A simulation platform for connected vehicle based traffic signal control

# Introduction

This chapter mainly describes a simulation platform for connected vehicle (CV) based traffic signal control, using VISSIM as the simulation software. The Drivermodel.dll API and Econolite’s ASC/3 virtual signal controller are required add-on components of VISSIM. The main purpose of the platform is to setup a CV environment, which tries to mimic the real-world situations as much as possible, so that signal control models that are tested in this environment can be implemented in the field with minimal modification. This chapter doesn’t intend to specify any traffic control algorithms, but to provide supporting data structure and interfaces. Users can plug-in their own algorithms in the platform for testing, validation and comparison.

In general, CV based traffic signal control systems utilize CV data (i.e., Basic Safety Messages (BSMs)) or a combination of CV data and infrastructure based sensor data (e.g., loop-detector) as input to make control decisions. A map, which describes the geometric structure of the intersection, is required to localize vehicles and calculate or estimate traffic information. Combining the estimated traffic information (or in terms of performance measures such as delay or queue length) and Signal Phasing and Timing (SPaT) information, the signal control model generates optimal signal plans and applies to the signal controller (virtual controller in this case).

# Platform overview

Figure 1 shows the structure of the platform. The DriverModel.dll is used to generate BSMs and the ASC3 virtual controller is used to push SPaT information. An intersection map file is constructed, which contains the intersection geometry information. Combining the MAP data and BSM data, the map matching algorithm locates vehicles on the map and calculates traffic information. The SPaT and traffic information servers as the input to the signal control algorithm. The signal plan generated by the algorithm is sent to the controller interface component, which uses NTCIP commands to execute the signal plan. In the follow, we will describe each component in details. Section 3 introduces how to DriverModel.dll to generate BSMs. Section 4 introduces how to use Econolite ASC3 virtual controller to generate SPaT information. Section 5 describes the map file, map data structure, and map matching algorithm. Section 6 introduces the signal controller interface. Examples will be provided in each section.

Note that all BSM, SPaT and MAP messages used in the simulation platform are NOT standard SAE J2735 messages, but contain necessary information to encode to standard J2735 messages (i.e., UPER encoding). For encoding and decoding standard messages, readers can refer to <https://lionet.info/asn1c/compiler.html>, an open source ASN.1 compiler for more information.

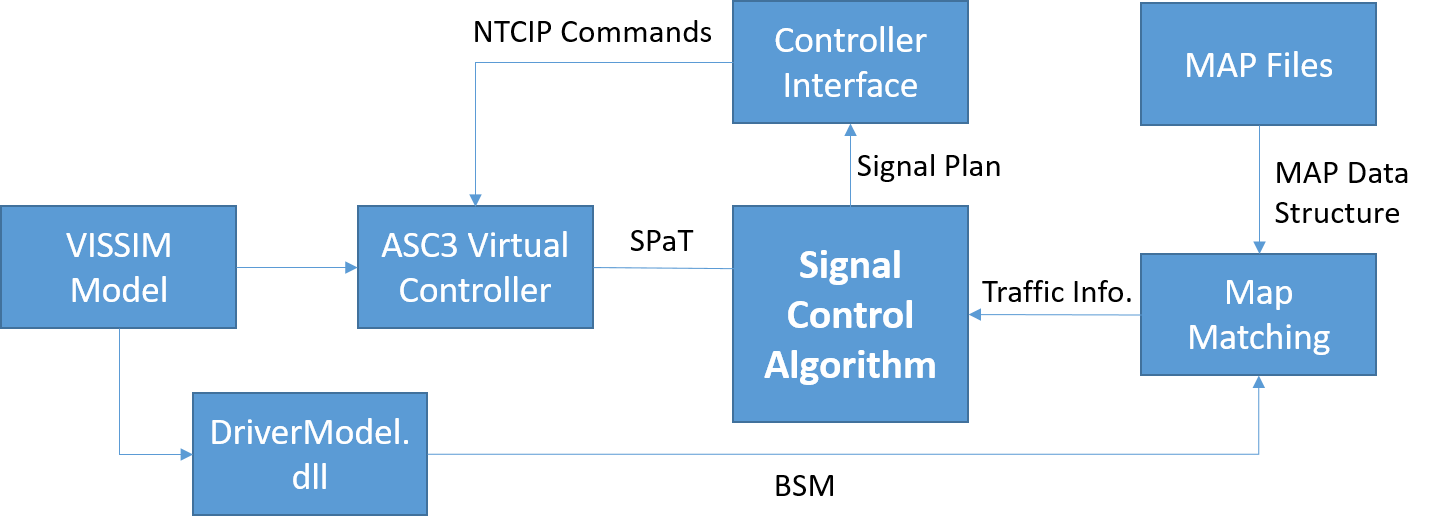


Figure Platform Overview

The illustration of the simulation platform is based on a real-world intersection (Huron Pkwy & Plymouth Rd) in Ann Arbor, Michigan. The VISSIM model of the intersection is contained in the folder: VISSIM Model\_Huron. Source codes and examples of other components of the simulation platform will be introduced along each section.

# BSM Generation

The BSM is the most important CV messages from the vehicle side for both safety critical and mobility applications. The specified transmission rate is 10 times per second and is often referred to as the “Here I am” message since it reports current vehicle location and states. The BSM has two parts. Part I contains the mandatory data elements that need to be broadcast by all vehicles. Part II contains optional data elements and the content is dependent on the requirements of the specific applications. Most of the BSM data can be obtained only from the vehicle CAN bus. In the VISSIM simulation platform, the following data elements can be obtained from the DriverModel.dll:

VehID: Vehicle temporary identification

Position: Latitude, Longitude, and Elevation (a coordinate transformation algorithm is needed)

Motion: Speed, Heading, and Acceleration

Vehicle Size: Length and Width

The DriverModel.dll is used to generate BSMs. The process of generating BSM using the API is described below:

Step 1: Initialization: Setup UDP socket communication and read IP address and port of the target application (map matching algorithm in this case)

Step 2: Read vehicle information from VISSIM through DriverModelSetValue() function

Step 3: Coordinates transformation which transform the vehicle position coordinates from local X, Y to GPS coordinates (WGS-84) applying the transformation algorithm described in (Farrell and Barth, 1999). This algorithm first transforms local X, Y coordinates to the earth-centered earth-fixed (ECEF) rectangular coordinates. The ECEF coordinates has its *x* axis extended through the intersection of the prime meridian (0° longitude) and the equator (0° latitude). The *z* axis extends through the true North Pole. The *y* axis completes the right-handed coordinate system, passing through the equator and 90° longitude. Then the ECEF coordinates are transformed to GPS coordinates.

Step 4: Generate BSMs by packing all data elements into one package.

Step 5: Broadcast BSMs through the UDP socket.

Note: users can enable the DriverModel.dll to different types and compositions of vehicles to simulate different penetration rates.

The sample code for BSM generation is contained in the folder: Dll Source code\_Huron.zip.

# SPaT data

The Signal Phase and Timing (SPAT) data is used to convey the current status of the traffic signals. Combined with the MAP message, the receiver is able to know the current signal status, remaining phase time and the next phase. In addition, current signal preemption and priority status are included in the message. The required transmission rate is 10 times per second. The SPAT data includes the following contents:

Intersection ID: Identification number of a particular intersection

Intersection Status: the operational status of an intersection

Signal Status: current signal status of vehicle signals, pedestrian signals, and overlap signals

Time to change: minimal and maximal time to change of the current status of vehicle signals, pedestrian signals, and overlap signals

Time stamp: current time in seconds and milliseconds

The Econolite ASC/3 virtual signal controller is an add-on in VISSIM, which full replicates the functionality of real ASC/3 controllers. After setup, the virtual controller can automatically push the SPaT data through a UDP socket while the VISSIM is running. The configuration and enabling of the ASC/3 virtual controller in VISSIM can be found in the following document:

BattelleSPAT\_MIBSupportDocument-v.02.docx

Note in this platform, the destination of the SPaT data should be the signal control algorithm.

# MAP Generation and map matching

The MapData (MAP) message is used to describe the geometric information of an intersection defined at the lane level. The required transmission rate is 1 time per second. The MAP message includes the following contents:

Intersection ID: Identification number of a particular intersection

RefPoint: The GPS reference point from which other lane nodes are offset.

Approaches: Data structure to describe an approach including a set of related egress and ingress approaches at the intersection. Each egress or ingress may have multiple lanes.

Lanes: Data structure to describe a lane including lane number, lane width, lane attributes, node list, and connection lanes. Each lane may have multiple lane nodes.

Nodelist: A sequence of points (Xs and Ys) that builds a path for the lane. The values of Xs and Ys are signed offsets from the last point.

In this simulation platform, for each intersection, the map data is saved in an .xml file as shown in Figure 2 (the complete description is contained in MAP\_PLYMOUTH\_Huron\_XML\_New.xml). The node points of the .xml is shown in the Google Earth file: Plymouth & Huron.kml



Figure Map Description file

This file contains map of the intersection of Huron Pkwy and Plymouth Rd in Ann Arbor, Michigan.

A data structure is designed to save the map information. The data structure is also designed in a hierarchical way, similar to the map description file. The description of the data structure is contained in NMAP.h. A sample function (ParseIntersection\_XML()) is provided to read the map description file and save corresponding data to the data structure. This function is contained in NMAP.cpp.

Finally, combining the map and BSM data, the LocateVehOnMap () function uses vehicles location, speed and position (all from BSM) as the input and calculates the vehicles approach, lane, requested signal phase, state (approaching or leaving the intersection), and estimated time of arrival (ETA) as output. This function is contained in the CV\_Trajectory\_Awareness.cpp file.

Beside the map matching function, the CV\_Trajectory\_Awareness tracks and maintains an active list of CVs that are approaching the intersection. All CV data are contained in the active list, which can be readily read by the signal control algorithm. For more detailed information regarding the CV trajectory awareness, readers can refer to Chapter 4 of [ref]

# Signal Controller Interface

The signal controller interface assumes a NEMA durl-ring barrier controller structure. It receives a signal control plan and controls the controlling according to the plan using NTCIP commands including force-off, hold, omit, and call. The interfaces requires input (through UDP socket) in the format of “schedule”, defined in a C++ data structure as:

class Schedule

{

public:

int time; //time point for the schedule

int phase; // which phase do we operate

int action; //FORCE\_OFF:0; HOLD:3; CALL:2; OMIT:1

// -----------------Methods-------------------------

};

One signal plan can be packed with multiple phases sequentially together. For example, a package can contain the following information

10 1 3 -> hold phase 1 until 10s

10 2 2 -> call phase 2 at 10s

10 1 0 -> force-off phase 1 at 10s

20 2 3 -> hold phase 2 until 20s

20 3 2 -> call phase 3 at 20s

20 2 0 -> force-off phase 2 at 20s

Note 1: It is assumed that the signal controller is working under “free operation” with no detector input. If no control command is sent, then the signal controller will be rest in current phase.

Note 2: always send “call” command before “force-off”. Otherwise, the controller can’t force-off because it doesn’t know which phase to go.

Note 3: the interface doesn’t check the dual-ring structure (e.g., end two rings at the same time at the barrier). The input signal plan should already consider such constraints. Otherwise, the interface may not be able to control the controller as expected.

Once a new signal plan is received, the old signal plan will be discarded immediately.

The sample code of the signal controller interface program can be found in the folder: TrafficControllerInterface