaluating target CAV systems without the time-consuming need to establish an HIL testbed and database. Two example CAV tests are presented to demonstrate the detailed data collection and performance evaluation procedure. This can help other California institutes to design customized CAV experiments with the HIL test tool.

The proposed data management functions provide a convenient means for researchers and practitioners to debug and analyze the CAV test results. It also offers flexibility for research teams to generate result evaluations by using stored historical data from real-world test systems. The HIL test tool also supports pulling historical data in several common file types, which enables researchers to export datasets to other tools. The HIL test tool with the proposed data management functions enables an easy data collection and analysis process for CAV systems. It helps researchers use existing traffic simulation tools, traffic signal controllers and CAV control algorithms in their HIL tests, even if they are not experts in traffic flow modeling or vehicle dynamics control. It can greatly streamline CAV data collection and quality. This will accelerate the implementation and evaluation of research for broader applications in California.

---

[1]This project was sponsored by the U.S. Department of Energy (DOE) Vehicle Technologies Office (VTO) under the Systems and Modeling for Accelerated Research in Transportation (SMART) Mobility Laboratory Consortium, an initiative of the Energy Efficient Mobility Systems (EEMS) Program.

Contents

Chapter 1. Introduction

Background

Hardware-in-the-Loop Evaluation of Connected Automated Vehicle Applications

Connected Automated Vehicles (CAV) are vehicles that are equipped with connectivity and autonomous technology. Different from Automated Vehicles (AV), CAVs can talk to each other and the infrastructure around them. The connectivity and autonomous vehicle technologies enable CAVs to significantly improve the existing transportation system by reducing congestion, traffic incidents, and vehicle fuel consumption and emissions [1].

Testing the performance of CAVs in various road facilities and traffic scenarios is a critical step towards successful deployment of this advanced technology. Since large scale field CAV tests are difficult to conduct due to resource limitations, safety constraints and high expense, the hardware-in-the-loop (HIL) experiment is an alternative approach to expedite CAV performance evaluation and system implementation.

Hardware-in-the-loop is appropriately named to represent the loop-like interaction between real systems and test systems that simulates reality. In a HIL test, researchers often put real-world CAV vehicles or their subsystems (e.g., vehicle dynamics controllers) in a controllable test environment (e.g., a test track or a chassis dynamometer)[2]. They then use computer simulation tools to generate repeatable traffic flow conditions and insert a virtual traffic flow into the test environment. The performance of the subject CAVs is then measured as the real system interacts with the artificial environment. As an example, a HIL test assessing CAV performance in congested traffic may have CAVs operating on a physical test track and receiving virtual information representing a simulated congested traffic scenario, such as the speed and distance of other vehicles. In response to the virtual information, the CAVs will behave like they were driving in congested traffic.

Current HIL Test Tool

A functional HIL test should provide a platform for multiple real-world and virtual systems to interact with each other. To this end, we developed a HIL (testbed) test tool as displayed in Fig. 1 in a previous study. The test tool includes a test CAV fleet, a test track, a microscopic traffic simulation model that simulates car-following and lane-changing behaviors of vehicles, a real-world traffic signal control system, a communication layer, and a server program. The test tack and traffic control system offer a physical test environment for the CAVs. The simulation model is responsible for generating virtual traffic flow. The server program is used to coordinate the operation of each test system via various communication mediums, which helps with the synchronized operation of all test systems. In addition, the server program manages the data flow (as described in Figure 1). It contains the encoding and decoding algorithms that facilitate the data transfer via different communication channels.

A typical experiment in the test tool contains the following steps:

- Step 1: Develop a simulated road network based on the physical layout of the test track.
- Step 2: Initiate the traffic simulation to create a virtual traffic stream.
- Step 3: Synchronize the clocks of the simulation, traffic signal controller, and test CAVs.
- Step 4: Build a connection between the traffic simulation and the traffic controller; start updating traffic signals based on the virtual traffic.

- Step 5: Build a connection between the traffic simulation and the test CAVs; start the data interchange between the real and virtual vehicles.
- Step 6: Begin the test; start collecting test data.

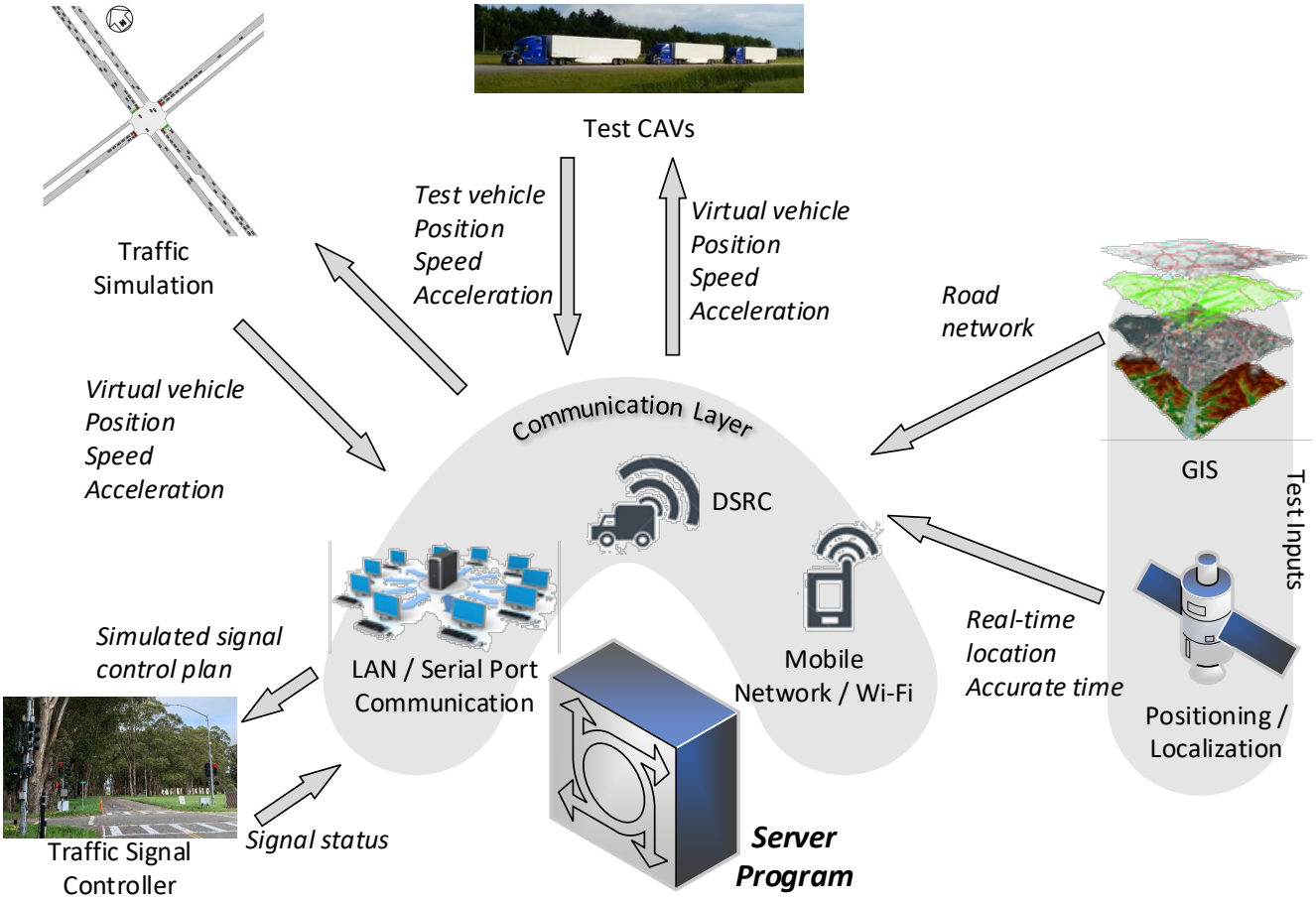

Figure 1. Components and data flow of the current HIL test tool

Problem Statement

While our current HIL test tool has functions to coordinate the execution of different test systems and manage the data flow, it does not contain data storage and data processing functions to process metrics such as time, vehicle distance covered, speed, and acceleration over a long period of time to facilitate further analysis.

Research Objective

This study provides guidance for better CAV data collection and evaluation through the development of a HIL CAV test tool that provides an easy data collection and analysis process. This study also serves as a resource for other researchers interested in using existing traffic simulation tools, traffic signal controllers, and CAV control algorithms in their HIL tests,

even if they are not experts in traffic flow modeling or vehicle dynamics control which will greatly streamline CAV data collection and quality.

Specifically, this study adds a data management module to the current HIL test tool (circled in Figure 2), which will enable the test tool to automatically store real-time data and will also allow the test tool to process historical data from each test system after a test run is completed. The research team also conducted sample CAV tests to demonstrate the implementation of the proposed data management tool to help users easily get started with the proposed tool.

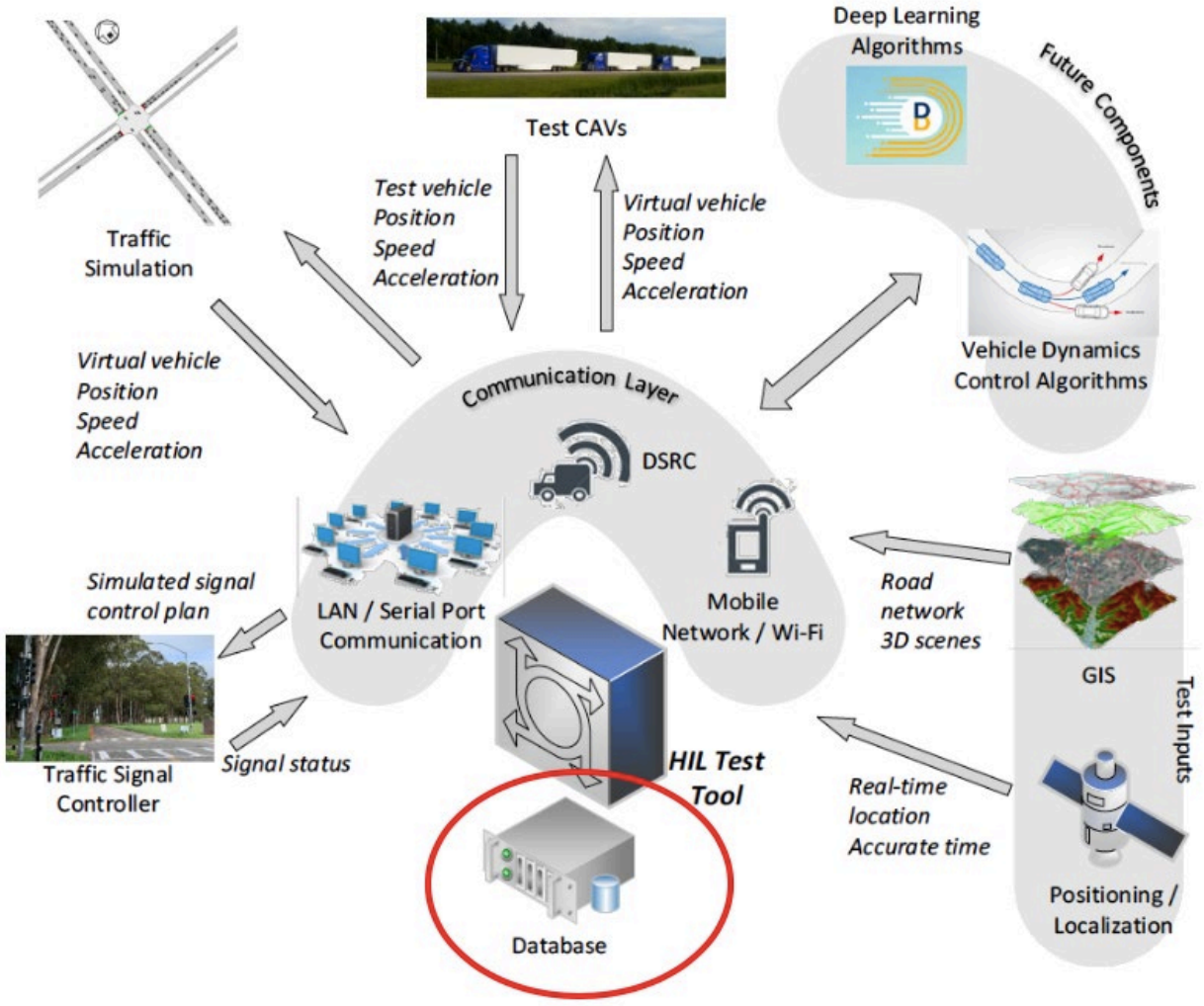

Figure 2. Test systems and data flow of the proposed HIL test tool

Report Organization

The next chapter compares different modern databases based on a literature review. Chapter 3 describes our database instance design including Entity-Relationship Diagram and Attributes Table design. Chapter 4 describes the logic flow and the detailed implementation of the test tool. In chapter 5, we provide a setup guide for the test tool and in chapter 6 we

offer a step-by-step user guide for researchers to use our test tool in an easier way. Two specific example CAV test experiments are described in Chapter 7. The conclusions are given in Chapter 8.

Chapter 2. Database Comparison and Selection

Available Database Tools

The proposed data management functions require a database for storing data and accessing data. Both Non-relational Databases (NoSQL) and Relational Database Management Systems (RDBMS) are available choices. RDBMS emerged in the 1970's and became widely used since then. It stores data according to a schema that allows data to be displayed as tables with rows and columns[3]. The tables in a relational database have keys associated with them, which are used to identify specific columns or rows of a table and facilitate faster access to a particular table, row, or column of interest. NoSQL databases were developed as a popular alternative to relational databases due to their flexibility to take a variety of forms[4]. The critical difference between NoSQL and RDBMS is that RDBMS schemas rigidly require that all data inserted into the database must be typed and composed, while NoSQL can be schema agnostic, allowing unstructured and semi-structured data to be stored and manipulated[5]. The most popular and widely used open-source databases are MongoDB, MySQL and PostgreSQL[6]. Among them MongoDB is a non-relational database and the other two are relational databases. They were the candidate database tools in this project. A brief description of them is given as follows.

MongoDB

MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era[7]. It is a NoSQL database program and it uses JavaScript Object Notation (JSON)-like documents with optional schemas. Because of this, it is frequently used for projects developed in JavaScript code and the JSON facilitates the exchange of data between web apps in a human-readable format.

MySQL

MySQL is an open-source relational database management system which employs the concept of storing data in rows and tables which are further classified into the database. MySQL is the world's most popular RDBMS database[7]. It uses Structured Query Language (SQL) to access and transfer data. And it applies commands such as "SELECT" and "INSERT" to manage the data.

PostgreSQL

PostgreSQL is a free and open-source relational database management system emphasizing extensibility and SQL compliance[8]. It is the world's most advanced RDBMS database with rich features such as automatically updatable views, materialized views, triggers, foreign keys and stored procedures. It is designed to handle a range of workloads, from single machines to data warehouses or Web services with many concurrent users.

Comparison and Final Choice

A comparison of MongoDB, MySQL and PostgreSQL is summarized in Table 1[9]. Because the attributes that we wanted to store are in relational format and the tables in RDBMS databases are easier for users to read and access, the two RDBMS databases are preferred. Since MySQL embeds SQL as the data manipulation language, and it offers APIs to

connect the database with a C++ application, we chose MySQL as the database tool to develop the proposed data management functions.

Table 1. Comparison of MongoDB, MySQL and PostgreSQL

	MongoDB	MySQL	PostgreSQL
Description	One of the most popular document stores available both as a fully managed cloud service and for deployment on self-managed infrastructure	Widely used open source Relational Database Management System (RDBMS)	Widely used open source Relational Database Management System (RDBMS). Developed as object-oriented DBMS(Postgres), gradually enhanced with "standards" like SQL.
Primary Database Model	Document Store	Relational DBMS	Relational DBMS
Secondary Database Model	Search engine	Document Store	Document Store
Implementation Language	C++	C and C++	C
Data Schema	schema-free	yes	yes
SQL	Read-only SQL queries via the MongoDB Connector for BI	yes	yes
Supported programming languages	C/ C++/ Java/ Matlab/ Python/ PHP/ R/ Ruby	C/ C++/ Java/ Python/ PHP/ R/ Ruby	C/ C++/ Java/ Python/ PHP
Foreign keys	no	yes	yes
Transaction concepts	Multi-document ACID Transactions with snapshot isolation	ACID	ACID

Chapter 3. Database Instance Design

A database instance design lays the foundation for coding the proposed database tool. The database instance design produces an Entity-Relationship diagram and its corresponding tables that describe the framework of the database. The details regarding the database instance design are shown in this chapter.

Entity-Relationship Diagram

An Entity-Relationship diagram (E-R diagram) is a data modeling technique that graphically illustrates an information system's entities and the relationships between those entities. An entity is a real-world object described by a set of attribute values. A relationship is an association among two or more entities.

As Figure 3 shows, in our E-R diagram, we have four different modules:

1) Server, a program used to coordinate the operation of each test system via various communication mediums.
2) Controller, representing the traffic signal controller in the real world.
3) Simulator, providing the simulated traffic flow. It contains two sub-modules, one for regulating the movements of the virtual cars and the other one for generating traffic signal control plans based on the virtual traffic flow.
4) DSRC Radio, representing the communication layer for test CAVs. In our test track and test plan, three test trucks were used, so this module contains three sub-modules, each of them represents a single test truck.

Each entity's corresponding attribute values that are to be stored are shown in white ovals in Figure 3.

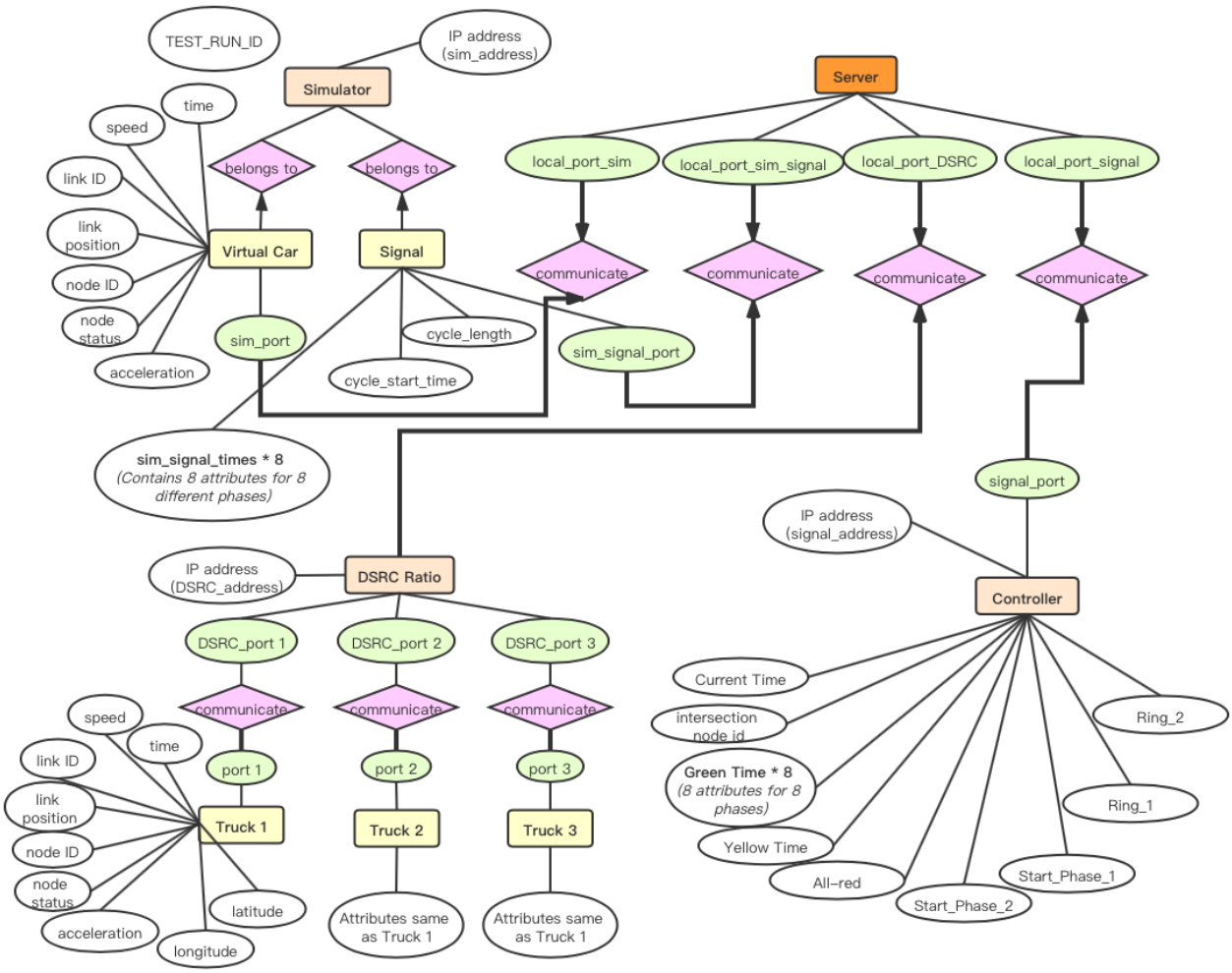

Figure 3. Entity-Relationship Diagram of the database design

Attributes Table Design

The attributes table design provides further information about the attribute values to be stored in the database. This work is also beneficial for streamlining the coding work of the database because it describes the framework of the tables in the database.

Six attributes tables have been designed. Each table corresponds to a sub-module/module of the E-R diagram described above in Figure 3. The tables not only list the attributes, but also their data type in both the Server Program implemented in C++ and the database implemented in MySQL. This can be beneficial for developers to find the corresponding attribute variables in the HIL test tool and write the correct scripts to create the database in MySQL.

Table 2. Attributes Information of Signal Controller

Attribute	Type in C++	Data Type in database	Attribute	Type in C++	Data Type in database
TEST_RUN_ID	double	DOUBLE	Next_Cycle_Start_Time	double	VARCHAR(30)
Next_Cycle_Start_Time_ms	double	DOUBLE	Intersection_Node_ID	double	INT
Green Time_1	double	DOUBLE	Green Time_2	double	DOUBLE
Green Time_3	double	DOUBLE	Green Time_4	double	DOUBLE
Green Time_5	double	DOUBLE	Green Time_6	double	DOUBLE
Green Time_7	double	DOUBLE	Green Time_8	double	DOUBLE
Yellow Time	double	DOUBLE	All Red	double	DOUBLE
Start Phase_1	int	INT(1)	Start Phase_2	int	INT(1)
Ring_1	int	INT(3)	Ring_2	int	INT(3)

Table 3. Attributes Information of Simulator Signal

Attribute	Type in C++	Data Type in database	Attribute	Type in C++	Data Type in database
TEST_RUN_ID	double	DOUBLE	Time	double	VARCHAR(30)
Time_ms	double	DOUBLE	Speed	double	DOUBLE
Link_ID	double	INT	Link_Position	double	DOUBLE
Node_ID	double	INT	Node_Status	bool	INT(1)
Acceleration	double	DOUBLE			

Table 4. Attributes Information of Virtual Car Table

Attribute	Type in C++	Data Type in database	Attribute	Type in C++	Data Type in database
TEST_RUN_ID	double	DOUBLE	Time	double	VARCHAR(30)
Time_ms	double	DOUBLE	Speed	double	DOUBLE
Link_ID	double	INT	Link_Position	double	DOUBLE
Node_ID	double	INT	Node_Status	bool	INT(1)
Acceleration	double	DOUBLE			

Table 5. Attributes Information of Truck 1 Table

Attribute	Type in C++	Data Type in database	Attribute	Type in C++	Data Type in database
TEST_RUN_ID	double	DOUBLE	Time_1	double	VARCHAR(30)
Time_1_ms	double	DOUBLE	Speed_1	double	DOUBLE
Acceleration_1	double	DOUBLE	Position_1	double	DOUBLE
Longitude_1	double	DOUBLE	Latitude_1	double	DOUBLE

Table 6. Attributes Information of Truck 2 Table

Attribute	Type in C++	Data Type in database	Attribute	Type in C++	Data Type in database
TEST_RUN_ID	double	DOUBLE	Time_2	double	VARCHAR(30)
Time_2_ms	double	DOUBLE	Speed_2	double	DOUBLE
Acceleration_2	double	DOUBLE	Position_2	double	DOUBLE
Longitude_2	double	DOUBLE	Latitude_2	double	DOUBLE

Table 7. Attributes Information of Truck 3 Table

Attribute	Type in C++	Data Type in database	Attribute	Type in C++	Data Type in database
TEST_RUN_ID	double	DOUBLE	Time_3	double	VARCHAR(30)
Time_3_ms	double	DOUBLE	Speed_3	double	DOUBLE
Acceleration_3	double	DOUBLE	Position_3	double	DOUBLE
Longitude_3	double	DOUBLE	Latitude_3	double	DOUBLE

Chapter 4. Database Implementation in the HIL Test Tool

Major Steps

The data management function in the HIL test tool is implemented through the following steps: 1) creating a connection to the MySQL server, 2) generating a distinct test run ID for different test runs, 3) finding corresponding attribute variables in the current HIL Server Program, 4) transforming attribute variables into strings in response to the particular format of MySQL's API functions, and 5) writing attribute values to the database in MySQL.

Once those steps are completed and the Server Program is executed, the test data can be automatically stored in the database.

Step 1: Create Connection to MySQL Database

At the beginning of the test, the connection between the HIL test tool and the database hosted in MySQL can be created by using the API functions of MySQL.

Step 2: Generate Test Run ID

Since a large amount of data from different test runs will be stored in the database, it might be difficult for researchers to identify and extract data from a particular test run. Since it is common to conduct several tests sequentially, it is necessary to distinguish between the end of one test run and the start of another. In order to address this problem, the start time of the HIL tool in seconds (counting from 1970-1-1) becomes the test run ID for each simulation run and is stored as an attribute value for each collected data row.

Step 3: Find Corresponding Attributes

This step addresses data collection. The attributes that we listed in the design chapter are not provided in a sorted order by the existing HIL test tool. Fully understanding the current test tool framework and its core functions is very important. Most of the attribute variables are in the three main functions that handle the updates of the simulation, signal controller, and test CAVs. For the rest of the variables, tracking backwards through those main functions can help.

Step 4: Transform Attribute Variables into Strings

The attributes in the HIL tool are stored in different data types and format. In order to utilize the API function of MySQL in the correct way, it is necessary to convert those attribute variables into strings.

Step 5: Record Attribute Values

This step stores the collected data. The major API function of MySQL is "Insert." With the help of the Insert function, the newly collected data row can be easily inserted into the corresponding table of the database.

Implementation

Steps 1 and 2 added new functions to the current functions in the HIL test tool. To realize steps 3, 4, and 5, revising the current functions would be required. Specific details on implementing these steps can be found in the Appendix.

Chapter 5. Test Tool Setup Guide

Prerequisites

To set up the test tool, the following five software programs need to be installed:

1. Visual Studio 2017
2. Visual Studio 2013
3. Aimsun 8.2
4. Aimsun MicroSDK package
5. MySQL Server 5.7 (X86, 32 bit)
6. MySQL connector 1.19(X86, 32 bit)

Visual Studio 2017 is the software used for running the HIL test tool. Aimsun 8.2 is the simulation software for generating the virtual traffic stream. Customized simulation models were used to replace Aimsun's default models. The customized models were developed in Aimsun MicroSDK in Visual Studio 2013. MySQL Server 5.7 was used to run the database and MySQL connector 1.19 was used to connect MySQL and Visual Studio, so that the database could be manipulated through the HIL test tool.

Both MySQL programs (Nos. 5 and 6) should be installed in the X86, 32-bit version, since part of the elements in the Server Program are processed in the 32-bit version.

When installing MySQL Server, the user must set up their own user name (default is *root*) and corresponding password. This user name and the password need to be inputted in the database management tool as well.

MYSQL Connection to C++

The following detailed step-by-step tutorial has been provided for connecting MySQL to Visual Studio:

Step 1

Open the VS project in VS 2017, open the project configuration properties and click in the following path:

Configuration Properties→ C/C++ → General → Additional Include Directories

Add MySQL Server 5.7 include, MySQL Connector 1.1.9 include and project's own include folder as shown in Figure 4.

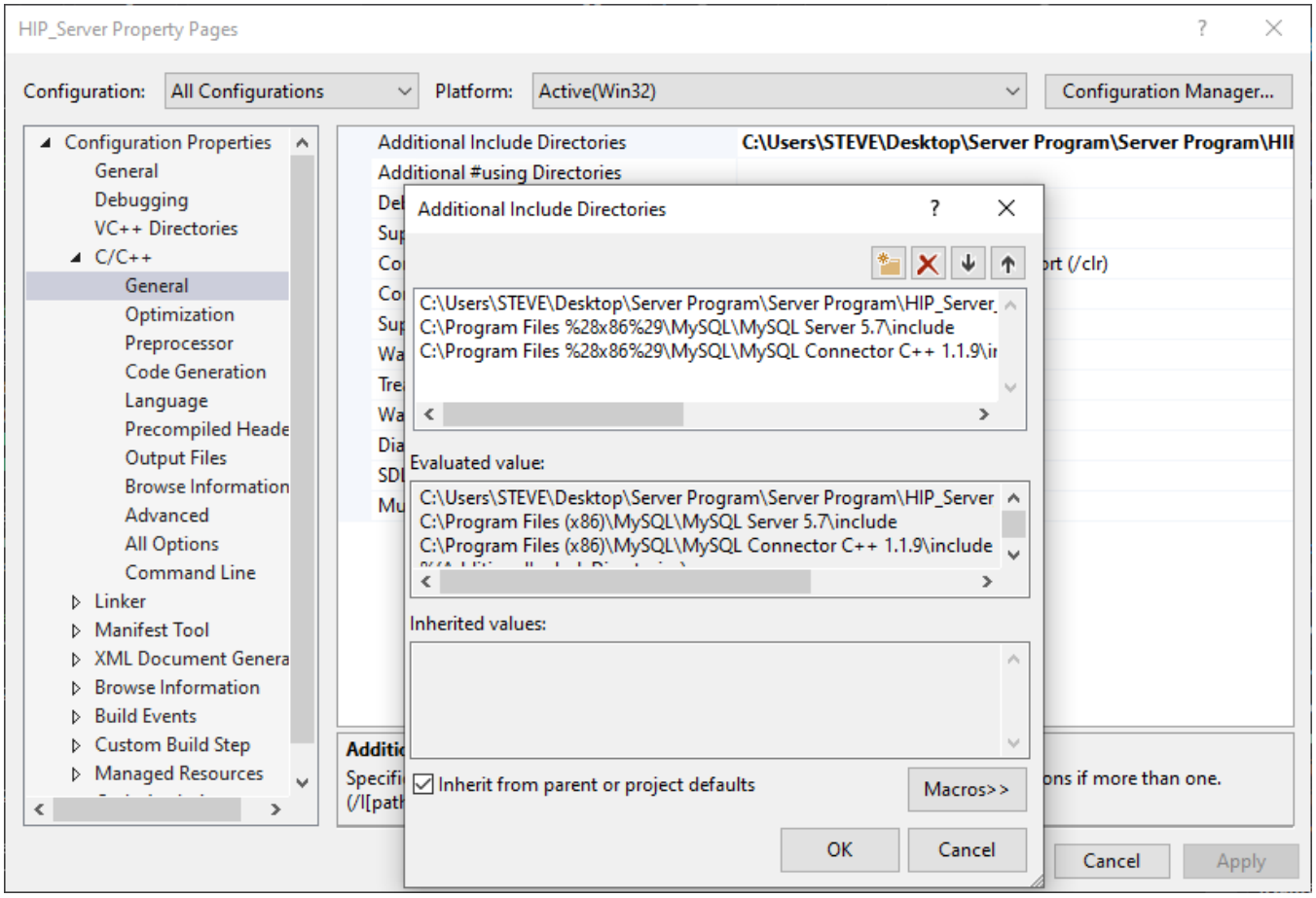

Figure 4. Add additional include directories

Step 2

Then go to the following path:

Linker → General → Additional Library Directories

Add MySQL Server 5.7 lib, MySQL connector 1.1.9 lib, MySQL connector 1.1.9 lib/opt, and project's own lib folder as shown in Figure 5.

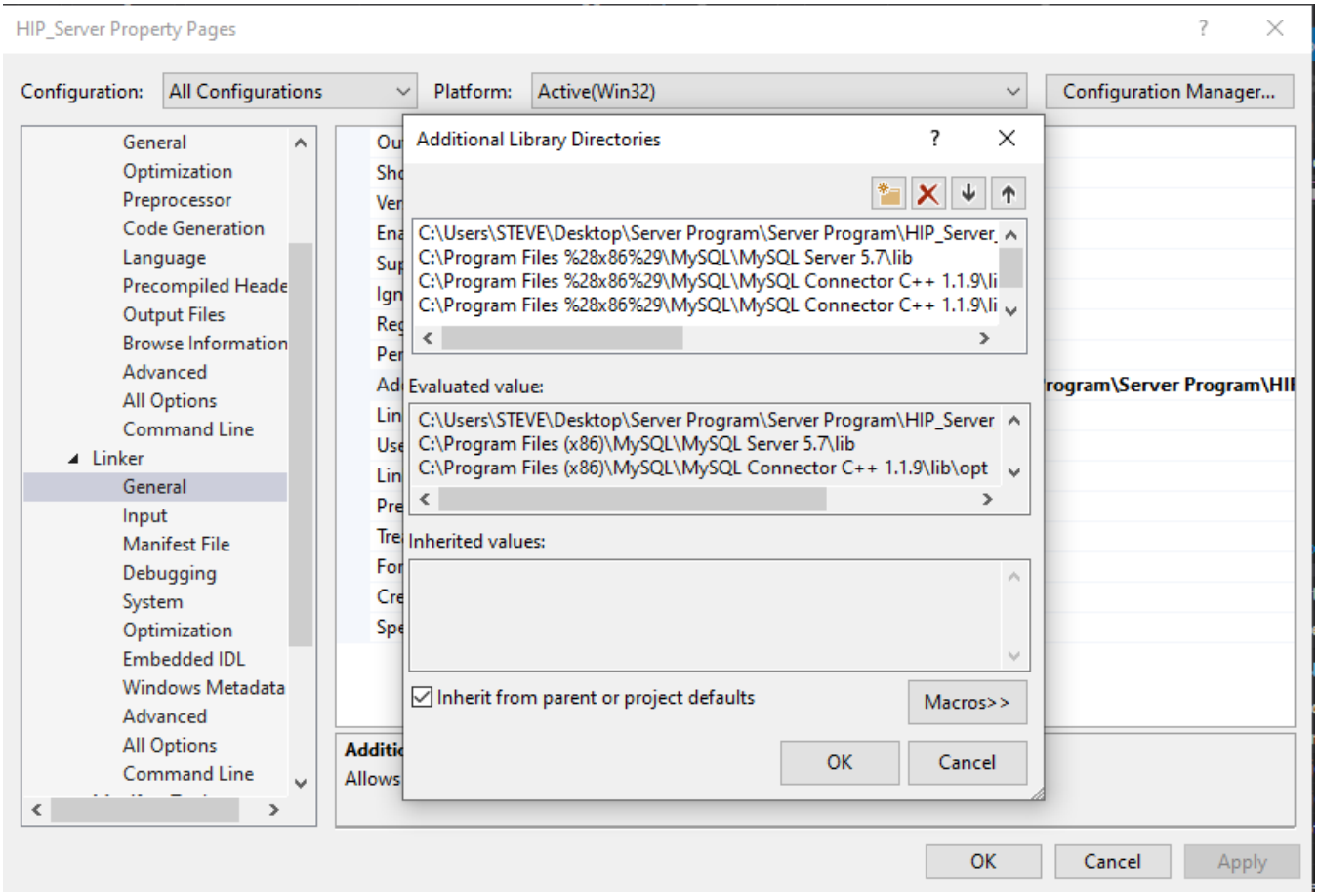

Figure 5. Add additional library directories

Step 3

Then go to the following path:

Linker → input → Additional Dependencies

Add mysqlcppconn.lib, libmysql.lib as Figure 6 shows.

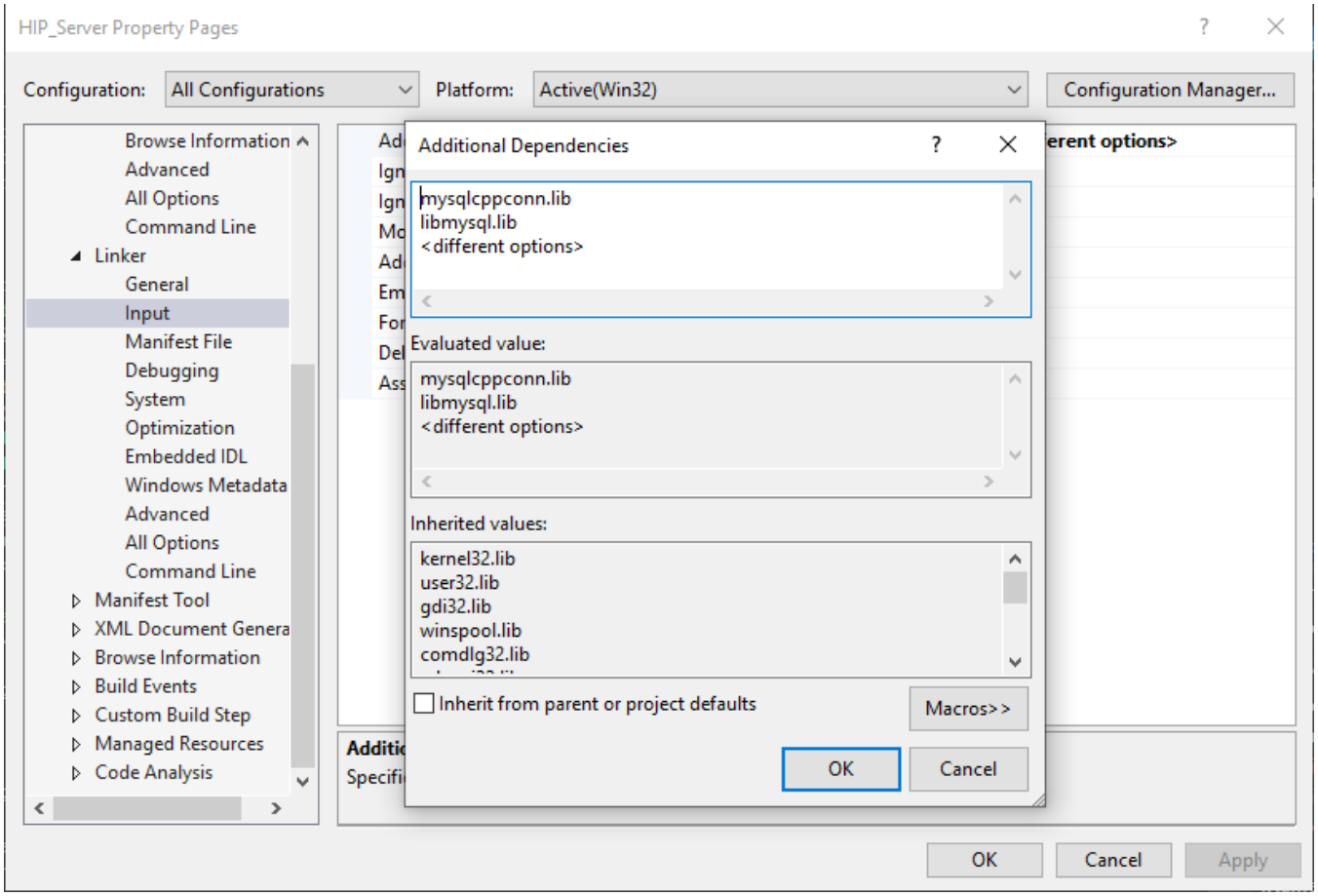

Figure 6. Add additional dependencies

Step 4

Click OK and Apply.

Then copy libmysql.dll from C:\Program Files\MySQL Server 5.7\lib to the executable directory.

After finishing the above four steps, you should now see all the MySQL related code without any compile error.

Database Establishment in MYSQL

Establishing the database is conducted in MySQL through MySQL Workbench. The required MySQL script to establish the database and check each table's data storage status are also provided. Copy those scripts into the MySQL Workbench and execute them.

Create Database

MySQL script:

```
CREATE DATABASE db2020;

USE db2020;
```

Create Tables

MySQL script:

```
CREATE TABLE Sim_Signal (TEST_RUN_ID DOUBLE, Cycle_Length DOUBLE, Cycle_Start_Time VARCHAR(30),
Cycle_Start_Time_s DOUBLE, Sim_Signal_Times_1 DOUBLE, Sim_Signal_Times_2 DOUBLE, Sim_Signal_Times_3
DOUBLE, Sim_Signal_Times_4 DOUBLE, Sim_Signal_Times_5 DOUBLE, Sim_Signal_Times_6 DOUBLE,
Sim_Signal_Times_7 DOUBLE, Sim_Signal_Times_8 DOUBLE);

CREATE TABLE Virtual_Car (TEST_RUN_ID DOUBLE, TIME VARCHAR(30), TIME_MS DOUBLE, Speed DOUBLE,
Link_ID INT, Link_Position DOUBLE, Node_ID INT, Node_Status INT(1), Acceleration DOUBLE);

CREATE TABLE Truck_1 (TEST_RUN_ID DOUBLE, TIME_1 VARCHAR(30), TIME_1_MS DOUBLE, Speed_1 DOUBLE,
Acceleration_1 DOUBLE, Position_1 DOUBLE, Longitude_1 DOUBLE, Latitude_1 DOUBLE);

CREATE TABLE Truck_2 (TEST_RUN_ID DOUBLE, TIME_2 VARCHAR(30),TIME_2_MS DOUBLE, Speed_2 DOUBLE,
Acceleration_2 DOUBLE, Position_2 DOUBLE, Longitude_2 DOUBLE, Latitude_2 DOUBLE);

CREATE TABLE Truck_3 (TEST_RUN_ID DOUBLE, TIME_3 VARCHAR(30), TIME_3_MS DOUBLE, Speed_3 DOUBLE,
Acceleration_3 DOUBLE, Position_3 DOUBLE, Longitude_3 DOUBLE, Latitude_3 DOUBLE);

CREATE TABLE Controller (TEST_RUN_ID DOUBLE, Next_Cycle_Start_Time VARCHAR(30),
Next_Cycle_Start_Time_MS DOUBLE, Intersection_Node_ID INT, Green_Time_1 DOUBLE, Green_Time_2 DOUBLE,
Green_Time_3 DOUBLE, Green_Time_4 DOUBLE, Green_Time_5 DOUBLE, Green_Time_6 DOUBLE, Green_Time_7
DOUBLE, Green_Time_8 DOUBLE, Yellow_Time DOUBLE, All_Red DOUBLE, Start_Phase_1 INT(1), Start_Phase_2
INT(1), Ring_1 INT(3), Ring_2 INT(3));
```

Check Each Table Status

MySQL script:

```
USE db2020;

SELECT * FROM Virtual_Car;

SELECT * FROM Truck_1;

SELECT * FROM Truck_2;

SELECT * FROM Truck_3;

SELECT * FROM Sim_Signal;

SELECT * FROM Controller;
```

Chapter 6. Test Tool User Guide

The following detailed user guide has been provided to make it easier for researchers to use the test tool.

Required Software: Visual Studio 2017, Visual Studio 2013, Aimsun 8.2, MySQL Workbench

Step 1

Check the WIFI properties on your laptop to get the IP address. For example, in Figure 7 the IP address would be shown in the IPv4 address row.

In some cases, user's WIFI IP address might change, to prevent any errors, it should be checked every time when starting the Server Program.

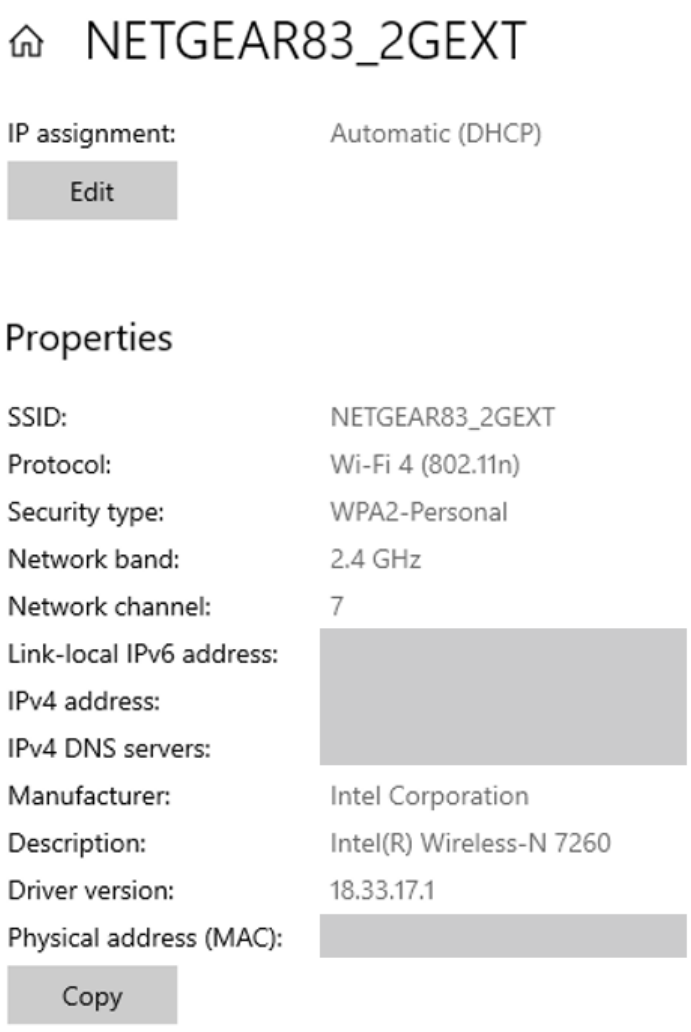

Figure 7. Screenshot of WIFI Properties

Step 2

Open PATH_MODEL.suo in the MicroSDK package, choose the mybehaviorModel.cpp to open. Find the line "#define Server_Address" and revise the default Server_Address to our WIFI address recorded in step 1.

The code before revision can be found in Figure 8. and as Figure 9. shows, the Server_Address has been revised to our WIFI address.

```
// For intersection-in-the-loop test
#define Traffic_Scenario_Start_Time 30 // in seconds,
#define Server_Address "10.142
//#define Server_Address "192.
//#define Server_Address "10.1
#define Server_Port 8800
#define Server_Port_Signal 8900
#define Local_Port 8000
#define Local_Port_Signal 8100
```

Figure 8. Screenshot of the code in mybehaviormodel.cpp before revision

```
// For intersection-in-the-loop test
#define Traffic_Scenario_Start_Time 30 // in seco
#define Server_Address "192.168.
//#define Server_Address "192.16
//#define Server_Address "10.142
#define Server_Port 8800
#define Server_Port_Signal 8900
#define Local_Port 8000
#define Local_Port_Signal 8100
```

Figure 9. Screenshot of the code in mybehaviormodel.cpp after revision

After the revision, rebuild the whole project in order to make sure the simulation software functions as expected.

Step 3

Open Server Program through Visual Studio 2017, then open Server.h file.

Change the lines *user = "root", *password = "path2020" (as shown in Figure 10) to the user's own user name and password established in the MySQL set up procedure.

```
// Zhe: for connection to MySQL
//using namespace std;
MYSQL* conn;
const char *user = "root";
//const char *password = "12345";
const char *password = "path2020";
const char *host = "localhost";
const char *dbname = "db2020";
unsigned int port = 3306;
const char *qstr;
```

Figure 10. Screenshot of the code in Server.h

Step 4

Execute the Server Program through Visual Studio 2017, then a window like the one in Figure 11 below will pop up.

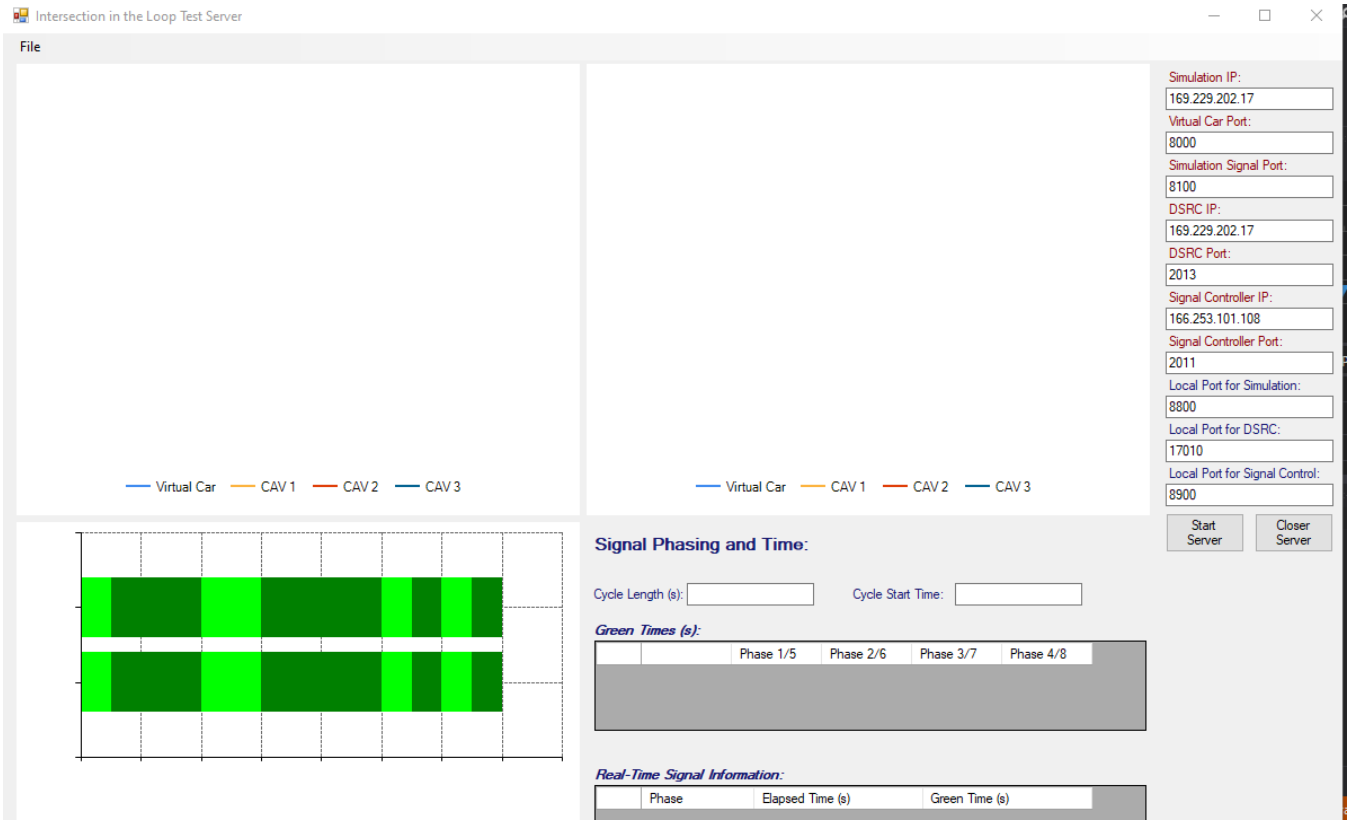

Figure 11. Pop-up window when executing Server Program

Revise the IP address correspondingly. For example, when running the Server Program and the simulation on the same laptop, the user should change the Simulation IP and DSRC IP from default to the user's WIFI IP Address as recorded in Step 1. Example of changing IP address has been shown in Figure 12.

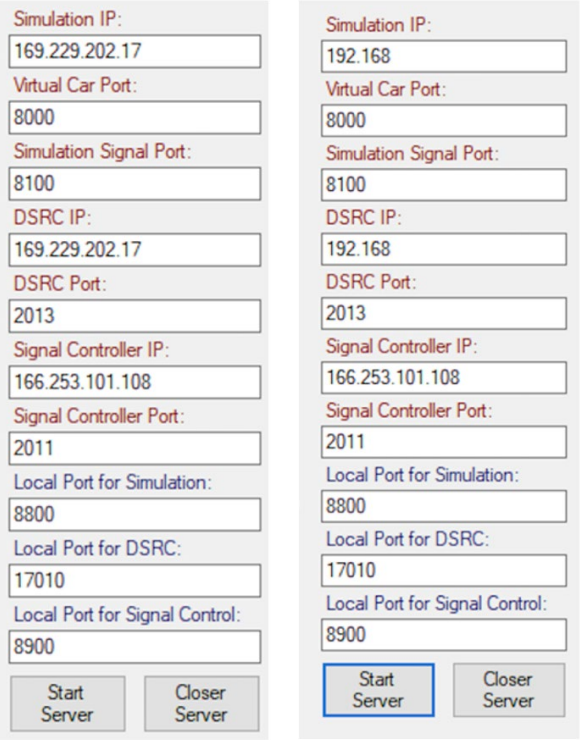

Figure 12. Screenshots of pop-up window. Left: before IP address change; Right: After change

Step 5

Click the "Start Server" button in the pop-up window of the Server Program, then go back to the DOS window to check the controller status.

Three key sentences should be shown in the DOS window, as shown in Figure 13:

a) Successfully connection to Database!
b) Server Ready!
c) Controller Ready!

The first two sentences will be shown at the beginning and when the third sentence appears, the Server Program is ready to communicate with the simulation software and database.

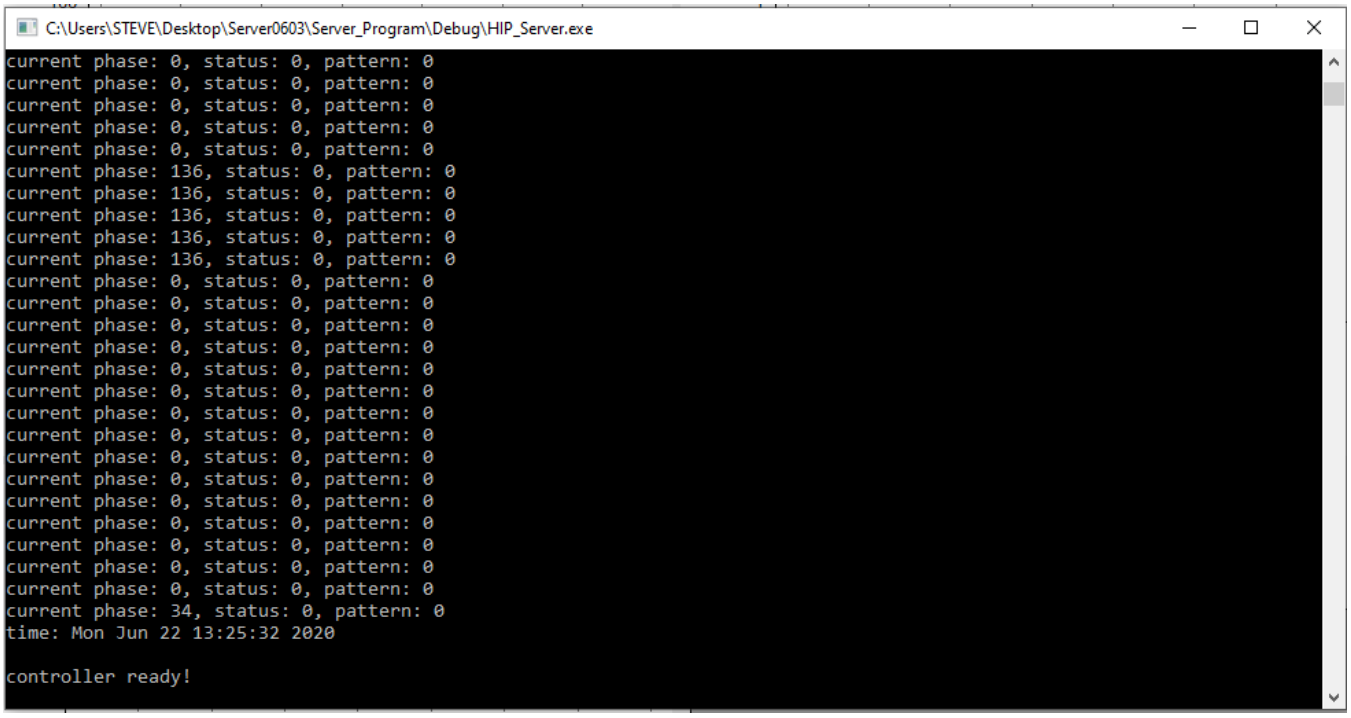

Figure 13. Screenshot of DOS window when Server Program is ready

The user might notice that the DOS window becomes stuck at the line “Server ready!” as shown in Figure 14. This is because the controller is handling another request. Since the controller can only handle requests sequentially, wait until the previous request has been completed.

Figure 14. Screenshot of DOS window when controller is handling another request

When the Server Program is ready, the graphs within the pop-up window should appear. Some straight lines representing the three CAVs should appear on both graphs as shown in Figure 15.

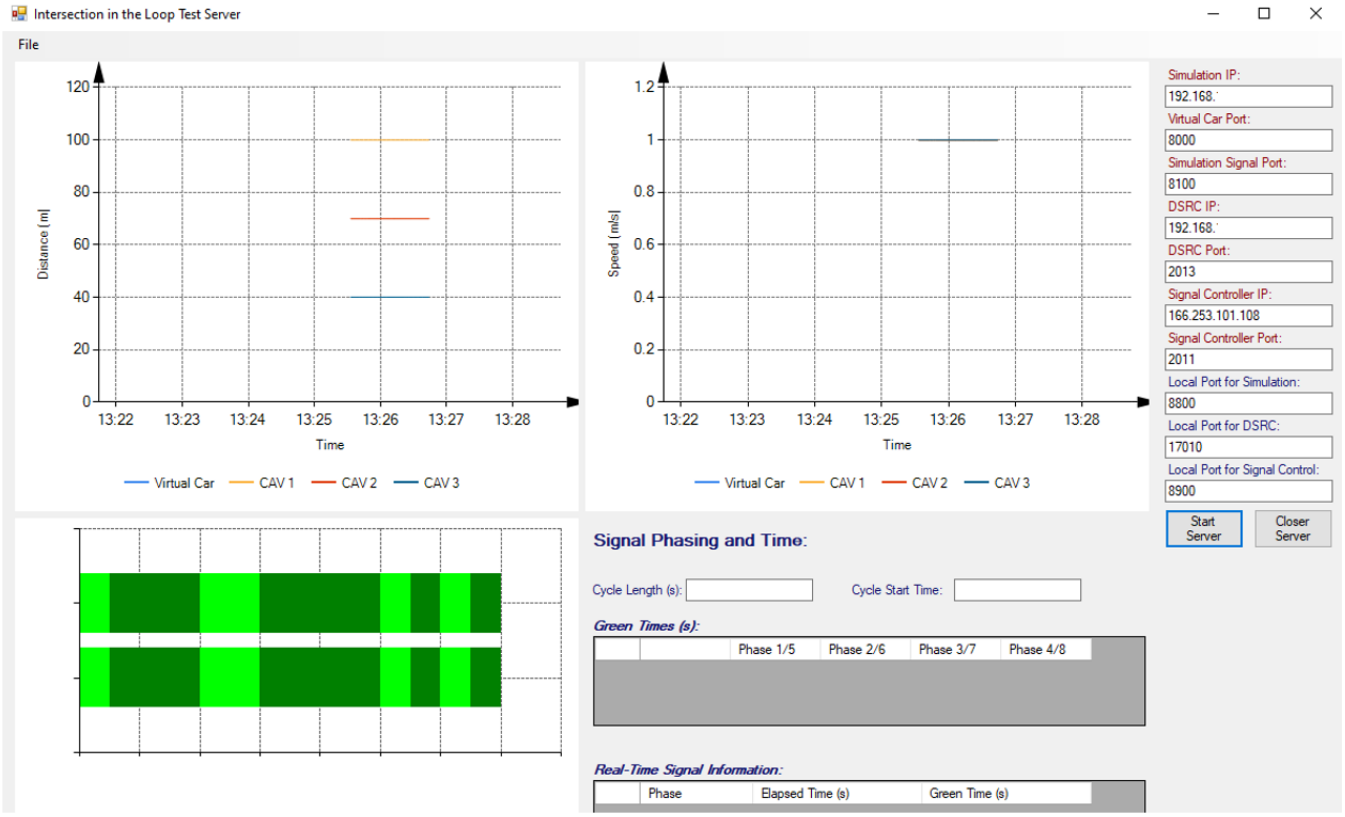

Figure 15. Pop-up Window when Server Program is ready

Step 6

Open the Aimsun 8.2 simulation software and open the simulation file. Then run the selected replication. The vehicle animations can be seen in Figure 16.

The animation might freeze while running the test. It is designed to do this when the virtual test vehicles are synchronizing with the simulated test vehicles.

After starting the simulation in Aimsun, go back to the pop-up window of the Server Program to check the status of the simulation. The in-time signal phasing and time, time-space diagram and time-speed diagram are all shown in the pop-up window. Figure 17 shows the screenshot of the pop-up window when all elements function well.

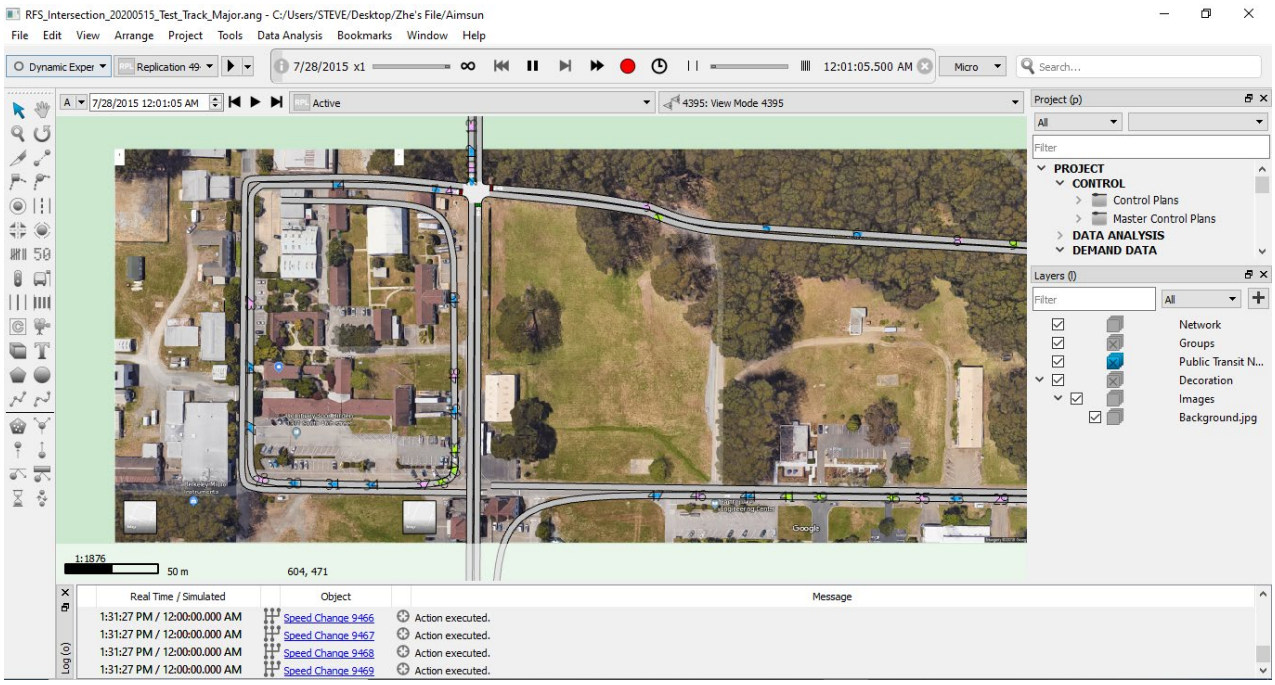

Figure 16. Screenshot of Aimsun

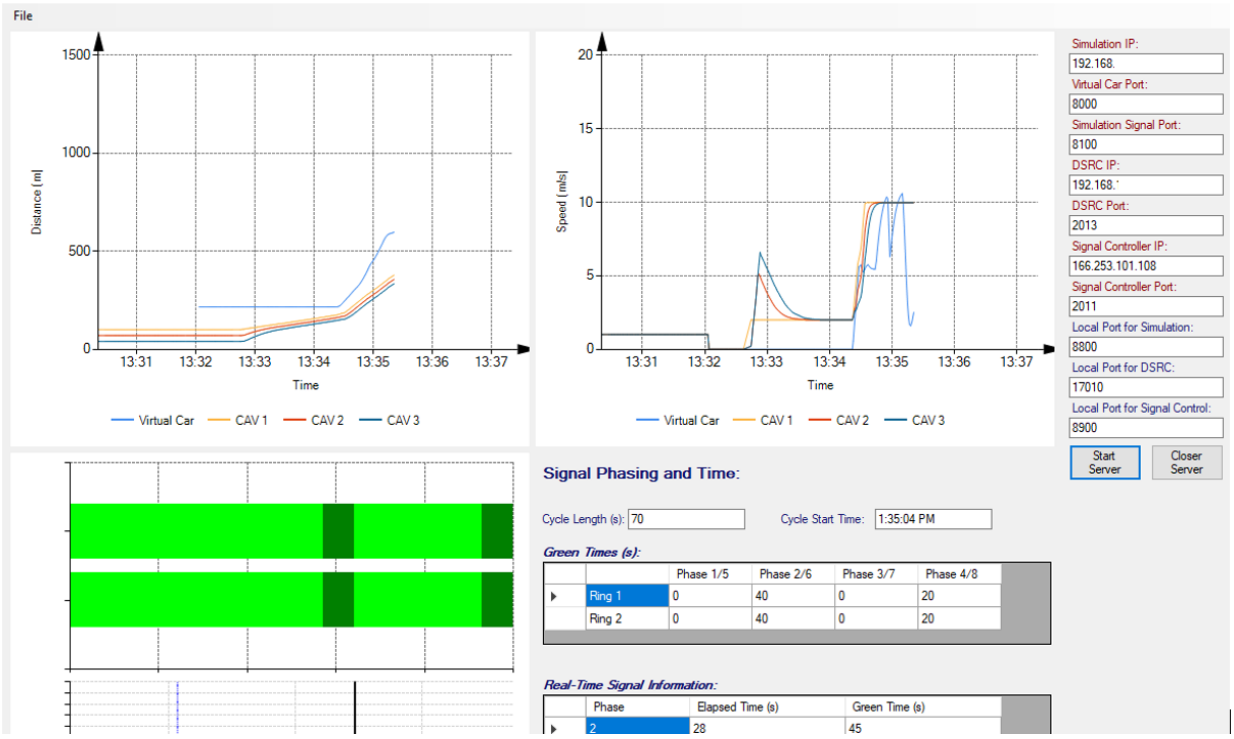

Figure 17. Pop-up window when all elements function well

Step 7

When the simulation and Server Program work well, all the required data is stored simultaneously and automatically in the database built in MySQL. Those data can be checked during the simulation or after the simulation through MySQL Workbench.

As Figure 18 shows, all the data for truck 1 can be found in the Truck_1 table.

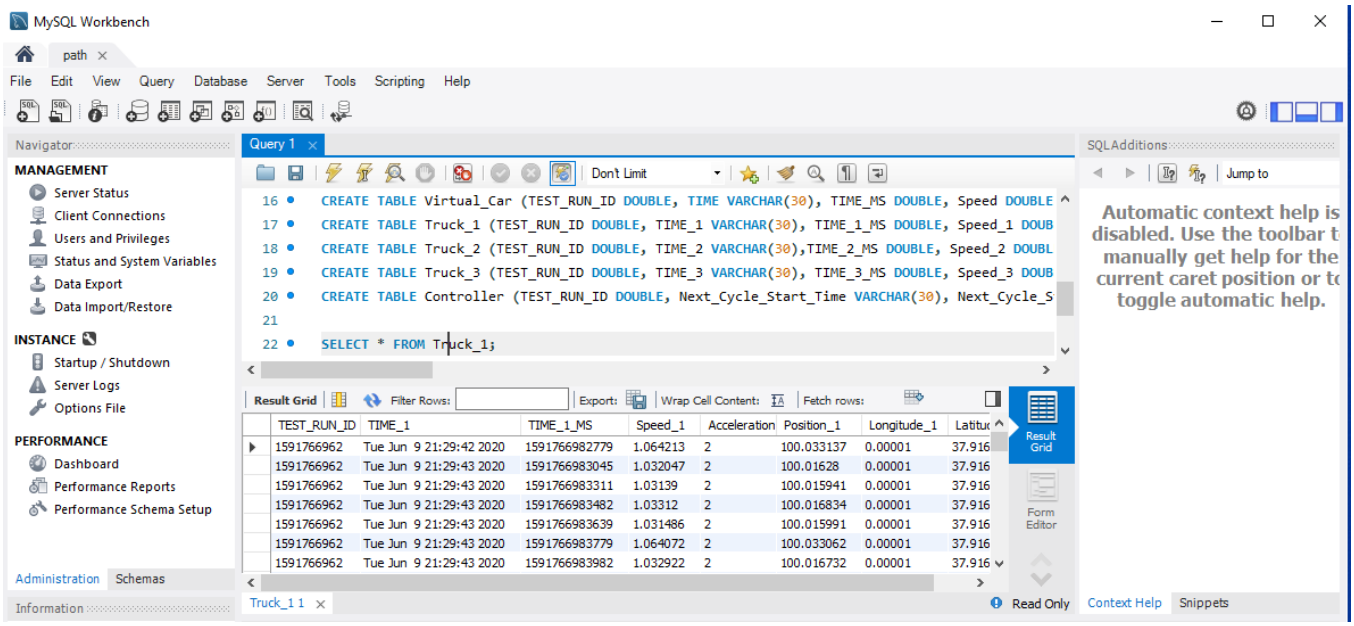

Figure 18. Screenshot of MySQL Workbench

Step 8

Simple data statistics such as number of data rows, maximum value, minimum value, and average value can be obtained by SQL commands in MySQL Workbench. However, for further data analysis and aggregated data plots, extracting the data needed in a common format will be better and more user friendly.

Exporting data from MySQL Workbench is very simple, first select the particular data needed (e.g. SELECT * FROM Truck_1 WHERE TEST_RUN_ID = 1591766962;). When the result is shown in the result grid, click the “Export” button above the result grid, as indicated by the red circle in Figure 19. Then select the desired file type and storing directory as shown in Figure 20.

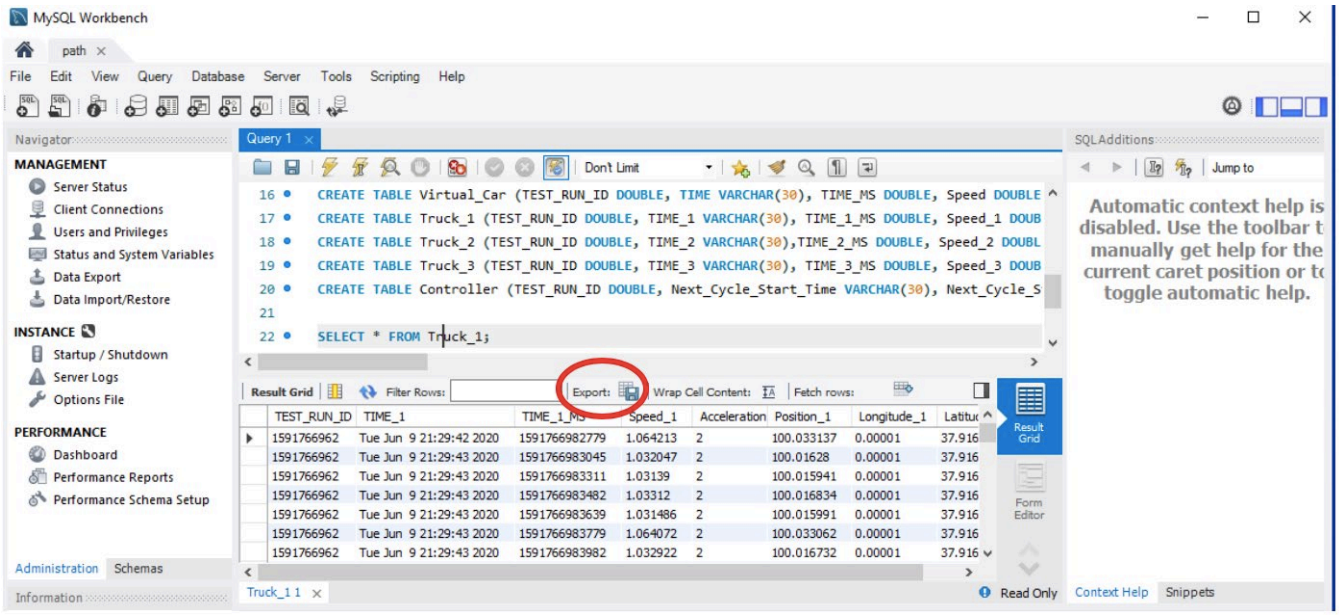

Figure 19. Export function in MySQL Workbench

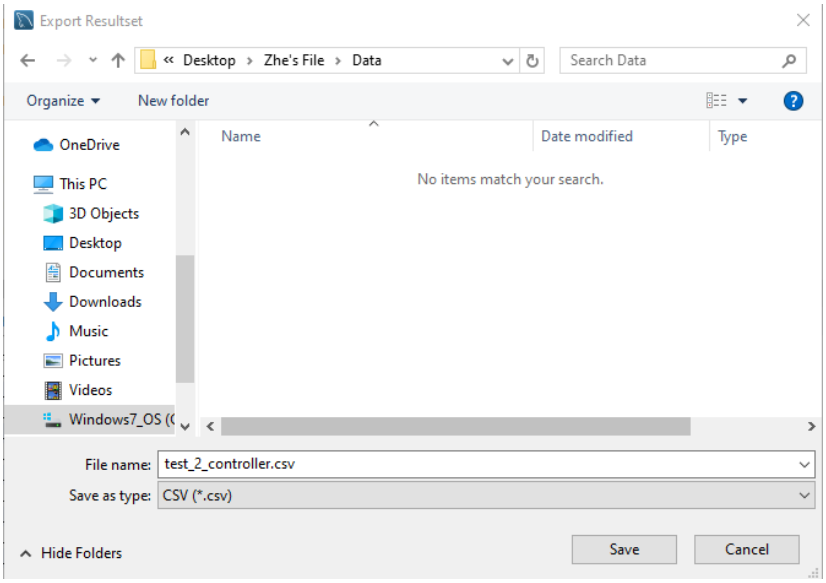

Figure 20. Export directory and file type

Chapter 7. Example CAV Test Experiment Analysis

Experiment Design

In order to provide a guide for other research teams to easily use the test tool that we developed, two example test experiments were designed and conducted. The two cases have the same simulation road network, based on the physical layout of the test track at UC Berkeley's Richmond Field Station (RFS). All the parameters for the simulation are the same except one case used the default fixed time signal control plan, and the other used the cooperative signal control plan, which will change cycle length corresponding to different traffic status.

For ease of comparison, two cases will be executed for same travel distance, which is 100 loops of the test track. Since the average speed of CAVs may varies in the two cases, the total travel time is expected to be different. Our database is designed to record vehicle data every 0.1s, so the number of records will be different among the two cases.

In the example tests, the research team used a computer program instead of physical CAVs to generate the test vehicle data. The program implemented a vehicle dynamics model to depict the movements of test CAVs in response to the virtual traffic. The vehicle program could communicate with the traffic simulation via the same communication channels a real test CAV would use. With such an experiment setting, we could investigate a complete data generation and storage process without incurring high test cost. The test program generates vehicle speed, acceleration, and position data points in 10 Hz. Those data points were sent to the presented database tool via encoded messages. The messages followed the Basic Safety Message (BSM) format defined in the J2735 standard. In this study, a BSM contained a standard core message and a customized message that stores data fields required by the test vehicles. The BSMs were encoded into binary streams so that the communication devices could send or receive them properly. The encoder and decoder use the Abstract Syntax Notation One (ASN.1) interface to serialize the messages.

The traffic simulation provided virtual traffic streams in which the test CAVs could interact with simulated vehicles. The simulation used rectangles as placeholders to represent the test CAVs in the simulation graphic. At each simulation interval, the simulation received the location, speed, and acceleration from the test CAVs. This information was adopted to specify the speed and location of the placeholders in the simulated environment. Afterward, the remaining simulated vehicles updated their movements with the new status of the placeholders considered. In the meantime, the simulation also shared the location, speed, and acceleration of the simulated vehicle that virtually leads the first test CAV. The information was the basis for the first CAV to update the movements. The traffic simulation, test CAVs, and the traffic intersection controller needed to synchronize their update intervals. In a test, the simulation and the intersection controller would start first. The real-time simulation first generated a virtual traffic stream in simulation time. Under the same update speed, the intersection controller would compute signal plans based on the virtual traffic stream. After those warm-up steps were completed, the two systems ran in real-time. To synchronize the real-time update intervals of the simulation and the intersection controller, the simulation would pause at the end of a signal cycle to wait for the real-world traffic controller to finish the current cycle. The simulation and the intersection controller then started updating at the same time when a new cycle began. At this time point, the test CAVs would join the test by starting to send and receive messages. This completed the synchronization process for the three systems.

Test Procedure

Test Setup

Follow the User Guide in Chapter 6 to export data to record the TEST_RUN_ID from the DOS window when starting the Server Program.

The only difference between the two cases is the signal control plan. The case with a TEST_RUN_ID of 1591851416 is the one using a cooperative signal control plan, the other case with TEST_RUN_ID of 1591855836 is the one using a fixed time signal control plan.

Processing

The test tool will automatically save the real-time control data from the simulation, signal controller and test CAVs. As Figure 21 indicates, the data collected during the experiment can be checked in the MySQL database.

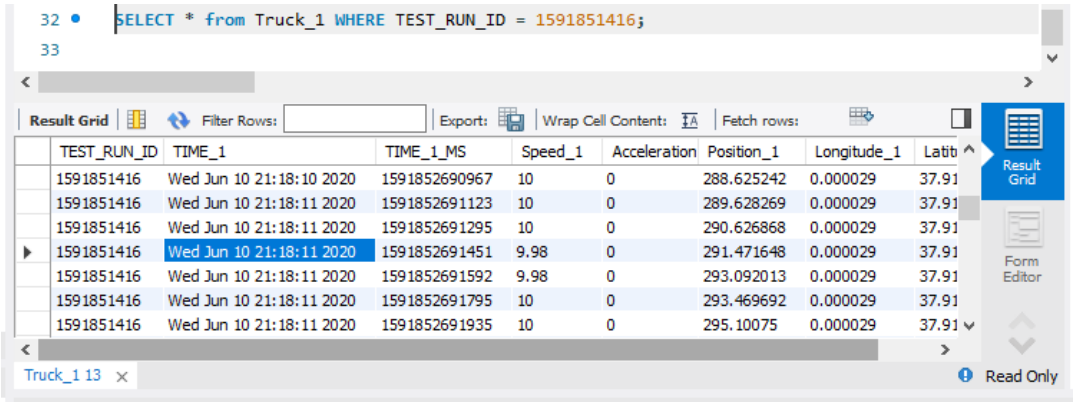

Figure 21. Truck_1 data checked in the process of the experiment

Test Completed

The test tool can also pull historical data from each test system after a test run is completed. As shown in Step 8 of the User Guide in Chapter 6, MySQL Workbench can export the desired data table to the file format needed. This approach is mainly used to obtain the performance data sets from each test system.

Data Analysis

Through MySQL

Some simple statistics can be obtained through SQL commands in MySQL Workbench. The number of data rows, maximum, minimum and average of a particular metric can be easily obtained through a few command lines.

As Figure 22 – 25 show, Truck_1 Table of Test 1591851416 has 24926 total rows of data, the maximum speed of truck_1 in that test is 10.21 m/s, the minimum speed is 0 m/s, and the average speed is 4.62 m/s.

```
30 •  SELECT COUNT(TIME_1) from Truck_1 WHERE TEST_RUN_ID = 1591851416;
31 •  SELECT * from Truck_1 WHERE TEST_RUN_ID = 1591851416;
32
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

COUNT(TIME_1)
24926

Figure 22. "Count" functionality in MySQL

```
29 •  SELECT MAX(Speed_1) from Truck_1 WHERE TEST_RUN_ID = 1591851416;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

MAX(Speed_1)
10.205796

Figure 23. "Maximum" functionality in MySQL

```
29 •  SELECT MIN(Speed_1) from Truck_1 WHERE TEST_RUN_ID = 1591851416;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

MIN(Speed_1)
0

Figure 24. "Minimum" functionality in MySQL

```
29 •  SELECT SUM(Speed_1)/COUNT(Speed_1) from Truck_1 WHERE TEST_RUN_ID = 1591851416;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

SUM(Speed_1)/COUNT(Speed_1)
4.623430211675929

Figure 25. "Average" functionality in MySQL

Further Analysis through Python

The export function in MySQL provides more flexibility to analyze the data. With the .csv files extracted from the database, we can use an appropriate tool to conduct further analysis. Here we used Python as an example tool.

In the original test tool, we could only view a temporary 5-minute time-space diagram and time-speed diagram. Now we can obtain aggregated ones that provide a better view of the whole process. Figure 26 and 27 are the aggregated time-space diagram and time-speed diagram using Truck_1 data from Test_1 as an example. Other similar plots can also be easily generated.

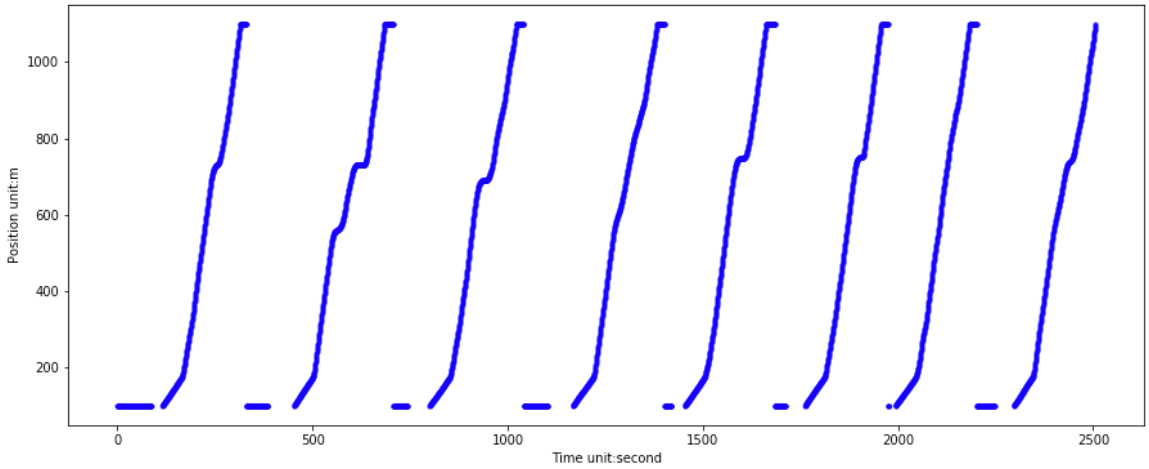

Figure 26. Aggregated Time-Space Diagram of Truck_1 in Test #1

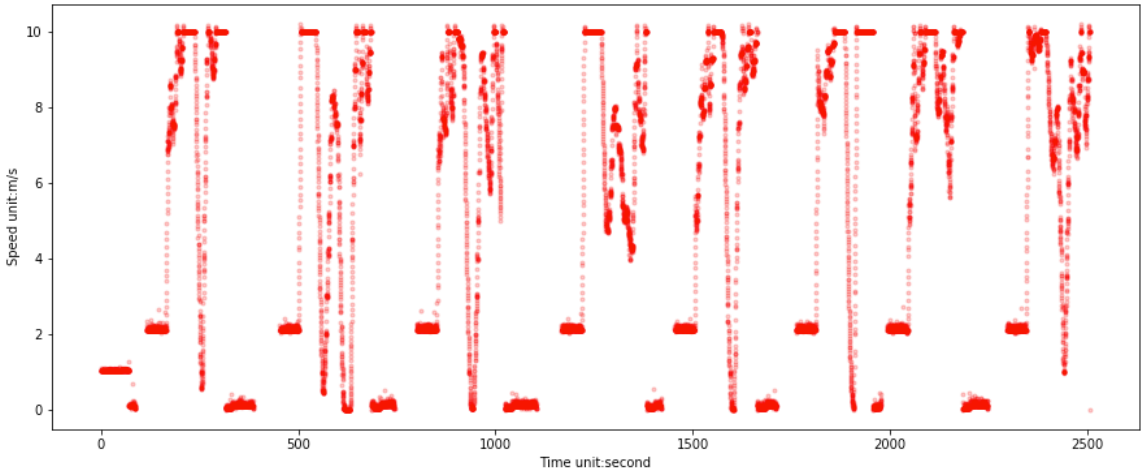

Figure 27. Aggregated Time-Speed Diagram of Truck_1 in Test #1

Besides the aggregated plots, our test tool can be used to further study specific metrics and compare the performance of different targets and different tests.

For example, we may want to compare trucks' performance near an intersection. Data preprocessing including extracting the data with position between 400-800 (near intersection area) and removing abnormal data (e.g., records with strange acceleration) was conducted.

After data preprocessing, in the cooperative signal control case, truck #1 has 71919 records, truck #2 has 72910 records, truck #3 has 73267 records; in fixed signal case, truck #1 has 95538 records, truck #2 has 96139 records, and truck #3 has 96056 records. The two cases have different numbers of records and the three trucks' number of records vary from each other, which is in line with our expectation since each truck in a different case may have different average speed.

Some basic statistics in terms of speed and acceleration are listed in Table 8 and Table 9. It is very obvious that the average speed in the cooperative signal case is larger than that in the fixed signal case. As for acceleration, just producing basic statistics cannot provide us with explicit insight, we need deeper analysis.

Table 8. Basic Statistics for Speed

	Cooperative			Fixed		
Unit: m/s	Truck_1	Truck_2	Truck_3	Truck_1	Truck_2	Truck_3
Maximum	10.222	10.157	10.148	10.227	10.139	10.132
Minimum	0	0	0	0	0	0
Average	5.551	5.475	5.447	4.178	4.151	4.154

Table 9. Basic Statistics for Acceleration

	Cooperative			Fixed		
Unit: m/s^2	Truck_1	Truck_2	Truck_3	Truck_1	Truck_2	Truck_3
Maximum	2	2	2	2	2	2
Minimum	-3	-3	-3	-3	-3	-3
Average	-0.0220	-0.0074	-0.0220	-0.0066	0.0031	-0.0084

Besides basic statistics, histogram plots can directly show the target metric's distribution and provide us with more explicit comparisons.

For example, in Figure 28, we plotted the three trucks' speed distribution in the same histogram. By looking at the speed distribution, we can see the similarity of the different trucks' behavior. The left plot represents Test #1, (cooperative signal control plan), and the right plot Test #2 (fixed signal control plan).

The three trucks have different speed distributions. The speed of truck #1 is fluctuates more than truck #2, and truck #3 has the most "stable" speed when looking at the distribution in extreme values.

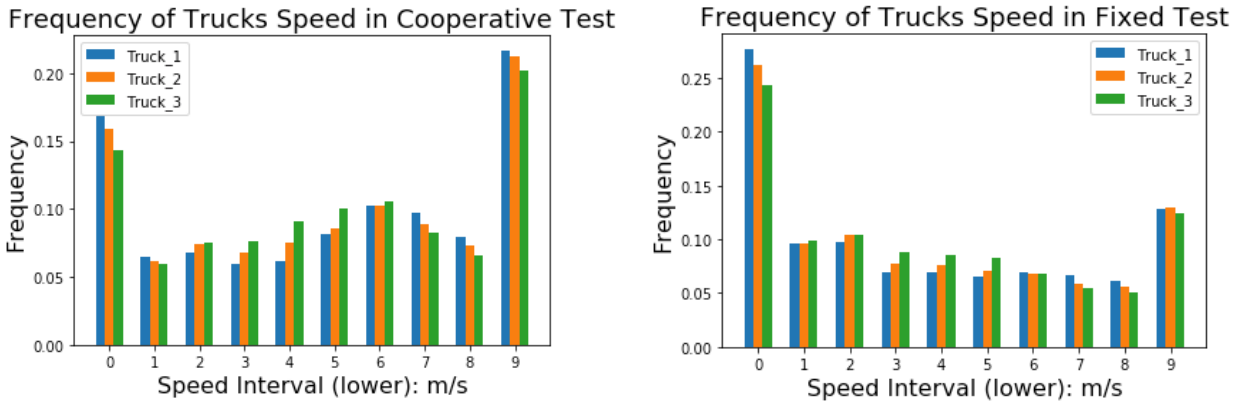

Figure 28. Comparison of truck's speed in the same test. Left: Test #1; Right: Test #2.

To further compare the different tests, we also generated plots for all three trucks as shown in Figure 29. The upper left one compares truck #1's speed in the two tests with the blue bar representing the speed in Test #1 and the orange bar representing the speed in Test #2. Similarly, the upper right plot compares truck #2's speed and the bottom plot compares truck #3's speed.

Although all three trucks performed differently, they show similar trends. In Test #1 (cooperative signal control plan), all vehicles registered more frequent speeds greater than 5 m/s, while in Test #2, (fixed signal control plan), they had more instances of speeds in the lower speed ranges (less than 5 meters per second).

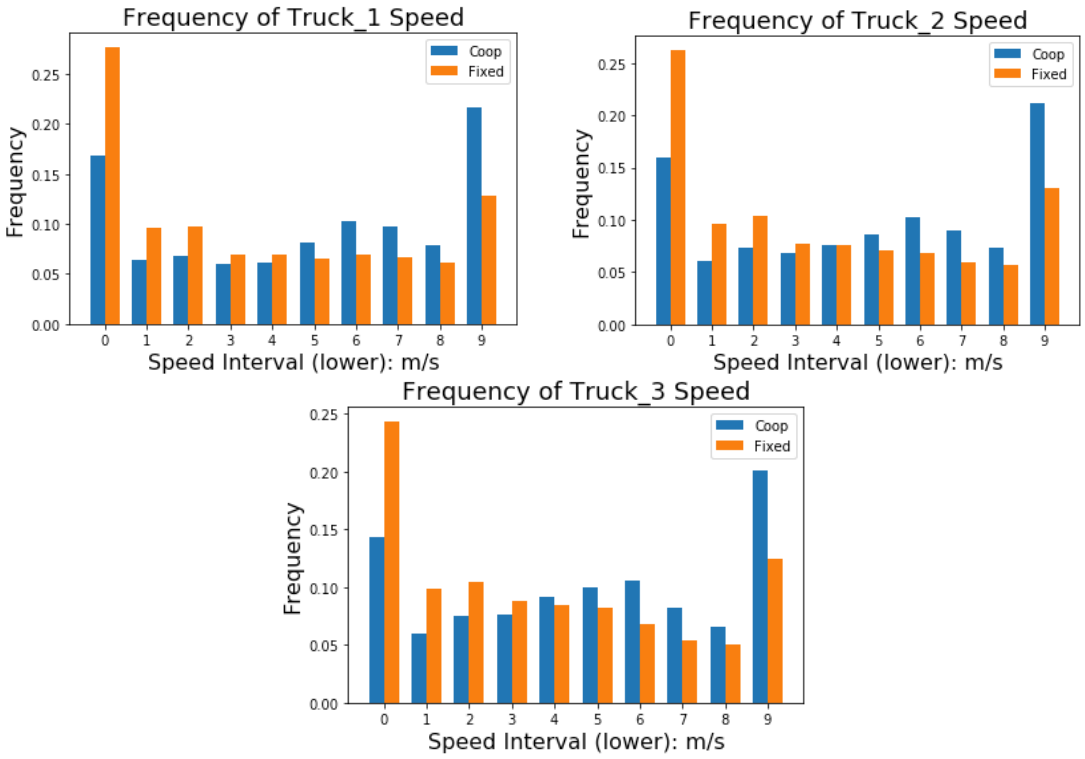

Figure 29. Comparison of truck's speed in different tests

Similarly, we can also compare the "Acceleration" metric. It is worth mentioning that we often care more about the accumulated time cost of acceleration, since it indicates a passenger's comfort level in the car. Therefore, instead of comparing the frequency of acceleration in each interval, we will compare the average accumulated time spent during each acceleration interval.

It is necessary to be aware of the difference of average travel time between the two cases if we want to compare the acceleration in term of accumulated time. Table 10 shows each truck's average travel time in each case. The cooperative signal case has significantly shorter average travel time than the fixed signal case.

Table 10. Comparison of Average Travel Time

	Cooperative Case	Fixed Case
Truck_1	79.47 s	106.09 s
Truck_2	79.66 s	105.15 s
Truck_3	80.05 s	105.06 s

As for the acceleration, truck #1, the lead vehicle of the three CAVs, behaves very differently from the following vehicles. According to Figure 30, truck #1 spent the most time accelerating and had more instances of acceleration that were larger than 1 m/s^2. However, the following vehicles, truck #2 and truck #3 had a more balanced distribution, spending some time on deceleration.

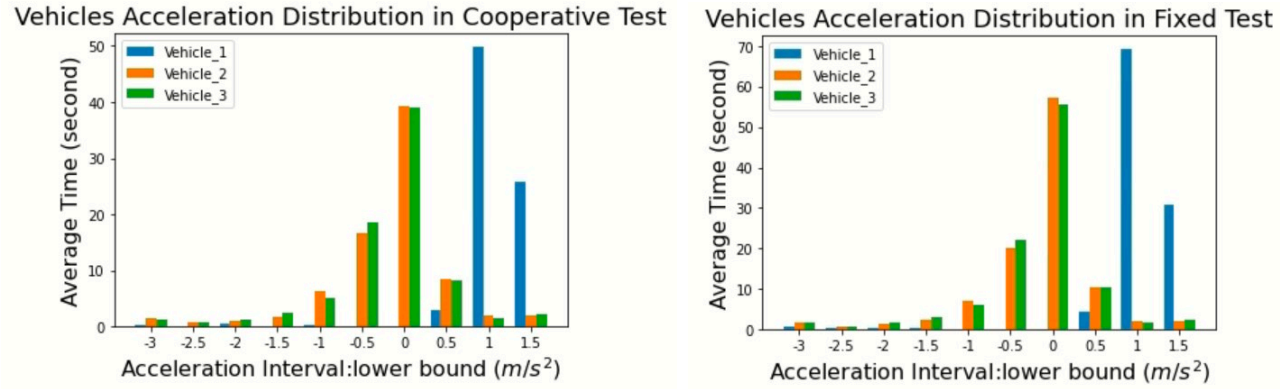

Figure 30. Comparison of truck's acceleration in the same test. Left: Test #1; Right: Test #2.

Figure 31 compares the acceleration for each truck between the two tests; the upper left one is for truck #1's acceleration with the blue bar representing the acceleration in Test #1 and the orange bar representing the acceleration in Test #2. The upper right plot is for truck #2's acceleration and the bottom plot is for truck #3's.

The acceleration distribution trend of the two tests are similar, however, in Test #1 (cooperative signal control plan), the trucks have more duration in nearly every interval than in Test #2, (fixed signal control plan).

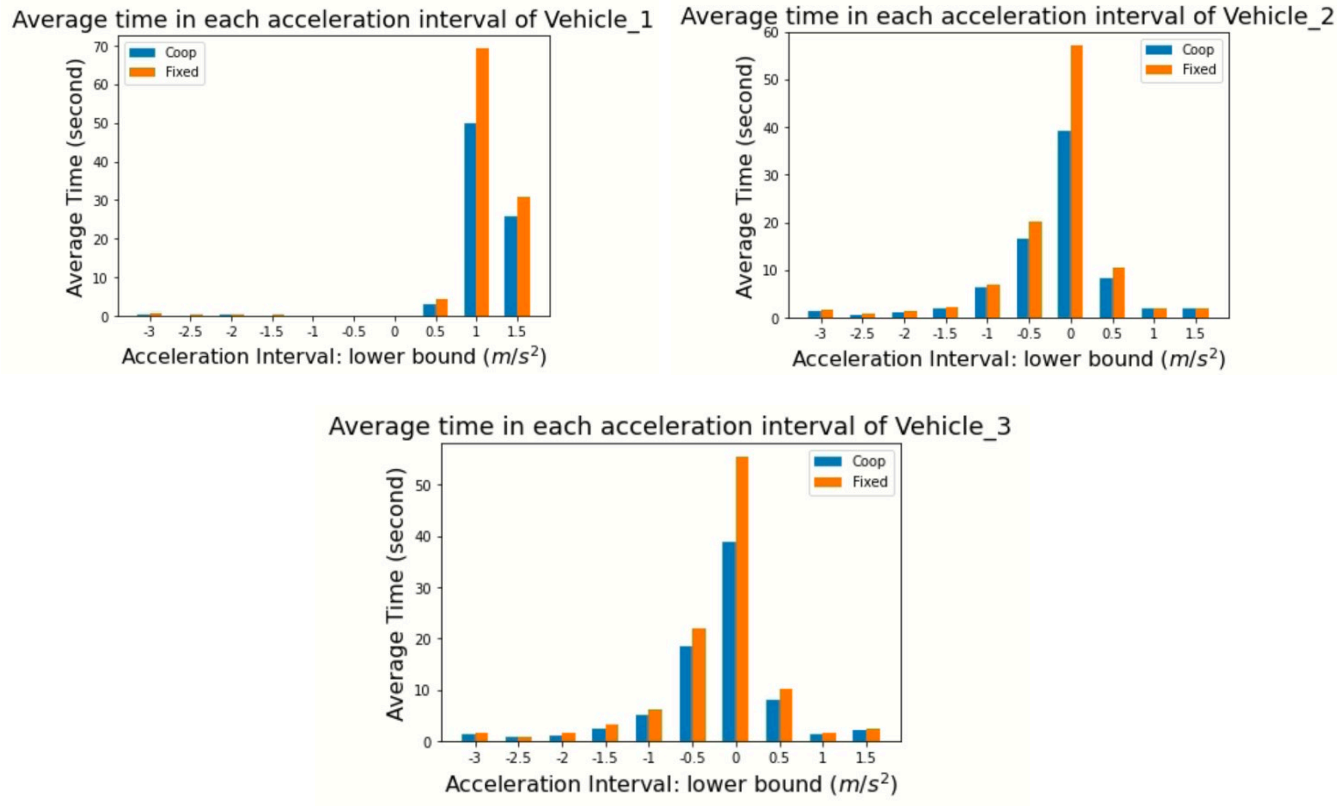

Figure 31. Comparison of truck's acceleration in two tests

To further compare trucks' performance in two cases, we can measure their average travel time delay. We used 10 m/s as the free flow speed to calculate the estimated travel time and the difference in the average travel time in each case. The result of each truck's average travel delay in two cases is shown in Table 11.

Table 11. Comparison of Average Travel Delay

	Cooperative Case	Fixed Case
Truck_1	39.47 s	66.09 s
Truck_2	39.66 s	65.15 s
Truck_3	40.05 s	65.06 s

It is not surprising that the trucks in the cooperative signal case has significantly less delay than those in the fixed case since Table 10 has already shown that trucks in test #1 would have less travel time.

When evaluating CAV's performance, we always regard number of stops and average stop duration as important metrics. Stops are closely relevant to fuel consumption and emission, moreover, they can also represent passengers' comfort level.

With careful analysis of trucks' trajectory data, we defined a stop as a period during which the vehicle remains at a speed lower than 0.3 m/s for longer than 3 seconds. With this criterion, we calculated the number of stops as shown in Table 12. From the previous results we knew that trucks in the cooperative signal case would have fewer stops than those in the fixed signal case. Moreover, the leading vehicle has more stops than the following vehicles.

Table 12. Comparison of Number of Stops

	Cooperative Case	Fixed Case
Truck_1	70	133
Truck_2	62	124
Truck_3	61	119

As for the average stop duration, we can obtain different metrics with different denominators. Table 13 shows the average stop duration per stop, which was calculated by the total stop duration divided by the number of stops. Obviously, trucks in cooperative signal case would have an average stop duration about 2 second less than those in fixed signal case. It is also very interesting that each vehicle would have significant differences on this metric: the leading vehicle would have the longest stop duration while the last following vehicle would have the shortest.

Table 13. Comparison of Average Stop Duration (per stop)

	Cooperative Case	Fixed Case
Truck_1	15.408 s	17.986 s
Truck_2	13.542 s	15.339 s
Truck_3	11.604 s	13.882 s

Table 14 shows the average stop duration per loop, which was calculated by the total stop duration divided by the number of loops (in this experiment, it is a constant number, 100). More obvious differences were shown by using this metric, not only between the two cases, but also between different trucks. This was because the leading vehicle in the fixed signal case had longer stop duration and more stops.

Table 14. Comparison of Average Stop Duration (per loop)

	Cooperative Case	Fixed Case
Truck_1	10.786 s	23.922 s
Truck_2	8.396 s	19.020 s
Truck_3	7.078 s	16.520 s

The benefits of a cooperative signal control plan are obvious. Based on the detailed comparison of speed, acceleration, travel delay, number of stops and average stop duration, we can conclude that the vehicles in the cooperative signal case travel at higher and more constant speeds, less travel delay, and fewer and shorter stops.

Chapter 8. Conclusion

In this research, we completed and improved an existing HIL test tool by adding user-friendly data management functions. The HIL test tool can coordinate the execution of different test systems including test CAVs, communication systems, traffic controllers and traffic flow models. Moreover, it can manage data flow by facilitating data transfer via different communication channels. In addition, the HIL test tool will save the large data sets generated by different test systems automatically when it receives the real-time data and can help pull historical data from each test system for further analysis.

The added database provides more convenience for researchers to debug and analyze test run results. It also offers more flexibility for research teams to generate the results that they need by using historical data from any test system. The HIL test tool also supports pulling historical data in several common file types, which will enable researchers to export the datasets to other tools.

We have also provided potential users with a detailed procedure for designing and establishing a database. The process is beneficial for extending the data management functions to other test systems and will enable users to develop and improve CAV database systems by using the Entity-Relationship diagram to sort out the necessary attributes.

The detailed setup guide and specific user guide presented in this project makes our test tool more user-friendly, and allows researchers to concentrate on developing and evaluating target CAV systems without the time-consuming necessity of establishing an HIL testbed and database.

Two example CAV tests were conducted to demonstrate the detailed data collection and performance evaluation procedure. The examples not only demonstrate the functionalities of the test tool but will also assist potential users easily adapt the tool for their customized tests. This can help other California institutes design CAV experiments with the HIL test tool.

The HIL test tool enables easy data collection and analysis for CAV systems. It will help researchers use existing traffic simulation tools, traffic signal controllers and CAV control algorithms in their HIL tests, even if they are not experts in traffic flow modeling or vehicle dynamics control. It can greatly streamline the CAV data collection and quality. This will accelerate the implementation and evaluation of research outcomes for broader applications in the State of California and will ultimately make California the leading state in CAV technologies deployment.

References

[1] Liu, H., Kan, X. D., Shladover, S. E., Lu, X. Y., & Ferlis, R. E. Modeling impacts of cooperative adaptive cruise control on mixed traffic flow in multi-lane freeway facilities. Transportation Research Part C: Emerging Technologies, 95, 2018. 261-279.

[2] Bacic, M. On hardware-in-the-loop simulation. In *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005. Pp. 3194-3198

[3] Corbellini, A., Mateos, C., Zunino, A., Godoy, D. and Schiaffino, S. Persisting big-data: The NoSQL landscape. *Information Systems*, 2017.*63*, pp.1-23.

[4] Lee, K.K.Y., Tang, W.C. and Choi, K.S. Alternatives to relational database: comparison of NoSQL and XML approaches for clinical data storage. *Computer methods and programs in biomedicine*, *110*(1), 2013. Pp.99-109.

[5] Padhy, R.P., Patra, M.R. and Satapathy, S.C. RDBMS to NoSQL: reviewing some next-generation non-relational database's. *International Journal of Advanced Engineering Science and Technologies*, *11*(1), 2011. Pp.15-30.

[6] Fraczek, K. and Plechawska-Wojcik, M. Comparative analysis of relational and non-relational databases in the context of performance in web applications. In *International Conference: Beyond Databases, Architectures and Structures*. 2017. Pp. 153-164

[7] Kuyumdzhiev, I. Comparing backup and restore efficiency in MySQL, MS SQL Server and MongoDB. *International Multidisciplinary Scientific GeoConference: SGEM*, *19*(2.1), 2019. Pp.167-174.

[8] Sharma, M., Sharma, V.D. and Bundele, M.M, November. Performance analysis of RDBMS and No SQL Databases: PostgreSQL, MongoDB and Neo4j. In *2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)* 2018. pp. 1-5.

[9] Aghi, R., Mehta, S., Chauhan, R., Chaudhary, S. and Bohra, N. A comprehensive comparison of SQL and MongoDB databases. *International Journal of Scientific and Research Publications*, *5*(2), 2015. pp.1-3.

Appendix

Database Implementation in HIL Test Tool

New Functions

Createconnection

Syntax

void Server_HIP::createconnection()

Description

This function creates the connection between our Server Program and MySQL Server. It will print out information about whether the connection to the MySQL Server has been successfully connected to.

Inputs Arguments

None

Output Arguments

None for function itself.

Print “Successfully connection to Database!” if the connection succeeds, and “Connection to database has failed” if the connection is not successful.

Test_Start_Time

Syntax

int Server_HIP::Test_Start_Time()

Description

This function helps generate the Test Run ID. It will record the start time when the user starts the Server Program and then calculate the difference from 1970-1-1 0:0:0. The time difference will be converted into seconds and used as the Test Run ID.

Inputs Arguments

None

Output Arguments

An integer, which is the time difference in seconds.

Insertdata

Syntax

Void Server_HIP::insertdata(MYSQL * conn, const char *qstr)

Description

This function utilizes the API function of MySQL and will help inserting the data row into the database.

Inputs Arguments

MYSQL * conn: a variable to identify the connection of the Server Program and MySQL Server.

const char *qstr: an insertion command.

Output Arguments

None

Displaydata

Syntax

Void Server_HIP::displaydata(MYSQL * conn)

Description

This function helps developers to debug and show certain collected data values in the Server Program.

Inputs Arguments

MYSQL * conn: a variable to identify the connection of the Server Program and MySQL Server.

Output Arguments

None for the function itself but it will display the data row in the DOS window.

Deletedata

Syntax

Void Server_HIP::deletedata(MYSQL * conn, const char *qstr)

Description

This function helps researchers to delete the unwanted data rows from the database.

Inputs Arguments

MYSQL * conn: a variable to identify the connection of the Server Program and MySQL Server.

const char *qstr: a deletion command.

Output Arguments

None

Revision on Current Functions

UDP_Create

Syntax

int Server_HIP::UDP_Create(int sim_port, int truck_port, int signal_port)

Description

This function is the first function that will be called when researchers start the Server Program. It will initialize the test truck parameters, Winsock and help establish the communication between different ports.

Revision

Since it is the first function to call, the first two major steps: Create connection to MySQL database and generate test run ID can be added here.

The createconnection() and Test_Start_Time() are added in this function.

ASN1_Decode_Truck_Message

Syntax

List<double>^ Server_HIP::ASN1_Decode_Truck_Message(char* buf, int size)

Description

This function helps decode the truck message that is received from DSRC Radio and is beneficial for the further use of the Server Program.

Revision

In the current version, only the variables for time, speed, acceleration, and distance are stored as the output of this function. However, according to our database design, the longitude and latitude of the trucks are also important and need to be collected.

The longitude and latitude of the truck have also been added as an output of this function.

Simulation_Communication_Handle

Syntax

void Simulation_Communication_Handle()

Description

This function helps receive virtual car information from the simulation and sends virtual car information to the trucks. It will be triggered when a new message from the simulation arrives.

Revision

This function contains all of the attribute variables needed for the virtual car. This function transforms variables to strings and records attribute values to the database.

Truck_Communication_Handle

Syntax

void Truck_Communication_Handle()

Description

This function helps receive and store messages from the trucks.

Revision

This function contains all of the attribute variables needed for trucks. This function transforms variables to strings and records attribute values to the database.

Signal_Communication_Handle

Syntax

void Signal_Communication_Handle()

Description

This function helps receive and store messages from simulated signals.

Revision

This function contains all of the attribute variables needed for the simulated signal and signal controller. This function transforms variables to strings and records attribute values to the database.