Traffic Signal Simulation Cookbook version 1.0

Last Date: June 2022

Traffic Signal Simulation Cookbook

DOI for citing this report:

DOI:10.5281/zenodo.6636406

Github Link for source codes and example models:
https://github.com/trbahb252/traffic_signal_sim_cookbook

A LinkedIn Group:
https://www.linkedin.com/groups/12278423

Main Contributors: Qing He, Pengfei "Taylor" Li, Aleks Stevanovic, Chris Day, Jongsun Won, Qichao Wang, Tracy Zhou, Yiheng Feng, Guoyuan Wu, Milan Zlatkovic, Dusan Jolovic

Disclaimer

This document is NOT a publication of Transportation Research Board (TRB). The chapters, models and source codes are contributed by a handful of volunteers who have expertise in traffic signal simulation. Traffic signal simulation subcommittee of TRB Traffic Signal Systems Committee (ACP25) facilitates the efforts in developing this document. The information in this cookbook is intended solely for practitioners and academic researchers non-commercial use of the user who accepts full responsibility for its use. The developer assumes no responsibility or liability for any errors or omissions in the content of the materials.

The information contained in this cookbook is provided on an "as-is" basis with no guarantees of completeness, accuracy, usefulness, or timeliness, and without any warranties of any kinds whatsoever, express or implied. The developer does not warrant that this site and any information or material downloaded from this site will be uninterrupted, error-free, omission-free, or free or virus or other harmful items.

Specific examples or methods described or demonstrated are provided as examples only and are not specifically endorsed by the model developer. The views expressed in this cookbook are not necessarily those of the U.S. Governments, the Transportation Research Board Traffic Signal System Committee, nor the Signal Simulation Subcommittee.

Table of Contents

1. Introduction

Contributor(s): Qing He, Pengfei "Taylor" Li, and Aleks Stevanovic

Traffic signal control is one of the most cost-effective approaches to mitigating traffic congestion. In practice, before implementing a new signal system, traffic signal simulation is usually performed to estimate the impact of the proposed solution. In other words, microsimulation tools provide a huge amount of output results that can be used for the performance assessment of signal control systems. To mimic the real-time signal operations, microscopic traffic simulation is typically adopted in traffic signal simulation, aiming to model the individual vehicle movements on a second or sub-second basis.

Traffic signal simulation is in many aspects different than traditional traffic simulation. First, traffic signal simulation has to involve a signal system that configures the signal phasing and timing plan. Such signal systems could be established on top of a very simple two-phase setting or a complicated NEMA eight-phase setting depending on the purpose of the simulation. Second, signal simulation has to take into account the special features of real-world signal controllers, such as transit signal priority, signal preemption, and so on. Sometimes an advanced software-in-the-loop simulation is preferred. Third, signal simulation usually integrates vehicles with signals, enabled by connected vehicles v2i technology. New technologies such as SPat and BSM, shall be considered in the traffic signal simulation.

This cookbook aims to provide the traffic signal community with easy-to-follow procedures and examples in conducting different aspects of traffic signal simulation. This cookbook is a collection of tiny simulation "recipes" that each demonstrates a particular signal control concept. The use of this book will aid in the consistent and reproducible application of traffic signal microsimulation models and will further encourage the development of signal control algorithms by the entire community. As a result, practitioners and researchers will be equipped to conduct quick signal-related simulations. Depending on the project-specific purpose, need, and scope, elements of the process described in this cookbook may be enhanced or adapted to support the decision-makers. It is strongly recommended that the respective stakeholders and partners consult prior to and throughout the application of any signal simulation model.

1.1 Organization of Cookbook

It is worth noting that this cookbook is still under development. Many of the proposed chapters are waiting for the inputs from volunteers. The current version (1.0) of the signal simulation cookbook is organized into the following chapters:

Chapter 1 (this chapter) introduces the traffic signal simulation and provides an overview of this cookbook.

Chapter 2 addresses the simulation of signalized intersections.

Chapter 3 discusses the steps to simulate non-signalized intersections.

Chapter 4 provides guidance on simulate signals with connected and automated vehicles.

2. Signalized Intersection Control

Contributor(s): Jongsun Won, PE, Betsy LaRue, PTV America, Inc.

2.1 Modeling Intersection Control Logics

2.1.1 Basic Components

The purpose of this chapter is to provide step by step guidance on modeling signalized intersection(s) in microscopic traffic simulation model. More specifically, basic components of the signalized intersection modeling are covered in this chapter and it is structured as shown below:

- Basic components of the signalized intersection modeling
- Applications of the signalized intersection modeling

In order to make this guidance more practical, we have selected a specific software, PTV Vissim, and provided more direct guidance rather than staying at the general level. This guidebook will only cover modeling signalized intersections and assumes that the reader is already aware of the basic microscopic simulation modeling process. If you need guidance on any other part of the modeling process, refer to other resources or training opportunities. Also, if you would like to go in depth to the details on any traffic signal operation, we would ask you to refer to "Traffic Signal Timing Manual (2015)[1]".

There are five (5) basic components for the signalized intersection modeling in microscopic simulation modeling. Some of the five (5) components represent the signal control objects that you can find in the field and some represent the behavior of the drivers at the signalized intersections. The five (5) basic components are:

1. Signal Controller
2. Signal Timing Plan
3. Signal Head
4. Priority Control for Permissive Movements
5. Detector

Fortunately, the five components listed above typically need to be created in sequential order because most components are connected to each other and affect the simulation model collectively. Therefore, the overall process can also be understood in five steps to add these five basic components. The flowchart in the figure below shows the process of signalized intersection modeling with its basic components.

[1] Signal Timing Manual, Transportation Research Board, 2015 (https://ops.fhwa.dot.gov/publications/fhwaho`p08024/index.htm)

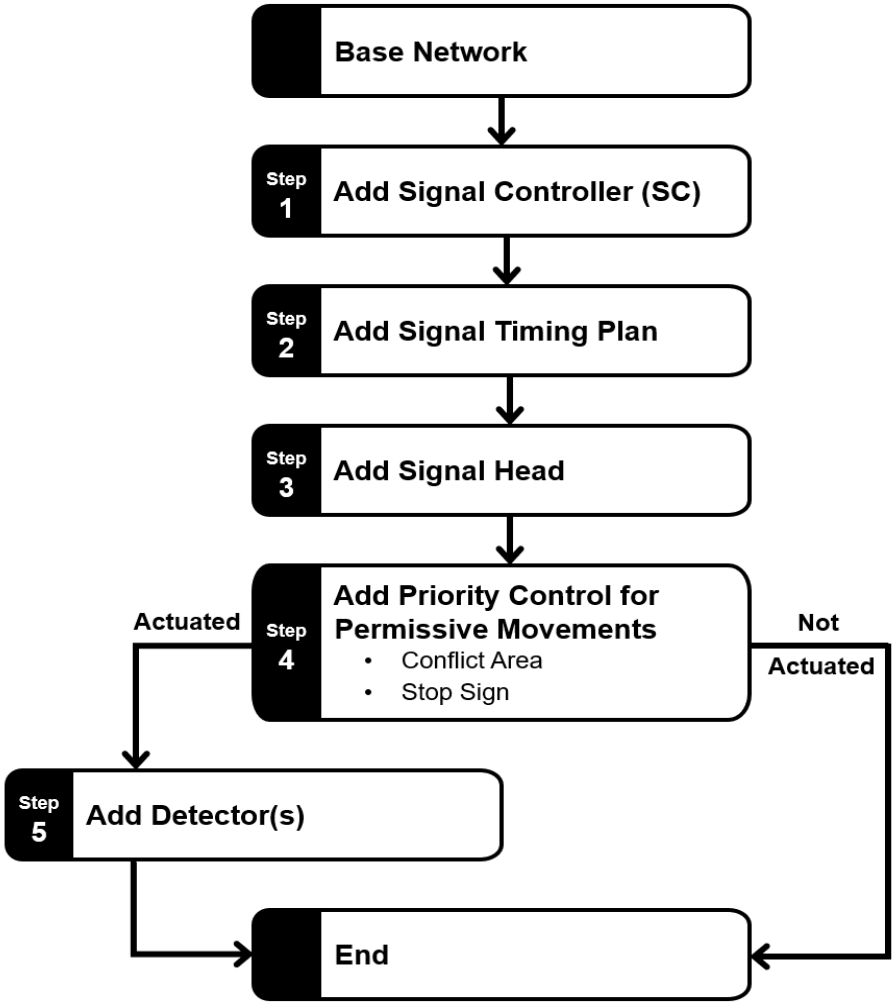


Figure 2.1-1: Basic components for the signalized intersection

Details on each step above are described in the following sections.

2.1.1.1 [Step 1] Add Signal Controller (SC):

Before working on any further steps, a signal controller (SC) must be defined and added to the network. Without a SC, no other objects related to the signal control operation can be created.

You can access the SCs window in PTV Vissim user interface by going to "Signal Control > Signal Controllers" menu.

Then, "Signal Controllers" window (SCs window) will be loaded as shown in the figure below. The window might contain a list of existing SCs which are already defined in the model.

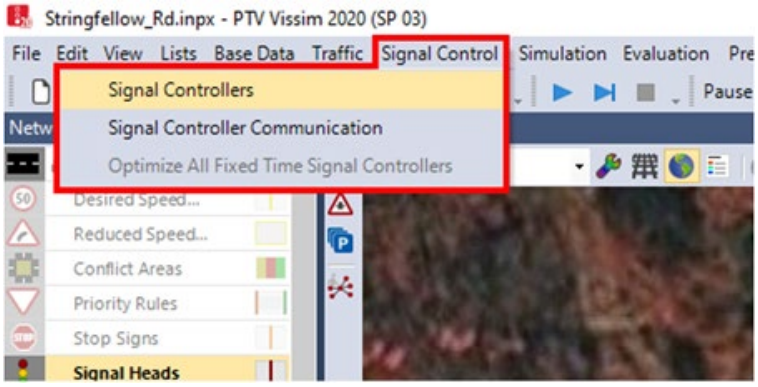


Figure 2.1-2 SC Window

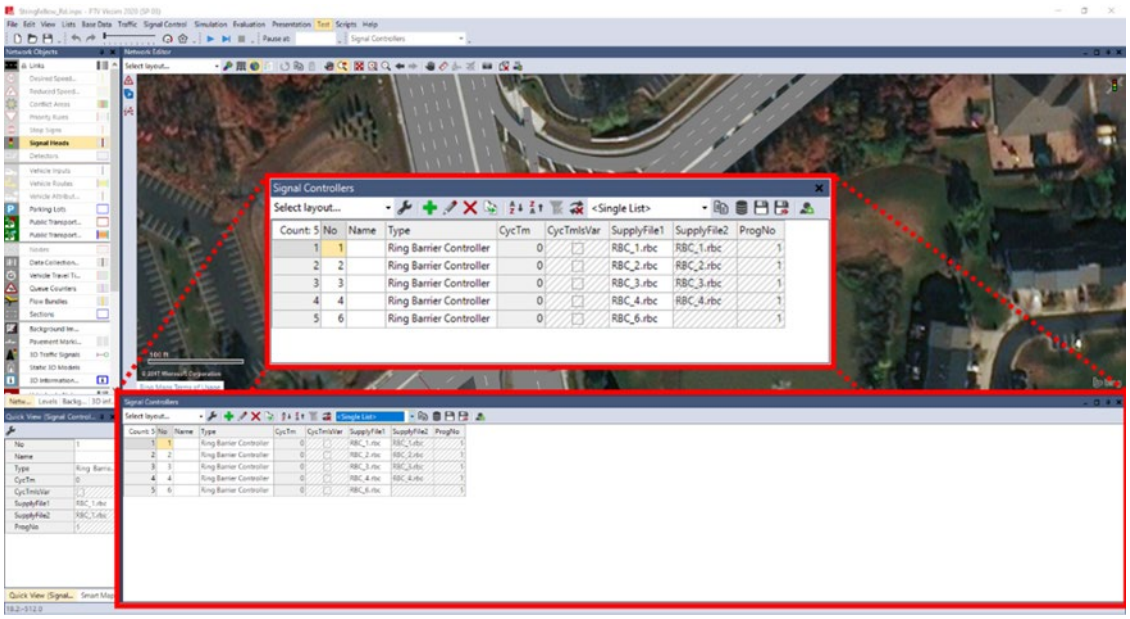


Figure 2.1-3: Window containing existing signals

On the SCs window, first, look for configuration buttons on the top of the list. Click the green add button (+) to create a new SC or click the red remove button (X) to delete existing SC or click the edit button ( ) to modify or view existing SCs.

As stated above, click the green add button (+) to create a new SC and the "Signal Controller" window will open. On this window, you will define 1) SC Number, 2) SC Type, and 3) SC Configuration. All other parameters can be adjusted if needed.

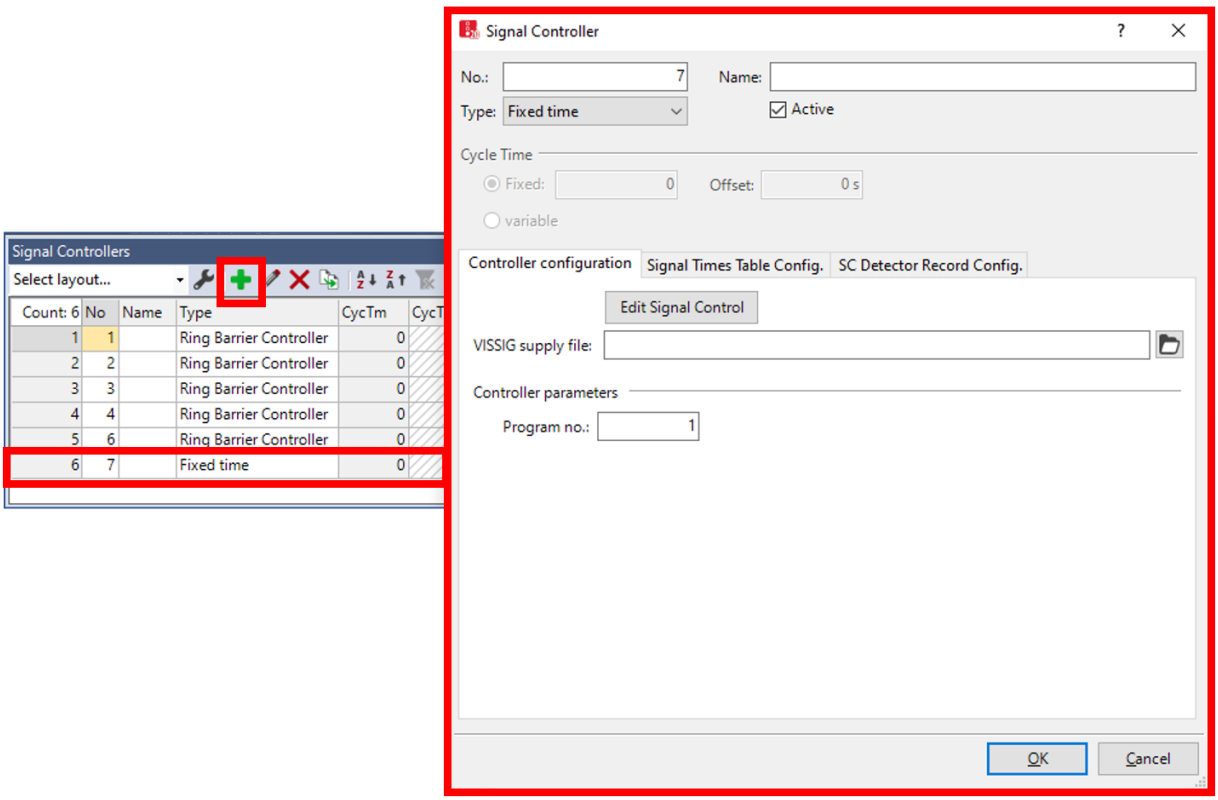


Figure 2.1-4: Add or delete a Signal Control

SC Number

This is a unique identification number assigned to each SC. This number will be used to connect SC and other objects (e.g. signal head, detector, etc.). If you do not have the SC number defined yet, it is recommended to use a number identical to any type of predefined intersection ID

SC Type

There are many different types of SCs built-in to PTV Vissim, which can be accessed by clicking on the "Type" dropdown box. Look for "Ring Barrier Controller (RBC)" and select it unless you have a need for another SC type available in the selection. RBC is selected as a default signal controller in the North America region and this guidebook will only describe details on RBC and no other SC types.

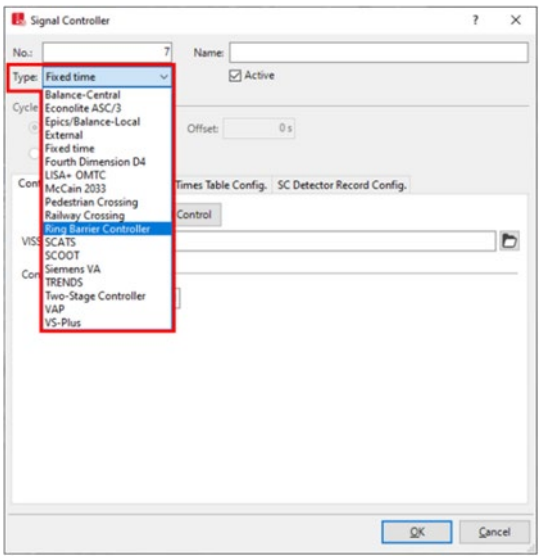


Figure 2.1-5: SC Type

Exercise

1. Open "Step_1_Add_SC" folder and load "Stringfellow_Rd_Step_1.inpx" file.
2. Go to "Signal Control > Signal Controllers" to open "Signal Controllers" window.
3. Click Add button (+) on the top menu field and wait for "Signal Controller" window to open.
4. Update following data in the signal controller window:
   - No.: 4
   - Name: Stringfellow Rd @ Fair Lakes Pkwy
   - Type: Ring Barrier Controller
5. Click on "Browse" button for "Data file" field in "Controller configuration" tab and look for "RBC_1.rbc" file in the same folder as the Vissim input file.
6. If you leave the data file field blank and save, you may receive an error message mentioning about an incomplete file path.
7. Click "OK" button to save.

2.1.1.2 [Step 2] Add Signal Timing Plan:

In Vissim, the signal timing plan can be accessed via the RBC editor window and the data is saved to an external file (*.rbc). In other words, the signal timing plan data is not saved in the network file (*.inpx) at all and it will require external signal files (*.rbc) to conduct any simulation model runs.

The RBC editor can be accessed by going to the individual SC window and clicking on the "Edit signal group" button as shown in the figure below.

The RBC editor consists of four (4) parts including: 1) menu (top), 2) selection panel (left top), 3) tables (right), 4) timing display and log (bottom). Signal timing data will be displayed in the "tables" field and the RBC editor will typically display a blank field upon loading. You can turn on any parameters by using the "selection panel"; however, the set of commonly used signal timing parameters can be loaded by using a default layout. Go to menu and click on "View>Basic View" to load default layout.

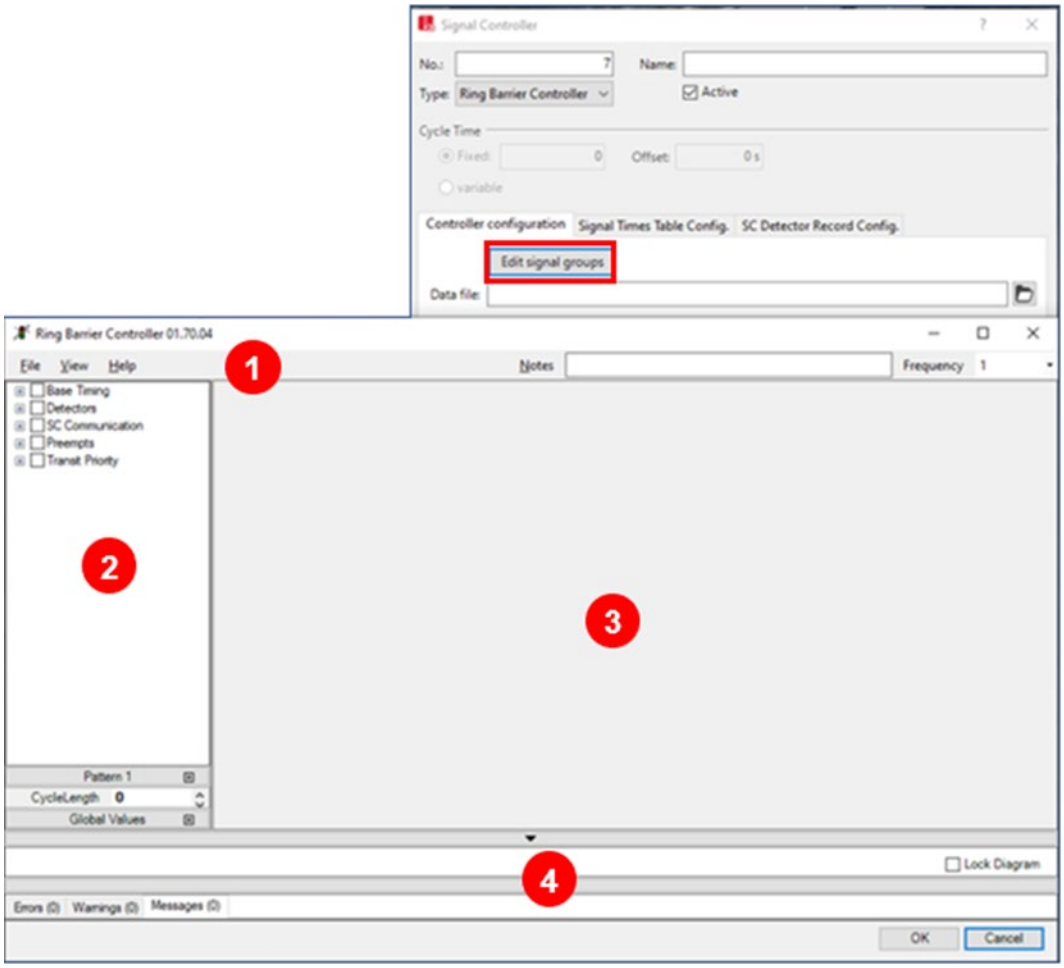


Figure 2.1-6: RBC Editor

With this, you can start adding (or editing) the timing parameters. This guidebook will only discuss the list of parameters selected in the default layout and details on other parameters can be found in the RBC user manual.

First, here are the steps that you will need to follow (with parameter definitions) if you are defining a new signal timing plan:

You will need to define *Signal Group (SG) number* for all vehicular SGs that you will need. Once the RBC editor is open, go to the "SG Number" row and start defining desired SG numbers. You will simply need to enter any integer value (less than 1,000) in each cell.

If you already have a signal timing plan provided, you should stay consistent with the provided signal timing plan. If not, you can use the standard National Electric Manufacturers Association (NEMA) phase numbering system for typical four-legged intersection with protected left-turn movements. It is shown in the figure below along with pedestrian SG and movement numbers.

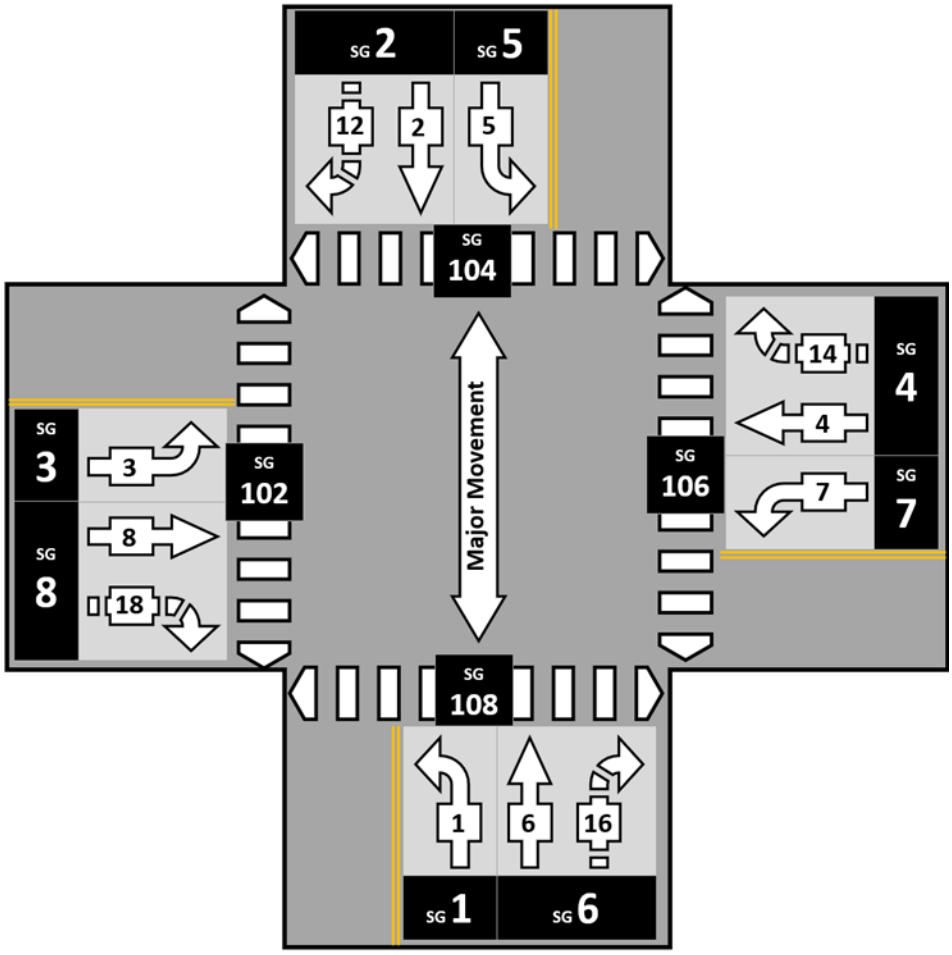


Figure 2.1-7: NEMA phase numbering system

With SGs defined, enter the following set of parameters in the "Basic Timing" section:

SG Name

Name for each SG.

Minimum Green (0-255)

The minimum green time (Min Green) that each SG will serve before changing to yellow. In the absence of any extension (demand), the SG has to serve this minimum green time before it is eligible to terminate.

Vehicle Extension (Passage Time, 0.1-25.5)

The allowed time between successful vehicle extensions (calls) before a SG will gap out.

Max 1 (1-255)

This parameter defines the maximum green time that the SG will be allowed to extend before it will max-out. A max-out will make a SG eligible to terminate, even though it may not have gapped-out and have more demand to be served.

Splits (1-255)

The amount of time allocated in the cycle for each SG to time. The split includes the time it will take the green, yellow, and all-red intervals to time for each SG. The split should at least accommodate the SG Min Green time, yellow clearance time, and red clearance time, but it doesn't necessarily need to accommodate the full pedestrian service time for an actuated pedestrian SG. The sum of the splits of all SGs in each ring should add up to the Cycle Length.

Coordinated

In case of signal coordination modeling, there must be a coordinated SG in each ring of any ring group that will be coordinated. All coordinated SGs must be in the same barrier group. If coordinated SGs are not defined, the controller will run in free running mode.

In RBC, you will need to define the schedule of signal operation. In most cases, you will not model signal timing changes in your model. In this case, you still need to select the pattern number (including Free running) and respective start time. If the signal timing plan changes during the same model run, you can specify the plan here and set it to change up to seven (7) times.

Pattern Number

This is the pattern that will run starting at the defined Pattern Start Time. The Free running mode will run the Basic and Advanced timing.

Pattern Start

This is the simulation start time, in seconds, at which the defined pattern will start running. If more than one pattern is defined for the same start time, the last listed will be the pattern that is run for that time. Active patterns will only end when another pattern begins.

Define and edit the sequence of SGs in the Sequence Editor. Only SG numbers that have been defined above can be used in the sequence once. A SG must be included in the sequence for the SG to time; otherwise the timing information will be ignored. Clicking on the column header will create a barrier to the right of the column below. A maximum of 8 barriers and 4 rings are available.

If you need to add an overlap SG, go to the "Overlap" section and add the following parameters:

Overlap SG

SG numbers for corresponding overlap. The SG number that will be created in the model will be used to create signal heads that use the timing defined for this overlap

Parent

These are the SGs that the overlap will be allowed to time with. When one parent SG is timing and another parent SG is next, the overlap will remain green. When the last parent SG terminates, the overlap will also terminate.

To model actuated signal operation, you will need to add detectors. Before adding the detector object to the network, you will need to define detector settings in SC in the "Detectors" section for both vehicles and pedestrians (push button). Following are further details on each parameter in the detectors section:

Vehicle Detectors

Detector Number

The detector number that should be used within the model to call or extend the vehicle SGs selected. There are 32 available vehicle detectors.

Call

SGs that are called when the detector input is on.

ExtendSGs

SGs that are extended when the detector input is on.

Pedestrian Detectors

Detector Number

The detector number that should be used within the model to call the pedestrian SGs selected. There are 16 available pedestrian detectors.

Call Peds

This parameter defines the pedestrian SGs that are called when the detector input is on.

There are a couple of parameters that you will need to define for each pattern at the higher level that can be found under the "Pattern" section right below the "Selection Panel."

Pattern

A total of 8 patterns are available. Patterns must be used for coordinated control, otherwise the controller is in free running mode. In general, patterns are coordinated but they will run in free running mode if the cycle length is set to zero or if no coordinated SGs are defined. Any values set within pattern that are duplicates of variables within base timing override the base timing; zero values within pattern are ignored. However, for those checkboxes which are duplicated, the pattern can only turn on checkboxes that are off in base timing. If they are on in base timing, they will still be on when the pattern is running. If you would like to switch to another pattern, you can right click on the header bar.

Cycle Length (30-255)

This value defines the cycle length of the pattern. This is the maximum time it will take for each SG to cycle once. The cycle length is only used for coordination. If a cycle length is not defined (set to zero), the pattern will run in free running mode.

Offset (0-255)

When coordinated, the local cycle timer will be offset from the master cycle timer by the defined offset time.

Max Green Mode

This setting determines the maximum green mode that will be used for all SGs while the coordination pattern is active. This selection is only valid for coordinated patterns; if used in free running mode, the value will be ignored. The selections are 1) Max Inhibit (all SGs will ignore their max green time), 2) Max 1, Max 2, Max 3 (all SGs will observe their selected max green time).

Permissive Mode

This setting defines the permissive mode for the coordination pattern. The permissive mode controls the method in which permissive periods are opened and closed for all non-coordinated SGs.

Finally, there are two (2) parameters at the global level that are applied to all patterns. To edit these parameters, go to the "Global Values" section. Two parameters in this section include:

Offset Reference

This is the point in the cycle where the master cycle timer will be equal to the defined offset time when the controller is coordinated and not in transition (offset seeking).

Transition Mode

This is the mode that all coordination patterns will use to transition when the local dial does not have the correct offset to the master clock, simulation.

Click the "OK" button to save the signal timing plan and define the desired file name (e.g. Intx01.rbc). Note that it is suggested to save all RBC files in the same folder as the Vissim input file (*.inpx) to avoid any complications.

Exercise:

Open “Step_2_Add_Timing_Plan” folder and load “Stringfellow_Rd_Step_2.inpx” file.

Go to “Signal Controllers” window and select “SC 4: Stringfellow Rd @ Fair Lakes Pkwy”.

Click on “Edit” button on the menu () to open “Signal Controller” window.

Click on “Edit signal groups” button in “Controller configuration” tab to open RBC editor.

Enter the following data in RBC editor:

Basic Timing

Table 2.*1*-1: Basic Timing

SG Number	1	2	3	4	5	6	7	8
Min Green	5	5	5	5	5	5	5	5
Max 1	25	85	5	49	34	76	6	48
Yellow	3	3	3	3	3	3	3	3
Red Clearance	1	1	1	1	1	1	1	1
Ped SG Number		102		104		106		108
Walk		5		5		5		5
Ped Clear (FDW)		15		27		18		24
Start Up		X				X		
Max Recall	X	X	X	X	X	X	X	X

Sequence

Table 2.1-2: Sequence

Ring 1	2	1	3	4
Ring 2	5	6	7	8

Click “File > Save File As” and specify file path and file name and click “Save” button.

And click “OK” button to save.

If you are importing existing signal timing plan in RBC format (*.rbc), from Vistro or Synchro Import, here are the steps that you will need to follow:

In the RBC window, go to “File > Import File”.

Find and select the desired RBC file and click the “Open” button.

If you do not see any parameters or tables open, either adjust your view by selecting parameters on the selection panel on the left side or go to “View > Basic View” to load basic parameters.

Once all the signal timing data is loaded in the RBC window, check the accuracy of each parameter and click “OK” to save and connect to the network file.

Exercise

Open “Step_2_Add_Timing_Plan” folder and load “Stringfellow_Rd_Step_2.inpx” file.

Go to “Signal Controllers” window and select “SC 4: Stringfellow Rd @ Fair Lakes Pkwy”.

Click on edit button on the menu () to open “Signal Controller” window.

Click on “Edit signal groups” button in “Controller configuration” tab to open RBC editor.

Go to “File > Import File” and select “RBC_1.rbc” located in the same folder as Vissim input file.

Click “Open” to load imported timing plan file.

And click “OK” button to save.

2.1.1.3 [Step 3] Add Signal Head:

Signal heads, like stop signs, are placed in a lane and force a vehicle to decelerate to standstill condition while showing red. However, unlike a stop sign, a signal head, while showing red, will not allow vehicles to cross. Once the signal head shows green, vehicles may accelerate back to their desired speed.

In Vissim, a signal head works as both a signal head and stop bar in the field; therefore, it is important to place signal heads at the stop bar location. Follow the steps below to place a signal head in Vissim:

1. Turn on “Background maps” by clicking on the globe icon () in the network editor window and change the network display to wireframe mode (). This way, you can see the background map along with the skeleton of your network.
2. Click “Signal Heads” in the network objects window as shown in the figure to be in signal head insert mode.
3. Zoom in to the location where the new signal head will be placed.
4. Left click to select link or connector where signal head will be placed.
5. Press <CTRL> + Right click to place the signal head and Signal Head window will open as shown below.

6. Define the following data for this signal head.

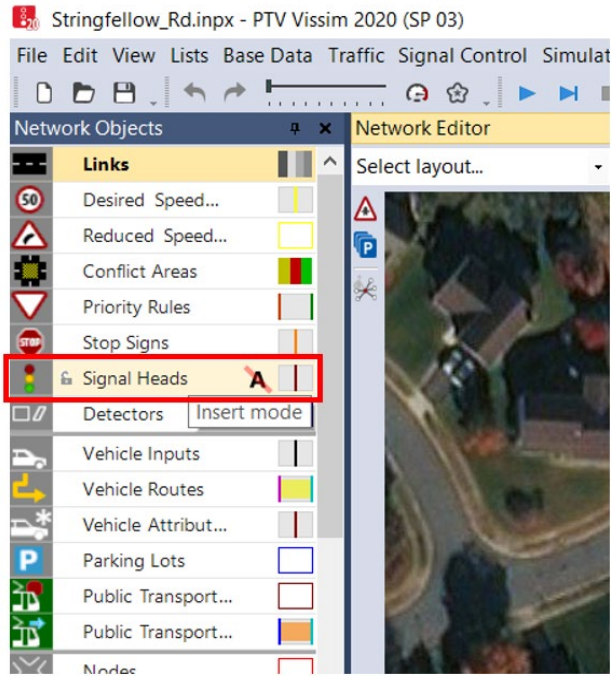


Figure 2.1-8: Signal Head

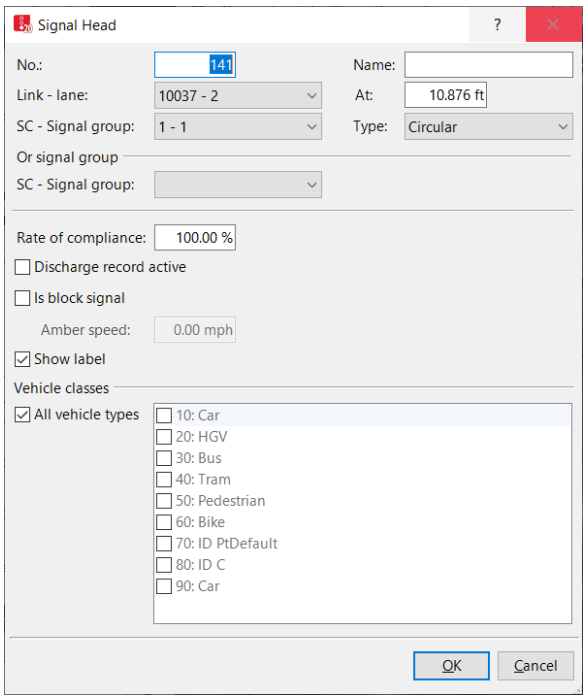


Figure 2.1-9: Signal Head Window

SC – Signal group

SC and SG that this signal head will need to be connected to.

Type

Display of the signal head during a simulation run including circular, left arrow, right arrow, and invisible.

Or Signal group – SC

Signal group: secondary SG connected to this signal head. If this option is selected, this signal head will display green when either primary or secondary SGs are showing green or it will show yellow when either (or both) of them are showing yellow. Unlike "overlap SG", it will not display green automatically when it is transitioning in between primary and secondary SGs and show yellow when needed.

Vehicle Classes

Vehicle classes that will obey the signal display at this signal head. If there will be any vehicle classes that will ignore this signal head, you should unselect "All vehicle types" and select desired vehicle classes.

Exercise

1. Open "Step_3_Add_Signal_Head" folder and load "Stringfellow_Rd_Step_3.inpx" file.
2. Go to "Signal Controllers" window and select "SC 4: Stringfellow Rd @ Fair Lakes Pkwy".
3. Right click on row selected above and select "Zoom" in the context menu to zoom into the study intersection.
4. Update network display as needed. It is beneficial to switch to "Wireframe mode" (<CTRL> + A) and turn on labels for the signal head object and select signal group attribute to show SG numbers on the network editor as shown in the example file.
5. In addition to a few signal heads which are already placed, complete signal head settings along Stringfellow Road (N-S corridor) for the following movements:
   - NBL: SG 1
   - NBT & NBR: SG 6
   - SBL: SG5
   - SBT & SBR: SG2
   - Crosswalk across EB approach: SG102
   - Crosswalk across WB approach: SG106
6. Click "Save" icon and "Run" simulation and observe.

2.1.1.4 [Step 4] Add Priority Control for Permissive Movements

Even though most conflicts at the signalized intersection will be controlled by the signal, there are still a significant number of conflicts which are not directly controlled by the signal. It is mainly caused by permissive movements such as right-turn-on-red (RTOR), permissive left-turns, etc. In this step, two different types of objects including 1) stop sign and 2) conflict area will be used. Details on each object can be found in the following section.

Stop Sign (Right Turn on Red, RTOR)

When a vehicle is allowed to make a right turn during a red, the driver is supposed to treat the signal as they would a stop sign; therefore, they will come to a complete stop, then check for conflicting traffic. In Vissim, an approach is configured for RTOR using a stop sign object. Stop signs can be configured to work in conjunction with a specific signal group.

Note that stop signs must be used along with conflict areas so that the actual conflict that happens after the stop sign can be controlled. Follow the steps below to add stop signs for RTOR:

1. Double click on right turn signal head object to edit. It will open signal head window.
2. Since this signal head will not be obeyed by vehicles but only be used for visualization purposes, unselect "All vehicle types." and select vehicle class which does not travel on this link (i.e. Pedestrian).
3. Click "OK" button to save.
4. Click "Stop Signs" in the network objects window as shown in the figure to be in stop signs insert mode.
5. Left click to select the link or connector where the stop sign will be placed. This stop sign should be overlapping to where right turn signal head is placed. If the right turn lane is shared with another movement (i.e. thru movement), you may will need to have a longer connector so that each turn can have its own signal heads placed.
6. Press <CTRL> + Right click to place stop sign and Stop Sign window will open as shown below.

7. Go to “RTOR” tab and check “Only on Red” and specify SC and SG connected to this RTOR operation. (It typically will be SG for concurrent thru movement.)
8. Click “OK” button to save.

Figure 2.1-10: Stop Sign Menu

Figure 2.1-11: Stop Sign Dialogue Box

Exercise

1. Open “Step_4-1_Add_Priority_Control_Stopsign” folder and load “Stringfellow_Rd_ Step_4-1.inpx” file.
2. Click “Stop Signs” in the network objects window to be in stop signs insert mode.
3. Zoom into signal heads for NBR or SBR and left click on the link (or connector) where the signal head is placed to select.
4. Press <CTRL> right click on top of right turn signal head to place stop sign.
5. Under “RTOR” tab, select appropriate SC-signal group to match up with SG assigned to right turn signal head.
6. Click “OK” button to save.
7. Select signal head for right turn movement below stop sign placed above and double click on it to open “Signal Head” window.
8. In “Vehicle classes” section, uncheck “All vehicle types” and check “50: Pedestrian”. This way, this signal head will not be obeyed by any vehicles, but you can still observe signal status changes while the simulation is running.
9. Repeat same steps for those places where RTOR is allowed.
10. Click “Save” icon and “Run” simulation and observe.

Conflict Area

In order to manage conflicts caused by permissive movements such as permissive left turns and RTORs, it is necessary to add objects which can manage conflicts in between unsignalized (permissive) movements. In traffic there are generally three types of conflicts; merging, diverging

and crossing; however, merging will be the most common type of conflict that will be caused by permissive movements at the signalized intersections.

Conflict area objects in Vissim should be used to handle these unsignalized conflict management cases as shown in the figure on the right. On the contrary, it is also important to not place conflict areas for those movement conflicts which are managed by the signal (i.e. protected left turn and opposing thru movements).

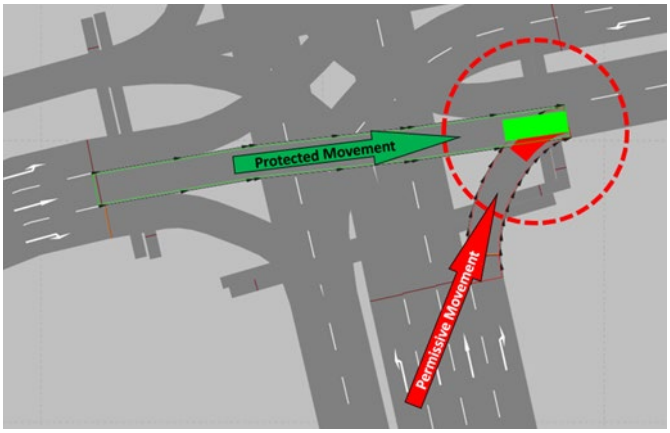


Figure 2.1-12: Conflict Area

Conflict areas are automatically generated any time two or more links occupy the same space. A conflict area has three states:

- Passive (displayed yellow)
- Active (movements are displayed red and green depending on priority setting)
- Undetermined (both links are displayed red and it is first-come-first-served)

As vehicles on the minor street (displayed as red) approach the conflict area they determine whether there is a large enough gap for them to pass through the conflict without disturbing the major movement traffic. If need be, a vehicle will change speed or come to a complete stop to avoid a "crash". Vehicles on the minor street will attempt to minimize their delay so if they do not need to stop. Conflict areas use a few parameters to calculate acceptable gap including "front gap", "rear gap", and "safety distance factor" however these will not be discussed in this guidebook. Refer to Vissim user manual for details on each parameter. All overlapping sections of links will create a conflict area automatically and simply require the user to determine priority scheme. Follow the steps below to add conflict areas:

1. Click "Conflict Areas" in the network objects window as shown in the figure to be in conflict areas insert mode.
2. Zoom into the conflict area to define priority scheme.
3. Left click on the conflict area you would like to update priority scheme on. If you cannot select specific conflict area that you want, left click as close as possible and press "Tab" key to select underlying object(s).
4. Press <CTRL> + Right click to change priority scheme and repeat until you see desired priority scheme.

5. Repeat for all permissive movement conflicts for each intersection, including all vehicle-to-pedestrian conflicts as shown below.

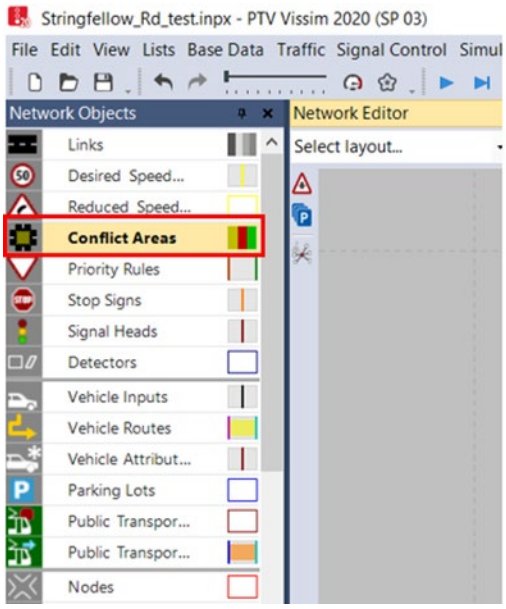


Figure 2.1-13: Conflict Areas

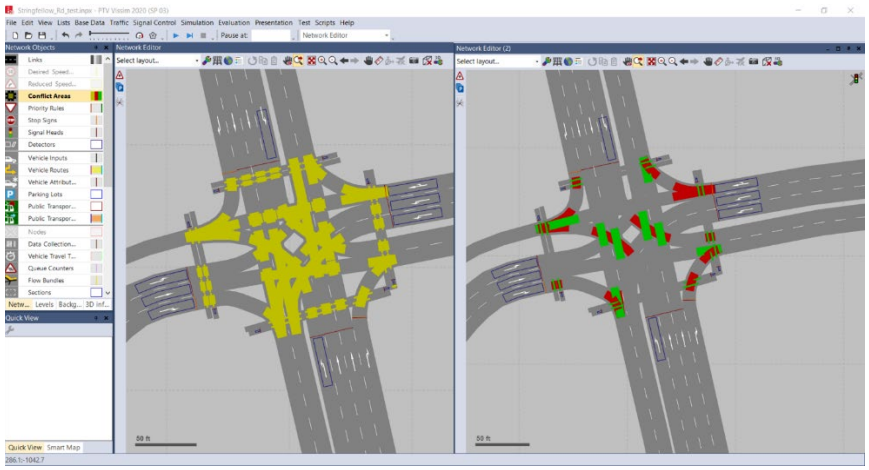


Figure 2.1-14: Assigning Conflict Areas

Exercise

1. Open “Step_4-2_Add_Priority_Control_Conflict Area” folder and load “Stringfellow_Rd_ Step_4-2.inpx” file.
2. Click “Conflict Areas” in the network objects window to be in conflict area insert mode.
3. Zoom into south west quadrant of the intersection as shown in the figure.
4. Left click on any conflict (yellow area) to update conflict control status to a realistic combination. If it cannot be selected in the network editor window, click as close as possible and use the “Tab” key to toggle.
5. Update conflict configurations for EBR movement and pedestrian crossing and SBT by pressing <CTRL> + right click.

6. Complete all priority settings for permissive movements and click “Save”.
7. “Run” simulation and observe.

Figure 2.1-15: Conflict Areas Exercise

2.1.1.5 [Step 5] Add Detector:

In the case of modeling actuated signals, you will need to add a detector object to the network. Detector objects work much the same way as in the field: as soon as the front of a vehicle crosses the detector, a detection signal is sent to the controller and as soon as the rear of that vehicle leaves the detector, the signal changes to no detection. This information is then interpreted by the signal control logic to determine what to do next.

Detection in the field is achieved using various methodologies including induction loops, video cameras, push buttons, track circuits, etc. Vissim models each detector type in the same way; wherever the calibrated detection zone is, is where a detector should be placed:

Induction loops- place the detector in the same location on the link.

Video cameras- place the detector on the link within the same zone a camera will read a vehicle as detected.

Push button- place a detector on the ground where a pedestrian would stand to press the button.

To create new detectors on a link, follow the steps below:

1. Click “Detectors” in the network objects window as shown in the figure to be in detectors insert mode.
2. Zoom into the link (or connector) where detectors will be placed and left click to select link. (If you need to check background map (or image), switch to wireframe model by pressing <CTRL> + A.)
3. Move mouse cursor to the location on the link where detector should start.
4. Press <CTRL> + Right click and hold; then, drag it along the link toward the direction of travel.
5. Release all keys and buttons and the new detector will be added to the selected link.
6. When the “Detector” window is open, enter the following data based on intersection design:

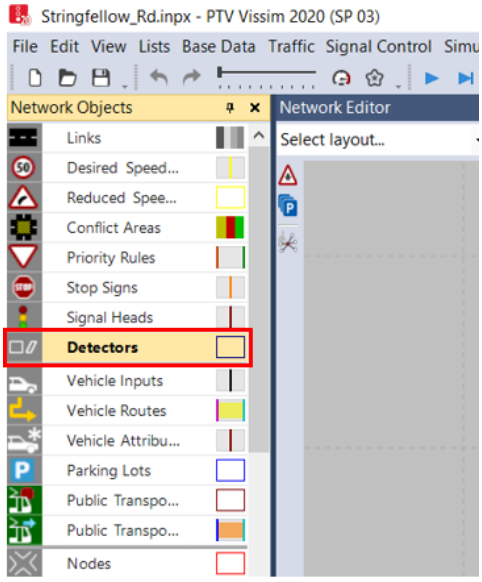


Figure 2.1-16: Detectors Menu

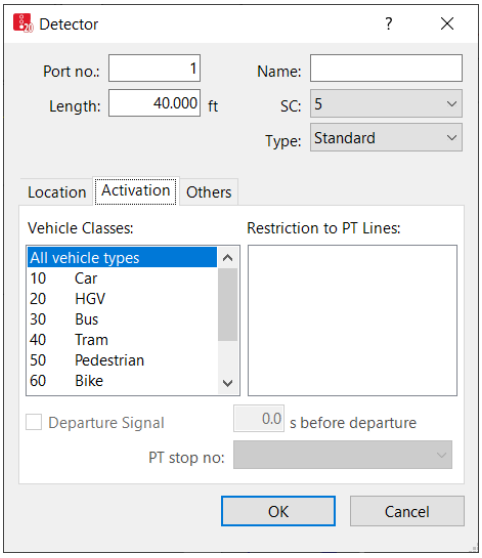


Figure 2.1-17 Detectors Window

Port No

This is identical to the physical port number that you can see in the field connecting SC and detector(s). This ID should be unique for detector to SC connection. Typically, the port number should match with the detector number defined for vehicle (or pedestrian) detectors in SC settings.

SC

SC that the detector is connected (or assigned) to.

Type

Detector type that can be selected from the list below.

Standard: Standard detectors detect vehicles and pedestrians.

Pulse: Impulse detectors do not send information regarding occupancy to the control procedures.

Presence: Presence detectors do not send information regarding the impulse via the front end or back end of the vehicle to the control procedures.

PT calling point: This detector type is only applicable to public transit. This detector will allow you to model short range communication in between vehicle and SC exchanging data more than detection such as vehicle ID, lateness, transit line number, etc. Standard type should be applicable for most signalized intersection modeling.

Activation / Vehicle Classes

If this detector should only detect certain vehicle classes (such as public transit, heavy vehicle, etc.), you can select a set of vehicle classes instead of leaving it at “All vehicle types”.

Click “OK” button to save settings.

Exercise

Open “Step_5_Add_Detector” folder and load “Stringfellow_Rd_Step_5.inpx” file.

Click “Detectors” in the network objects window to be in conflict area insert mode.

Zoom into EB approach where detectors are not placed yet. For this process, it is beneficial to have label turned on for signal heads (signal group attribute) and detectors (port number attribute).

Select link where detector will be placed (Link 8) by left clicking and press <CTRL> + right click at the starting point of this detector and drag over following the direction of travel.

Once you reach the end, release both <CTRL> key and right mouse button and it will create new detector and open “Detector” window.

Enter the following details to detectors for each EB lane:

EBL: SC number 4 / Port number 3 / Standard Type / Length 40ft

EBT (Both lanes): SC number 4 / Port number 8 / Standard Type / Length 40ft

To model push buttons for pedestrians, zoom into crosswalk links across the EB approach and add following detectors:

Crosswalk (Both directions): SC number 4 / Port number 102 / Standard Type / Length 4 ft.

Once detectors for both directions are placed, make sure that each detector is located upstream of respective signal head and double click on each detector and enter 4.0 ft to “Before stop” parameter and it will be placed exactly at the signal head and upstream of it.

Complete all detector settings and click “Save”.

“Run” simulation and observe.

2.1.2 Applications

There are two main categories applied to basic signal operations such as 1) Adaptability and 2) Connectivity. First, you can update adaptability by using actuation techniques and setting it to operate in a same pattern regardless of traffic condition or in a dynamic pattern depending on the traffic condition by using detectors. In addition, if there are multiple signalized intersections along the corridor and they are located close enough so that one intersection is affecting the operations of another intersection significantly, you can consider connecting them to provide progression along the desired path. The chart below shows four common types of signal operation divided by the level of connectivity and adaptability.

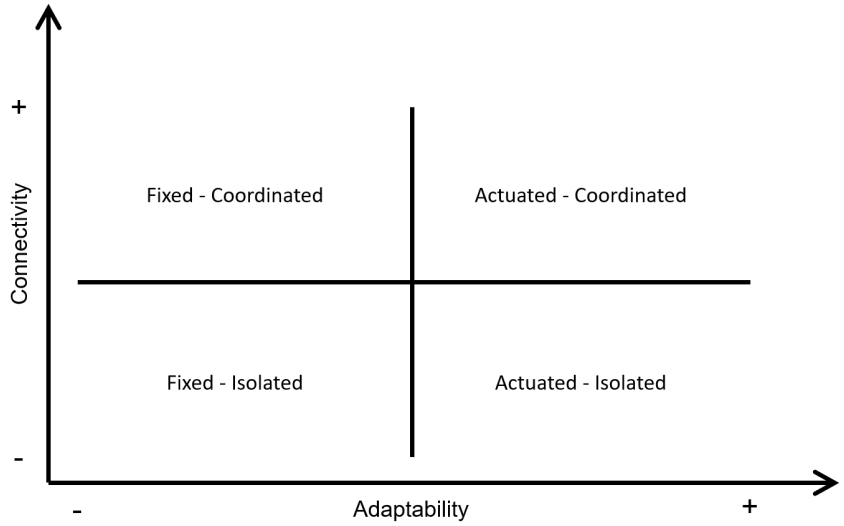


Figure 2.1-18: Main types of Signal Operations

The basic steps to model each of these techniques stays identical for the most part; however, there are few objects and parameters which may need to be added or further modified to achieve the desired type of operation. In this section, details on objects and parameters that require additional attention are described for different types of signalized intersection operation strategies.

2.1.2.1 Individual Intersection: Fixed Time Control:

Modeling fixed time control does not require any new objects to be added to the network; however, it will require a few parameters in the signal timing to be modified.

2.1.2.2 [Step 2A] Signal Timing Plan Update for Fixed Time Control Modeling:

Following is the list of four (4) parameters in the basic or pattern sections of the signal timing plan that need to be checked and modified to model fixed time signal control:

Max Green

If the signal is in free running mode or max green mode is not set to be "Max Inhibit", max green time will determine how long each SG should be. The length of green time may be extended if any SG in concurrent barrier group is set to be longer.

Max green time is typically used for isolated intersections.

Splits

If the signal is running any (time of day) pattern and max green time does not interfere by having 1) Max Inhibit selected for max green mode and/or 2) long enough max green time to serve splits, the duration of green time will be defined by split length.

Split is typically used for signalized intersection which is a part of coordinated corridors.

Max Recall

This is a checkbox that is assigned to each vehicular SG. Once max recall is turned on, the selected vehicular SG will get continuous vehicle call automatically so that it will be served for maximum duration in time regardless of traffic condition.

To model fixed time control, turn this parameter on so that the duration of green time assigned to each SG will stay fixed.

Pedestrian Recall

This is a checkbox that is assigned to each pedestrian SG. Once Max recall is turned on, the selected pedestrian SG will get pedestrian call automatically so that it will be served for Walk and Ped clearance (FDW) without any pedestrian demand.

To model fixed time control and if there is pedestrian crossing, turn this parameter on so that the duration of Walk and Ped clearance time assigned will stay fixed.

Exercise

Open "Step_2A_Fixed_Time" folder and load "Stringfellow_Rd_Step_2A.inpx" file.

Go to "Signal Control > Signal Controllers" to open "Signal Controllers" window and select SC number 4.

Click on Edit button on the menu () to open "Signal Controller" window.

Click on "Edit signal groups" button in "Controller configuration" tab to open RBC editor.

Update the following parameters in RBC Editor:

Max Recall for all SGs: Checked

Ped Recall for all Ped SGs: Checked

Click "OK" in RBC editor and Signal Controller window to save.

"Run" simulation and observe (especially signal times table on the right bottom corner) fixed time control operation.

2.1.2.3 Individual Intersection: Actuated Control:

Actuated signal control can be modeled by modifying the list of parameters below in the signal timing plan. Depending on whether this signalized intersection is isolated or a part of a coordination group, the list of required parameters varies. Note that all recall modes will have to be turned off to activate actuated control.

2.1.2.4 [Step 2B] Signal Timing Plan Update for Actuated Control Modeling:

Following is the list of four (4) parameters in the basic or pattern sections of the signal timing plan that need to be checked and modified to model actuated time signal control:

Min Green

This defines the minimum duration of green time that each SG will serve without any successive call(s). Therefore, this is how short each SG will be in case of actuation.

Max Green

If max green mode is not set as "Max Inhibit", the max green time defines the maximum duration of green time that each SG will serve.

Max green time is typically used for isolated intersections.

Splits

If max green mode is set as "Max Inhibit" and/or max green time is longer than the time allowed by split settings, the maximum duration of green time for each SG is set by split.

Split is typically used for signalized intersection which is a part of coordinated corridors.

Vehicle Extension

This defines the duration of acceptable time gap in between vehicles to extend green times for each SG. Vehicle extension must be defined; otherwise, the SC will assume that there is no call after serving min green time.

In order to detect the traffic condition, the detector object must be added and connected to the corresponding SC. This configuration can be set for both vehicle and pedestrian detectors and the following is a set of parameters which need to be defined:

Vehicle Detector

1. Define vehicle detector numbers on "Detector Number" row. Note that this number must match with "Port Number" used for corresponding detector object setting. Typically, detector numbers are set to be identical with SG numbers unless there is any data provided.
2. Select desired SG(s) for both "Call" and "Extend SGs" parameters. Call will place initial call to the detector and Extend SGs will place calls for extensions. Typically, identical SG(s) is selected for both call and extend SGs.

Pedestrian Detector

1. Define pedestrian detector numbers on "Detector Number" row. Note that this number must match with "Port Number" used for corresponding detector object setting.
2. Select desired pedestrian SG for "Call Peds" parameter. Call will place initial call and it works the same as "Push button" in the field.

2.1.2.5 [Step 5A] Add Detector for Actuated Movements:

The detector object shall be added to each location based on background map or intersection design drawings. In case of semi-actuated control, such as coordinated movement along major movement, you may not need detectors for certain lane(s).

Following is the minimum set of parameters that will need to be set for actuated operations:

Port no.

This is the port number that will allow communication in between SC and detector. As described above, this number must match with the detector number set in the signal timing plan. Multiple detectors can have the same port number if necessary.

SC no.

Select SC number that this detector is connecting to. If it is necessary to have the detector connecting to multiple SCs, multiple detectors should be created for each SC.

Type

Select "Standard" type since this type transmits both pulse and presence calls and it is suitable for typical signal operation cases.

Length and Location

Specify the length of detection range. The value "0.000" is e.g. permissible and useful for modeling trolley wire contacts and pedestrian sensors. Location contains info on which link/lane and how far from start of the link/signal head is detector.

Vehicle Type

Select "All Vehicle Types" unless this detector is used to model any special case.

Exercise

1. Open "Step_2B_5A_Actuated_Time" folder and load "Stringfellow_Rd_Step_2B_ 5A.inpx" file.
2. Go to "Signal Control > Signal Controllers" to open "Signal Controllers" window and select SC number 4.
3. Click on Edit button on the menu () to open "Signal Controller" window.
4. Click on "Edit signal groups" button in "Controller configuration" tab to open RBC editor.
5. Update following parameters in RBC Editor:
   - Max Recall for all SGs: Uncheck
   - Ped Recall for all Ped SGs: Uncheck
   - Vehicle Detectors

Table 2.1-3: Vehicle Detector

Detector Number	1	2	3	4	5	6	7	8
Call	1	2	3	4	5	6	7	8
Extend SGs	1	2	3	4	5	6	7	8

- Ped Detectors

Table 2.1-4: Ped Detectors

Detector Number	102	104	106	108
Call Peds	102	104	106	108

6. Click “OK” in RBC editor and Signal Controller window to save.
7. Click “Detectors” in the network objects window to be in conflict area insert mode.
8. Zoom into EB approach where detectors are not placed yet. For this process, it is beneficial to have label turned on for signal heads (signal group attribute) and detectors (port number attribute).
9. Select link where detector will be placed (Link 8) by left clicking and press <CTRL> + right click at the starting point of this detector and drag over following the direction of travel.
10. Once you reach to the end, release both <CTRL> key and right mouse button and it will create new detector and open “Detector” window.
11. Enter following details to detectors for each EB lane:
    - EBL: SC number 4 / Port number 3 / Standard Type / Length 40ft
    - EBT (Both lanes): SC number 4 / Port number 8 / Standard Type / Length 40ft
12. Complete all detector settings and click “Save”.
13. “Run” simulation and observe (especially signal times table on the right bottom corner) fully actuated time control operation.

2.1.2.6 Signalized Corridor: Coordination:

To update SC as a part of a signalized intersection corridor with coordination, a number of parameters in the signal timing plan needs to be updated.

2.1.2.7 [Step 2C] Signal Timing Plan Update for Coordinated Signal Control Modeling:

There are couple of parameters which are required to be updated in addition to settings required for either fixed or actuated time control above.

Coordinated

In order to apply coordination, SGs which are coordinated must be defined. You cannot have multiple coordinated SGs on the same ring and all coordinated SGs must be in the same barrier group. If coordinated SGs are not defined, the controller will run in free running mode.

Identify SGs to be coordinated and turn on all of them for desired patterns.

Cycle Length

Cycle length is the total duration in time to serve complete sequence of SGs for each SC. To coordinate any SCs, it is necessary to have consistent cycle length (except for half cycle or double cycle) and maintain consistent green time band throughout the corridor.

Set desired cycle length and make sure that the sum of the splits of all SGs in each ring adds up to the cycle length.

Offset

When SCs are set to be coordinated, it is necessary to define the starting point of each SC clock as a relative time stamp. In the simulation model, unlike in the field, the master clock does not have to be defined. The simulation model will use its own simulation clock as master and all other times are understood as relative to it.

It still plays a critical role in the signal coordination and it is very important to enter an accurate value. Also, it is important to ensure that the “Offset Reference” point is checked thoroughly

because different offset reference point may result in significant and unexpected interruption to the green band coordination.

Permissive Mode

This setting defines the method in which permissive periods (the window of opportunity that each SG will accept calls within given cycle) are opened and closed for all non-coordinated SGs. The controller will only yield to SGs that are permissive following the end of green on each coordinated SG (yield point). If you do not have information on permissive mode setting, it is recommended to stay at "Single Band" mode which is default in RBC. The permissive modes are as follows:

Single Band

The permissive period for non-coordinated SGs will open:

At the beginning of the coordinated SG green for SGs in the same ring and concurrent barrier group as the coordinated SG, or

At the beginning of the lagging coordinated SG green for SGs outside of the same concurrent barrier group as the coordinated SGs.

The permissive period for non-coordinated SGs will close:

When there is no longer enough time to clear all timing SGs and serve the longer of the minimum green or permissive green on the SG, or

When the SG is in a different concurrent barrier group then the coordinated SGs and any coordinated SG has yielded to a SG that is sequentially before the coordinated SG, in the same ring and concurrent barrier group.

Multi Band

The permissive period for non-coordinated SGs will open:

The same as single band permissive operation above, but only for the first SG in each ring that sequentially follows the coordinated SG. For each subsequent SG, the permissive period will open once the previous SG's permissive period closes.

The permissive period for non-coordinated SGs will close the same as they do for single band permissive operation above.

Reservice

The permissive mode will operate the same as single band permissive until the coordinated SG yields to a non-coordinated movement. All SGs will be allowed to reserve. After the coordinated SGs yield once:

SGs in the non-coordinated barrier group will be allowed to reserve if there is enough time to serve the minimum green time and still be able to have the leading coordinated SG green by the start of its split.

SGs in the coordinated barrier group will be allowed to reserve if there is enough time to serve the minimum green time and still be able to have the coordinated SG in the same ring green by the start of its split.

Additionally, there are two different parameters for coordinated signal operation which will be set for the entire SC.

Offset Reference

This is the point in the cycle where the cycle timer will be equal to the defined offset time when the controller is coordinated. The selections are:

LagFO (Lagging Force-Off): The reference point will be at the force-off point for the lagging coordinated SG.

LeadGreen (Leading Start of Green): The reference point will be at the start of the leading coordinated SG green.

LagEnd (End of Lagging Red): The reference point will be at the end of red clearance for the lagging coordinated SG.

CoordEnd (End of Coordinated Group Red): The reference point will be at the end of red for the last SG in the concurrent barrier group with the coordinated SGs.

The timing diagram below shows each of the offset reference points.

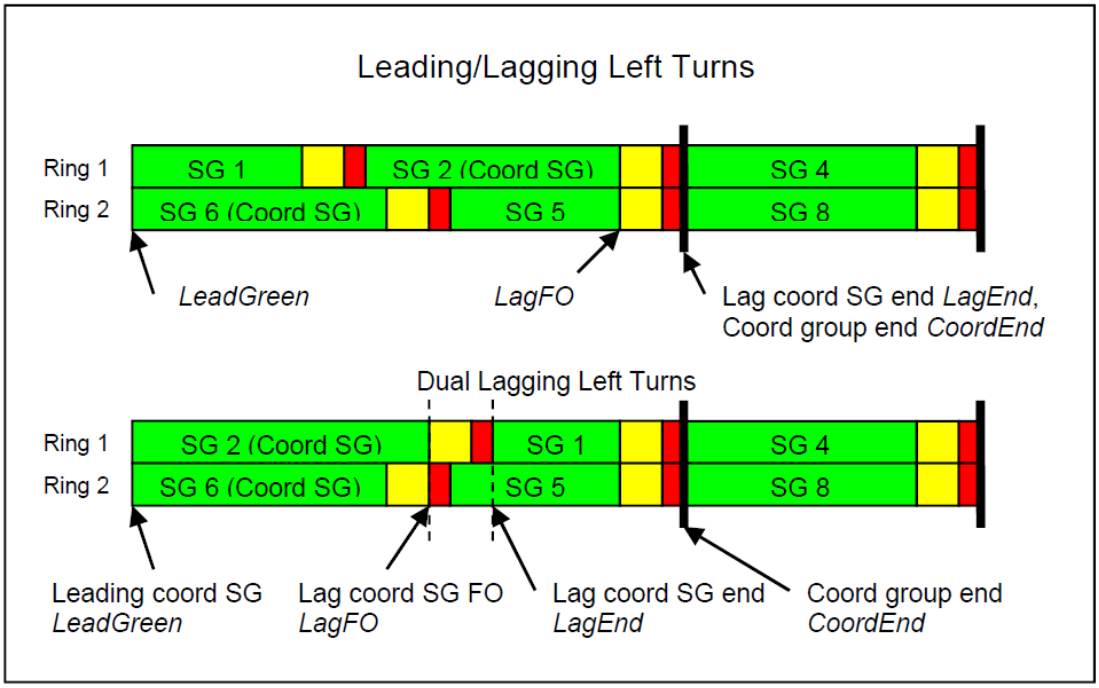


Figure 2.1-19: Timing Diagram

Transition Mode

This is the mode that all coordination patterns will use to transition when the local dial does not have the correct offset to the master clock in Vissim. The max transition values are automatically set to the split value plus 20% and will be adjusted whenever a split value is changed. The selections are:

Best: The controller will determine whether implementing one or more shorter cycles, or one or more longer cycles will achieve coordination the quickest. Min Green and pedestrian SG timing restrict how short a transition cycle can be.

LongMode: The controller will implement one or more longer cycles in order to achieve coordination.

BestIgnorePed: Same as “Best” above, except when determining how short a transition cycle can be, the controller will ignore minimum pedestrian SG timing for all actuated pedestrian movements.

Exercise

Open “Step_2C_Coordinated_Control” folder and load “Stringfellow_Rd_Step_2C.inpx” file.

Go to “Signal Control > Signal Controllers” to open “Signal Controllers” window and select SC number 4.

Click on Edit button on the menu () to open “Signal Controller” window.

Click on “Edit signal groups” button in “Controller configuration” tab to open RBC editor.

Update the following parameters in RBC Editor:

Max Recall for all SGs: Uncheck

Ped Recall for all Ped SGs: Uncheck

Pattern 1:

Table 2.1-5: Pattern 1

Signal Group	1	2	3	4	5	6	7	8
Splits	29	89	9	53	38	80	10	52
Coordinated		X				X		

Schedule:

Table 2.1-6: Schedule

Pattern Number	**Pattern Start**
1	0

Vehicle Detectors:

Table 2.1-7: Vehicle Detectors

Detector Number	1	3	4	5	7	8
Call	1	3	4	5	7	8
Extend SGs	1	3	4	5	7	8

Ped Detectors:

Table 2.1-8: Ped Detectors

Detector Number	102	104	106	108
Call Peds	102	104	106	108

Pattern Global:

Table 2.1-9: Pattern Global

Cycle Length	180
Offset	21

Click “OK” in RBC editor and Signal Controller window to save.

“Run” simulation and observe coordinated operation.

2.2 Special Controls

Contributor(s): Tracy Zhou from Texas A&M Transportation Institute (TTI)

2.2.1 Conventional Diamond Interchange

A conventional diamond interchange is an at-grade interchange that has two closely spaced intersections of an arterial with a one-way pair of frontage roads. Figure 2.2-1 illustrates an example diamond interchange with signalized control.

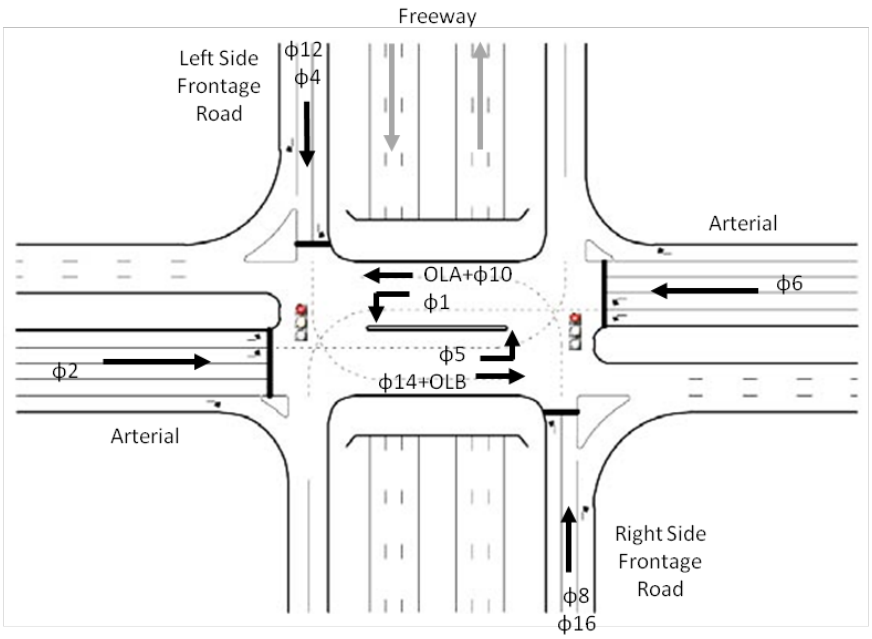


Figure 2.2-1: Conventional diamond interchange

Signal control at conventional diamond interchanges often adopts the Three-Phase operation or the Four-Phase operation. Figure 2.2-2 shows the typical phase sequence for each of the two operation methods.

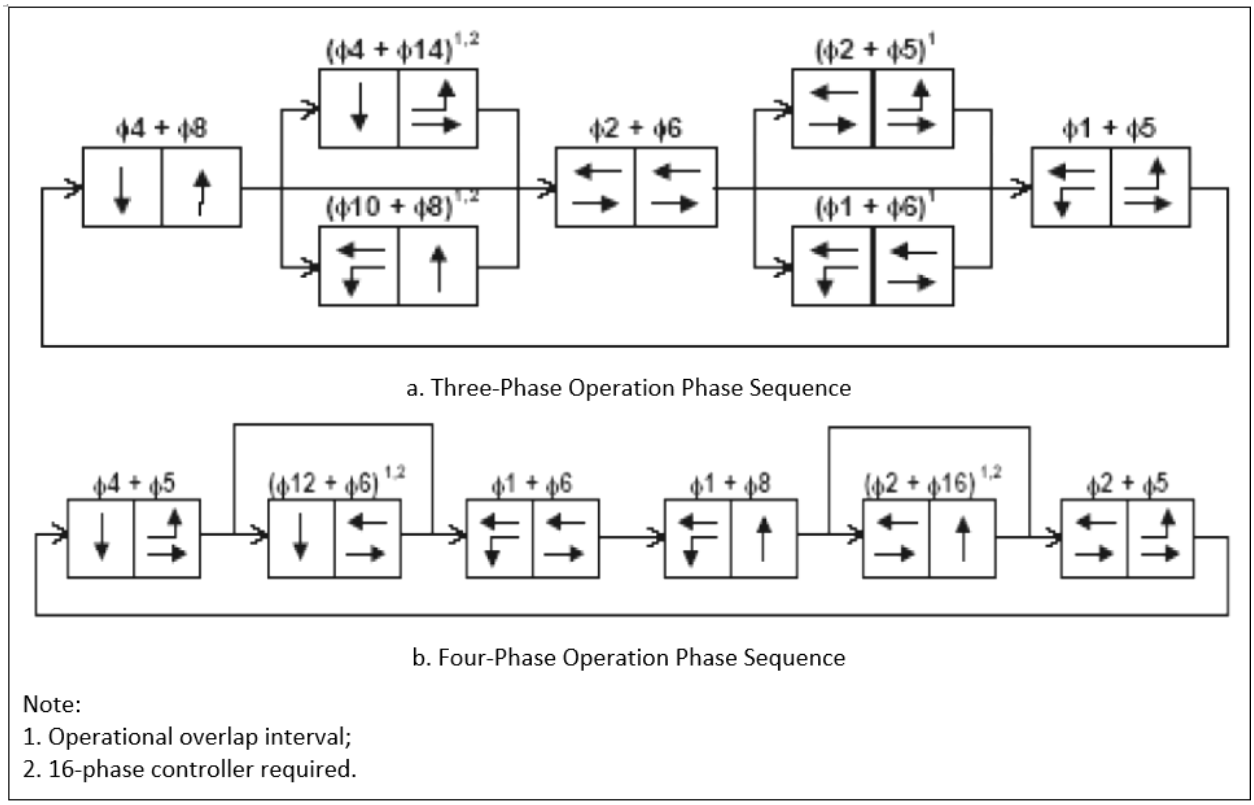


Figure 2.2-2: Typical Signal Operations at Conventional Diamond Interchanges

The Three-phase operation services external through movements simultaneously (ϕ2+ϕ6 or ϕ4+ϕ8) without having to be interrupted by interior left turns (ϕ1 or ϕ5). The Three-Phase operation is ideal for wide interchanges (long distance between the two intersections) in rural areas with light overall traffic and heavy through movements.

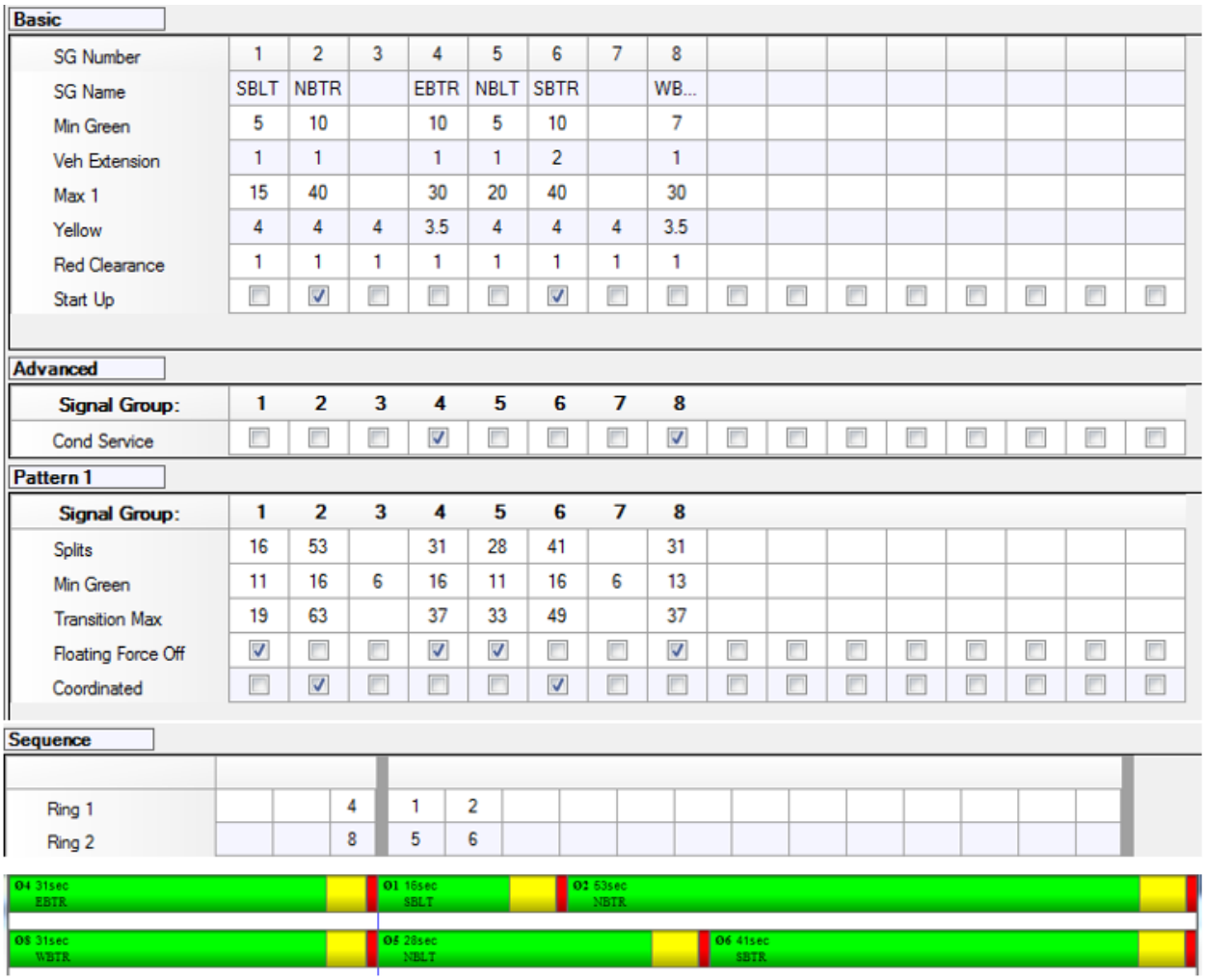
The Four-Phase operation runs each exterior movement independently (ϕ2, ϕ6, ϕ4, or ϕ8) and services interior movements in concurrent external phases efficiently. This operation aims to provide progression for all movements entering the interchange to minimize stops inside the interchange. The Four-Phase operation is most effective at tight urban diamond interchanges (intersection distance up to 400 ft) with heavy turning movements.

Simulation of signal operation at conventional diamond interchanges follows the general procedures provided by the FHWA Guidelines 2 in coding geometry and traffic information. This section demonstrates signal controller settings to realize Three-Phase and Four-Phase operations with two example models developed using PTV Vissim 8.00. The models can be found in the "Diamond_3-Phase" and "Diamond_4-Phase" VISSIM example folders.

Example of Three-Phase Operation

The "Diamond_3-Phase" model is developed for a rural diamond interchange with intersection spacing about 470 ft. The total traffic entering the interchange is less than 1000 vph during the peak hour. The interchange operates a Three-Phase control with a cycle length of 100 sec. The arterial external movements ϕ2 and ϕ6 are coordinated, and the frontage road movements ϕ4 and ϕ8 are conditional services. The interior left turns ϕ1 and ϕ5 lead the opposing arterial through movements ϕ2 and ϕ6, respectively. Floating force off is used to provide maximum capacity for arterial movements. Phases ϕ3 and ϕ7 are reserved as optional phases for future usage. Figure 2.2-3 shows the Base Timing settings in the RBC signal controller and the generated ring barrier diagram for the Three-Phase operation.

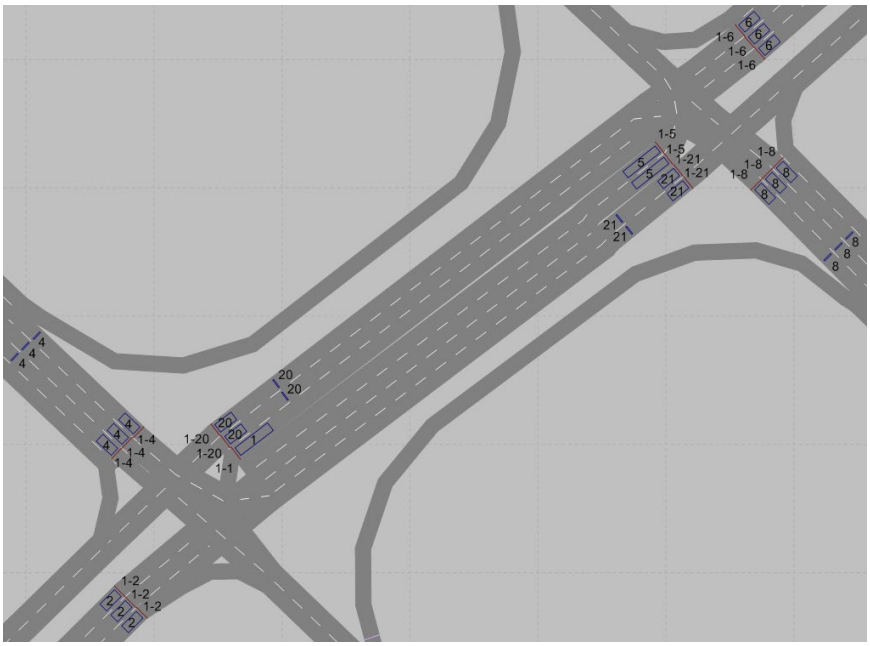
[2] Traffic Analysis Toolbox Volume III: Guidelines for Applying Traffic Microsimulation Modeling Software



Note: Settings not shown are left blank.

Figure 2.2-3: Vissim RBC Signal Controller Base Timing Settings for the Diamond_3-Phase Example

Note that the interior through movements run as overlaps of the concurrent interior left turns and the opposing through (ϕ1+ ϕ2 for SB interior through, ϕ5+ϕ6 for NB interior through). This is set by assigning detector #20 and #21 to call ϕ1+ ϕ2 and ϕ5+ϕ6, respectively, and placing these detectors on the corresponding interior through lanes (Detector PortNo #20 for southbound and PortNo #21 for northbound interior through lanes). Accordingly, two overlap signal groups #1-20 and #1-21 are assigned for the two interior through movements, and signal heads using the two overlap signal groups are placed at the stop lines of these interior through lanes. Figure 2.2-4 shows the relevant settings in the RBC signal controller to realize the Three-Phase operation.



Note: notation numbers with a dash indicate signal group IDs and numbers without a dash are detector IDs.

Figure 2.2-4(a): Assignment of Signal Groups and Detectors for South Intersection

Overlaps

Overlap SG	20	21													
Yellow Clearance	4	4													
Red Clearance	1	1													
Parent	1,2	5,6													

Veh Detectors

Detector Number	1	2	4	5	6	8	20	21
Detector Mode	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect
Added Initial Mode	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled
Call	1	2	4	5	6	8	1,2	5,6
Extend SGs	1	2	4	5	6	8	1,2	5,6

Note: Settings not shown are left blank.

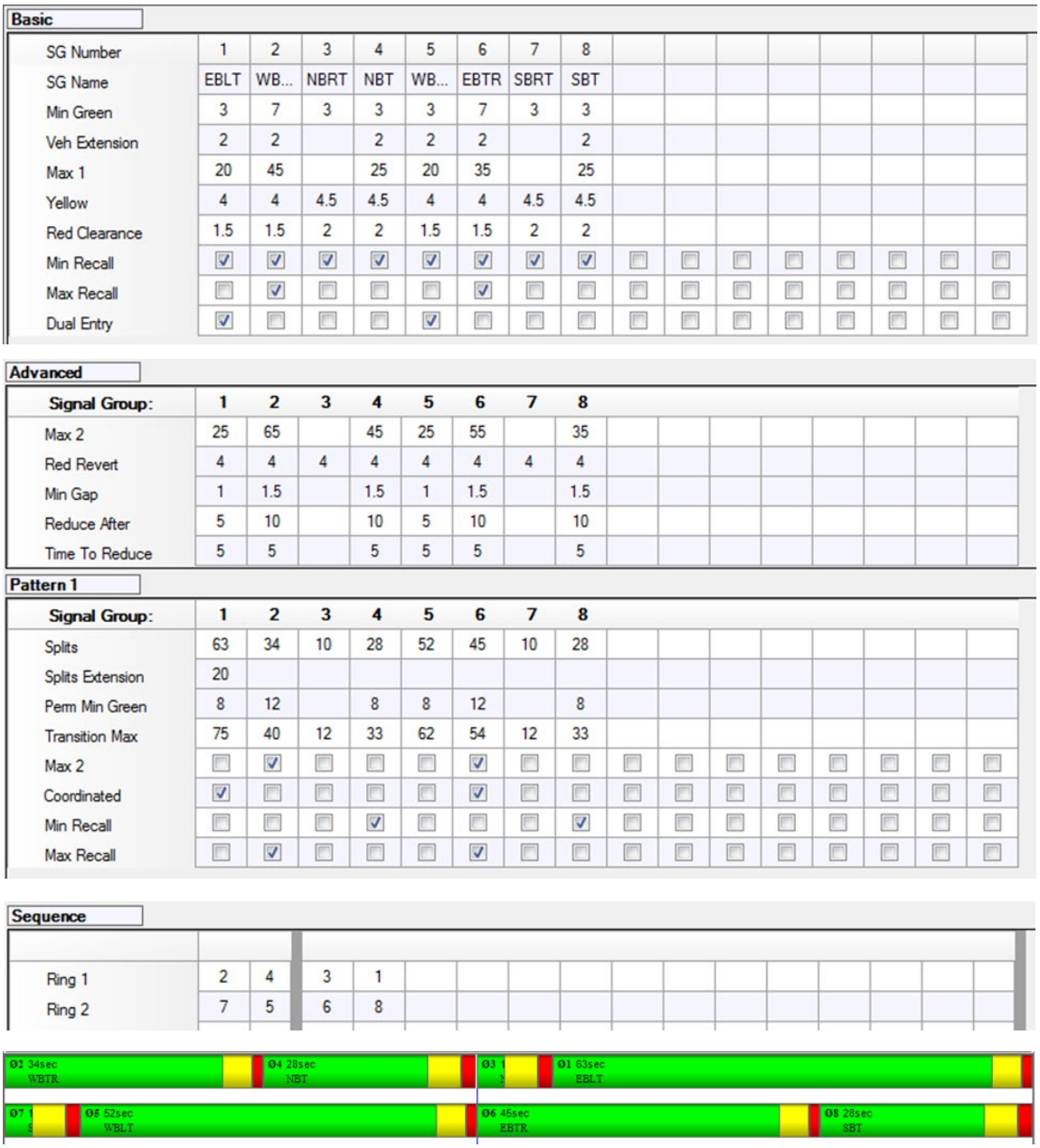
Figure 2.2-4(b): Overlap and Detector Settings for RBC Signal Controller

Figure 2.2-4: Overlap Signal Group and Detector Settings for Interior Movements in Diamond 3-Phase Example

Example of Four-Phase Operation

The "Diamond_4-Phase" model is developed for a tight urban diamond interchange with intersection spacing about 330 ft. The total traffic entering the interchange is close to 8000 vph during the peak hour. The interchange operates a Four-Phase control with a cycle length of 135 sec. Arterial eastbound movements ϕ6 and ϕ1 are coordinated. Interior left turns lead the opposing through. Due to heavy frontage right turn demand, ϕ3 and ϕ7, equivalent to ϕ12 and ϕ16,

respectively, are used to add additional capacity. Figure 2.2-5 shows the Base Timing settings in the RBC signal controller.



Basic

SG Number	1	2	3	4	5	6	7	8
SG Name	EBLT	WB...	NBRT	NBT	WB...	EBTR	SBRT	SBT
Min Green	3	7	3	3	3	7	3	3
Veh Extension	2	2		2	2	2		2
Max 1	20	45		25	20	35		25
Yellow	4	4	4.5	4.5	4	4	4.5	4.5
Red Clearance	1.5	1.5	2	2	1.5	1.5	2	2
Min Recall	☑	☑	☑	☑	☑	☑	☑	☑
Max Recall	☐	☑	☐	☐	☐	☑	☐	☐
Dual Entry	☑	☐	☐	☐	☑	☐	☐	☐

Advanced

Signal Group:	1	2	3	4	5	6	7	8
Max 2	25	65		45	25	55		35
Red Revert	4	4	4	4	4	4	4	4
Min Gap	1	1.5		1.5	1	1.5		1.5
Reduce After	5	10		10	5	10		10
Time To Reduce	5	5		5	5	5		5

Pattern 1

Signal Group:	1	2	3	4	5	6	7	8
Splits	63	34	10	28	52	45	10	28
Splits Extension	20							
Perm Min Green	8	12		8	8	12		8
Transition Max	75	40	12	33	62	54	12	33
Max 2	☐	☑	☐	☐	☐	☑	☐	☐
Coordinated	☑	☐	☐	☐	☐	☑	☐	☐
Min Recall	☐	☐	☐	☑	☐	☐	☐	☑
Max Recall	☐	☑	☐	☐	☐	☑	☐	☐

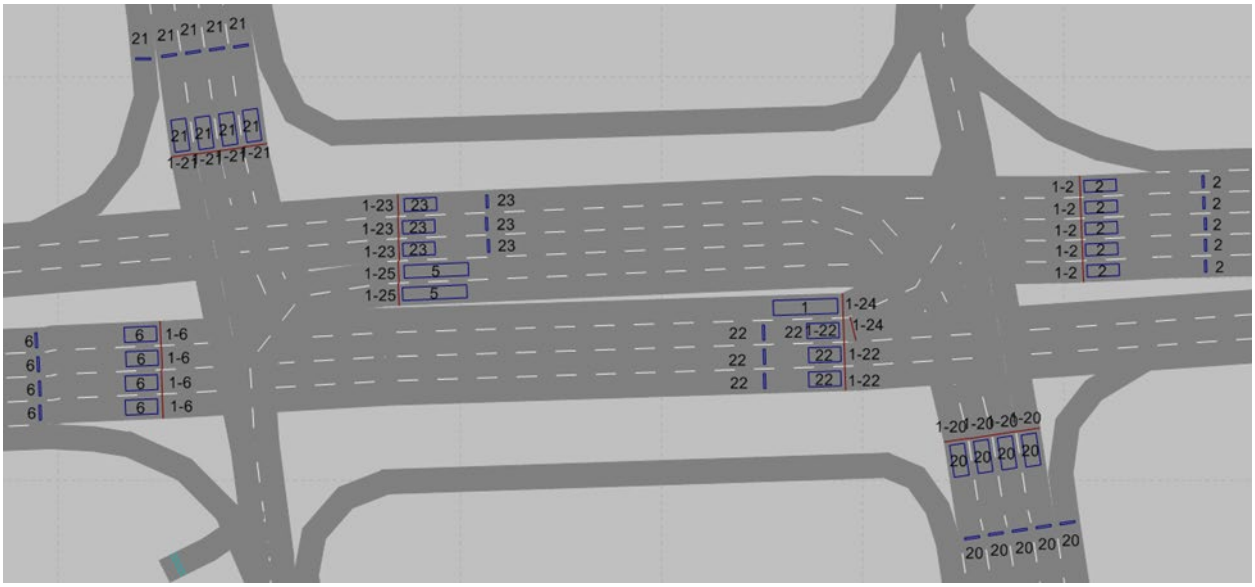
Sequence

Ring 1	2	4	3	1
Ring 2	7	5	6	8

Note: Settings not shown are left blank.

Figure 2.2-5: Vissim RBC Signal Controller Base Timing Settings for the Diamond 4-Phase Example

The frontage road movements run as overlap of the through and right turn phases (ϕ4+ϕ3 or ϕ8+ϕ7). The interior left turn signal run as the overlap of interior left turns and the concurrent frontage right turns (ϕ1+ϕ3 or ϕ5+ϕ7). Similar to Three-Phase operation, the interior through signals run as overlaps of the opposing through and interior left turns (ϕ2+ϕ1 or ϕ6+ϕ5). The method of numbering and placement of overlap signal groups and detectors are similar to that of the Diamond_3-Phase example and is shown in Figure 2.2-6.



Note: notation numbers with a dash indicate signal group IDs; numbers without a dash are detector IDs.

Figure 2.2-6(a): Assignment of Signal Groups and Detectors for South Intersection

Overlaps

Overlap SG	20	21	22	23	24	25
Parent	3,4	7,8	1,2	5,6	1,3	5,7

Veh Detectors

Detector Number	1	2	5	6	20	21	22	23
Detector Mode	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect
Added Initial Mode	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled
Call	1	2	5	6	3,4	7,8	1,2	5,6
Extend SGs	1,3	1,3	1,3	1,3	1,3	1,3	1,3	1,3

Note: Settings not shown are left blank.

Figure 2.2-6(b): Overlap and Detector Settings for RBC Signal Controller

Figure 2.2-6: Overlap Signal Group and Detector Settings for Interior Movements in Diamond_4-Phase Example

Note that the eastbound interior left turns use one exclusive left turn lane and one shared lane in this example. Two signal heads are used for this shared left turn lane: one left turn signal head (Overlap SG #1-24) is placed on the left turn lane connector, and one through signal head (Overlap SG #1-22) is placed on the link upstream of the left turn connector. An alternative way of modeling this is to assign both left turn and through signal groups (Overlap SG #1-22 OR #1-24) to the shared lane (signal head placed on the link, not on the connector).

2.3 Transit Signal Priority in VISSIM Microsimulation

Contributor(s): Milan Zlatkovic, PhD, PE, PTOE, University of Wyoming

2.3.1 Transit Signal Priority (TSP) Overview

Transit Signal Priority (TSP) is an operational strategy that facilitates the movement of transit vehicles (usually those in-service), either buses or streetcars, through traffic-signal controlled intersections. It makes transit faster, more reliable, and more cost-effective (*Smith, Hemily, Ivanovic, 2005*). Expected benefits of TSP vary depending on the application but include improved schedule adherence and reliability and reduced travel time for buses, leading to increased transit quality of service. Potential negative impacts consist primarily of delays to nonpriority traffic, and these delays have proven to be minimal.

A transit agency has two objectives for using TSP: improve service and decrease costs. Through customer service enhancements, the transit agency could ultimately attract more customers. Fewer stops also mean reductions in drivers' workload, travel time, fuel consumption, vehicle emissions, and maintenance costs. Greater fuel economy and reduced maintenance costs can increase the efficiency of transit operations. TSP can also help reduce transit operation costs, as reductions in transit vehicle travel times may allow a given level of service to be offered with fewer transit vehicles. Local transportation agencies can also benefit from TSP strategies when improved transit service encourages more auto users to switch to public transportation. Finally, reduced demand for personal car travel can help improve roadway service level.

TSP can be implemented in different ways, including passive, active, and adaptive TSP (*Smith, Hemily, Ivanovic, 2005*). Passive TSP is the simplest type of TSP. It does not require any hardware or software installations, but the priority operates continuously, based on knowledge of transit route and ridership patterns, and does not require a transit detection or priority request. This can be an efficient form of TSP when transit operations are predictable. A simple passive priority strategy is establishing signal progression for transit, where the signal timings plan takes into account transit operational characteristics such as the average dwell time at transit stops, or considering that dwell times are highly variable, and use as low a cycle length as possible. Sometimes, a simple retiming of signal plans in order to improve progression along a corridor can be beneficial for transit vehicles too.

Active priority strategies provide priority treatment to a specific transit vehicle following detection and subsequent priority request activation. There are different types of active priority strategies that may be used within the specific traffic control environment. A green extension strategy extends the green time for the TSP movement when a TSP equipped vehicle is approaching. This strategy only applies when the signal is green for the approaching transit vehicle. This is one of the most effective forms of TSP since a green extension does not require additional clearance intervals yet allows a transit vehicle to be served and significantly reduces the delay to that vehicle relative to waiting for an early green or special transit phase. An early green strategy, also known as red truncation, shortens the green time of preceding phases to expedite the return to green for the movement where a TSP equipped vehicle has been detected. This strategy only applies when the signal is red for the approaching transit vehicle. Usually, green extension and early green strategies are implemented simultaneously within TSP enhanced control environments, and the controller uses one of them depending on the specific situation. These strategies are shown in Figure 2.3-1 a). Some other active TSP strategies are actuated transit phases, where a specific phase, usually a left turn phase, is displayed only when a transit vehicle is detected; phase insertion,

where a special priority phase is inserted within the normal signal sequence when a transit vehicle is detected and a call for priority is placed (Figure 2.3-1 b)); and phase rotation, where a normal sequence of signal phases is rotated when a priority call is placed, in order to serve the priority phase first (Figure 2.3-1 c)). Any, or a combination, of the active priority strategies can be used depending on the specific situation and traffic and transit operations.

Adaptive TSP is the most comprehensive strategy that takes into consideration the trade-offs between transit and traffic delay and allows graceful adjustments of signal timing by adapting the movement of the transit vehicle and the prevailing traffic condition. It can also consider some other inputs, such as if the transit vehicle is running on time or it is late, the headway between two successive transit vehicles, the number of passengers on board, etc.

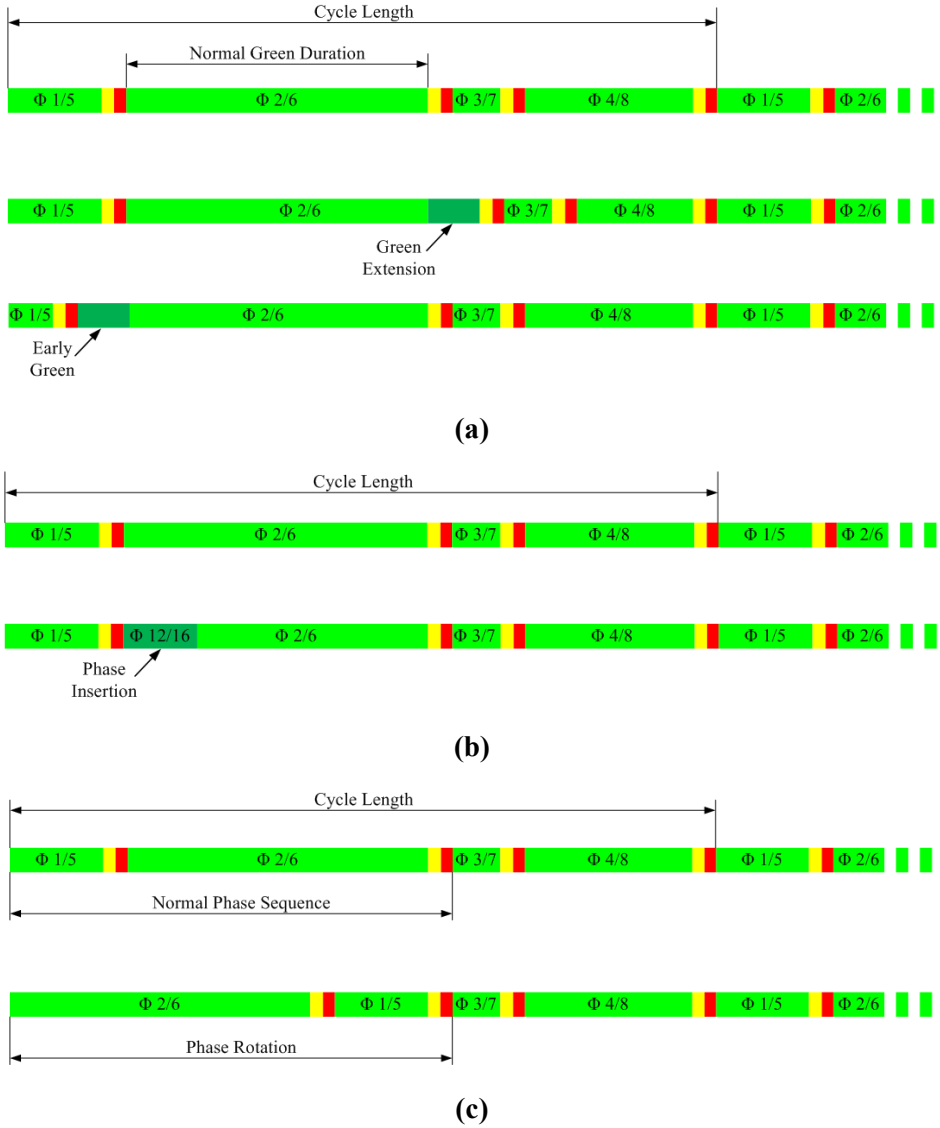


Figure 2.3-1: Active TSP Strategies (Φ 2/6 transit phases)

2.3.2 TSP in Simulation

In general, simulation is defined as the process of replication of a real-world situation. Computer models are the most common tool in assessing TSP characteristics. The model is used to predict system performance based on interactions between its components. Performance of a traffic system is evaluated through different Measures of Effectiveness (MOE), such as travel times, system throughput, delays, queues, etc.

Traffic simulation models can be classified into three categories, based on the fidelity of the model and the level of details. These categories are macroscopic models, with a low level of details, mesoscopic models, with a moderate level of details, and microscopic models, with the highest level of details and simulation of individual vehicles and their interactions. Microsimulation models provide extensive reporting capabilities, on a sub second-by-second basis, which is an important requirement for a TSP evaluation. Due to their ability to simulate individual vehicles in the network, only microscopic models are able to simulate TSP applications at the individual intersection level, which is why they are widely used for this purpose, so these guidelines will only focus on microscopic models.

Traffic simulation allows for testing of new systems, or changes made in old systems, before they are implemented in the field. It also allows for testing of different alternatives and "what if" scenarios, which cannot be tested in the field. Some of the advantages of traffic simulation are the following (*Smith, Hemily, Ivanovic, 2005*):

- Providing a cost effective way of testing and evaluating different scenarios
- Allowing the user to test scenarios faster than real time
- Offering an insight into the important characteristics of traffic system operations, and allowing the user to make a more informed decision
- Providing outputs/animation that the public can understand

On the other hand, simulation has some disadvantages, such as the following:

- Requires collection of detailed data on field conditions
- Needs calibration and validation of the model prior to testing scenarios
- Requires an understanding of how the model works before assessing the outputs

There are many traffic simulation packages available on the market today, such as AIMSUN, CORSIM, NETSIM, PARAMICS, SIMTRAFFIC, TRANSIT 7F, VISSIM, etc. Each of them can be used to simulate TSP. The most significant limitations include the ability to simulate different characteristics of transit systems, and signal control logic and detection. TSP simulation in VISSIM can be achieved through built-in TSP strategies in the Ring Barrier Controller (RBC) emulators, or through software-in-the-loop (SIL) traffic control simulation (such as ASC/3 SIL controllers that incorporate complex TSP strategies). In this document, TSP in microsimulation will be described using VISSIM as the simulation platform.

2.3.3 SP Settings in RBC Controllers

RBC controllers are the built-in traffic control emulators in VISSIM. They allow TSP settings for up to eight signal groups. Each TSP signal group is tied to a vehicular signal group, defined as the parent signal group (*PTV Group, 2014*). Priority service for any transit signal group can be enabled. When a transit signal group operates in a priority mode, signal groups that conflict with the parent signal groups of a transit signal group can be abbreviated or omitted based on the defined parameters. The controller will attempt to adjust its operation so that it can have the transit signal group green by the time the vehicle arrives at the intersection. When the signal controller is recovering from a TSP operation, the recovery green is proportional to all upcoming signal groups. It covers all signal groups following the TSP signal groups up through the coordinated signal groups. The proportional recovery green to each signal group is computed as a percentage between the minimum split (based on minimum green or priority minimum green) up to the full split. The percentage used to calculate the recovery is the percentage of how far the TSP phase extended versus the absolute maximum TSP extension allowed, which is calculated automatically by the controller based on minimum splits for all signal groups.

RBC controllers introduce user friendly GUI, where the TSP settings are defined step by step. First the transit inputs need to be enabled, by checking the box next to "Transit Inputs" in the menu tree, as shown in Figure 2.3-2.

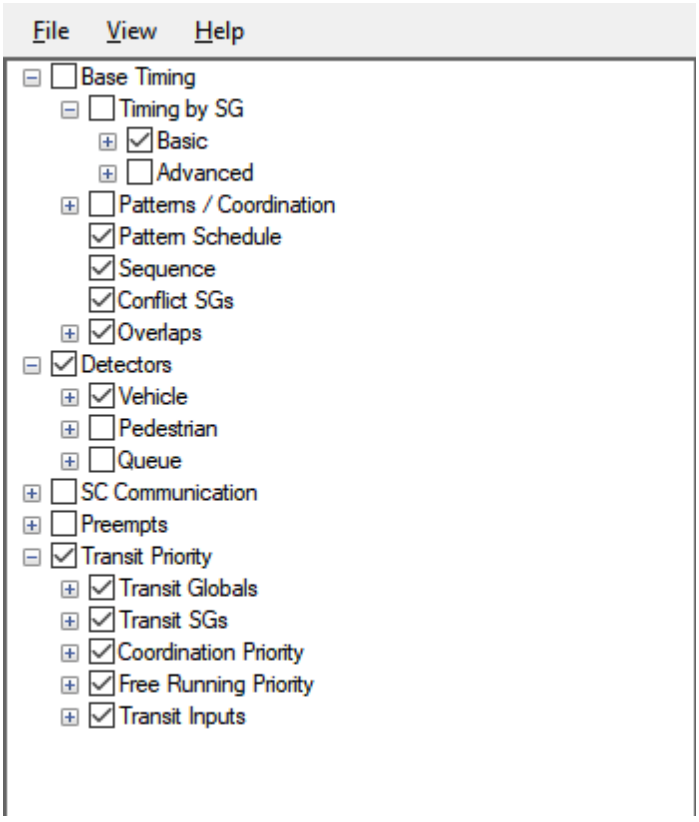


Figure 2.3-2: Enabling TSP in RBC

The transit global inputs are shown in Figure 2.3-3. The "Detector Slack" is the maximum time that can elapse since detector actuation before the transit call is checked out. The "Detector Adjust

Threshold" is the number of consecutive transit detector max-outs or gap-outs that will trigger a positive or negative adjustment to the detector travel time respectively (*PTV Group, 2014*). "Free Alt. Sequence" and Coord. Alt. Sequence" define an alternative sequence of signal phases that can be displayed once the TSP call is placed, in free and coordinated modes, respectively. This is achieved by selecting one of the eight patterns from the drop-down menu. This function can be used for the phase rotation strategy, as described later in the document.

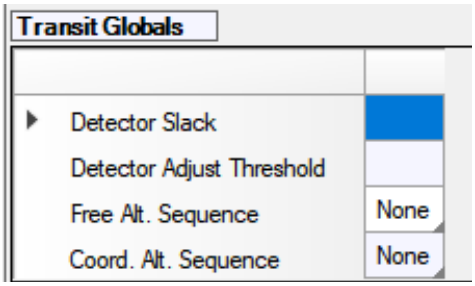


Figure 2.3-3: Transit Globals in RBC

Under the "Transit SG" settings (Figure 2.3-4) a user defines transit signal groups that will be in use (*PTV Group, 2014*). The maximum number of transit signal groups is eight. The signal group numbers for transit priority are hard coded as SG 301-308. The transit signal groups will not be available within VISSIM unless the "Use as VISSIM SG" option is flagged.

Transit SGs

Transit SG	301	302	303	304	305	306	307	308
Use as Vissim SG	☐	☑	☐	☐	☐	☑	☐	☐
Parent SGs		2				6		
No Call SGs								
Priority SGs								
Min Green								
Yellow								
Red Clearance								
Adv. Call Time								
Extension								
Call Mode	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked
Priority								
Reservice Inh. Same								
Reservice Inh. All								

Figure 2.3-4: Transit SGs in RBC

"Parent SGs" are the signal groups that must be timing for the transit priority to time, which means that this parameter links a transit signal group to a vehicular signal group. Any signal groups flagged for the "No Call SGs" parameter will not be called when the associated transit priority has a call. "Priority SGs" are the signal groups that the controller will choose as the priority movements

that will serve concurrently with the Transit SG. If this parameter is left undefined, the controller will choose the parent signal groups as the priority movements.

The “Min Green” parameter defines the minimum green time that a transit signal group will serve before changing to yellow. In the absence of any extension, the transit signal group will serve this minimum green time before it is eligible to terminate. “Yellow” and “Red Clearance” respectively define the time that a transit signal group will time yellow before advancing to red, and the time the signal group will time a red indication before any signal groups that conflict with its parent signal groups will be allowed to begin timing. The “Adv. Call Time” parameter links the travel time of a transit vehicle from the check-in point to the intersection and the moment when a TSP call will be placed for that transit signal group. The “Extension” parameter defines the time that a transit signal group will extend its green interval following the dropped detection, either loss of presence on a Presence detector, or Check Out on a Check In/Check Out detector. During this extension, the transit signal group will remain green unless parent signal groups are terminating before the extension expires. The “Call Mode” option determines the way a transit signal group is called. Four call modes are available: Recall, Locked, Non-Locked and Soft Recall. If the call mode is set to “Recall,” the transit signal group will receive an automatic call without any detector actuation, and will place calls on all parent signal groups. The transit signal will change green whenever all parent signal groups are green. Under the “Locked” mode, transit detectors will place a locked call to the transit signal group anytime there is an actuation. Calls that are dropped on Presence detectors or Check In/Check Out detectors will still call the transit signal group until it is served. Parent signal groups will receive a call until the transit signal group is served. If the “Non-Locked” mode is set, transit detectors will place a call to the transit signal group anytime there is an actuation, but will not lock the call. While the call is present, Parent signal groups will receive a call. In case of a “Soft Recall,” the transit signal group will not place an automatic call for itself or its parent signal groups. If all parent signal groups are green, the transit signal will still change green. The “Priority” parameter defines the sequence of the TSP service. By default, transit priority requests with lower estimated travel times are served ahead of transit priority requests with higher estimated travel times. For higher priority transit movements, the Priority should be defined to a higher value than that of lower priority transit movements. The “Reservice Inh. Same” and “Reservice Inh. All” define the time for which a TSP call will not be serviced after the last call, for the given or all priority services, respectively.

Figure 2.3-5 presents the Coordination Priority settings, which are used in case of coordinated signals along a corridor (*PTV Group, 2014*). The “Vehicle SG Omits” and “Pedestrian SG Omits” parameters define the vehicle and pedestrian signal groups that will be omitted when a TSP call occurs. The “Priority Mode” option defines the priority mode of the transit signal group. Three modes are available: None, Early/Extend and Extend Only. If the “None” mode is selected, the transit signal group will not adjust any controller operation in order to achieve priority service.

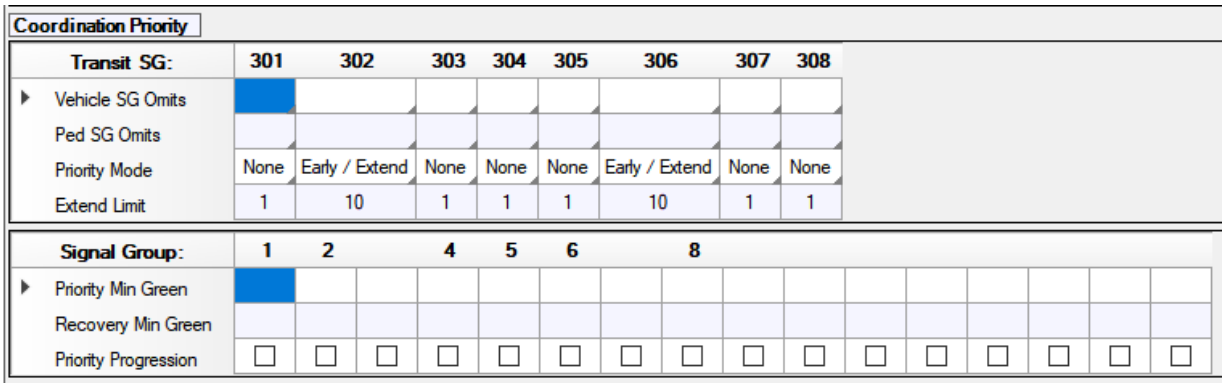


Figure 2.3-5: Coordination Priority in RBC

The "Early/Extend" mode defines both the early green and the green extension at the same time. The transit signal group may adjust signal group timing and sequencing based on these parameters in order to achieve priority service. Signal group split adjustments can be made to allow for either an early return to or extension of the priority parent signal groups. The Transit SG may adjust signal group timing and sequencing based on the "Extend Only" parameter in order to achieve priority service. Signal group split adjustments will only be made to allow for an extension of the priority parent signal groups. When using a Coordinated Priority Mode other than "None," the "Extend Limit" parameter is the maximum time that a transit signal group will be allowed to extend its parent groups beyond their normal coordinated force-off times before the priority extension will be aborted. When a signal group that conflicts with a priority signal group is timing, the controller will abbreviate it in order to serve the priority signal groups faster. The "Priority Min Green" parameter will guarantee the longer of the signal group minimum green or priority minimum green time before it will be terminated to serve a priority signal group. When computing the maximum duration for a priority extension, the controller will take into account the minimum timing for each signal group in the recovery cycle (the cycle following the priority extension). The controller can only extend the priority signal groups to the point where it can still serve the greater of the min green time or "Recovery Min Green" time to all subsequent signal groups by the end of the next cycle. The "Priority Progression" parameter modifies the way the signal group splits are shortened during a priority request.

Similar settings are used for Free Running Priority, as shown in Figure 2.3-6. One addition in this case is the "Recovery" mode, which can be defined as "Normal" or "Serve Omit". "Normal" will continue timing normally from the current signal groups after the priority service. "Serve Omit" returns to the first signal groups that were omitted during the priority service, if they were flagged as such.

Free Running Priority

Transit SG:	301	302	303	304	305	306	307	308
Vehicle SG Omits								
Ped SG Omits								
Priority Mode	None	None	None	None	None	None	None	None
Recovery Mode	Normal	Normal	Normal	Normal	Normal	Normal	Normal	Normal
Extend Limit	1	1	1	1	1	1	1	1

Signal Group:	1	2		4	5	6		8								
Priority Min Green																

Figure 2.3-6: Free Running Priority in RBC

Figure 2.3-7 shows the RBC GUI that is used to define TSP inputs. Special input parameters are defined for each of the eight allowed TSP signal groups (*PTV Group, 2014*). The "Call" parameter defines signal groups that will receive a vehicle call and extension while the transit detector input is on. "Call Transit SGs" defines the transit signal groups that are called and extended while the transit detector input is on. If the "Checkout Detector" option is flagged, it will automatically check out any additional transit inputs, when this transit detector receives a check-out call. "Delay Time" is the amount of time a detector input must have continuous presence before placing a call to "Call Transit SGs" and "Call" signal groups. "Extend Time" is the amount of time that the transit detector will continue to extend TSP signal groups after the transit detector has been checked off. "Travel Time" is the estimated time it will take the transit vehicle to arrive at the intersection once it passes the transit detector. This travel time will be used to adjust intersection timing if priority service is enabled for this transit signal group.

Transit Inputs

Inputs	1	2	3	4	5	6	7	8
Call		2				6		
Call Transit SGs		302				306		
Checkout Detectors								
Delay Time								
Extend Time								
Travel Time		7				7		
Travel Time Slack								
Adjust Step								
Adjust Max								
Calling Pt. Detector	☐	☐	☐	☐	☐	☐	☐	☐
Lateness	0	0	0	0	0	0	0	0
Check Out Limit								
Check Out Mode	Normal	Normal	Normal	Normal	Normal	Normal	Normal	Normal
Detector Type	Presence	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Presence
Presence	301							308
Check In		312	313	314	315	316	317	
Check Out		322	323	324	325	326	327	

Figure 2.3-7: Transit Inputs in RBC

"Travel Time Slack" is a parameter that identifies the uncertainty of the actual travel time of the transit vehicle from the local detector to the intersection. The value entered for this parameter is the average length of time beyond the local detector "Travel Time" that the transit vehicle is

expected to arrive at the intersection. “Check Out Limit” defines the time that a transit Check In detector will remain on before automatically being checked-out (in case the transit Check Out detector is not triggered). “Check Out Mode” determines the mode of the check-out detector. Two options are available: “Normal” and “Stop Bar”. If the “Normal” option is selected, the transit signal group will be checked out as soon as the Check Out detector is activated. If the mode is set to “Stop Bar,” a check-in call will be placed when the Check Out detector is activated. Once the Check Out detector is deactivated (transit vehicle departs from the detector zone), the Transit SG will be checked out. The “Detector Type” parameter determines which type of detectors will be defined for the input and which detector number(s) will be shown for the detectors. The detector type can be either a Presence detector, or Check In/Check Out detectors. The detector numbers are hard coded and they will automatically appear in the “Presence” (301 – 308), “Check In” (311 – 318) and “Check Out” (321 – 328) fields once the type has been selected.

2.3.4 Examples of TSP Strategies in VISSIM/RBC

2.3.4.1 Green Extension / Early Green TSP

An example of GE/EG strategies is provided in the corresponding model “**700 E Curbside Bus TSP GE-EG**”. This model shows a north-south curbside bus line (the magenta colored link in the model) with enabled TSP. Signal phase settings for one intersection along the corridor are shown in Figure 2.3-8. Transit phases correspond to vehicular phases 2 and 6, in the northbound and southbound direction respectively. These are the transit signal group phases. Transit detection is defined as check-in/check-out. The same TSP control can be achieved with presence detectors. The bus detection begins about 300 ft before the stop line. In this example, since transit lanes are exclusive, the activation of these detectors does not have to be defined as “bus only”. If the buses are running in mixed traffic, TSP detector activation must be enabled for transit vehicles only.

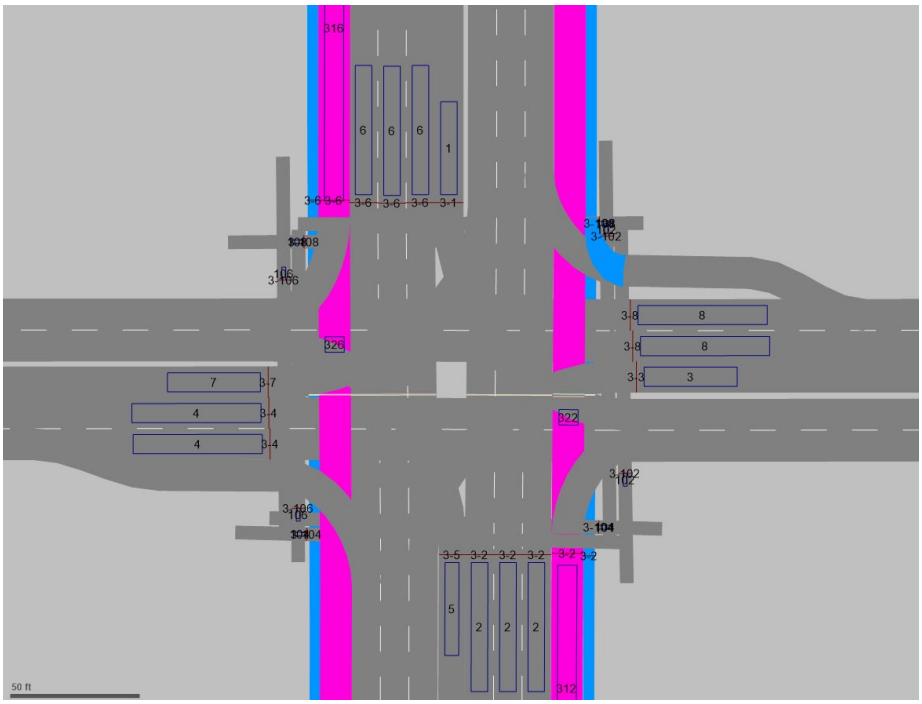
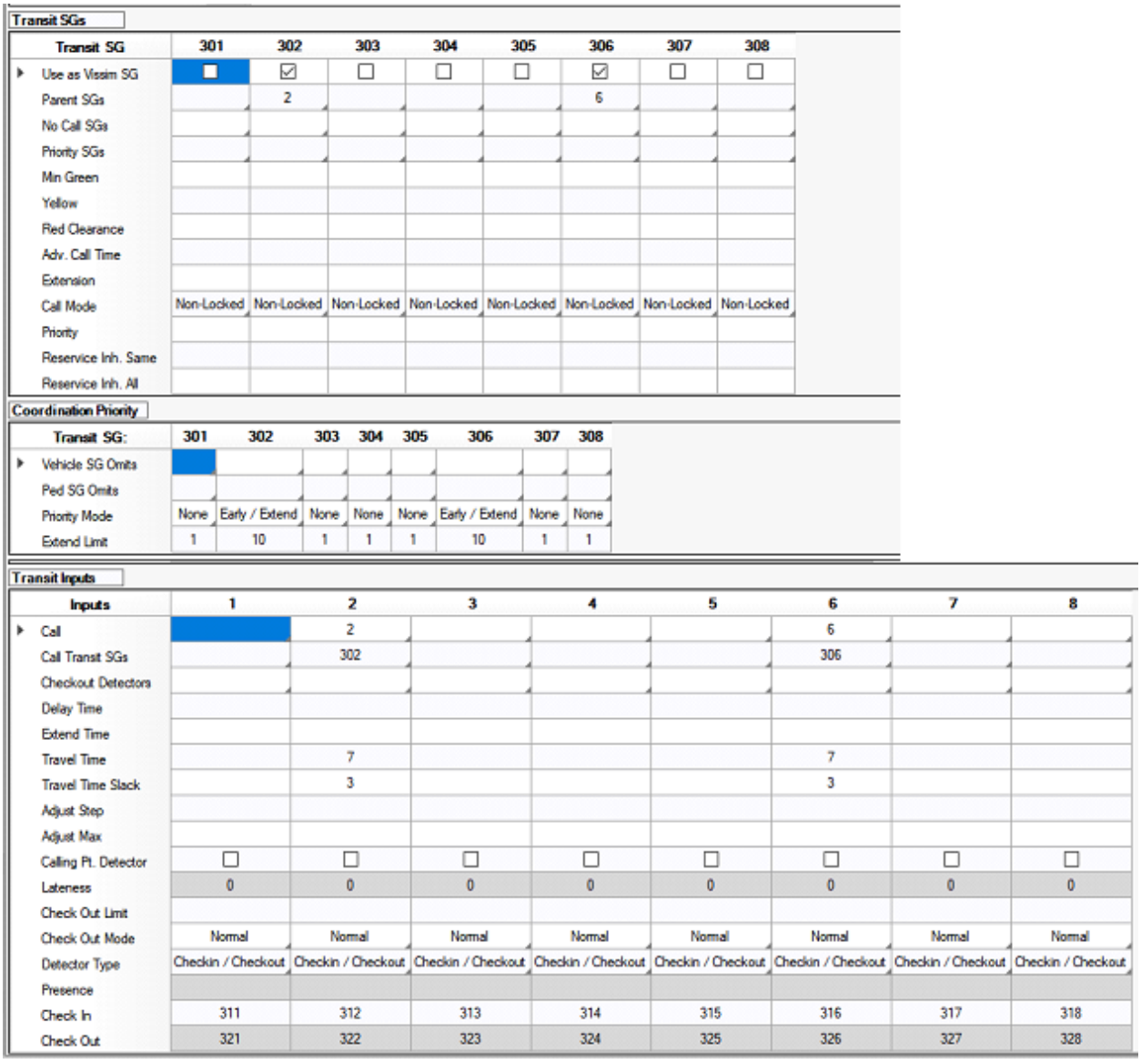


Figure 2.3-8: Intersection with Curbside Bus Lanes and TSP

The TSP is defined as Early/Extend with 10 s of extra green time for buses. No phases are omitted in this case, and no special minimum green times are defined for non-TSP phases. Bus travel times between the beginning of the transit detectors and the stop line are set to 7 s, with 3 s of slack time (this corresponds to the bus speeds of about 35 mph on this section). This travel time should be updated if the buses are running in mixed traffic, or if the bus is stopped at a near-side stop. In the case of a near-side bus stop, the TSP detection can be placed right after the stop to avoid uncertainties in bus travel times. Other TSP settings were not used in this case. These settings are shown in Figure 2.3-9.



Transit SGs

Transit SG	301	302	303	304	305	306	307	308
Use as Vissim SG	☐	☑	☐	☐	☐	☑	☐	☐
Parent SGs		2				6		
No Call SGs								
Priority SGs								
Min Green								
Yellow								
Red Clearance								
Adv. Call Time								
Extension								
Call Mode	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked
Priority								
Reservice Inh. Same								
Reservice Inh. All								

Coordination Priority

Transit SG:	301	302	303	304	305	306	307	308
Vehicle SG Omits								
Ped SG Omits								
Priority Mode	None	Early / Extend	None	None	None	Early / Extend	None	None
Extend Limit	1	10	1	1	1	10	1	1

Transit Inputs

Inputs	1	2	3	4	5	6	7	8
Call		2				6		
Call Transit SGs		302				306		
Checkout Detectors								
Delay Time								
Extend Time								
Travel Time		7				7		
Travel Time Slack		3				3		
Adjust Step								
Adjust Max								
Calling Pt. Detector	☐	☐	☐	☐	☐	☐	☐	☐
Lateness	0	0	0	0	0	0	0	0
Check Out Limit								
Check Out Mode	Normal	Normal	Normal	Normal	Normal	Normal	Normal	Normal
Detector Type	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout
Presence								
Check In	311	312	313	314	315	316	317	318
Check Out	321	322	323	324	325	326	327	328

Figure 2.3-9: TSP Settings for Curbside Bus Lane

2.3.4.2 Transit Overlaps for Center Running Transit

In cases when transit is crossing a signalized intersection in a center-running ROW, which is common with LRT or BRT, overlap phases must be used whether or not any other TSP strategy is implemented. The reason for this is to avoid a conflict between left-turning vehicles and transit vehicles in the center lane on the same approach. This conflict does not exist when transit is running in mixed traffic, or in curbside transit lanes. This application is shown in "State Street Center BRT Transit Overlaps" VISSIM example.

The transit overlap runs simultaneously with the through phase`, but displays red when the left turning phase on the same approach is displayed. This case is shown in Figure 2.3-10. Overlaps 12 and 16 are assigned to the northbound and southbound transit phases, respectively. Overlap 12 has phase 2 as the parent phase, and phase 5 as the negative green phase. Similarly, the parent phase for overlap 16 is phase 6, and phase 1 is the negative green. This prevents a conflict between the transit and left turning vehicles from the same approach. In this example, a 5 s delay is added to the overlap phases to ensure that the left-turning vehicles clear the intersection before the transit overlap starts. The overlap settings are shown in Figure 2.3-11.

Transit overlaps can be combined with other TSP strategies, such as GE/EG, phase rotation or phase insertion. In the provided example, TSP is not implemented. It only shows transit overlaps to avoid conflicts. The examples in the next sections provide combinations of transit overlaps with GE/EG and phase rotation.

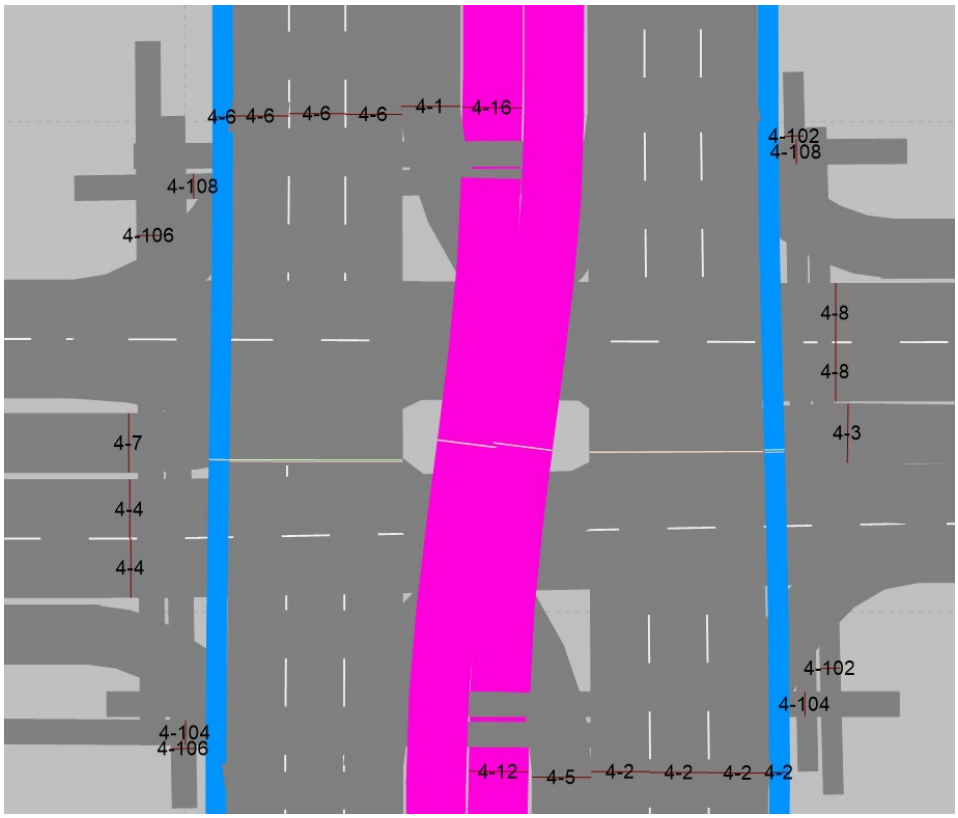


Figure 2.3-10: Intersection Signal Settings for Transit Overlaps

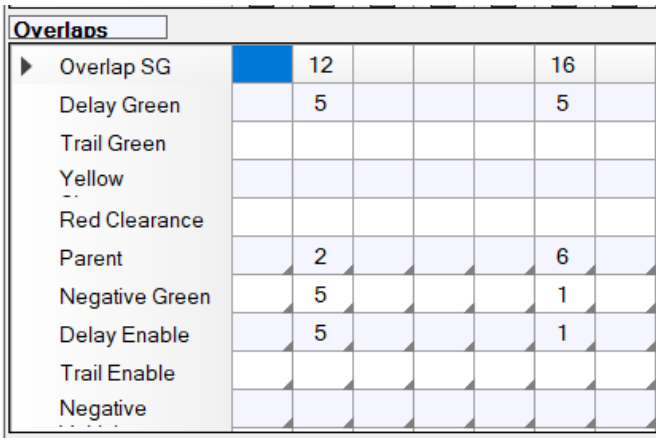


Figure 2.3-11: Transit Overlap Settings

2.3.4.3 Transit Overlaps with GE/EG for Center Running Transit

For semi-rapid transit modes, such as BRT and LRT running in center lanes, TSP should be provided to reduce transit delays and improve their speeds and reliability. The previous section describes transit overlaps which should be used alongside center-running lanes where conflicts with left-turning vehicles exist. Here, the transit overlaps are combined with GE/EG TSP strategies. This is shown in the example "**State Street Center BRT Transit Overlaps - GE-EG**". The signal phases are shown in Figure 2.3-12. In this case, transit detectors also exist in the model. They can be set as check-in/check-out or presence, as described earlier. Overlap and TSP settings are shown in Figures 2.3-13 and 2.3-14 respectively.

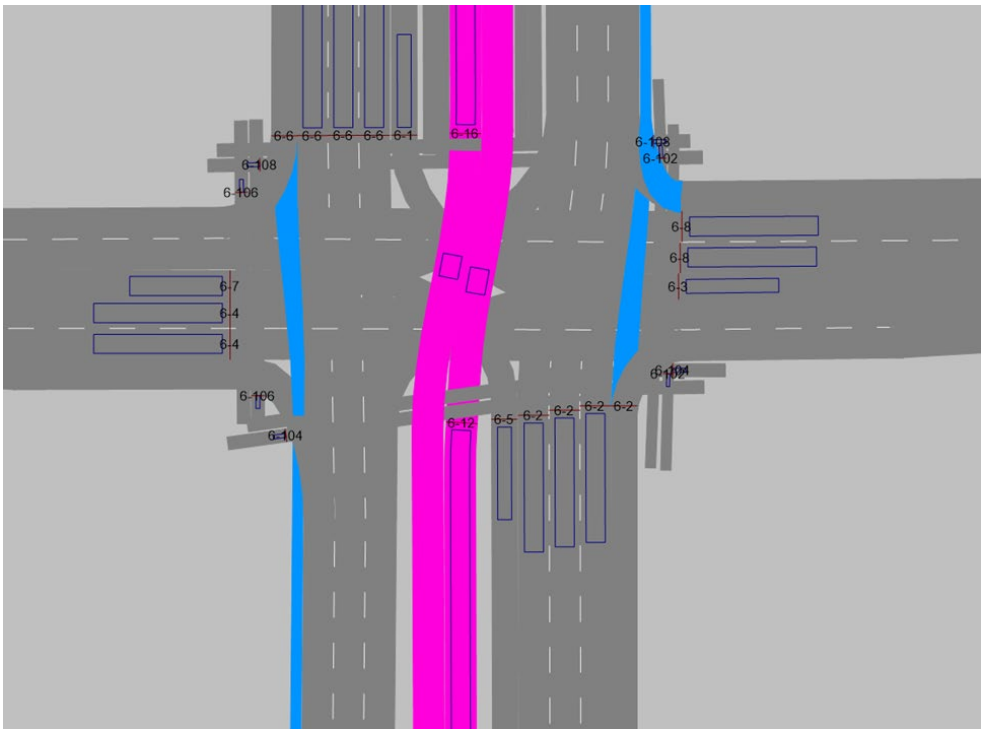


Figure 2.3-12: Intersection Signal Settings for Transit Overlaps and GE/EG

Overlaps

Overlap SG		12				16										
Delay Green		5				5										
Trail Green																
Yellow Clearance																
Red Clearance																
Parent		2				6										
Negative Green		5				1										
Delay Enable		5				1										
Trail Enable																
Negative Vehicle																
Negative Overlap																

Figure 2.3-13: Transit Overlap Settings

Transit SGs

Transit SG	301	302	303	304	305	306	307	308
Use as Vissim SG	☐	☑	☐	☐	☐	☑	☐	☐
Parent SGs		2				6		
No Call SGs								
Priority SGs								
Min Green								
Yellow								
Red Clearance								
Extension								
Call Mode	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked	Non-Locked
Priority								

Coordination Priority

Transit SG:	301	302	303	304	305	306	307	308
Vehicle SG Omits								
Ped SG Omits								
Priority Mode	None	Early / Extend	None	None	None	Early / Extend	None	None
Extend Limit	1	10	1	1	1	10	1	1

Signal Group:	1	2	3	4	5	6	7	8
Priority Min Green								
Recovery Min Green								
Priority Progression	☐	☐	☐	☐	☐	☐	☐	☐

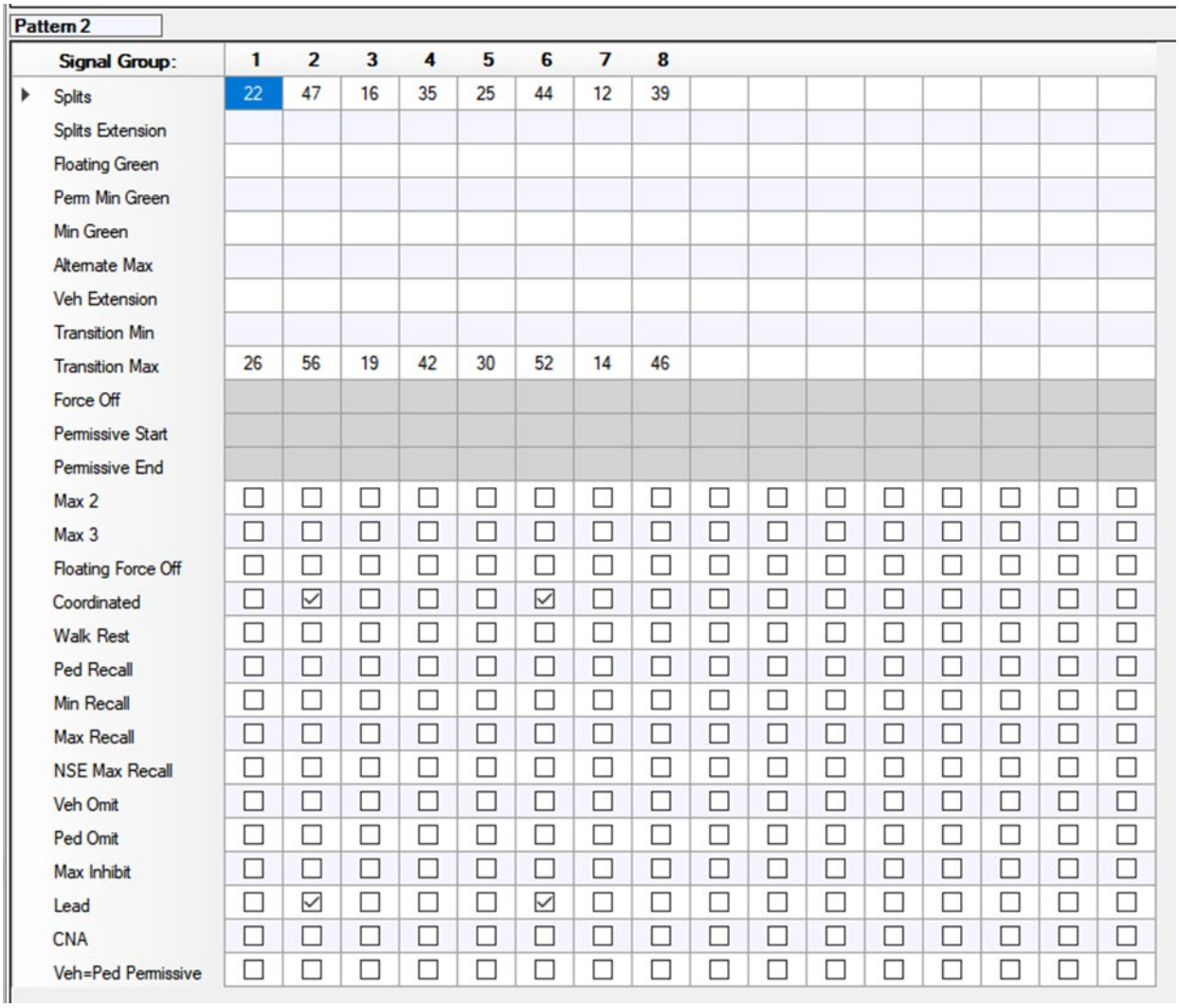
Transit Inputs

Inputs	1	2	3	4	5	6	7	8
Call		2				6		
Call Transit SGs		302				306		
Checkout Detectors								
Delay Time								
Extend Time								
Travel Time		7				7		
Travel Time Slack		3				3		
Check Out Mode	Normal	Normal	Normal	Normal	Normal	Normal	Normal	Normal
Detector Type	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout	Checkin / Checkout
Presence								
Check In	311	312	313	314	315	316	317	318
Check Out	321	322	323	324	325	326	327	328

Figure 2.3-14: TSP Settings for Transit Overlaps and GE/EG

2.3.4.4 Transit Overlaps with GE/EG – Phase Rotation for Center Running Transit

Phase rotation can be successfully implemented with transit modes to improve both operations and safety, regardless if the transit runs in the center lane, curbside lane or mixed traffic. The example "**State Street Center BRT Transit Overlaps - GE-EG - Phase Rotation**" is an extension of the previous model, which includes phase rotation in addition to GE/EG strategies. The overlap and GE/EG settings are the same as in the previous example. Phase rotation is achieved through an alternative pattern, which implements the same phase splits, but with leading northbound/southbound through phases. Pattern 2 is shown in Figure 2.3-15. The change in lead/lag phases is achieved by checking the "Lead" option under the pattern settings, since the phase sequence in the ring-barrier settings can be set only once. Figure 2.3-16 shows the call for Pattern 2 under "Transit Globals" settings. When this setting is defined, the controller will switch to the alternative pattern once the transit check-in (or presence) detector has been triggered.



Signal Group:	1	2	3	4	5	6	7	8								
Splits	22	47	16	35	25	44	12	39								
Splits Extension																
Floating Green																
Perm Min Green																
Min Green																
Alternate Max																
Veh Extension																
Transition Min																
Transition Max	26	56	19	42	30	52	14	46								
Force Off																
Permissive Start																
Permissive End																
Max 2	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Max 3	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Floating Force Off	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Coordinated	☐	☑	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Walk Rest	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Ped Recall	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Min Recall	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Max Recall	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
NSE Max Recall	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Veh Omit	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Ped Omit	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Max Inhibit	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Lead	☐	☑	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
CNA	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
Veh=Ped Permissive	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Figure 2.3-15: Pattern 2 Settings for Phase Rotation

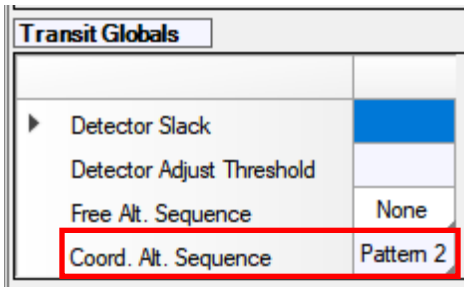


Figure 2.3-16: TSP Alternate Sequence Settings for Phase Rotation

2.3.4.5 Phase Insertion: Queue Jump Application

Phase insertion for TSP applications can be implemented in cases when a separate transit phase is needed to achieve pre-defined transit operations. The most common application of phase insertion is with Queue Jump TSP. This example is shown in the "**State Street Bus Queue Jump**" VISSIM model. In this case, bus queue jump lanes are integrated with exclusive right-turn lanes, which is the most common application. One intersection with signal settings is shown in Figure 2.3-17.

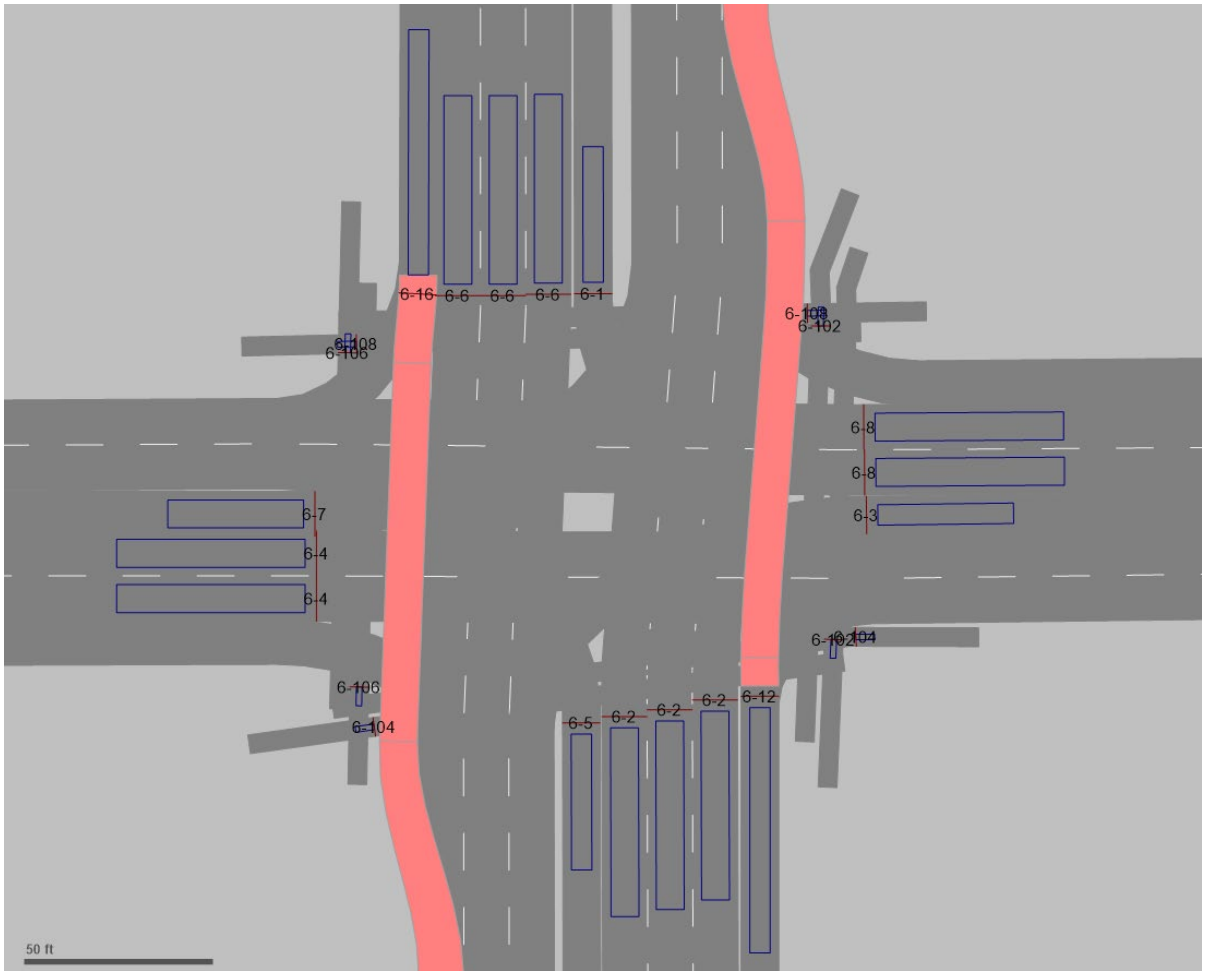
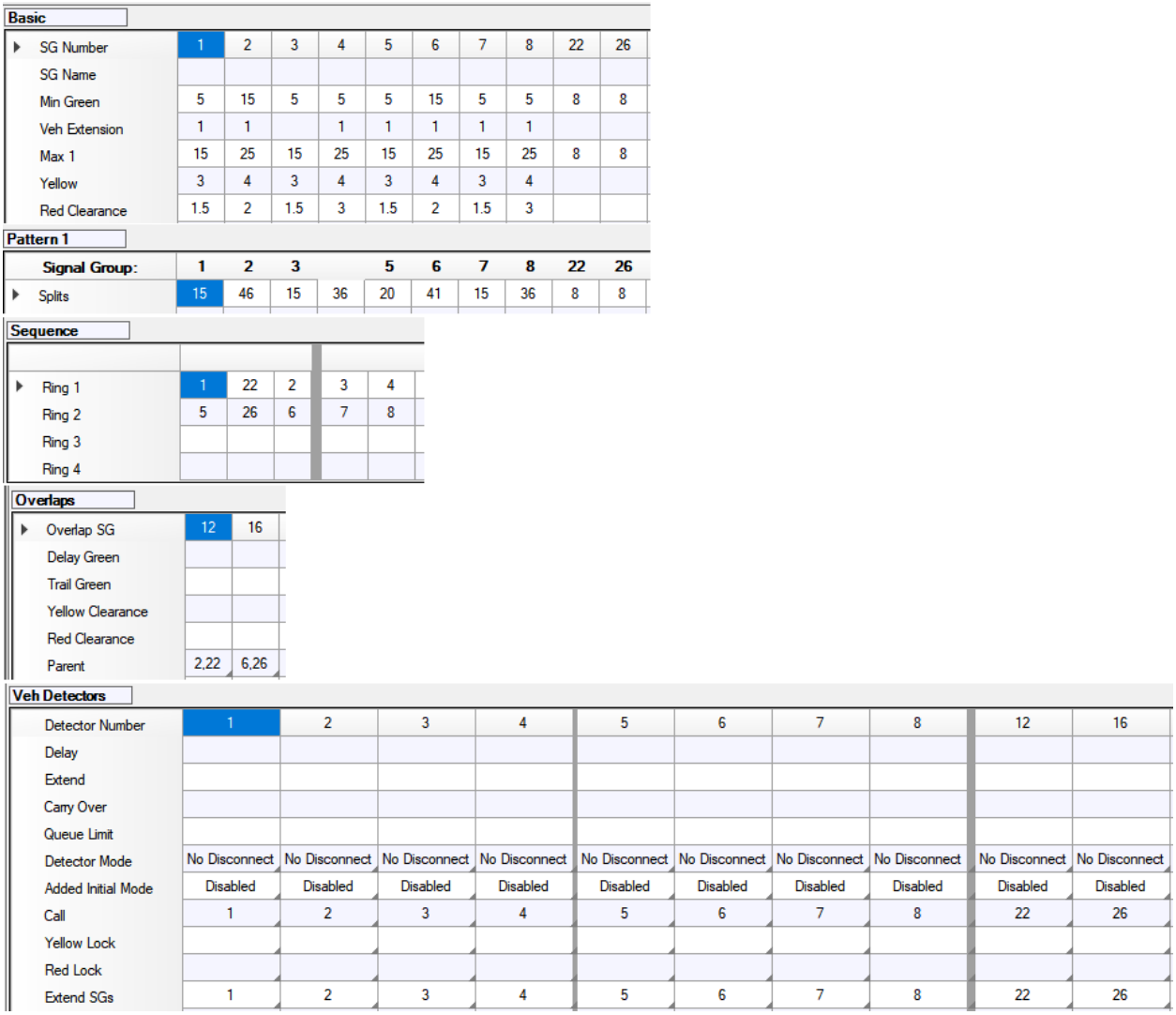


Figure 2.3-17: Intersection and Signal Settings for Queue Jump

The transit signal phasing is achieved through signal overlaps, which includes a leading transit phases. In the example signal timing plan, these transit phases are numbered 22 and 26, which correspond to phases 2 and 6, respectively. Transit overlaps 12 and 16 include phases 22/2, and 26/6, respectively. The split summation 22+2 and 26+6 corresponds to the original splits of phases 2 and 6, before phase insertion. In the given example, leading transit phases have a duration of 8 seconds, to allow the buses to cross the intersection and return to the traffic lane before other vehicles reach the merge point. This time will depend on the actual intersection configuration. To ensure that the phase time is always the same, both its maximum and minimum green times were set to 8 s, with no yellow/all read. Transit detectors call phases 22 and 26, and in this case they are set as bus-only detectors, since they are placed in the shared right-turn lane. Phase, sequence, overlap and detector settings are shown in Figure 2.3-18.



Basic

SG Number	1	2	3	4	5	6	7	8	22	26
SG Name										
Min Green	5	15	5	5	5	15	5	5	8	8
Veh Extension	1	1		1	1	1	1	1		
Max 1	15	25	15	25	15	25	15	25	8	8
Yellow	3	4	3	4	3	4	3	4		
Red Clearance	1.5	2	1.5	3	1.5	2	1.5	3		

Pattern 1

Signal Group:	1	2	3		5	6	7	8	22	26
Splits	15	46	15	36	20	41	15	36	8	8

Sequence

Ring 1	1	22	2	3	4
Ring 2	5	26	6	7	8
Ring 3					
Ring 4					

Overlaps

Overlap SG	12	16
Delay Green		
Trail Green		
Yellow Clearance		
Red Clearance		
Parent	2,22	6,26

Veh Detectors

Detector Number	1	2	3	4	5	6	7	8	12	16
Delay										
Extend										
Carry Over										
Queue Limit										
Detector Mode	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect	No Disconnect
Added Initial Mode	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled	Disabled
Call	1	2	3	4	5	6	7	8	22	26
Yellow Lock										
Red Lock										
Extend SGs	1	2	3	4	5	6	7	8	22	26

Figure 2.3-18: Signal Settings for Phase Insertion / Queue Jump

Phase insertion can also be combined with GE/EG and phase rotation, with the settings as described in previous sections.

2.3.5 References

1. H. R. Smith, B. Hemily, and M. Ivanovic. *Transit Signal Priority (TSP): A Planning and Implementation Handbook*. Intelligent Transportation Society of America. 2005.

2. *Ring Barrier Controller (RBC) User Manual.* PTV Group, 2014.

2.4 Traffic Responsive Pattern Selection

Contributor(s): Qichao Wang, Dusan Jolovic

2.4.1 Introduction

Traffic Responsive (TR) is one of the approaches to manage diurnal fluctuations in traffic. The TRPS control was first developed in the '70s when a UTCS federal project in Washington D.C. was executed [1]. This project set the standards for TR control for the last 35 years [2]. Traffic studies conducted so far showed that the TR systems could significantly improve overall traffic performances in the traffic networks. An advantage of the TR operations is that implementation of the signal timing patterns depends on the traffic fluctuations in the field and it is not predetermined by the time of the day. Also, upgrading signal system operations from TOD mode to a TR mode is much more cost-effective than some other alternatives (e.g., traffic adaptive).

The two common mechanisms of TR systems deployed in the field are threshold-based TRs and pattern-matching TRs. The threshold-based TRs use aggregated counts and occupancies from the detectors, while pattern-matching TRs utilize raw traffic data.

Two major strategies for implementing TR systems are 1) to develop several timing plans for every possible traffic conditions with no fine-tuning; 2) to develop typical tuned timing plans and a few of the 'special event' timing plans without fine-tuning (x). Note that the cycle lengths are fixed until a new plan is picked from the library of predefined plans. The parameters such as cycle, offset, and maximum splits are limited by the user, i.e., parameters will have upper and lower bounds.

Benefits of the TR systems can be evaluated in the field by comparing the data (number of stops, travel times, vehicle delays) collected in the field while TOD mode was running (before conditions), with the data collected after a TR system is fully operational in the same network (after conditions). However, if the TR parameters and thresholds are not set up properly, the field tests may cause disturbances in the traffic operations and deteriorate traffic performances on the corridor.

As an alternative, TR systems can be evaluated in the microsimulation software. However, no commercially available software comes with an embedded TR logic. Each TR logic, which is to be used in a microsimulation model, requires the development of an additional program/script that can replicate operations of a TR signal system. Still, by using microsimulation as an evaluation tool, one can test various and unusual traffic conditions such as traffic incidents, lane closures or increased traffic volumes due to traffic-generator events (concerts, sports games, etc.). Conducting such evaluations in the field would be unsafe, expensive, and much more difficult to execute.

2.4.2 Detector Setup

For the TR system detector placing, one should consider:

- Mid-block and free flow locations
- Characteristic locations for the network considered

Detectors should be placed outside of typical queueing areas. Preferred detector placing should be on the downstream side of the intersection. If the detectors are placed upstream of the intersection, the position must be outside of the queueing area.

If the segments near the middle of the corridor are near capacity the volume and occupancy levels may not change significantly throughout the day. In this case, segments leading in/from the middle of the corridor may provide the most relevant information as directional traffic increases and decreases throughout the day.

Some corridors may have volume and occupancy levels that increase and decrease throughout the day and have distinctly directional characteristics while segments leading into and out of the middle of the corridor have minimal fluctuations. In this case focus system detector locations in and around the middle of the corridor.

Another hint is to consider system detector locations that act as cordon detectors. They measure the volume and occupancy levels entering and/or exiting the system.

Major side streets should be selected, and system detectors placed to gather segment data for the relevant approaches that will have increasing and decreasing traffic levels throughout the day.

Eventually, the detectors will be classified as inbound, outbound, or cross-street. When determining system detector locations, one should think in terms of an inbound detector, outbound detector, or cross-street detector.

2.4.3 Traffic Responsive Pattern Selection Process or How to Use TRPS with Microsimulation

To implement a TRPS system with enough patterns to choose from, a Synchro, Vistro, or any other timing tool can be used to develop new signal patterns. These patterns should be based on the traffic volumes extracted for each intersection from calibrated and validated microsimulation (VISSIM) model. Despite its strength over TOD plans (it can reduce deterioration of signal timing plans, easily handle diurnal shifts in traffic demands, etc.) there are many parameters that need to be set and fine-tuned. This time-consuming TRPS-parameters fine-tuning process is one of the reasons why most of the traffic signal agencies that develop signal timing strategies in-house, stay away from implementation of TRPS systems. Note that when developing the TRPS system, it is desired to have at least a month of field traffic data to capture the variability of traffic flows.

This example focuses on a threshold-based logic, which is very similar to the one described in manuals for setting Naztec controllers [*2*].

The first step of setting up a TRPS is the definition of system detectors, which are used to control TRPS parameters. There are generally three groups of detectors:

- Inbound detectors
- Outbound detectors
- Cross street detectors

In the microsimulation model, these detectors are represented with the Data Collection Points since they are not really detectors that directly control vehicle actuation at signals. After the detectors are defined, initial simulation runs are conducted to get volume outputs from each detector. Maximum observed volume (in veh/min) serves to calculate "Full Rate Volume" defined with the equation [1]:

$$\text{Full Rate Volume} = \frac{\text{Maximum Observed Volume}}{0.80} \quad [1]$$

Full rate volume is defined for each detector class. The second parameter to be defined in this step is "Full Rate Occupancy". To increase the resolution of the data, this parameter can be reduced from the initial value of 100% to 20%-40%, depending on a detector group.

In the next step, 'smoothed volume' and 'smoothed occupancy' are calculated for all three groups of the detectors. To calculate these values, a constant called 'smooth value' has to be predefined for each detector in the groups previously defined (e.g., inbound, outbound, cross-street). This is a unique value for each detector that controls how much weight is given to new detector readings versus previous values, and once defined, it cannot be varied during the day. Equations [2,3] for calculation smoothed volume and smoothed occupancy are given as follows:

$$\text{Smooth Vol} = \frac{\text{New Vol}\cdot(100-\text{Smooth Val})+(\text{Old Vol}\cdot\text{Smooth Val})}{100}$$

$$\text{Smooth Occ} = \frac{\text{New Occ}\cdot(100-\text{Smooth Val})+(\text{Old Occ}\cdot\text{Smooth Val})}{100} \quad [2,3]$$

where:

New Vol – volume gathered from the detector for current 15-min period in (veh/min)
Smooth Val – a constant defined for each system detector
Old Vol – volume gathered from the detector for previous 15-min period in veh/min
New Occ – occupancy gathered from the detector for current 15-min period
Old Occ – occupancy gathered from the detector for previous 15-min period

By using smoothed values calculated with [2,3], volume and occupancy percentages are estimated for each detector group using the following equations [4, 5]:

$$\text{Vol\%} = \frac{\text{Smoothed Vol}}{\text{Full Rate Volume}} \quad [4]$$

$$\text{Occ\%} = \frac{\text{Smoothed Occ}}{\text{Full Rate Occupancy}} \quad [5]$$

In the third step, each system detector is assigned a volume (c_n) scaler and occupancy (k_n) scaler. Scalers are used as weights to give a relative importance to volume and occupancy, but also for the relative importance of volumes/occupancies from different detectors (if a flow from one detector is more important than the other, e.g., blocking a railroad tracks). Scalers can be assigned values from 0 to 9. If a certain detector is placed in an area where the traffic noticeably changes by time of day, that detector should have a higher scaler. If a detector is not adding any value to the pattern selection process (i.e., the detector is not placed in an area where the volumes are different from pattern to pattern), it should be removed from the calculation (i.e., assign scaler

value of zero to that detector). Once the scalers are determined, traffic responsive flow values for each detector group (inbound, outbound, cross street) are calculated. Proper calculation of the flow values is a very important step since all the later parameters are based on these values.

In fourth step, Cycle, Offset and Split Parameters are calculated based on the TR flow values, which are updated every 15 minutes. These parameters range from 0 to 100% and are used to perform an offset-table lookup to select the proper signal-timing pattern, which a signal controller executes in the field. These parameters are critical for the quality of a TRPS system. If they are not properly estimated, benefits of the TRPS system may be small or nonexistent. Following equations [6, 7, 8] show how Cycle, Offset and Split parameters are calculated:

$$\text{Cycle Index} = \text{Max}\left(\text{Flow}_{\text{inbound}}, \text{Flow}_{\text{outbound}}\right) \quad [6]$$

$$\text{Split Index} = \left(\frac{\text{Flow}_{\text{cross}} - \text{Cycle Index}}{\text{Flow}_{\text{cross}} + \text{Cycle Index}}\right) \cdot 50 + 50 \quad [7]$$

$$\text{Offset Index} = \left(\frac{\text{Flow}_{\text{inbound}} - \text{Flow}_{\text{outbound}}}{\text{Flow}_{\text{inbound}} + \text{Flow}_{\text{outbound}}}\right) \cdot 50 + 50 \quad [8]$$

The equations [6,7,8] show that the Cycle Index varies cycle length as a function of maximum directional hybrid value consisted of volume and occupancy. Similarly, the Split Index depends on the relative importance of the cross-street traffic whereas the Offset Index varies according to weights of directional hybrid (volume and occupancy) flows.

In order to implement this logic in microsimulation (VISSIM), the authors developed an algorithm, which retrieved (through the COM interface) detector data from VISSIM to calculate TRPS indices and further return specific signal timing patterns for that time of the day. The signal timing patterns are picked from matrices as cross-sectional values of Cycle Indices and Split Indices. These Cycle Index – Split Index matrices are developed for 3-4 Offset Indices which are usually based on diurnal periods where inbound/outbound traffic flows are dominant or not. (This algorithm is proprietary as of now, so one should develop it on its own.)

In practice, the indices and threshold values, which specify how quickly the algorithm moves within each matrix and between matrices, are based on the initial traffic flow values recorded during TOD operations. Later, the TRPS process can be reiterated a few times until the selection of the TRPS traffic signal patterns converges. The threshold values are contained in threshold tables which have increasing and decreasing columns. The values in these two columns are always different (for the same index) thus preventing TRPS to oscillate (if threshold values are selected properly) between two nearby traffic signal patterns. Table lookup procedure is the core of TRPS methodology, and it requires a significant amount of time to fine-tune TRPS parameters (threshold values and traffic signal patterns).

The outputs from Synchro and VISSIM can be used to determine the number of Offset indices (which determines several Cycle Index – Split Index matrices). Based on the obtained data – multiple matrices could be developed.

In the next step, the TRPS algorithm chooses a particular matrix to lookup for the traffic signal patterns. The lookup is based on an offset index and values calculated in [8]. Each column in the

matrix is assigned to a specific Split Index and each row is assigned to a Cycle Index. Traffic signal patterns are then assigned to cells according to when during the day they would be needed but observing that they fit within the ranges of Cycle and Split indices. On the other hand, multiple traffic signal patterns with the same cycle length but various splits will occupy the same row (the same Cycle Index) but various columns (various Split Indices); and vice versa. When the Cycle and Split indices are determined based on the parameters calculated with [6,7,8] and the thresholds are defined, the TRPS algorithm will look for the proper matrix's cell based on the indices assigned to that row/column. Traffic signal pattern within the cell will be selected for implementation for a specified period. If there are no changes in the indices from the previous iteration, the pattern will not change. Further explanations of the applied TRPS method can be found in the literature [2].

2.4.4 Literature on Traffic Responsive Systems

The biggest challenges researchers have regarding the development of the traffic responsive control systems are related to the signal timings plan selection for particular traffic patterns and transition between signal timing plans since directional flows at each intersection can vary significantly. Some of the previous researches conducted on TR systems are summarized herein.

Nelson et al. [3] focused on single controller diamond interchanges, the impact of emergency vehicle preemption on signalized operations and advanced real-time control parameters.

Yang [4] worked on the development of TR control optimization for special events. By using a neural network approach, the author successfully developed effective signal timings for a surge of the high traffic volume. The results showed a 4.6% decrease in average delay per vehicle when compared to existing timing plans.

Fouladvand [5] developed and tested a stochastic model on a single intersection with two one-way approaches. The author's TR algorithm was based on cut-off queue length and cut-off density. It was found that the developed system generated signal timing plans which were more efficient than the TOD method.

In his master thesis, Sharma [6] presented how to set up the TRPS mode in traffic controllers. He showed that the TRPS mode can be efficiently set up using a neural network approach.

Abbas and Sharma [7] recognized that the TRPS control is a pattern recognition problem of different traffic states. Using discriminant analysis (Bayesian-based approach) authors suggested a robust method for the TRPS optimal parameters and thresholds selection on the testbed in Texas. The authors pointed out that the timing plans should be assigned to distinct traffic states.

Abbas et al. [8] developed a methodology that selects optimal parameters, thresholds and timing plans for a TRPS control system. The methodology consists of data clustering, correlation analysis, O-D analysis, generation of patterns, and finding and validating best timing plans. Validation is conducted using the CORSIM tool and the research produced simplified guidelines for the TR system implementation.

Sayers and Bell [9] used fuzzy logic for the traffic responsive signal control strategy. To properly monitor variation in traffic flows the authors placed several inductive loop detectors on every intersection approach; in each lane at different distances from the stop line. The authors found that

this approach resulted in a quicker allocation of green time to appropriate approaches and a significant reduction in intersection delay.

Abbas and Sharma [10] used a Genetic Algorithm to select the best timing plans and to assign predetermined plans with grouped traffic flow states. The study concluded that fourteen timing plans identified in the study yield savings of 53% in delay and 19% in the number of stops.

Pesti et al. [11] implemented a TRPS control at four sites in Texas and conducted before and after studies to compare average travel speeds and delays. The outcome of the project was a step-by-step field manual to guide field technicians in process of TRPS implementation.

2.4.5 References

[1] Gartner N. (1982). Demand-Responsive Decentralized Urban Control. Part I: Single-Intersection Policies, Phase I Report, USDOT, Research and Special Programs Administration, Washington D.C., 89pg.

[2] Naztec Operations Manual, TS2 Closed Loop Systems (2004), Naztec Inc.

[3] Nelson, E. J., M. Abbas, G. E. Shoup, and D. M. Bullock. Development of Closed Loop System Evaluation Equipment and Procedures. Publication FHWA/IN/JTRP-2000/05. Joint Transportation Research Program, Indiana Department of Transportation and Purdue University, West Lafayette, Indiana, 2000. doi: 10.5703/1288284313191.

[4] Yang, J. S. (2004). Special Events Traffic Signal Timing Control via a SPSA Optimization Algorithm—A Case Study. CD-ROM: Transportation Research Board of the National Academies, Washington, D.C., 2004

[5] Fouladvand, M.E., Sadjadi, Z. and Shaebani, M.R. (2004). Optimized Traffic Flow at a Single Intersection: Traffic Responsive Signalization. In: Journal of Physics A: Mathematical and General, Vol. 37, pp. 561-576.

[6] Sharma, A. (2004). "Determination of Traffic Responsive Plan Selection Factors and Thresholds Using Artificial Neural Networks". Civil Engineering Faculty Publications. Paper 34. http://digitalcommons.unl.edu/civilengfacpub/34

[7] Abbas, M. and Sharma, A. (2004). "Configuration of Traffic-Responsive Plan Selection System Parameters and Thresholds: Robust Bayesian Approach". Civil Engineering Faculty Publications. Paper 16. http://digitalcommons.unl.edu/civilengfacpub/16

[8] Abbas, M., Chaudhary, N. A, Sharma, A., Venglar, S.P. and Englbrecht R.J. (2004). Methodology for Determination of Optimal Traffic Responsive Plan Selection Control Parameters. Report 0-4421-1. TxDOT, Research and Technology Implementation Office, Austin, TX, pg. 84.

[9] Sayers, T. and Bell, M. G. H. (1996). Traffic Responsive Signal Control Using Fuzzy Logic—A Practical Modular Approach. In: Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, pp. 2159-2163.

[10] Abbas, M. and Sharma, A. (2006). Multiobjective Plan Selection Optimization for Traffic Responsive Control. Journal of Transportation Engineering, Vol. 132(5), pg. 376-384.

[11] Pesti, G., Chaudhary, N. and Abbas, M. (2007). Implementation of Traffic Responsive Control on TxDOT Closed-Loop System. Report 5-4421-01-2, TxDOT, Research and Implementation Office, Austin, TX, pg. 56.

2.5 Adaptive Control

To be added later.

2.6 Econolite ASC/3 Software-in-the-Loop in VISSIM Microsimulation

Contributor: Milan Zlatkovic, University of Wyoming

2.6.1 ASC/3 Controller

The ASC/3 controller is a series of Advanced System Controllers by Econolite. It combines the requirements of NEMA TS2 and NTCIP and satisfies all industry and ISO quality standards (*Econolite, 2018*).

The most important features of the ASC/3 controller are as follows:

1) Control features:
    - Sixteen phases, eight configurable concurrent groups in four timing rings
    - All standard NEMA TS1, TS2, and NTCIP functions
    - Sixteen timed vehicle overlaps
    - Sixteen pedestrian phases (which can also be configured as pedestrian overlaps)
    - Soft vehicle recall
    - Conditional service
    - Dynamic Max operation
2) Coordination features:
    - 120 coordination patterns, each with its own cycle, offsets, and split plan selection
    - 120 split plans, each with its own coordinated phases, vehicle and pedestrian recall, and phase omits
    - Fixed or floating force-off
3) Preemption features:
    - Ten preemption sequences, where each may be configured as priority, first-come-first-serve, or preemption operation
4) Time Base features
    - 200 schedule programs, configurable for any combination of months, days of the week, and days of the month
    - Thirty-six fixed or floating exception day programs that override the day plan event on a specific day
    - Fifty-day plan events
    - 100 action plans
5) Detector features
    - Sixty-four vehicle detectors
    - Sixteen system or speed detectors
    - Unique detector types and operation
    - Individually assignable to phase and functions
    - Lock / Non-Lock function by detector

- Four detector plans
- Four detector diagnostic plans
- Logging of volume and/or occupancy assignable by detector
- Four pedestrian diagnostic plans

The ASC/3 controller is able to support very complex signal timing settings through Logic Processors, which can emulate external logic that is not included in the default settings. A total of 200 logic commands is available in the controller, and these commands can control and combine all the controller features.

The ASC/3 SIL version of the controller software has been specifically configured to operate as a virtual controller within the VISSIM environment. This allows full ASC/3 controller functionality to be used during simulations under VISSIM. Another benefit of the ASC/3 SIL is that it can run at 10 times normal speed during simulation, which greatly reduces the time needed to test a scenario in VISSIM.

The ASC/3 SIL is comprised of the following software components (*Econolite, 2009*):

- Data Manager
- Traffic Control Kernel
- Controller Front Panel Simulator
- VISSIM DLL interface

The Data Manager is a Windows application for managing the controller timing data of the simulated controllers while in the Windows environment. This software is more intuitive and easier to use than the controllers' normal front panel data entry screens. The database file for the ASC/3 SIL and an actual ASC/3 controller are identical.

The Traffic Control Kernel is the virtual ASC/3 core software that operates under Windows. It encompasses all internal processing that occurs between the mapped field inputs that are passed from VISSIM and subsequent calculation of commanded field outputs that are passed to VISSIM. This interface guarantees consistency in traffic control operation between the simulated ASC/3 SIL running under VISSIM and a physical ASC/3 controller.

The Controller Front Panel Simulator is a Graphical User Interface (GUI) designed to simulate the 16-line x 40-character display and keypad found on the ASC/3 physical controller. This GUI permits the display of status and data along with the changing of all user data settings within the simulated ASC/3 controllers running under VISSIM. Any changes made to the controller settings are stored in the simulated controller's database.

The VISSIM DLL interface couples the ASC/3 simulated controllers to VISSIM. It allows VISSIM to pass detector and other Input/output functions to the simulated ASC/3 controllers and to receive controller status information back.

All these components, unified under the Windows operating environment and integrated with VISSIM, provide the ability to simulate one or more intersections with a unifying controller management interface and the ability to model different signal timing strategies.

2.6.2 ASC/3 in VISSIM Simulation

An example VISSIM model with ASC/3 controllers is provided in "**State Street ASC3**" folder. With an add-on license, VISSIM allows Econolite ASC/3 Software-in-the-Loop (SIL) simulation. To enable this functionality, the user first needs to copy all the template files (except the two pdf manuals) from the ASC3 VISSIM folder under Program Files into the current VISSIM model folder. The path and the files are shown in Figure 2.6-1.

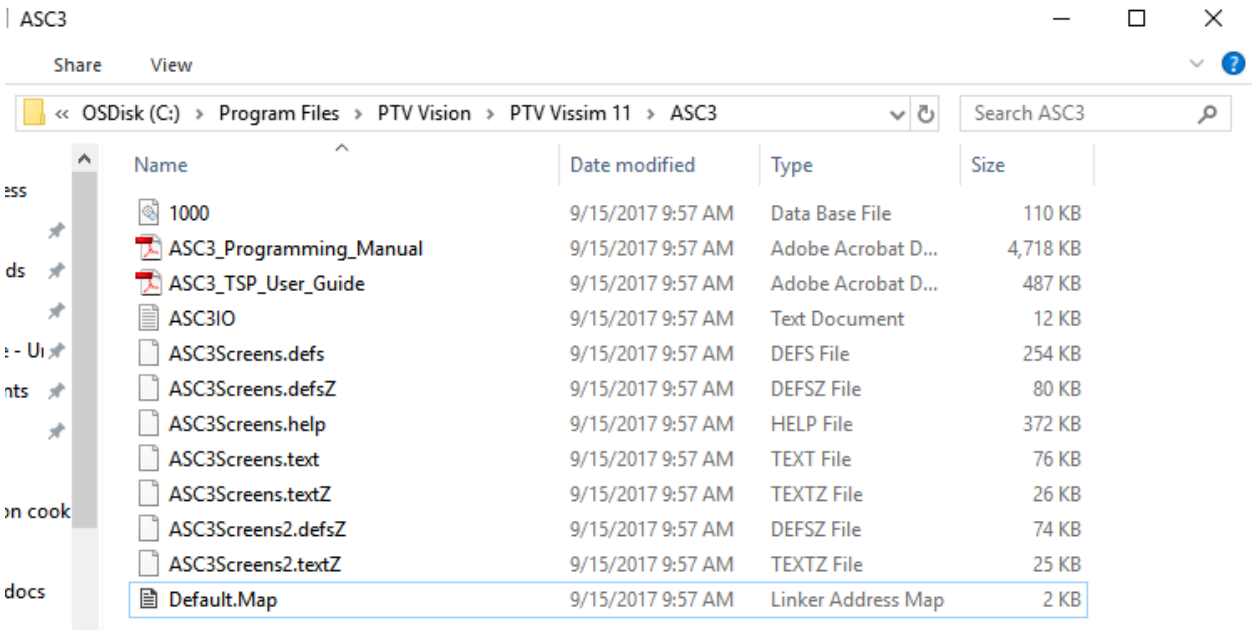


Figure 2.6-1: Figure 2.6-1: ASC/3 Template Files and Location

Any additional files will be created once the user starts defining settings in ASC/3.

For a successful ASC/3 simulation, the signal controllers in VISSIM need to be numbered sequentially starting from 1. For signal controller 1, the corresponding ASC/3 files will be numbered 0001, for 2 it will be 0002, and so on. Actual ASC/3 dll files can be copied from an existing control program. If it does not exist, the template 1000.dll data base file can be copied and renamed as 0001.dll, 0002.dll etc.

The process of creating ASC/3 SIL controllers in VISSIM is as follows:

- Add a signal controller; under "Type" select "External"
- Under "Controller configuration", select **asc3.dll** under "Program file"; **asc3gui.dll** under "Dialog DLL file"; upload **0001.dll** (or the other corresponding dll program file for the given intersection) under "Data File 1"; add **ASC3.wtt** under "WTT Files". asc3.dll, asc3gui.dll and ASC.wtt will be automatically referred to when the files are being uploaded

(they are located in the Exe folder of the PTV VISSIM installation). This is shown in Figure 2.6-2.

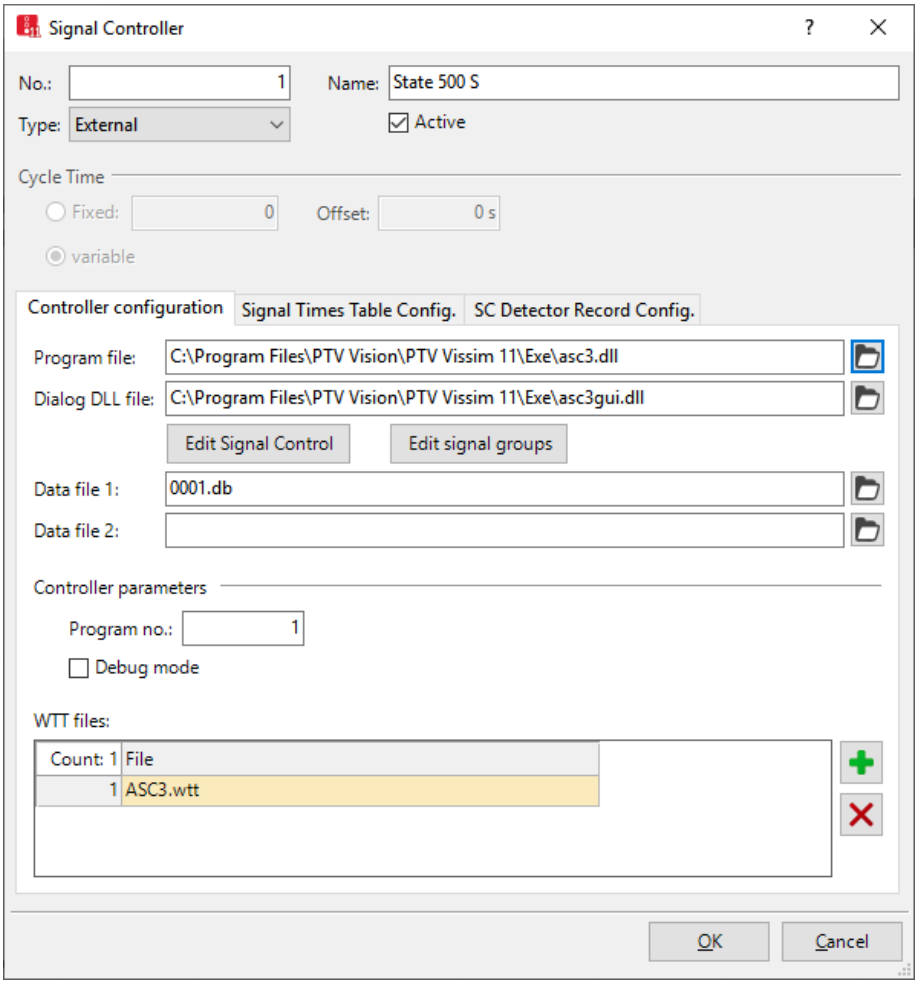


Figure 2.6-2: ASC/3 Signal Controller Definition

- Once the files are referred and uploaded, a click on "Edit Signal Groups" will open the ASC/3 SIL programming window (**Note**: the programming can be performed through Controller Front Panel Simulator once the simulation is started). The Database Editor is shown in Figure 2.6-3.

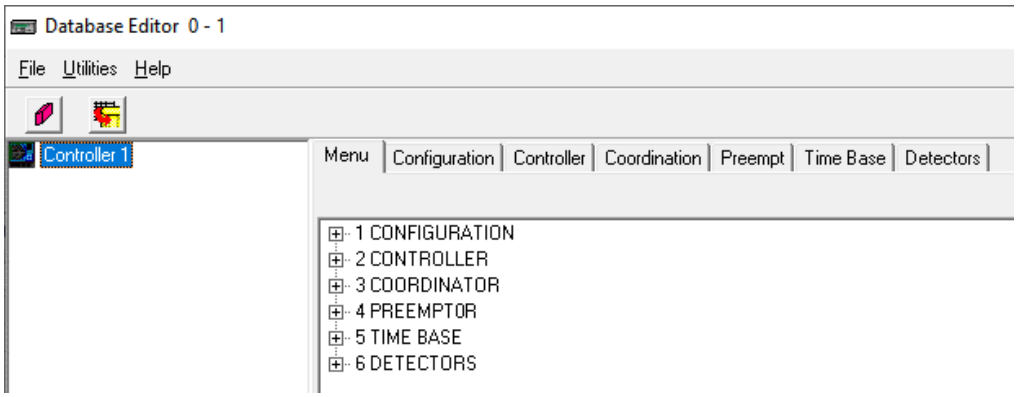


Figure 2.6-3: ASC/3 Database Editor

- The author's suggestion is to begin the programming by defining the "Time Base". This setting defines the Day Plan/Schedule and Action Plan as the time references for selecting timing and phasing plans. This is shown in Figure 2.6-4.

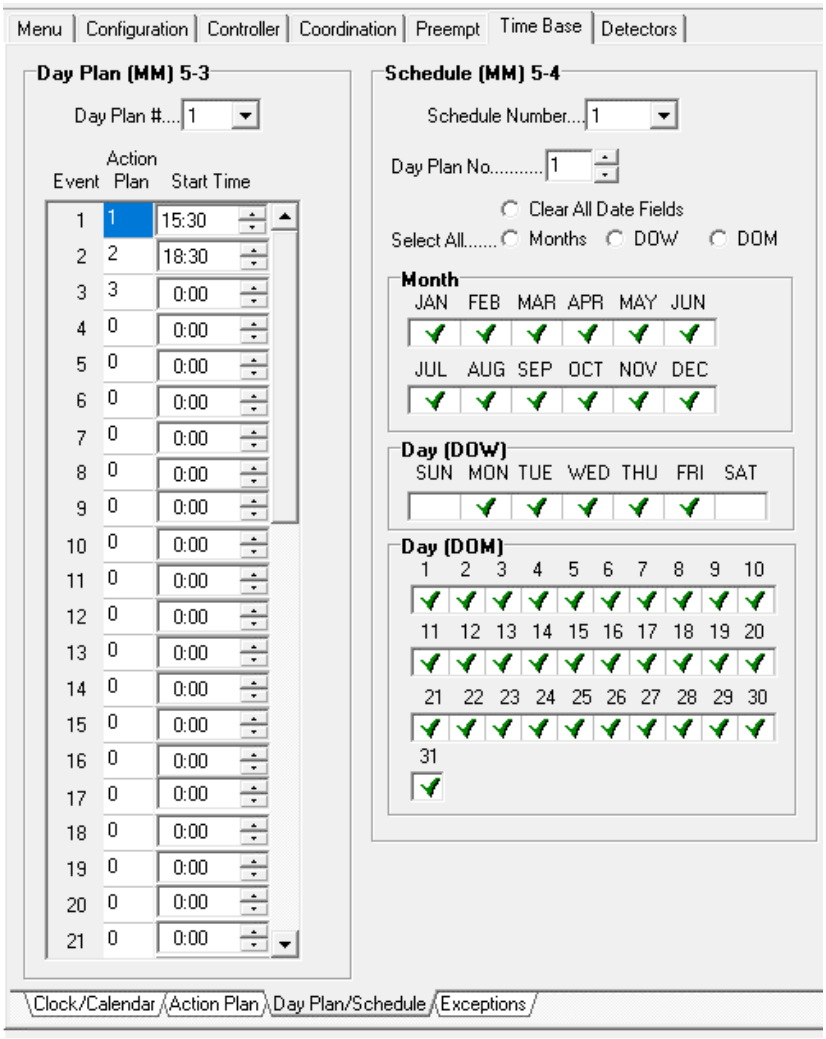


Figure 2.6-4: ASC/3 Time Base Settings

- "Configuration" settings define phase sequences, phase compatibility, phases in use, and logic processor, among other functionalities. This is shown in Figure 2.6-5.

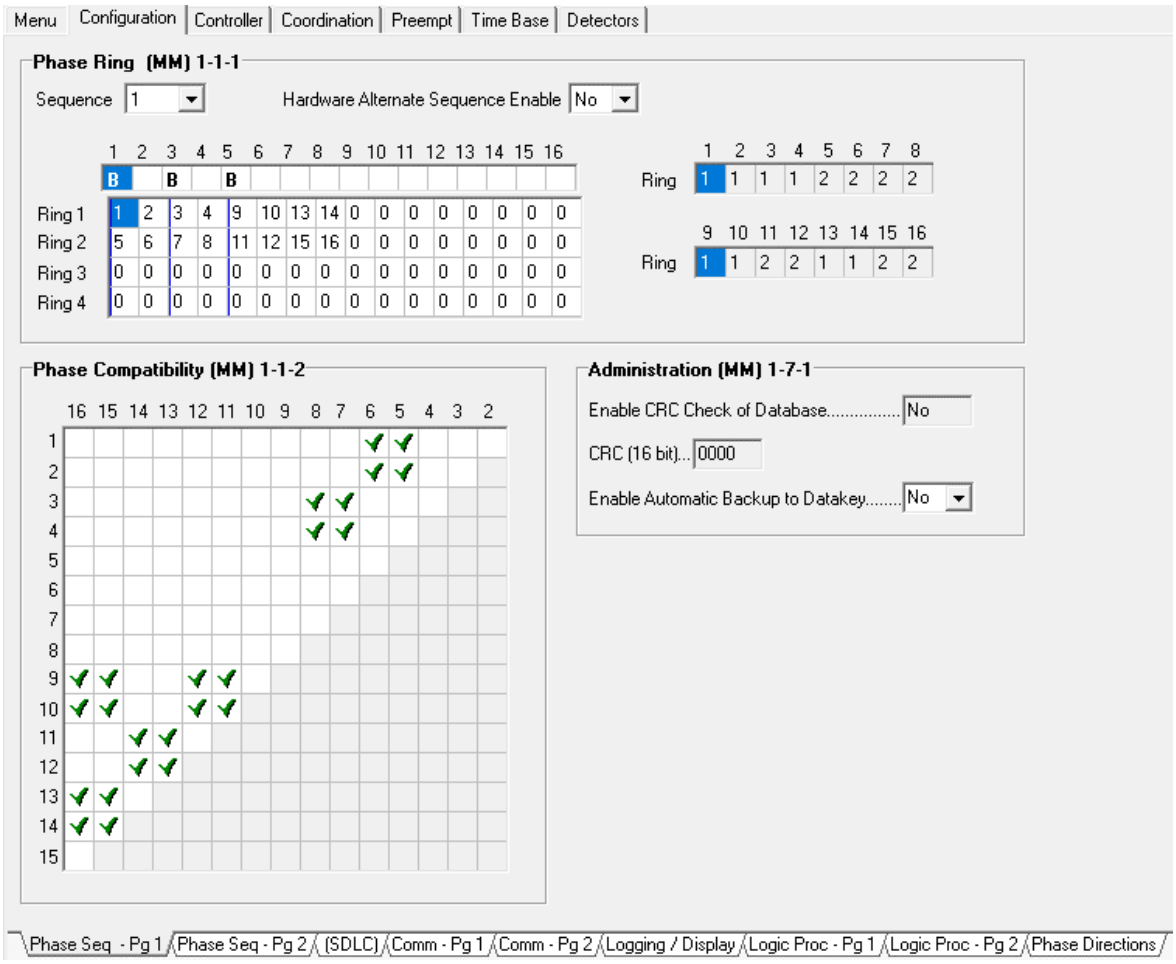


Figure 2.6-5: ASC/3 Configuration Settings

- The "Controller" settings define the timing plans, overlaps, flash options, and other phase options. A total of 4 signal timing plans can be defined under "Timing Plan". These settings are shown in Figure 2.6-6. (**Note**: according to the author's experience, phase overlaps do not function properly in microsimulation).

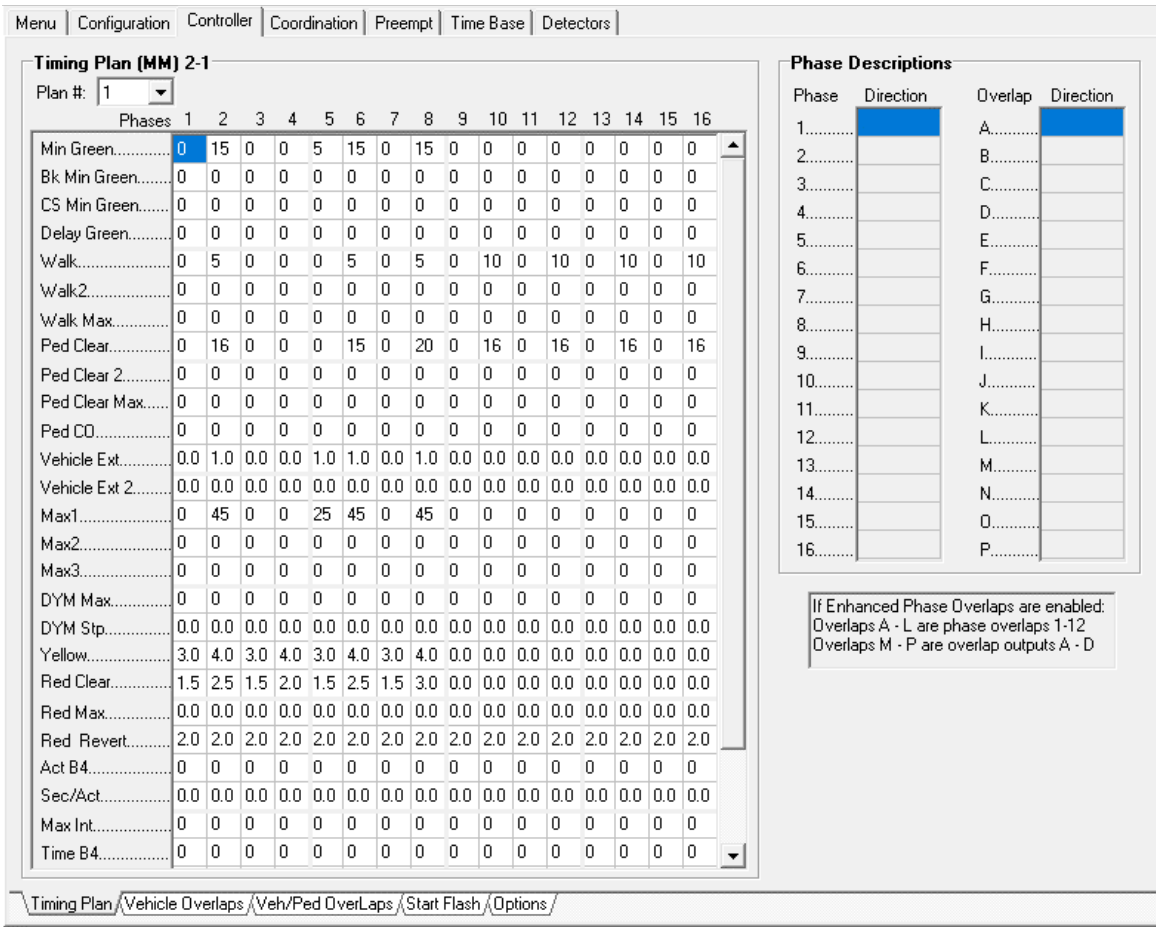


Figure 2.6-6: ASC/3 Controller Settings

- "Coordination" (Figure 2.6-7) define coordination settings (cycle lengths, offsets, phase splits, coordinated phases, phase recalls etc.). It is important here to relate the "Coordinator Pattern" with the correct Timing Plan, Sequence and Action Plan, so that VISSIM calls the correct pattern during simulation. On a side note, under "Simulation" settings in VISSIM, the user needs to define a correct date and time of the simulation that corresponds to the settings given in the "Time Base" window.

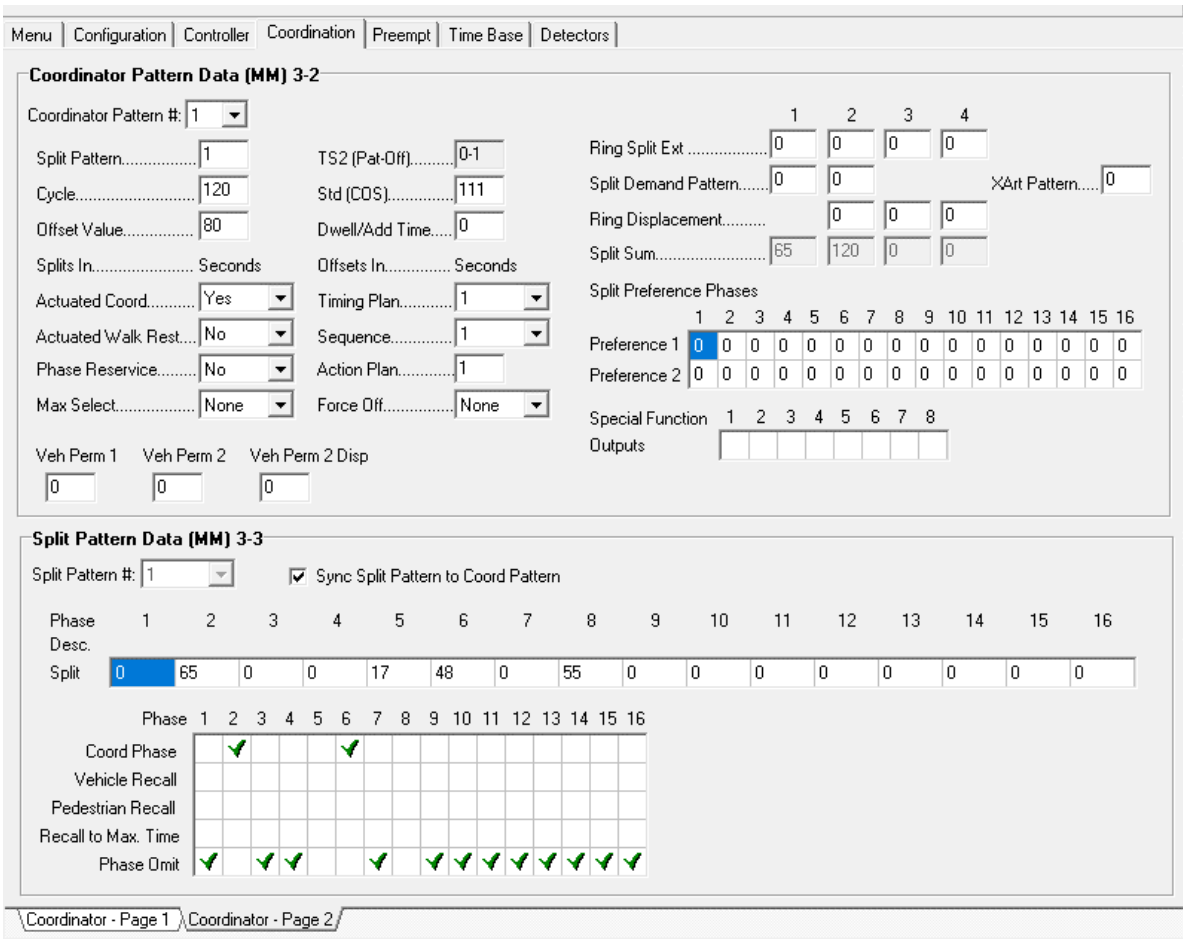


Figure 2.6-7: ASC/3 Coordination Settings

- "Preempt" settings define preemption and Transit Signal Priority (TSP) strategies (Figure 2.6-8). TSP introduces green extension and early green settings. Other types of TSP need to be created through the logic processor.

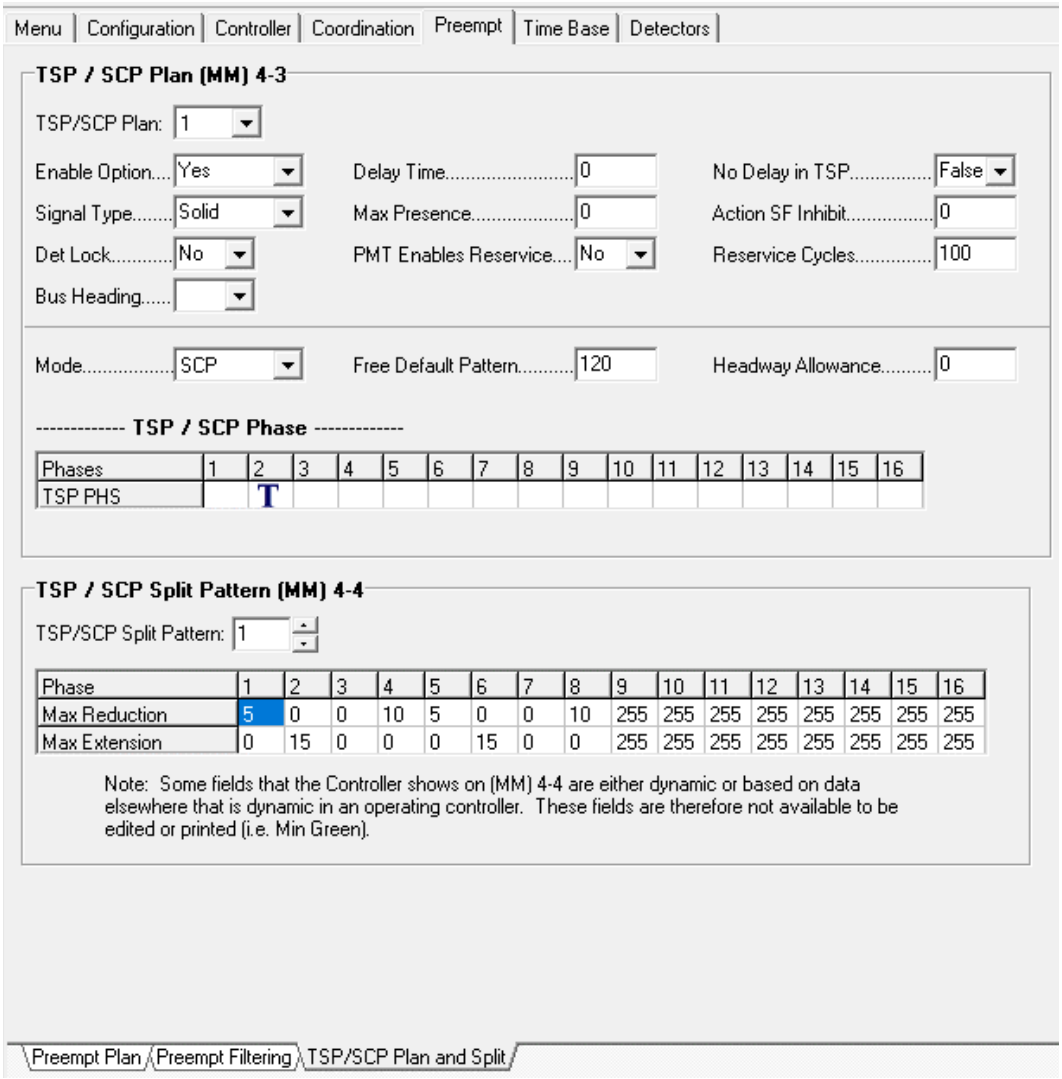


Figure 2.6-8: ASC/3 Preemption/TSP Settings

- "Detectors" setting define detector types and phase calls, among other settings for detection (Figure 2.6-9). A total of 4 different detector plans can be defined and used in simulation.

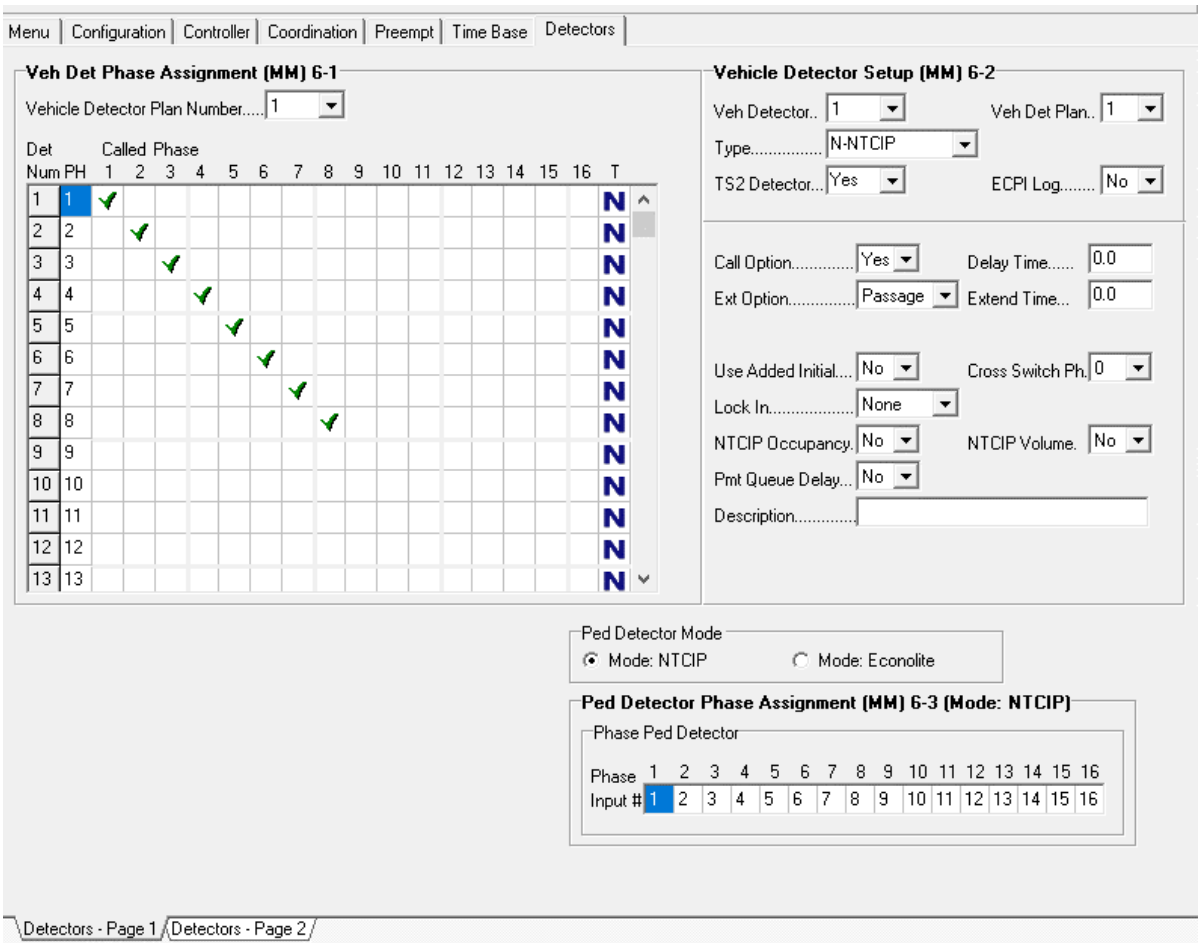


Figure 2.6-9: ASC/3 Detector Settings

Once the user defines all controller settings, a click on the “Store” icon on the left-hand side of the Database Editor (or Ctrl+S) will save the settings in the corresponding database file (0001.db, 0002.db etc.).

When the simulation is started, VISSIM will call all ASC/3 controllers and they will be shown as Controller Front Panel Simulators. Settings can also be changed through this application. A running ASC/3 VISSIM model is shown in Figure 2.6-10.

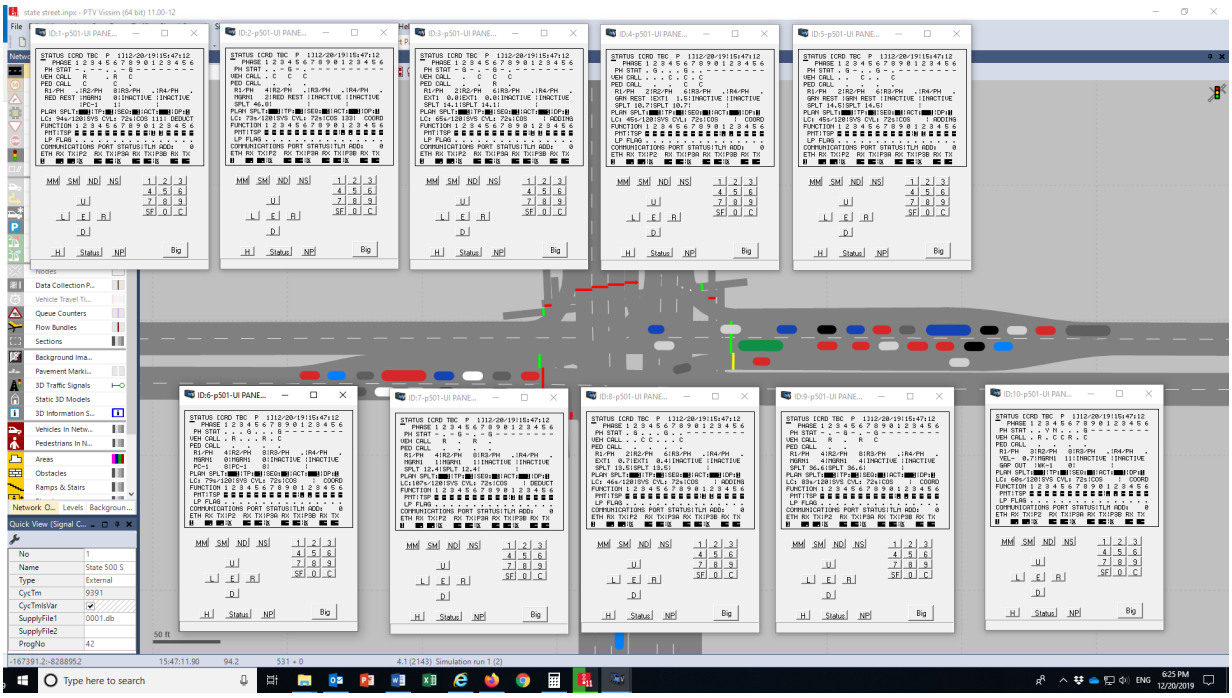


Figure 2.6-10: VISSIM ASC/3 SIL Model

Overlap Settings in ASC/3 SIL

ASC/3 allows 16 phases and 16 overlaps to be defined. The overlaps are numbered A-P. However, these overlaps in the VISSIM model will correspond to phase outputs 17-32. The matching between ASC/3 overlaps and VISSIM signal groups is as follows:

ASC/3 overlap	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
VISSIM SG	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

In order to use the ASC/3 overlaps in VISSIM, the SGs (17 – 32) need to be defined first in the VISSIM model. These can be done by using the VISSIM's RBC controller by creating overlaps numbered 17 – 32 (or as many as needed for the current simulation). Note that these overlaps do not need to be defined in RBC, only the overlaps numbers are needed to successfully connect ASC/3 overlaps with VISSIM SG outputs.

An example of overlap settings for a center-running BRT line can be found in this video.

2.6.3 Using NTCIP to manipulate ASC3 VISSIM Software in the loop

Contributor(s): Pengfei (Taylor) Li, University of Texas in Arlington

PTV VISSIM provides a high-fidelity signal emulator, the ASC/3 provided by Econolite. It is commonly referred to as software-in-the-loop simulation or SILS. Not only does the ASC/3 SILS contain all the real-world control logic but also the fully functional NTCIP protocols. This feature offers us with additional flexibility for traffic signal control system.

The family of NTCIP protocols is based on Simple Network Management Protocol or SNMP. The SNMP is a very common protocol for network management and so there are plenty of 3rd-party free libraries we can use. We just need to specify the specific OIDs defined by the NTCIP steering committee. The NTCIP-compliant traffic signal controller (like Econolite ASC/3) should immediately respond and return the formatted message once it receives a correct SNMP message.

The SNMP library I select is for C# language.

2.6.3.1 Setting up the ASC/3 SILS IP address

Once a VISSIM model is launched, each intersection controlled by ASC/3 will populate a screen. Users should set up its IP address as ("127.0.0.1") (main menu ->1->5->1). To distinguish multiple ASC/3, the users need to specify a different port for each controller (main menu ->1->5->5).

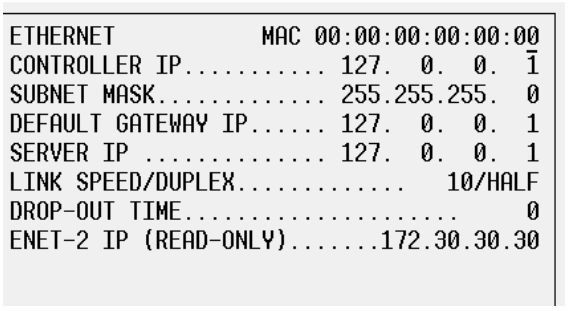


Figure 2.6.3-1:

Figure 2.6.3-2:

Once it is established, it will be stored permanently and the new IP address will be kept unchanged next time.

2.6.3.2 C# program

The C# program only includes the framework for users to communicate with ASC/3 or any NTCIP-compliant controllers. Please note the SNMP via Ethernet port is based on UDP protocol. The C# code is accompanied with many comments and so it should be easy to follow.

2.6.4 References

1. Econolite. *ASC/3 2070 Traffic Controllers for Traffic Operations*. Obtained through the Internet www.econolite.com/wp-content/uploads/sites/9/2016/10/controllers-

asc32070software-datasheet-2018.pdf
Accessed December 20, 2019.
2. Econolite. *Product Type: ASC/3 SIL – General Product Description*. Obtained through the Internet www.econolite.com. Accessed April 15, 2009.

2.7 Traffic Signal System Performance Evaluation with Simulation

Contributor: Chris Day, PhD, PE, Iowa State University

2.7.1 Aims of Performance Measurement

Traffic simulations are usually carried out to compare alternative scenarios (traffic control, network design, etc.). The basis of comparison is through the calculation of performance measures that quantify various aspects of the experience of roadway users in the network. From a traffic signal perspective, this frequently relates to how well different segments of traffic were served by different control scenarios.

When beginning a performance evaluation using simulation, it is important to consider the purpose of the study and the objectives of the traffic control. This will provide some insights as to which performance measures are best suited to evaluate whether these objectives are being met. It is essential to come to a decision on these points early enough in the study to ensure that appropriate data is obtained. Otherwise the simulations might need to be repeated.

It is not possible to enumerate every possible objective of traffic control, but we may consider a few basic questions to help relate a sense of the types of aspects of evaluation that would be relevant:

- What environment(s) are considered in the study? Urban, suburban, rural? Corridor, network, or isolated intersection? Is coordination an important objective of the operation?
- Are there special facilities included in the model that are important to the study outcomes, such as railroad grade crossings, interchanges, or non-signalized intersections?
- What modes of traffic are present and should be considered? Vehicle, pedestrian, bicycle, transit, or others? Should different segments of traffic be considered separately (e.g., passenger car versus freight)? Do they have differing objectives?
- What levels of traffic are considered? Low, moderate, high? Does the situation involve saturated or oversaturated traffic volumes? How might the objectives of the operation change with traffic level? Will different performance measures be needed to address these different levels?
- What mechanisms of control are included? Is control one-directional, meaning that information about vehicle demand are used solely to derive the signal control, or is it bidirectional, where the traffic control can supply vehicles with information such as green-light advisory? Is the vehicle routing a factor?

As an example it may be helpful to briefly consider the most common objectives of the types of signal control in common use in the field today for uncongested conditions. A starting point for addressing overall objectives is perhaps the following statement [3]:

> "We will do our best to avoid making drivers stop, but when we must make them stop, we will delay them as little as possible, within the context of safe operation."

This rather straightforward statement of operational objectives relates to an assumed road user objective of being able to travel between an origin and a destination within the shortest travel time possible, or to put it another way, while experiencing as little delay as possible. From this it may be tempting to conclude that all we need to do is measure delay, and that will solve all of our analysis needs. However, delay itself is not a completely straightforward measure as it can be tabulated in a number of different ways: as average and total delay, and by movement, movement type, intersection, etc. To seek what ways our delay measurements should be presented, we need to consider more specific objectives.

Two common objectives of uncongested signal operation are (1) to provide an *equitable distribution of green times* at intersections; and (2) to provide *smooth traffic flow*. Actually, both of these objectives suggest more specific sets of desirable outcomes that are in tension with each other.

- Equitable distribution of green: the overall cost to the traveling public—the *total delay* of the intersection—should be as low as possible, yet no individual movement should be subject to excessive amounts of delay (i.e. the *average delay* of individual movements should be considered).
- Smooth flow: we wish to avoid stopping traffic whenever possible—thereby reducing the *number of stops* or perhaps reducing the *travel time* on certain route through the system; yet this usually requires imposing delay on other, lower-priority movements.

Because multiple objectives often exist, multiple performance measures are often appropriate. As an example, even in terms of conventional signal control, in many if not most cases, operating the signals in a fully-actuated mode will likely yield lower total delay than using a coordinated signal plan. However, coordination is used to improve the performance of certain priority routes through the network. To capture both aspects of performance we could, for example, examine both total delay and travel time on the coordinated routes, or perhaps tabulate delay by movement type (coordinated and non-coordinated movements), etc.

In addition to such metrics as delay, travel time, and other quantities derived from vehicle motion, there are other metrics that can be developed that can capture additional elements of operation. For example, a change in delay or travel time on a route through a traffic signal system would imply an improvement or worsening of the signal coordination. To establish the cause of the shift, other observations would be needed.

In recent years, a methodology for obtaining a record of control system activity has emerged in real-world applications, opening the way for numerous performance measure applications [4]. The data type, "event data" or "high-resolution data", consists of records of discrete events occurring at traffic signals, such as the start of green, yellow, or red on a particular phase, or when individual detectors turn on or off. A variety of metrics and visualizations have been developed around this data type, which have lately come to be known under the name of "Automated Traffic Signal Performance Measures" (ATSPM). Because of the cost in obtaining vehicle-side data in the real world, ATSPM have seen considerable real-world usage in recent years. However, this type of data may also be able to provide insights into simulated traffic control systems as well. There have been a handful of studies that have incorporated these types of performance measures into simulations. However, this author speculates that they are not used more frequently because of a lack of widely available information on how to produce such metrics. This document will attempt to help close this knowledge gap.

2.7.2 Extracting Performance Measure Data from Simulation

It is possible to divide performance measures and their related data types into two broad categories: the "vehicle" side and the "infrastructure" side.

Vehicle-side measures depends solely on the record of vehicle motion through the system: either the vehicle trajectories themselves in their most raw state, or various derivatives of trajectories, such as delay, travel time, stops, and queue length. Real-world traffic control systems presently do not often have direct access to data of this type, instead obtaining traces of it through detection or sampling by other means. This state of things is likely to change with the introduction of concepts like vehicle-to-infrastructure communication. In simulation, such data is rather easy to obtain and there are often many mechanisms for exporting such data.

Infrastructure-side measures reflect the record of control system activity: the duration of green times, and the record of what the control system was able to directly observe and respond to. While the vehicle data provides the essential performance information, a more comprehensive understanding of why those outcomes occurred can be obtained by a view into the control system through graphical and quantitative performance measures.

Table 2.7-1 provides a listing of various data sources available in some commonly used simulation programs and the real-world equivalents of these.

Table 2.7-1: Simulation Data Sources

	Real-World Equivalent	VISSIM	SUMO	AIMSUN
Vehicle-side data				
Vehicle trajectory	Automatic Vehicle Location Data (GPS trace)	Vehicle Record	?	?
Vehicle ID at location	Automatic Vehicle Identification Data	Data Collection Point, Travel Time Section	?	?
Travel time and/or delay measurement	Using AVI/AVL data, or through direct observation	Travel Time Section, Node Data	?	?
Queue length measurement	Direct observation	Queue Counter	?	?
Infrastructure-side data				
Signal state	High-resolution data	Signal state record	?	?
Detector state	High-resolution data	SC detector record	?	?
Controller state	High-resolution data (certain objects)	SC detector record (certain objects)	?	?

- *Vehicle trajectories* record the vehicle ID, position, speed, etc. of each vehicle in the network on every time step.
- *Vehicle ID at location* refers to records of vehicle ID registering the simulation time when the vehicle arrives at a certain position. These may be logged as times of observation at a single location (such as with VISSIM "data collection points") or as pairs of observations at two locations, with the travel time or delay between (such as with VISSIM "travel time sections").
- *Travel time and delay measurements* can be observed by simulation programs; often these are "out-of-the-box" measurements relying on some intended configuration of an object for measuring them. In VISSIM, for example, "nodes" are drawn around intersections and the software can be configured to supply delay measurements based on them, either as an aggregate report or as disaggregate, per-vehicle records.
- *Queue length measurements* are also possible in some programs; in this case the length of the queue would be continuously measured and logged in some way. The queue length is a physical vehicle-related quantity that is not necessarily tied to individual vehicle IDs (although the IDs of vehicles in queue could be worked out).
- *Signal state* data consists of the times when signal outputs change their display states: typically red, yellow, green for vehicle outputs and walk, flashing don't walk, and don't walk for pedestrian outputs (although any possible state could be registered).
- *Detector state* data consists of the times when detector inputs change their states from an active occupied state or an inactive unoccupied state. This binary scheme is analogous to

real-world detector operation which measures vehicle presence. Detector data is not necessarily strictly limited to this information, but this is still the most common way of representing detector states.

- *Controller state* data consists of the trace of any variables internal to a signal controller. This data can be simple or complex depending on the variable data type within the controller. For example, the data may consist of simple Boolean variables or counters, or data arrays. Usually there has to be a specific mechanism to extract data internal to the controller. For example, in VISSIM, a variable defined within controller logic would need to have a definition to be available for output in the SC (signal controller) detector record files. In other words the developer must choose to expose those variables.

Setting Up Objects for Delay and Travel Time Measurements

Some simulation programs may provide delay measurements without having to do any configuration; this is possible if the intersection has a special object for modeling intersections, such as the SimTraffic simulation that is bundled with Synchro. Other programs, such as VISSIM, do not have objects to directly model intersections, but instead the network consists of links which may intersect, and the user can freely define what constitutes an intersection.

Figure 2.7-1 shows an example of two alternative ways to set up an intersection for delay measurement in VISSIM. The *node* object is intended to be drawn in as a cordon around the central portion of intersection, not necessarily including the approaches to the intersection. The software will automatically register vehicles entering the node, note their origins and destinations, and track their delay, providing it either as an aggregated value or as raw records for each vehicle that traverses the node. From this perspective, nodes primarily focus on vehicle movement through intersections. Pedestrian movements at intersections may be more difficult to extract using this type of object.

Another way to obtain delay measurements is to use travel time sections to demarcate points on the roadway where vehicles enter and exit an intersection. VISSIM can calculate a delay for these vehicles in addition to their travel times. In this case, care has to be taken that the starting point of the travel time sections extend beyond the largest extent of queues at the intersection as well as sufficient deceleration distance, or they will not capture all of the delay. Figure 2.7-1 shows travel time sections for one of the approaches; for a typical 4-leg intersection there would need to be at least 12 sections in all, or 16 if U-turn movements are included.

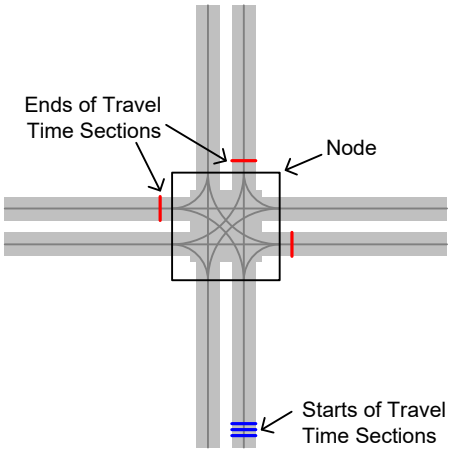
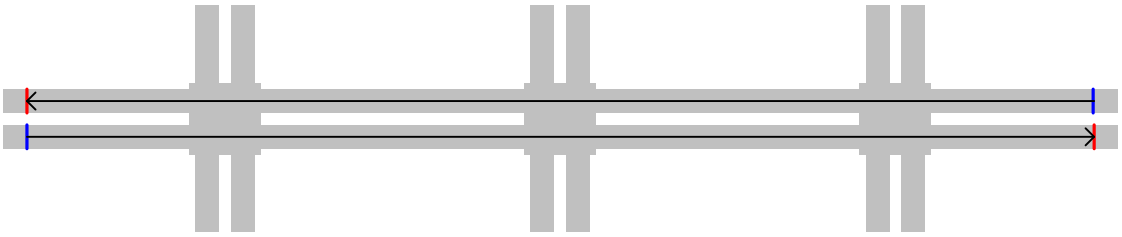


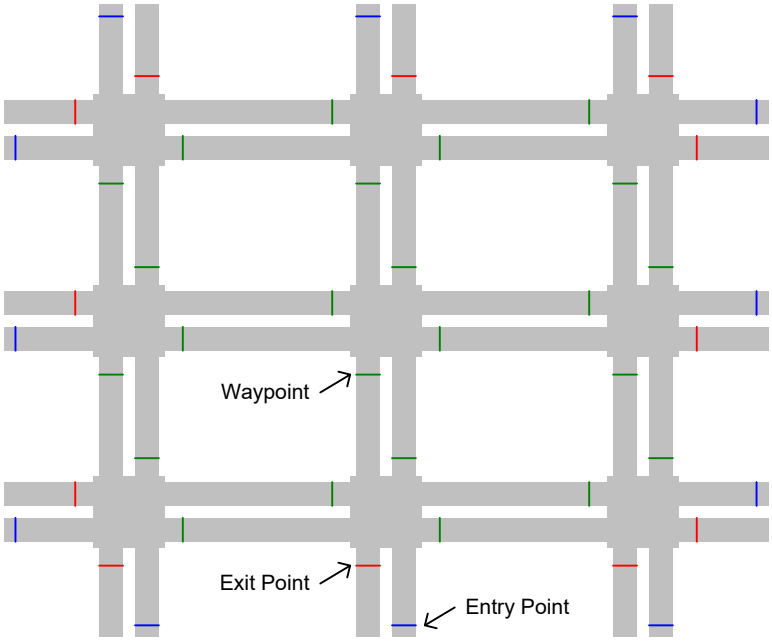
Figure 2.7-1: Outfitting an intersection for delay measurements in VISSIM.

The intended purpose of travel time sections is to record travel times, and they can of course be used for that purpose in many networks where it is desirable to record travel times along certain routes. In the case of linear arterial networks, it is often desirable to record travel times across the entire arterial; travel time sections can be established for this purpose as shown in Figure 2.7-2(a). Here also it is desirable to locate the travel time sections such that they are not influenced by delay and queuing at the first intersection.

In more complex networks where there are many possible routes, travel time sections of this nature may not always be feasible to lay out, or more specifically their layout might not capture all of the routes that would be useful for analysis. In that case it would be possible to instead take an approach of capturing vehicle IDs at select locations and then performing a match on the vehicle IDs to calculate travel times. Figure 2.7-2(b) shows a grid network that has been laid out with data collection locations where vehicle IDs would be written down, on the entry to the system and at the entry/exit for each internal link. A network set up with travel time sections on every approach to every intersection (as in Figure 2.7-1) would end up looking like the network in Figure 2.7-2(b). In this case it is possible to measure a travel time for practically any route in the system.



(a) Simple linear arterial.



(b) More complex network with many origin-destination paths.

Figure 2.7-2: Setting up travel time measurements.

Setting up Signal Event Data Collection

There are two basic ways to extract the equivalents of "high-resolution data" from simulation programs.

The first method is to obtain high-resolution data logs directly from the signal controllers themselves. This can be done with any controller type that can provide an internal data log. This can be done through hardware-in-the-loop (HITL) simulation that use physical controller units. In that case, the data can be extracted from the controller in the same manner as controllers operating intersections in the field. Software-in-the-loop (SITL) controllers can also be used to accomplish this. At present, the Econolite ASC/3 is the only SITL controller known to this author to be able to produce the high-resolution log data files, and at present the Econolite ASC/3 SITL controller is only available for use in VISSIM.

Once the high-resolution data has been obtained through either HITL or SITL controllers, it must then be *translated* from its compressed format into CSV files, from which further analysis can be undertaken. This requires the acquisition of an appropriate translator utility from the controller manufacturer. The

uncompressed data can then be developed into performance measures, either directly by the user, or by insertion into a software utility for this purpose. An example of such a software program for ATSPM is the open-source software published by the Utah Department of Transportation [5].

Once established, a setup consisting of these components (HITL or SITL controllers, translator utility, and ATSPM back-end) will achieve the equivalent of a real-world ATSPM implementation, allowing a variety of performance measures to be tabulated. However, it seems that this particular setup will not be immediately implementable by the majority of simulation users, or would require a rather onerous amount of setup time. Problems may include: a lack of access to HITL facilities, or the ability to use the one SITL controller capable of providing high-resolution data logs; as lack of infrastructure for operating software capable of handling these data logs; and difficulties in setting up the software for delivering ATSPMs in their final format. However, importantly, none of these elements are absolutely required for producing most of the performance measures. For this reason, the current version of this document instead focuses on how to use the other data in simulation programs to create the *equivalents* of high-resolution data. (Future versions may include expanded notes on the other approach.)

Although there are many different data elements defined in the enumerations for high-resolution data [6], most of the existing performance measures falling under the umbrella of ATSPM are derived from two types of events:

- The state change times of signal outputs (phases, pedestrian phases, overlaps, etc.)
- The state change times of detector inputs.

Thus by extracting these data elements from a simulation, it is possible to derive nearly all of the performance measures contained within any ATSPM system.

To begin, as mentioned earlier, many simulation programs are capable of recording the state change times of signal outputs. For example, VISSIM produces an output called “signal changes” which registers the time when any *signal group* in the network changes its state. This is a relatively straightforward data type to use, because it consists of the simulation timestamp, signal controller, signal group, and new state going into effect at that time.

The detector data may be a bit more challenging to obtain from the simulation depending on the manner in which it is exported. In VISSIM, for example, the detector states are not provided in an event format, but can be logged by use of the “signal control detector record”. The user selects which data they would like VISSIM to report during each simulation time step, which can include the on/off state of any detector, along with other variables.

Figure 2.7-3 shows an excerpt from a signal control detector record file. Here we see a number representing the simulation time (at 0.1-second resolution), followed by a single character encoding the state of a detector at that timestamp. A vertical bar symbol indicates that the detector is occupied, while a dot indicates that the detector is unoccupied. Continually logging the state on each timestamp requires as many entries as “ticks” of the simulation. In contrast, logging only the events when the states change (in this case, an “on” event and an “off” event) requires far fewer entries.

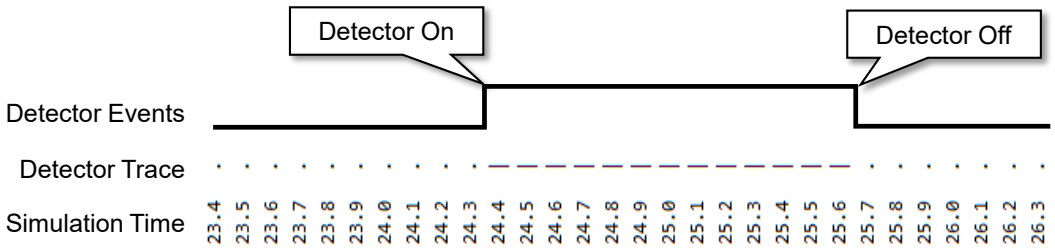


Figure 2.7-3: View of a detector state trace from VISSIM and equivalent events.

Some performance measures do not make use of detector occupancy, and an arrival time alone might be sufficient. In that case, data from a data collection point or travel time section might be usable (in VISSIM), or their equivalents (in other programs).

There are certain event types that may not necessarily be accessible through these means. For example, those events reflecting internal control parameters may not be directly accessible if the controller developer has not made it possible for them to be accessed (as through the signal control record file in VISSIM). However, these are mostly used for providing supplemental information, such as the time-of-day plan boundaries. There may be some circumstances in which such information would be useful to visualize (for example, testing a traffic responsive algorithm). For those situations some other provisions would be needed to obtain the internal controller data.

Table 2.7-2 provides a breakdown of different categories of events defined in high-resolution data enumerations, and alternative ways of obtaining this data from simulation data. The perspective of this table is oriented toward use of VISSIM, but similar concepts may be available in other simulation programs.

Table 2.7-2: Simulation Data Sources

High-Resolution Events	**Simulation Equivalents**
Signal output state changes having a direct output (red, yellow, green; walk, flashing don't walk, don't walk) – for both phases, overlaps, and other types of signal groups	Signal changes
Internal signal timing intervals that do not have a direct output (end of min green, phase check, etc.)	Imputation using signal changes with knowledge of active min green, red clearance time, etc.
Phase termination (gap-out, max-out, force-off)	Not available unless exposed by controller developer
Detector on/off events	Signal control detector record Data collection objects may be usable for certain detection information; e.g., vehicle arrival times in a travel time section when a detector 'on' event is used to measure a vehicle arrival
Phase calls, holds, and omits	Not available unless exposed by controller developer
Initiation of preempt and priority requests	May be implied by detector states
Pattern changes and adjustments, internal control logic functions, etc.	Not available unless exposed by controller developer

Illustration of High-Resolution Data Equivalents

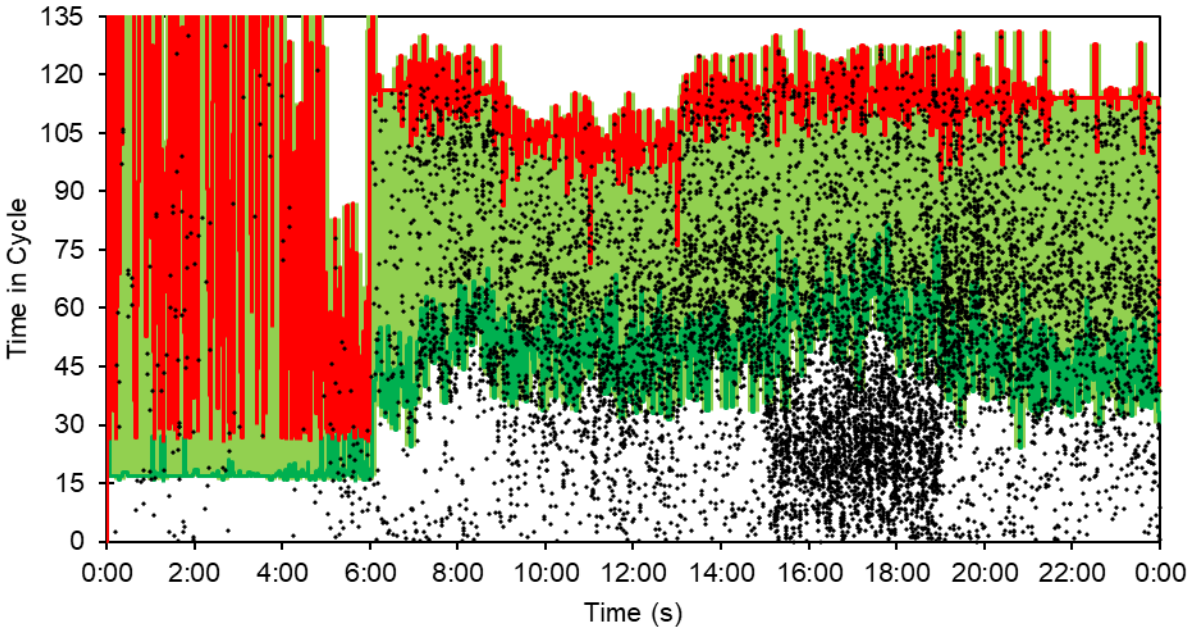
As an illustration of the possibility of obtaining equivalent data via simulation objects as would be produced by controller log files, Table 2.7-3 presents two views of phase events obtained from the same simulation run using both the signal changes data and the controller high-resolution data. The data are limited to green, yellow, and red events only for a particular signal group (SG) of a particular signal controller (SC). Each row shows the equivalent event in the two data sets. The signal ID in the high-resolution data is the same as the number used for the SC.

The main difference between the two methods of storing the data is the manner in which the event timestamps are stored. The signal changes data uses the simulation time, meaning the time elapsed since the start of the simulation, in seconds. The high-resolution data uses a timestamp including date and time. Whereas the signal changes data obtains its timestamp directly from the simulation, the high-resolution data uses the time registered in the controller clock. Note that all of the phase event timestamps in the high-resolution lag that in the signal changes data by 0.6 seconds. This indicates that 0.6 seconds of simulation time elapsed before the controller clock began to operate, a minor and in most cases negligible difference yet one that may be worth being cognizant of depending on the application of the data.

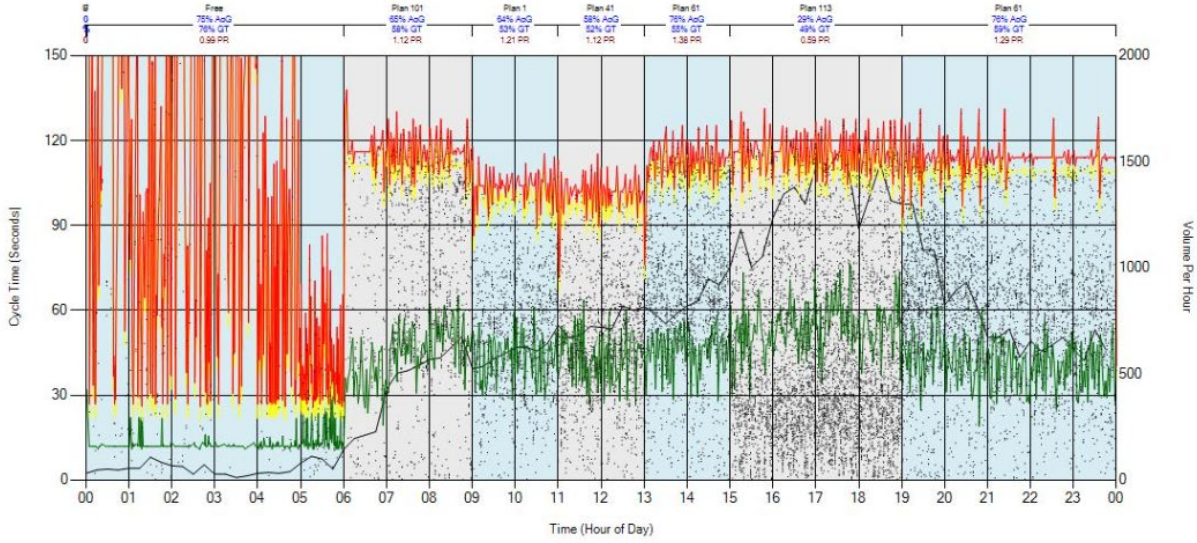
Table 2.7-3: Comparison of phase state events from alternative data sources

Signal Changes Data				Controller High-Resolution Data				
Time	SC	SG	State	Signal ID	Timestamp	Event Code ID	Event Param	State
10.9	1002	2	green	1002	2020-06-10 00:00:11.5	1	2	green
20.9	1002	2	yellow	1002	2020-06-10 00:00:21.5	8	2	yellow
25.9	1002	2	red	1002	2020-06-10 00:00:26.5	10	2	red
58.0	1002	2	green	1002	2020-06-10 00:00:58.6	1	2	green
242.7	1002	2	yellow	1002	2020-06-10 00:04:03.3	8	2	yellow
247.7	1002	2	red	1002	2020-06-10 00:04:08.3	10	2	red
259.6	1002	2	green	1002	2020-06-10 00:04:20.2	1	2	green
269.6	1002	2	yellow	1002	2020-06-10 00:04:30.2	8	2	yellow
274.6	1002	2	red	1002	2020-06-10 00:04:35.2	10	2	red
286.6	1002	2	green	1002	2020-06-10 00:04:47.2	1	2	green
362.5	1002	2	yellow	1002	2020-06-10 00:06:03.1	8	2	yellow
367.5	1002	2	red	1002	2020-06-10 00:06:08.1	10	2	red

Figure 2.7-4 compares coordination diagrams built from two different data sources being recorded during the same simulation run. Figure 2.7-4(a) was constructed using the signal event times available from the signal changes data, in conjunction with vehicle arrival times obtained using a travel time section located at the same detector location as the setback detector. Figure 2.7-4(b) shows a coordination diagram constructed using the UDOT open source software, after the simulation data was loaded into its database and necessary detector definitions were entered. The charts exhibit identical information about the relationship between vehicle arrivals and the green times, since they are based on largely equivalent data. However, the UDOT chart includes additional information about the timing plan in effect and displays the time-of-day pattern boundaries, which are only available in the high-resolution data.



(a) Native simulation data



(b) Controller high-resolution data

Figure 2.7-4: Coordination diagrams for the same approach from alternative data sources

2.7.3 Managing Simulation Data

Most traffic simulations are executed for at least two scenarios for comparison. More extensive studies may require numerous scenarios to be undertaken. In many cases, it may be possible to manage data by storing the output files and performing comparisons using ordinary spreadsheet tools or the like. However, in more extensive studies it is advantageous to employ a database management system to manage the attendant larger amounts of data and perform calculations much more quickly.

There are many ways which such data could be stored, but Figure 2.7-5 presents one possible configuration of tables that could be used to store a variety of simulation outputs. These are established to try to store the data in a more "normal form," that is, without numerous duplicate entries within the data tables, which may grow quite large with many different simulations (especially if trajectory data are logged). The data produced by simulation programs tends not to be provided in this manner: for example, trajectory and node data will write the vehicle type along with the vehicle ID for every single row in those tables, whereas it is possible to write the vehicle type *once* for the vehicle ID (assuming the vehicle type does not change), in a vehicle table as shown here.

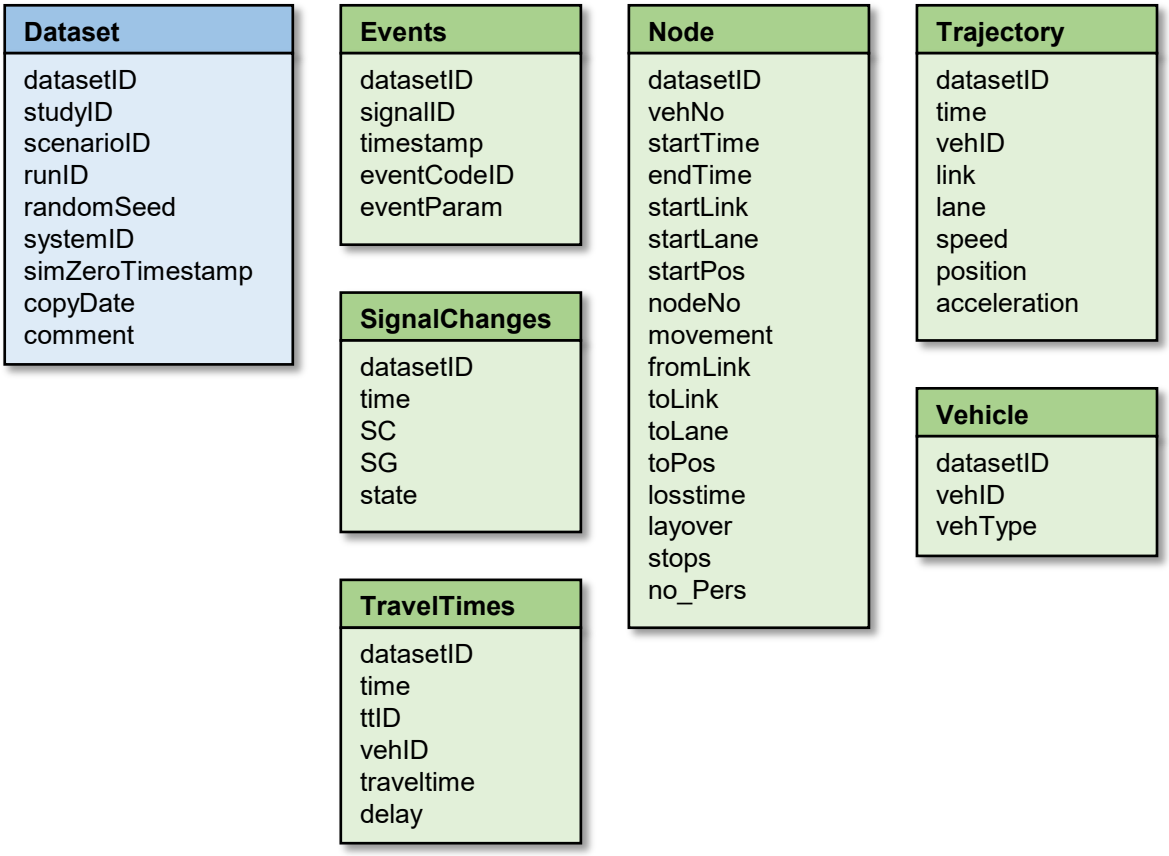


Figure 2.7-5: Database tables for storing large amounts of data

The dataset ID is used to link data across all of the tables involved. It represents an individual execution of the simulation program: one particular iteration or run of a particular simulation model. The *dataset* table is intended to store the basic information about this particular run; these columns are arbitrarily defined and could be adjusted according to the preferences of the analyst, but their purposes of some columns are worth discussing:

- The study ID, scenario ID, and run ID are intended to organize data by a particular study purpose, a particular scenario (e.g. alternative schemes to be compared), and by iteration number. A further system ID is added in case it might be useful to store additional information such as the simulation network (which might be used in separate studies).
- Space for information such as the random seed and user notes are provided.
- The "simZeroTimestamp" is intended to relate the *date* that would be present in the high-resolution data to the dataset. In some cases this may be useful to know.
- The "copyDate" is used to store the time when the data was copied into the database.

By making the datasetID an automatically generated number in a process to copy the data into the database, it will be possible to avoid having duplicate data.

The data tables for Signal Changes, Travel Times, and Nodes follow largely from the method in which VISSIM exports the data, with some simplifications to avoid redundancy. For example, the Signal Changes table lists only the state changes and excludes other information such as the controller type. The Trajectory table is intended to include entries from the VISSIM Vehicle Record: this is not the default configuration but includes some added information about the vehicle speed and acceleration.

The Events table is intended to store high-resolution data. In this case the dataset ID is included as a separate column, which can separate different data sets. If such a column were not included, then it would be important to manage the dates for different simulation runs. High-resolution data is intended for real-world use, and thus each day's data is unique; in a simulation environment we can repeat the same day many times. When bringing high-resolution data into an external utility such as the UDOT open-source ATSPM software, it is important to ensure that distinct dates are used for different simulation runs. Otherwise errors may occur because data from multiple runs would be conflated because they use the same range of timestamps.

The Vehicle table is used to store the vehicle type for entries in the Node and Trajectory table so that this does not have to be written repeatedly in those tables.

Figure 2.7-6 presents a workflow for managing data using a scheme such as this. The initial decision point depends on how the data is provided by the simulation. If flat files (e.g., CSV or similar text files) are provided, these must first be loaded into the database system using a bulk insert operation. It is safer to lad these into a set of initial tables before adding them to the tables for permanent storage. That way it is possible to check for errors that occur during the bulk insertion process prior to committing to keep the data. Some software programs may allow writing data directly to the database at runtime. VISSIM, for example, offers this functionality (although not for certain data types such as data collection points). In that case, it is best to allow the software to write to an initial set of tables according to its own default format, and then execute a query to copy the data to the permanent storage location.

Prior to this copying process, it is important to establish a unique ID for the dataset which will make it easier to retrieve the data later on. One way to maintain unique dataset IDs is to establish the dataset ID as an automatically-numbered column in the Dataset table. If this is done, then whenever a new row is added to the Dataset table, a new, unique ID will be generated. One can add a row containing the other information about the current dataset, and then query the table for the maximum row number to obtain the newly created dataset ID for use in the subsequent copying process. If such a strategy is not used, it is important to manage the list of datasets to avoid duplicate entries.

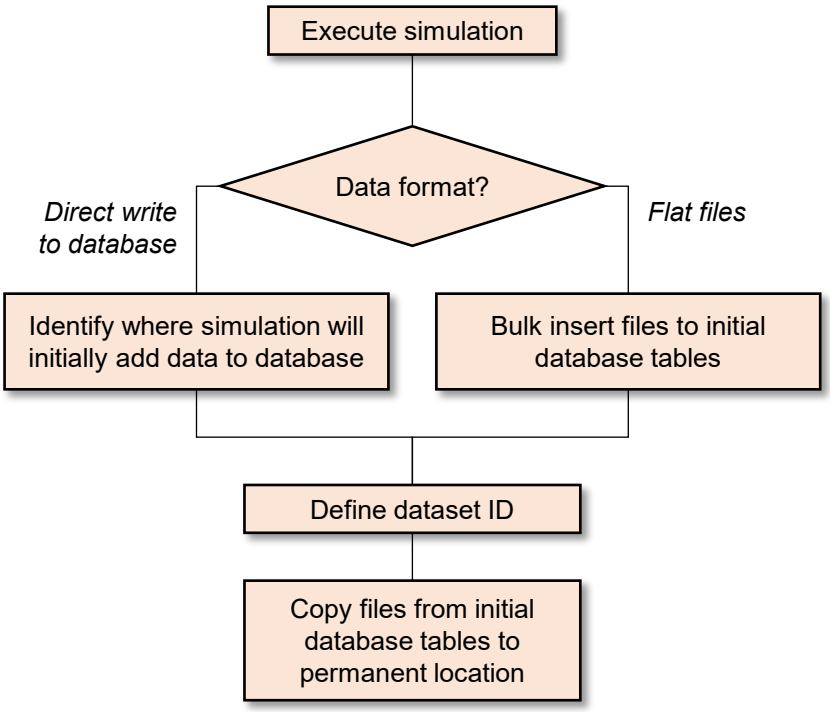


Figure 2.7-6: Flowchart for use of database for managing simulation data

2.7.4 Delay and Travel Time Performance Measures

Delay and travel time are perhaps the most common performance measures used in simulations of traffic signal systems, in part because average delay is the basis for the Highway Capacity Manual level of service, total delay is convenient for aggregating across multiple movements and intersections, and travel time is a rather succinct metric for expressing performance of signal coordination. The metrics are derived entirely from the vehicle movement and most simulation software has provisions for ability providing such metrics.

The total delay in a system can be expressed in a variety of ways, either as a single value representing the entire state of the system, or as segments of traffic, as illustrated by Figure 2.7-7. In this particular example, the total delay experienced under two alternatives is quite similar for the two scenarios examined (Figure 2.7-7a). Scenario B has slightly higher delay, yet the difference is probably less than 5%. The apparent difference could possibly be exaggerated by changing the vertical scale, but it would probably be better to dig a little deeper by dividing the total delay into segments, to see if any insights arise. Two examples are shown here where the delay is shown by movement type (Figure 2.7-7b) and vehicle type (Figure 2.7-7c, Figure 2.7-7d). In each case the total delay is very similar but the breakdown among segments of traffic illustrate tradeoffs. In some cases it may be helpful to rearrange the data to emphasize one of the elements (compare Figure 2.7-7c versus Figure 2.7-7d), depending on the story the analyst wishes to convey with the visualization.

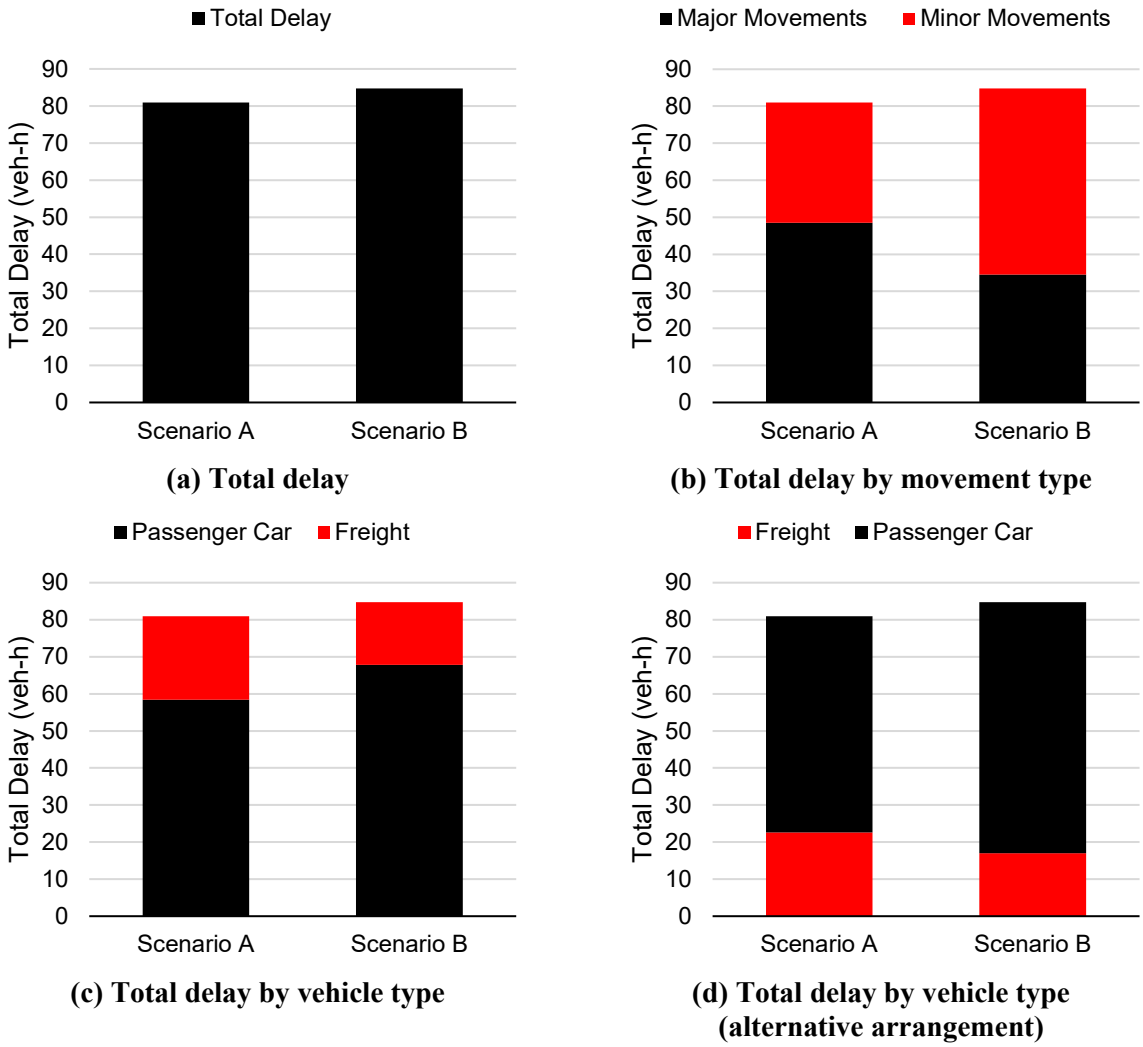


Figure 2.7-7: Visualizations of total delay

An alternative layout for when tradeoffs occur would be to arrange scenarios on two axes, as shown in Figure 2.7-8. Here, four hypothetical scenarios are visualized with delay divided between freight and passenger car segments. One could imagine such an arrangement being used considering four alternative control options implementing some sort of freight signal priority, for example. By arranging the data in this way the tradeoff between two competing priorities can be visualized rather succinctly.

The average delay by approach or movement can be somewhat more challenging to display given the number of different approaches and movements that can exist within a system. The values could be displayed in a tabular or map format, but this may be rather difficult to visualize beyond a handful of intersections. Another option for creating a succinct view of average delays is to sort the delays for separate scenarios according to their value from greatest to least (or a "Pareto sort"), as shown in Figure 2.7-9. Although this does not allow direct comparison of any particular movement, it does show the overall distribution of delays throughout the system. In this particular view, we see that Scenario A generally has higher delay for most movements than the other three scenarios, whereas Scenarios B, C, and D have about

the same value for many scenarios, with Scenario D having a slight advantage for several of the worst-performing movements.

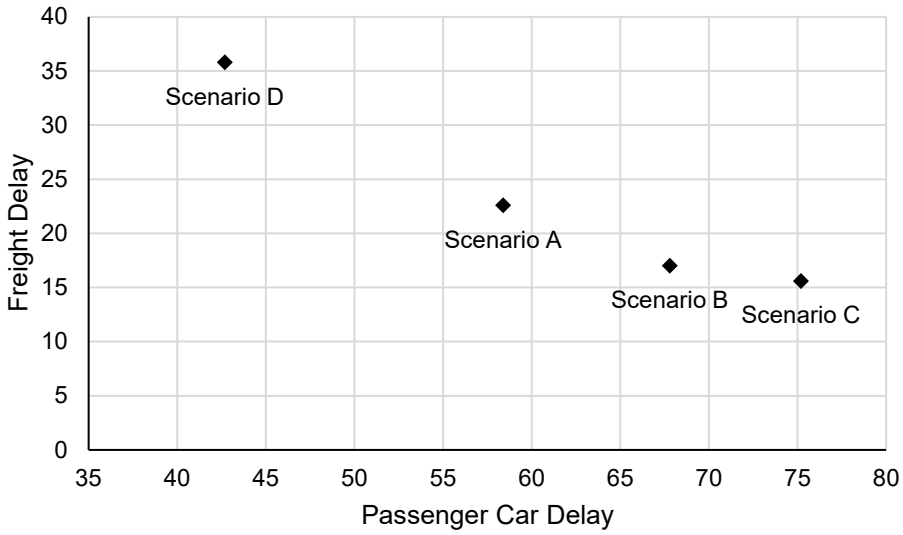


Figure 2.7-8: Visualizations of total delay by different segments of traffic

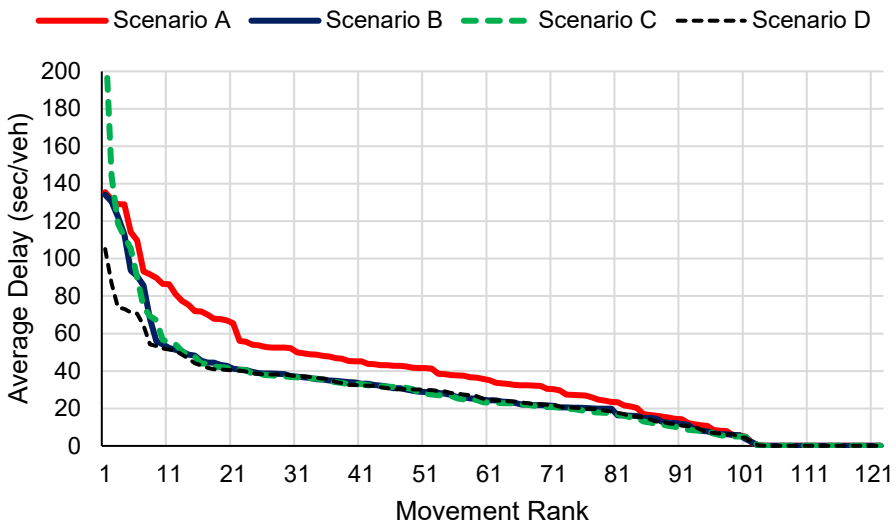
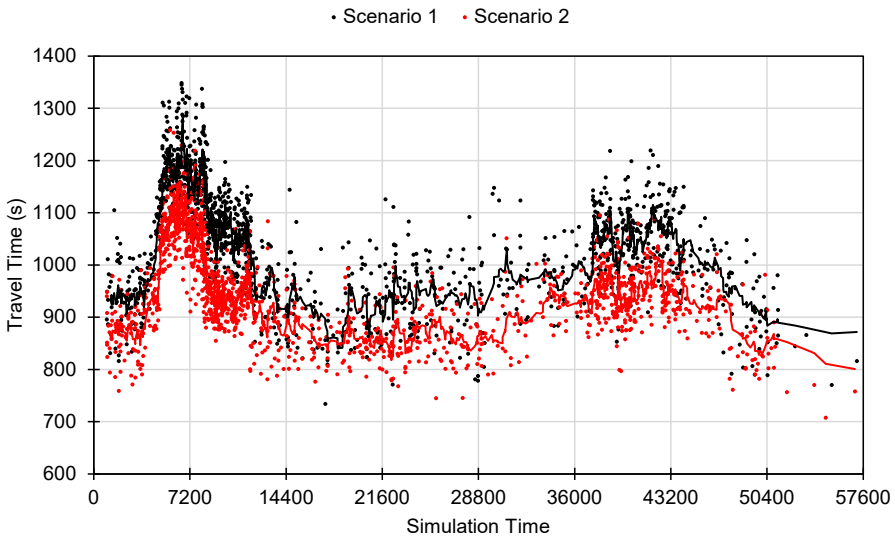


Figure 2.7-9: Average movement delay comparison between several scenarios

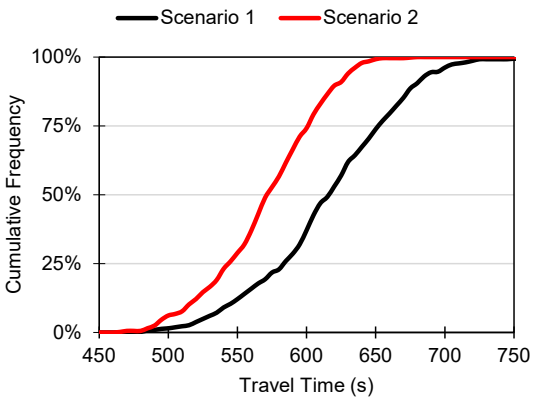
Travel times can also be useful to compare performance across alternative scenarios, especially for evaluating the effects of signal coordination. There are numerous techniques for visualizing these, but one of the most useful is to examine the cumulative frequency of travel time (the "cumulative frequency diagram" or "cumulative distribution function").

Figure 2.7-10 presents an example for a scenario in which a complex volume series was introduced, calibrated to field volumes that changed during each hour of a 16-hour simulation. Figure 2.7-10a shows all of the individual travel times of vehicles traversing a particular route (along with moving average trendlines). In this case, the two point clouds tend to be separated with Scenario 2 having somewhat lower travel times. However, in some cases the two data series might not be so distinct, making this view less useful.

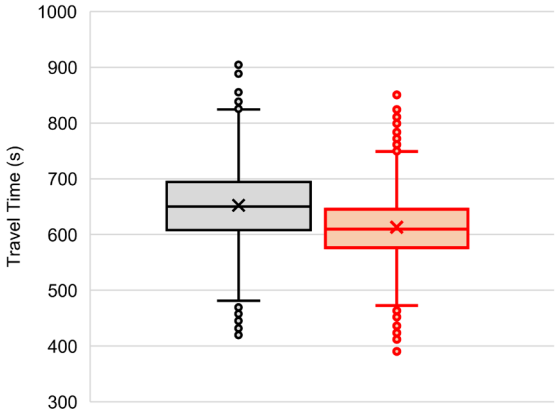
Figure 2.7-10b shows a cumulative frequency diagram representing travel times within a distinct time period (21600–28800 simulation seconds), which permits two distributions of travel time to be directly compared. In this case we can see that Scenario 2 has a lower median travel time, and in fact the entire distribution is wholly to the left of Scenario 1, indicating that travel times are reduced for all percentiles. It is difficult to visually ascertain whether the level of reliability has increased or decreased; that is made easier by presentation as a box-whisker plot, as shown in Figure 2.7-10c. This illustrates that the median (horizontal line in the middle of the "box") and average (the "x") have both decreased, while the variability has decreased (the height of the box marking the interquartile range is lower). These visualizations also make it rather easy to display several different scenarios at once.



(a) Chart of individual observations (with moving-average trendlines)



(b) Cumulative frequency diagram



(c) Box-whisker plot

Figure 2.7-10: Example comparisons of travel time distributions.

2.7.5 Performance Measures from High-Resolution Data

This section describes the calculation of performance measures that rely on "internal" data, or the phase state changes and equivalents to the detector information used in ATSPM. In some cases the "detector" data may be replaced with the equivalent vehicle information: e.g., the arrival times in a coordination diagram. The objective of this section is to show how such metrics can be calculated without the need to install a software package to prepare the metrics.

Some SQL queries are presented that rely on use of a schema similar to that shown in Figure 2.7-5. Similar arrangements can be managed using software such as Access. The outcomes of the queries could be achieved by alternative means as well, such as through spreadsheet lookup functions.

Phase Intervals and Durations

The basic starting point of many performance measures is what we shall call a "red-green-red" table, which lays out each *instance* of a phase as it occurs throughout the time period being analyzed. The idea is to consider a red interval followed by a green interval. This is done because vehicles that arrive during red are generally unable to proceed past the signal until the subsequent green interval, thus for analysis purposes it makes much more sense to pair a red interval with the following green rather than the preceding green [4].

The query below accomplishes the assembly of such a table using a series of join statements on the signal changes table. We begin by selecting only the red times of interest. Here, they are selected for signal group (SG) 2 at signal controller (SC) 1002, as shown in the "where" statement. These are specified only for the first reference to the signal changes table, because the join statements ensure that these propagate to the other references.

```
select
    r1.time as last_red,
    min(g.time) as green,
    min(r2.time) as next_red
from
    sim.dbo.signalchanges r1
left join
    sim.dbo.signalchanges g
    on r1.datasetID = g.datasetID and r1.SC = g.SC
      and r1.SG = g.SG and g.time > r1.time
left join
    sim.dbo.signalchanges r2
    on r2.datasetID = r1.datasetID and r1.SC = r2.SC
      and r1.SG = r2.SG and r2.time > g.time
where
    r1.datasetID = 3
    and r1.SC = 1002 and r1.SG = 2 and r1.state = 0
    and g.state = 1 and r2.state = 0
group by
    r1.time
order by
```

This query starts by obtaining a list of red times as table **r1**, which is then joined to tables **g** and **r2** using the requirement that the relevant timestamp in **g** and **r2** will be greater than the red time for each row of **r1**. The join will attach each row in **r1** to every row in **g** or **r2** for which this is true. The minimum function in

the select statement finds the which should occur immediately after the initial red time. At the end of all this, we will obtain a listing of red-green-red times as shown by the first three columns of Table 2.7-4.

Table 2.7-4: Phase state events and basic interval calculations.

Last Start of Red	Start of Green	Next Start of Red	Red Time (s)	Green Time (s)	Time Between Successive Reds (s)
48282.5	48313.8	48396.5	31.3	82.7	114.0
48396.5	48448.3	48516.8	51.8	68.5	120.3
48516.8	48564.1	48641.7	47.3	77.6	124.9
48641.7	48677.3	48748.8	35.6	71.5	107.1
48748.8	48802.4	48858.4	53.6	56.0	109.6
48858.4	48892.8	48972.8	34.4	80.0	114.4
48972.8	49006.9	49082.4	34.1	75.5	109.6
49082.4	49131.8	49205.8	49.4	74.0	123.4
49205.8	49234.3	49308.5	28.5	74.2	102.7
49308.5	49362.3	49422.5	53.8	60.2	114.0

The next three columns in this table can be obtained through rather simple calculations based on the red-green-red times: the red time being the difference between the start of green and last start of red; the green time being the difference between the next start of red and the start of green; and the time between successive reds being the difference between the next and last starts of red. From such a table we could, if we so desired, immediately produce a visualization of green times (Figure 2.7-11) or the times between successive starts of red (Figure 2.7-12). As with travel times, such data could be arranged into distributions or other such views if it is desirable to compare multiple scenarios, etc. This particular example is from a simulation that has been calibrated to run a 24-hour scenario including time-of-day plan changes.

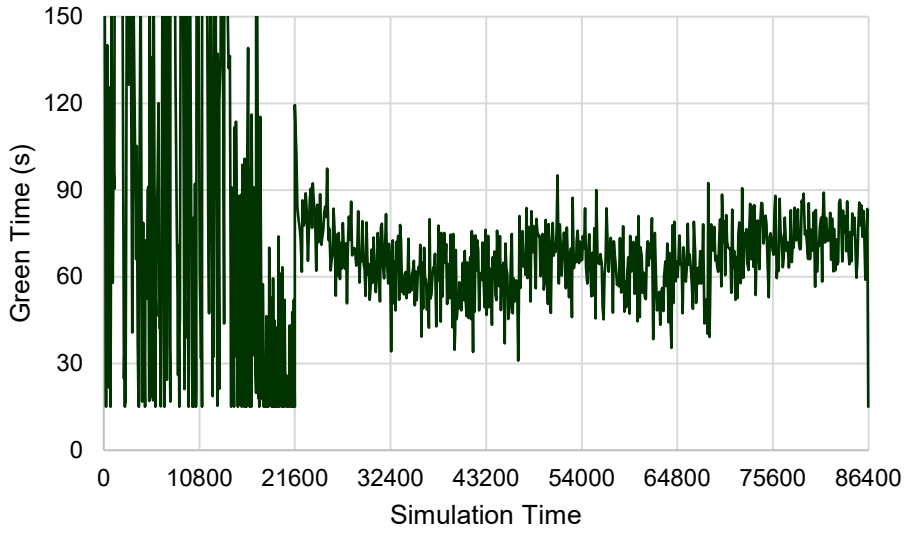


Figure 2.7-11: Green time per cycle.

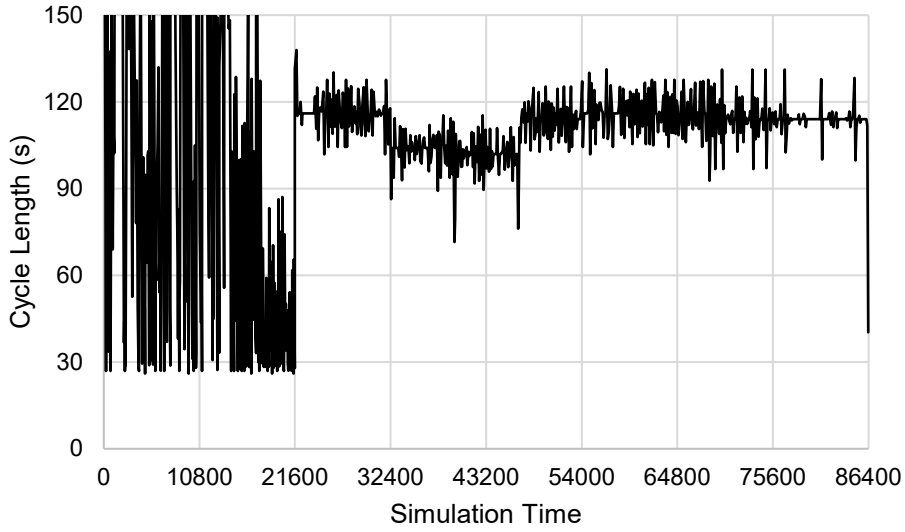


Figure 2.7-12: Time between successive red (or the effective cycle length, for some phases).

The time between successive red times can under some circumstances permit us to see the *effective cycle length*, for any phase that will occur during every cycle at an intersection. Whether such a phase exists depends on the specific configuration of the signal control. For typical eight-phase controllers, one of the major street through movements would frequently be a good reference point for determining the effective cycle length. For this example, phase 2 is "guaranteed" to occur in each cycle because it is a coordinated phase, and also under recall during fully-actuated control. This means that the time between successive starts of red for this phase makes a convenient reference point for calculating effective cycle length. There might not always be such a phase or signal group for which this is true under every circumstance. Use of a phase recall feature, for instance, might complicate this situation.

The "effective" cycle length refers to the time that the signal controller needed to serve all movements, regardless of whether the signal actually operates with a cycle length or not. In the above example, the signal operates under fully-actuated control prior to 21600 (6:00 a.m.), and so the effective cycle length often lasts for long periods of time as the signal rests in green for long periods of time in low-volume early morning conditions. During the coordinated portion of the day (after 21600), the effective cycle length hovers around a certain value, with variations occurring because of the use of an "early yield" feature [8].

Green times and cycle lengths can sometimes be useful "diagnostic" tools, e.g. to verify whether the control is operating according to expectations.

Vehicle Volume, Volume-to-Capacity Ratio, and Percent on Green

The next type of performance measure that we might consider relates the traffic count to the phase intervals. To accomplish this, we need to obtain a record of vehicle arrival or departure times (depending on what we wish to quantify) and relate it to the phase times to associate each record of vehicle movement with a phase interval.

In field studies, such measures rely on the placement of count detectors at some location relative to the intersection. In most cases, this is a location close to the stop bar. In that situation, the detection times relate to vehicle departures, so the numbers obtained refer to the number of vehicles *served* by the intersection. The alternative is to use a detector at a setback location, which would provide a count of arriving vehicles, which can permit a "percent on green" to be calculated. In the real world, it is nearly impossible to distinguish between different movements for setback detections—that is, we do not know whether a vehicle will continue straight, or turn left or right. In a simulation environment we are not limited in this way and can enjoy getting an arrival count for any movement we desire.

The example below uses a travel time section to obtain the vehicle count for a movement. In this case, a section with the ID 202 is used to identify *through vehicles only* for a particular movement. The arrival times are found by taking the **time** column (representing the record time – meaning the time when the vehicle crosses the exit of the travel time section) and subtracting the **traveltime** column. This relies on the configuration of the travel time section such that the entry occurs somewhat upstream of the intersection and the exit lies beyond the stop bar. Similar data could also be obtained from "Node" data.

The query starts from the red-green-red table, which is a nested query referred to as **q1**. For each row in this table, a series of vehicle counts are accomplished using some nested select statements contained under the top-level select statement. For example, the 2nd select statement indicates that for every row we will count up all applicable rows in the travel time table for which the arrival times occur between the last red and next red times. This gives us the arrivals occurring within each cycle. The 3rd select statement limits this to only those arrivals taking place during green.

```
select
    q1.green,
    (select count(*) from sim.dbo.traveltimes v where
      v.datasetid = 3 and v.ttid = 202
      and (v.time - v.traveltime) > q1.last_red
      and (v.time - v.traveltime) <= q1.next_red) as veh_count,
    (select count(*) from sim.dbo.traveltimes v where
      v.datasetid = 3 and v.ttid = 202
      and (v.time - v.traveltime) > q1.green
      and (v.time - v.traveltime) <= q1.next_red) as green_count
from
    (
    [the red-green-red table]
    ) as q1
group by
    q1.green, q1.last_red, q1.next_red
order by
```

The result of this statement is a count of total arrivals and a separate count of arrivals on green, as shown in Table 2.7-5. This table carries over the start of green time and the time between successive reds from

Table 2.7-4. The time between successive reds is the interval over which the count of total arrivals takes place. For example, the first row in Table 2.7-5 indicates that 14 vehicles arrived in 114 seconds, which would be rate of 14/114 = 0.123 veh/s. Multiplying by 3600 gives an equivalent flow rate of 442 veh/h. The percent on green is calculated by dividing the arrivals on green by the total arrivals. For example, 12/14 = 85.7%. With some additional analysis we could also calculate platoon ratio, arrival type, and other similar metrics. Figure 2.7-13 and Figure 2.7-14 present example calculations of the raw vehicle count and the equivalent flow rates for the example data, while Figure 2.7-15 displays the percent on green.

A detail in this method of calculation is that the arrivals during yellow will be counted as "arrivals on green." It would be possible to make some adjustments either to the cut-off time or to the arrival times to provide a somewhat more "accurate" indicator of the vehicle arrival characteristics: this will require some additional assumptions or analysis to be done and is beyond the scope of the present chapter, at least in this version.

Table 2.7-5: Vehicle counts and derivative measures.

Start of Green	Time Between Successive Reds (s)	Total Arrivals	Arrivals on Green	Flow Rate (veh/h)	Percent on Green
48313.8	114.0	14	12	442	85.7%
48448.3	120.3	11	9	329	81.8%
48564.1	124.9	13	9	374	69.2%
48677.3	107.1	22	21	739	95.4%
48802.4	109.6	12	5	394	41.7%
48892.8	114.4	11	7	346	63.6%
49006.9	109.6	15	13	492	86.7%
49131.8	123.4	13	11	379	84.6%
49234.3	102.7	13	12	455	92.3%
49362.3	114.0	10	7	315	70.0%

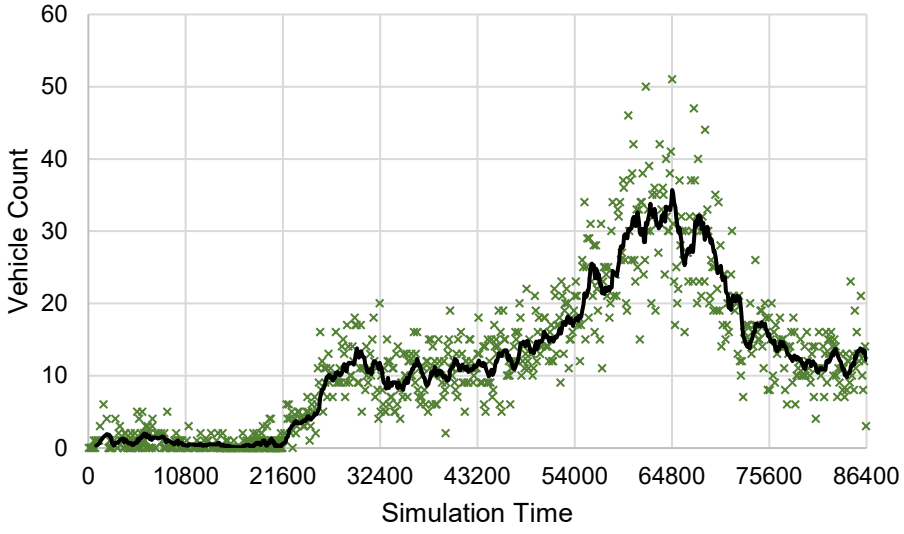


Figure 2.7-13: Vehicle count, with 10-cycle moving average.

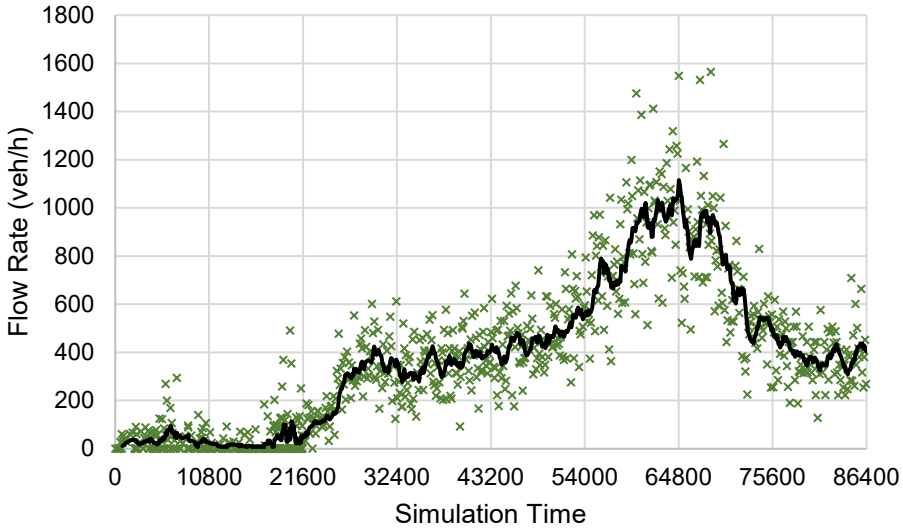


Figure 2.7-14: Flow rate, with 10-cycle moving average.

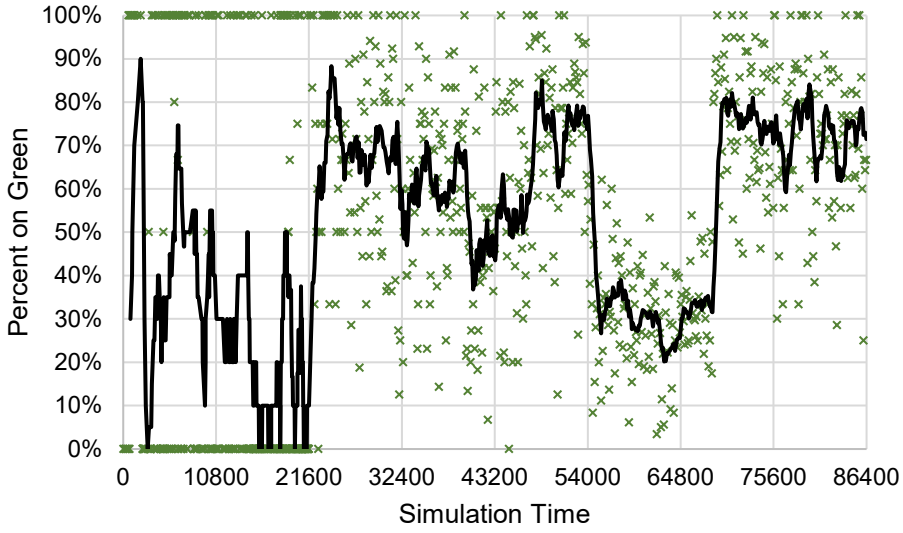


Figure 2.7-15: Percent on Green, with 10-cycle moving average.

There are several other metrics that can be developed from this type of information, such as the volume-to-capacity ratio, which would rely on the selection of an amount of capacity to associate with each unit of green time. In the previous example, for the first cycle shown in Table 2.7-4 and Table 2.7-5, there are 82.7 seconds of green available for the phase. We can assume that *effective green* takes the same value, if we assume that the start-up lost time is equal to the amount of clearance time available for vehicle movement [4]. If there are two lanes of traffic, and the saturation flow rate is assumed to be 1900 veh/h/lane, this means that the capacity would be equal to 1800 / 3600 × 2 = 1.06 veh/s, so in 82.7 seconds, it would be possible to serve 87.3 vehicles. Because there were 14 arrivals in total, the volume-to-capacity ratio would be 14 / 87.3 = 16%.

These methods will provide highly granular metrics on a cycle-by-cycle basis. In general, such detailed metrics are of rather limited use by themselves and can be more useful if aggregated or rearranged. One simple example is a view showing detail by phase, as presented in Figure 2.7-16. This chart shows the volume of vehicles being served by each of eight phases at a single intersection. In this case the relative demand for each individual movement is distinct: we can clearly see that phases 2 and 6 dominate and phase 7 has almost no traffic. In this example, the through/right turn movements have been configured to show through and right turn demands as separate series. It would also be possible to use separate series to display alternative scenarios, e.g. to compare volumes. A variety of potential applications of such metrics have been presented in previous reports on ATSPM [4, 10].

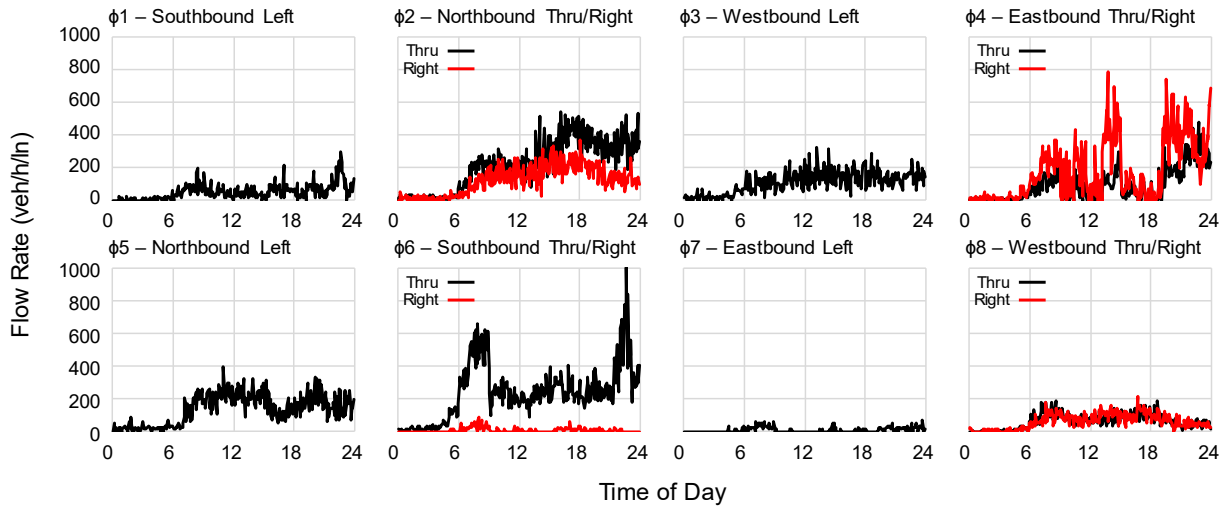


Figure 2.7-16: Volume by movement in eight-phase orientation.

Coordination Diagram

The last performance measure we will discuss in this chapter is what was introduced as the "Purdue Coordination Diagram" or PCD [11]. The coordination diagram may be thought of a disaggregate view of the cyclic flow profile used in tools such as TRANSYT [12] to describe average traffic flows on links between coordinated traffic signals. Rather than binning the arrival times, the diagram displays them individually and overlays a representation of when the signal is green. The visualization permits one to determine at a glance whether the vehicles are formed into platoons, how many platoons appear in a cycle, whether those arrive during green, whether the characteristics change by time of day, and so on. Here we will describe how to construct such a diagram from scratch using a spreadsheet tool.

The base data that is needed to construct a coordination diagram consists of two tables: a table of phase times (the same as the red-green-red events shown in Table 2.7-4), and a separate table of the individual vehicle arrival times. A detail in the development of this arrival table is distinguishing between the *detection time* and the *arrival time*. The location of the dot should reflect when the vehicle is likely to arrive at the traffic signal. Depending on how far away the detection point is, this may be a significant adjustment. For most field locations, setback detectors are located approximately 5 seconds upstream and we would thus add 5 seconds to the arrival times to reflect the state of the signal when the vehicle arrives at the stop bar. Similar adjustments might be made using simulation data, depending on how the arrival times are established.

The central component of a PCD is calculating the *time in cycle* of each vehicle arrival. This is done by first looking up the most recent start of red (LSOR) time, and calculating the difference between the arrival time and the LSOR. Table 2.7-6 shows what this looks like for the first handful of vehicle arrivals that we might obtain for a visualization. It would be possible to establish the LSOR time using a series of lookup functions. For each arrival time, we need to find the most recent LSOR. In a spreadsheet, the VLOOKUP function could be used, taking the arrival time as the reference, while the lookup range would be the list of LSOR times (e.g., the first column in Table 2.7-4).

The first step in creating a PCD is to plot the "time in cycle" (vertical axis) versus the "arrival time" (horizontal axis). Such a chart is shown in Figure 2.7-17. This chart does not yet have any phase information added to it; so far, we have only used the LSOR to calculate time in cycle. The next step will be to overlay the green state onto the diagram. For this, we need to refer back to the red-green-red table (Table 2.7-4) and use the phase times to identify the time in cycle for which the start and end of green occur. These are the same as the start of green (SOG) time and the time between successive reds (TBSR) in Table 2.7-4. Thus, we already have what we need to produce a coordination diagram.

The only challenge we have is in deciding how we will implement the layering of the green time onto the vehicle arrivals. Many views of the PCDs, including that in the open-source software, use red and green lines alone. An alternative and perhaps simpler view is to provide a green shaded region. In a spreadsheet, this can be accomplished by using a series of error bars. The anchor point for each error bar is determined from:

- Horizontal axis: the *midpoint* of each cycle (the LSOR + TBSR/2)
- Vertical axis: the SOG time

while the height of the error bar is the duration of the green. To set this up in a spreadsheet, we would first add a new series defined according to the above horizontal and vertical values, and then add vertical error bars. Those error bars should *only* be displayed for the positive value, and the lengths of the error bars should be set to the green durations. Be sure to then hide the symbols that were generated marking the anchor points. Once this has been done, the chart should resemble the PCD shown in Figure 2.7-18. The visualization is not completely perfect; since we are looking at many cycles, potential gaps or areas of overlap are hidden. In particular, the width of the error bars should be adjusted to provide appropriate coverage; if there are huge variations in cycle length, then it may be difficult to avoid gaps between adjacent error bars. If desired, it would be possible to add red and green lines on top of this, as shown in Figure 2.7-19.

Table 2.7-6: First few rows in a vehicle arrival table.

Arrival Time	LSOR Time	Arrival Time	LSOR Time	Time In Cycle
748.2	745.8	00:12:28.2	00:12:25.8	2.4
818.2	772.7	00:13:38.2	00:12:52.7	45.5
891.9	832.9	00:14:51.9	00:13:52.9	59
1038	996.9	00:17:18.0	00:16:36.9	41.1
1095.5	1065.8	00:18:15.5	00:17:45.8	29.7
1273.5	1176.6	00:21:13.5	00:19:36.6	96.9
1384.1	1278.9	00:23:04.1	00:21:18.9	105.2

	1384.9	1278.9	00:23:04.9	00:21:18.9	106
	1459.9	1278.9	00:24:19.9	00:21:18.9	181

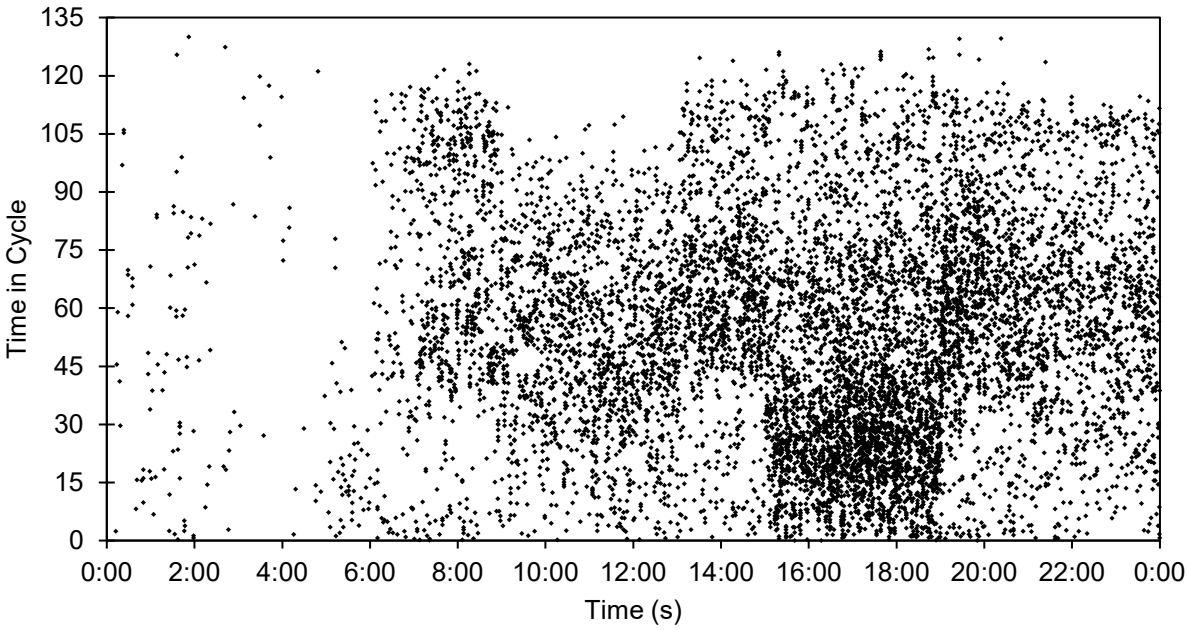


Figure 2.7-17: Time in cycle of individual vehicle arrivals.

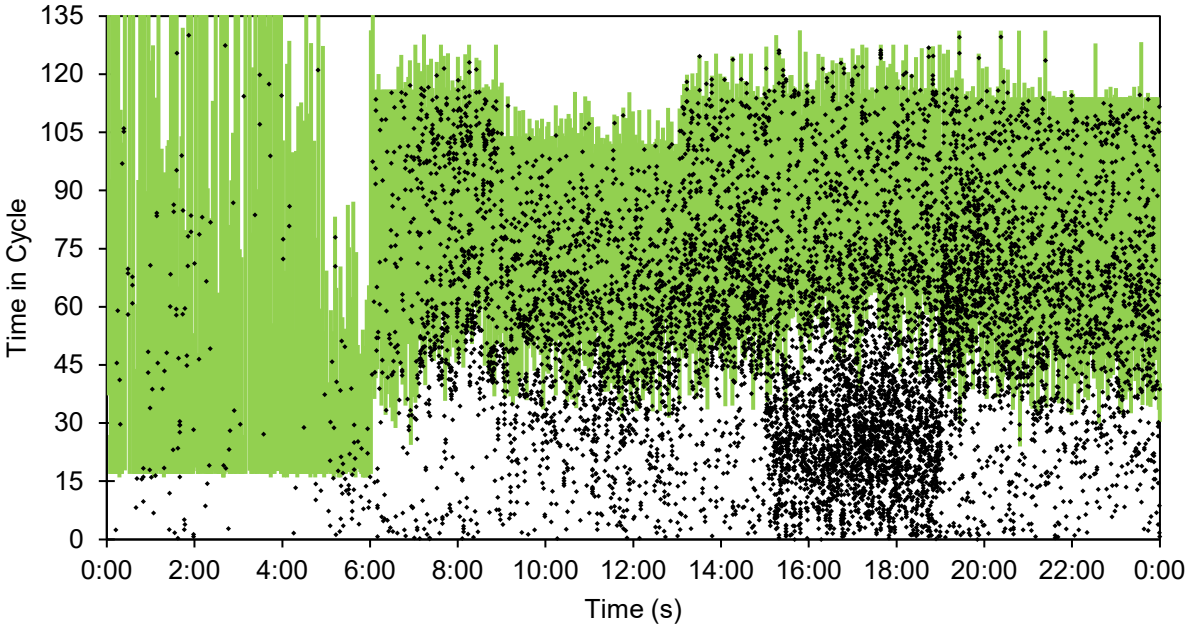


Figure 2.7-18: Coordination diagram using green shading to display when the phase is green.

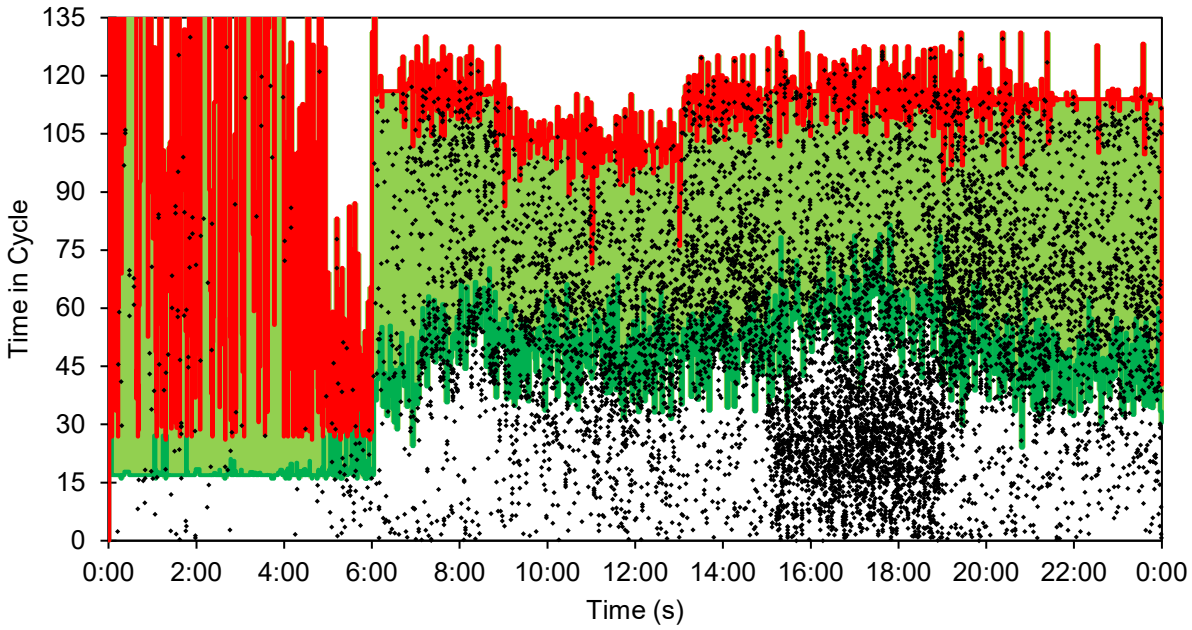


Figure 2.7-19: Coordination diagram including the red and green lines.

2.7.6 References

3. Denney, R.W. *Improving Traffic Signal Management and Operations: A Basic Service Model.* Report FHWA-HOP-09-055, Federal Highway Administration, 2009.

4. Day, C.M., D.M. Bullock, H. Li, S.M. Remias, A.M. Hainen, R.S. Freije, A.L. Stevens, J.R. Sturdevant, and T.M. Brennan. *Performance Measures for Traffic Signal Systems: An Outcome-Oriented Approach.* West Lafayette, Indiana: Purdue University (2014). http://dx.doi.org/10.5703/1288284315333

5. Utah Department of Transportation. "UDOT Automated Traffic Signal Performance Measures." Available online at https://udottraffic.utah.gov/ATSPM

6. Li, H., A.M. Hainen, J.R. Sturdevant, et al. *Indiana Traffic Signal Hi Resolution Data Logger Enumerations.* Indiana Department of Transportation and Purdue University, 2019. https://doi.org/10.5703/1288284316998

7. Day, C.M. and D.M. Bullock. "Investigation of self-organizing traffic signal control with graphical signal performance measures." Transportation Research Record No. 2620, 69-82, 2017.

8. Day, C.M., E.J. Smaglik, D.M. Bullock, and J.R. Sturdevant. "Quantitative evaluation of fully actuated versus non-actuated coordinated phases." Transportation Research Record No. 2080, 8-21, 2008.

9. Day, C.M. and D.M. Bullock. "Cycle length strategies for a diverging diamond interchange in a signalized arterial." Journal of Transportation Engineering, Vol. 142, 04016067, 2016.

10. Day, C.M., D.M. Bullock, H. Li, S.M. Lavrenz, W.B. Smith, and J.R. Sturdevant. *Integrating Traffic Signal Performance Measures into Agency Business Processes.* West Lafayette, Indiana: Purdue University, 2016. http://dx.doi.org/10.5703/1288284316063

11. Day, C.M., R. Haseman, H. Premachandra, T.M. Brennan, J.S. Wasson, J.R. Sturdevant, and D.M. Bullock. "Evaluation of arterial signal coordination: methodologies for visualizing high-resolution event data and measuring travel time." Transportation Research Record No. 2192, 37-49, 2010.

12. Robertson, D.I. *Transyt: A Traffic Network Study Tool*. Report No. LR 253. Road Research Laboratory, Crowthorne, Berkshire, England, 1969.

3. Non-signalized Intersection Control

(To be determined)

4. Intersection Control with Connected and Automated Vehicles (CAVs)

4.1 Control for Homogenous CAV Fleet

4.1.2 A Simulation Platform for Connected Vehicle Based Traffic Signal Control

Contributor(s): Y. Feng, G. Wu

4.1.2.1 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).

4.1.2.2 Platform Overview

Figure 4.1.2-1 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.

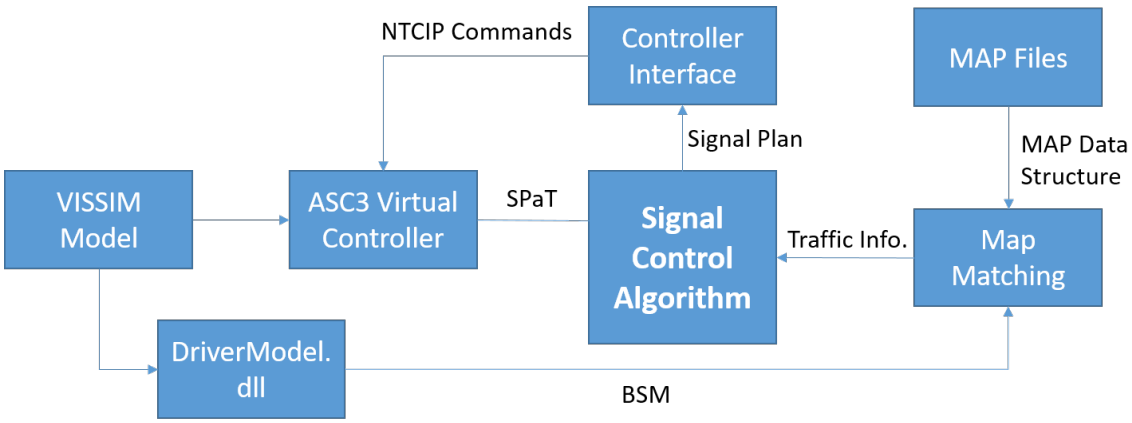


Figure 4.1.2-1: Platform Overview

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.

4.1.2.3 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° longitudes. 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.

4.1.2.4 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.

4.1.2.5 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 4.1.2-2 (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

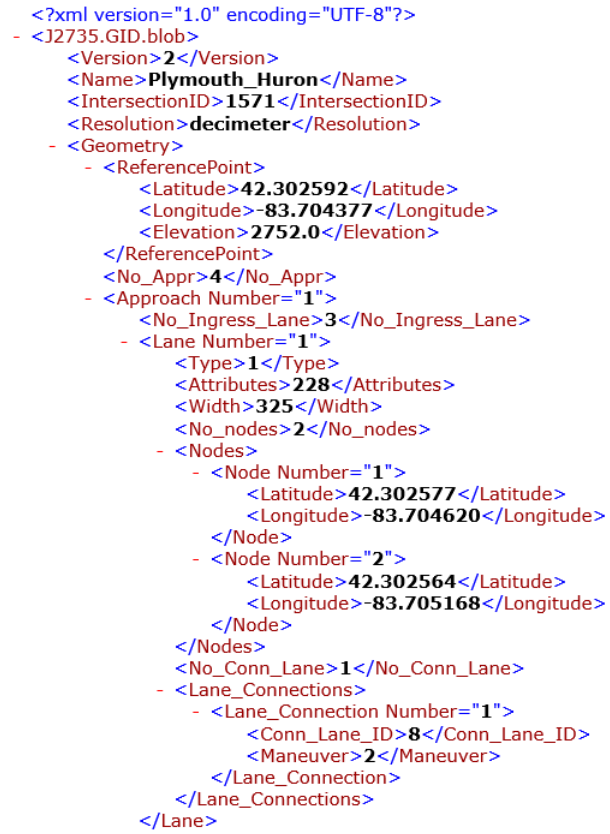


Figure 4.1.2-2 Map Description File

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.

4.1.2.6 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 require 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

4.1.3 Eco-Driving/Eco-signal Control

4.1.3.1 Introduction

The uninterrupted growth in transportation activities, for both people and goods movement, have been exerting significant pressure on our socio-economics and environment. Over the years, it has been consistently reported that transportation-related activities contribute around one third of total energy consumption and/or greenhouse gas (GHG) emissions. On the other hand, emerging technologies such as connected and automated vehicles (CAVs), are deemed as effective solutions to these energy and environmental problems. Representative examples of environmentally-friendly CAV applications include *Dynamic Eco-Driving* and *Eco-Signal Control*. In particular, the *Eco-Approach and Departure* (EAD) at signalized intersections, a flagship Eco-Driving application, has shown significant promise. In this system, an equipped vehicle can take advantage of the signal phase and timing (SPaT) and geometric intersection description (GID) information from the upcoming signalized intersection and calculate the optimal speed to pass on a green light or to decelerate to a stop in the eco-friendliest manner [1]. Speed recommendations may be provided to the driver using a driver-vehicle-interface (DVI) or to the vehicle systems that support automated longitudinal control capabilities.

A microscopic traffic simulator, such as PTV VISSIM [2], is deemed as a cost-effective tool to model and evaluate the emerging transportation technologies and services, including the EAD application. This chapter aims to provide tutorial for coding the External Driver Model DLL interface in PTV VISSIM for implementing the EAD at signalized intersections application. A case study is presented where sample codes for a proposed trajectory planning algorithm and U.S. EPA's MOVES (*Motor Vehicle Emission Simulator*) model [3] are presented. This tutorial is drafted based on Microsoft Windows 10 Pro, PTV VISSIM 9.0, and Microsoft Visual Studio 2017 Pro Version.

4.1.3.2 System Architecture and Key Components

The PTV VIISSIM External Driver Model DLL interface has been well described in previous sections. In this section, the system architecture for modeling the Eco-Approach and Departure (EAD) application is detailed, along with the description of key components.

4.1.3.2.1 System Architecture

Figure 4.1.3-1 illustrates the system architecture for modeling the EAD application in PTV VISSIM, including: 1) the Simulation Engine where by-default car following model, lane changing model and others are embedded; 2) Application Programming Interface (API) which supports the external Driver Model DLL, Signal Control DLL, Emission DLL, and other user-customized DLLs to evaluate system performance (e.g., VMT, VHT, queue length or other system-wide mobility metrics); 3) Eco-Approach and Departure algorithm which takes the dynamic information from the simulation and provides the longitudinal control signals (e.g., desired acceleration) to the equipped vehicles; 4) the MOVES model estimates the fuel consumption and pollutant emissions at different resolution, based on vehicle type, position (associated with the road grade if any), speed, and acceleration/deceleration; 5) the Surrogate Safety Assessment Model (SSAM) utilizes the .trj files output from VISSIM to predict the safety performance, such as time-to-collisions and post-encroachment time [4].

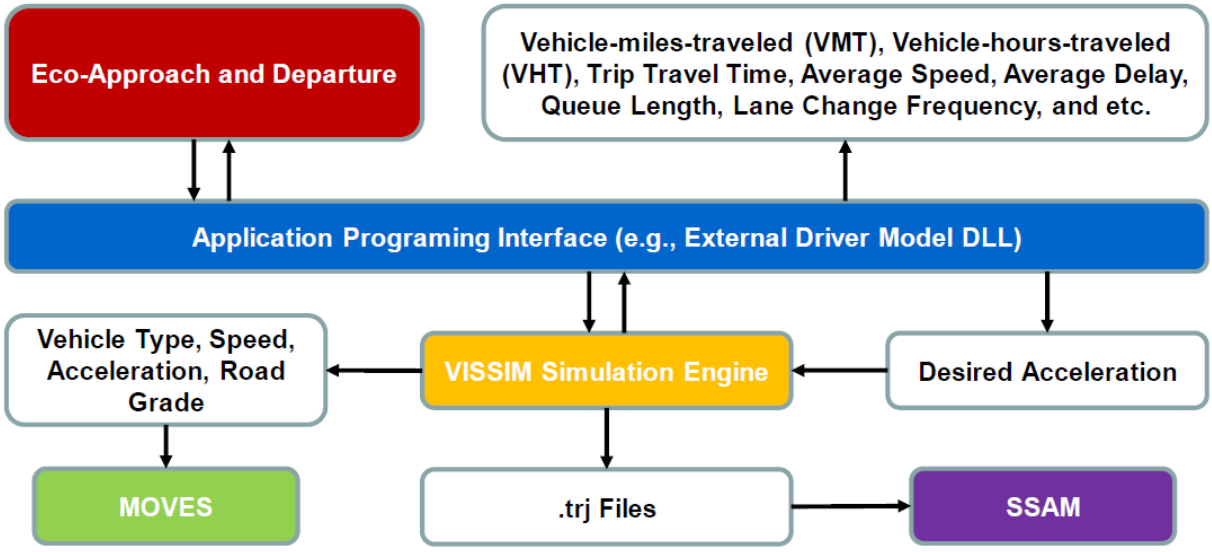


Figure 4.1.3-1: System architecture for modeling the EAD application in PTV VISSIM

4.1.3.2.2 Eco-Approach and Departure Module

As a dynamic eco-driving system for urban arterial travel, the Eco-Approach and Departure (EAD) module aims to perform appropriate longitudinal maneuvers (by either manually driven or automatic control) for a vehicle to minimize the fuel (or energy) consumption along the entire trip, which is mainly determined by the propulsion power, a function of vehicle mass, powertrain characteristics, tire feature, frontal area, vehicle speed, acceleration/deceleration, roadway grade, road surface, and meteorological condition (e.g., wind speed, wind direction). In particular, the vehicle speed at any point in time is constrained by many factors, such as the maximum engine power, upcoming traffic signal status, and downstream traffic conditions. Therefore, the environmentally friendly vehicle longitudinal control problem (in continuous form) is in essence a constrained nonlinear programming (CONLP), i.e.,

$$minx(t) \int_0^T \mathrm{f(x(t), u(t); p)}$$

subject to

$$\mathrm{C(x(t), u(t); p)} \leq 0$$

where $f(\cdot)$ is the fuel or energy consumption rate; T is the expected trip time; $x(t)$ represents the real value decision vector (e.g., speed); $u(t)$ denotes other time-varying variables, such as signal phase and timing (SPaT) information and the preceding vehicle's speed; and p denotes the deterministic variables (e.g., network topology, fuel type, and the vehicle's frontal area).

Although numerous EAD-like algorithms have been proposed over the past decade, this document adopts the piecewise linear-trigonometric function based EAD algorithm [5] (developed by the researchers from University of California at Riverside) as an illustration example, whose flowchart is presented in Figure 2. Basically speaking, this algorithm first estimates the amount of time available to reach the intersection during the current or next green phase, upon receiving the SPaT information from the upcoming traffic signals. It then calculates a recommended cruise velocity

for the equipped vehicle given the constraints of roadway speed limit, maximum acceleration or jerk, and downstream traffic conditions (if available).
Within the piecewise linear-trigonometric function family, the algorithm can find the most fuel-efficient speed profile (containing both acceleration/deceleration portion and cruising portion) for the equipped vehicle to travel through the signalized intersection. In traffic simulation, the algorithm will keep updating the desired acceleration/deceleration at each time step by using the latest information on the vehicle location, speed and SPaT, while ensuring the safety performance (i.e., collision avoidance). Please refer to [5] for more details if interested.

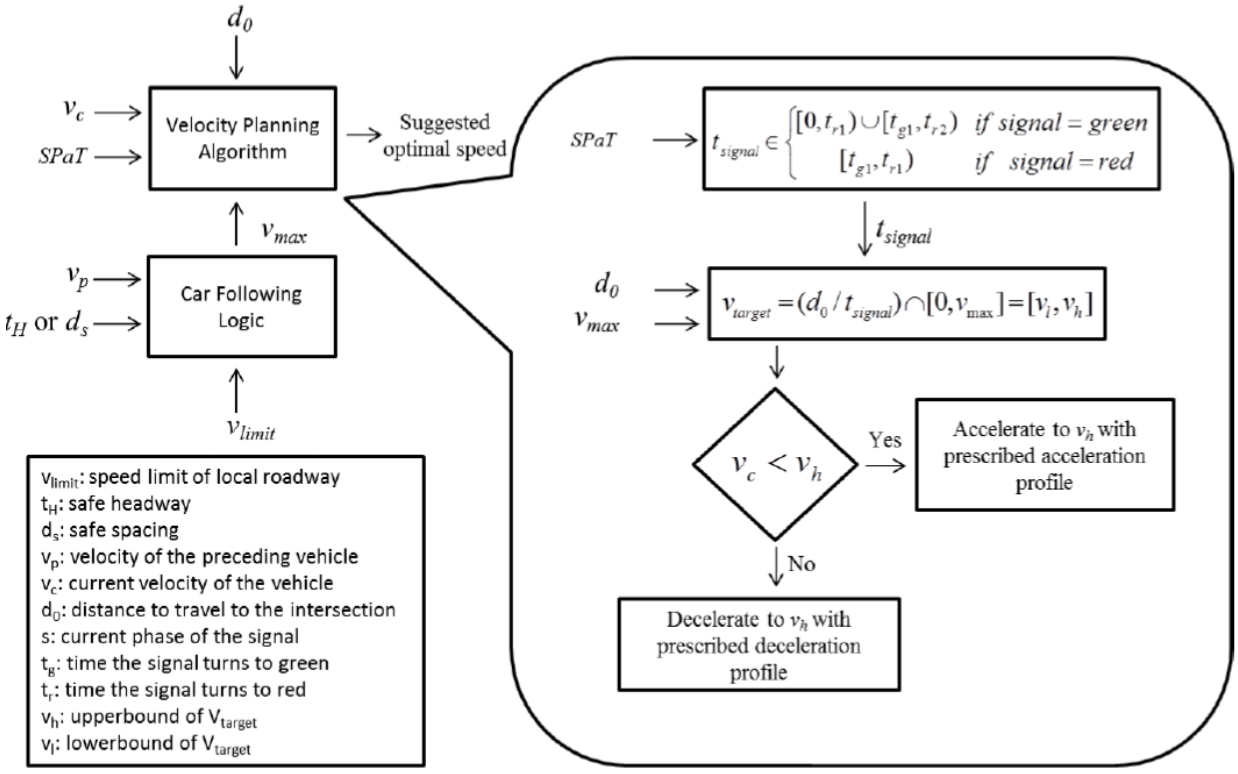


Figure 4.1.3-2: Flowchart of trigonometric speed profile-based EAD algorithm [6].

4.1.3.2.3 MOVES Module

Over the past years, U.S. EPA has developed MOVES (Motor Vehicle Emission Simulator), a state-of-the-science emission modeling system that estimates emissions for mobile sources at the national, county, and project level for criteria air pollutants, greenhouse gases, and air toxics [3]. However, the model is comprehensive enough to be not suitable for on-line interaction with microscopic traffic simulation (i.e., due to heavy computational loads). Therefore, the researchers in UC Riverside has developed an alternative approach to simplify the application of MOVES to simulation while keeping reasonable fidelity of original MOVES model.

Figure 4.1.3-3 depicts the proposed work flow of MOVES plug-in development for PTV VISSIM. Basically speaking, there are two major steps (sub-flows):

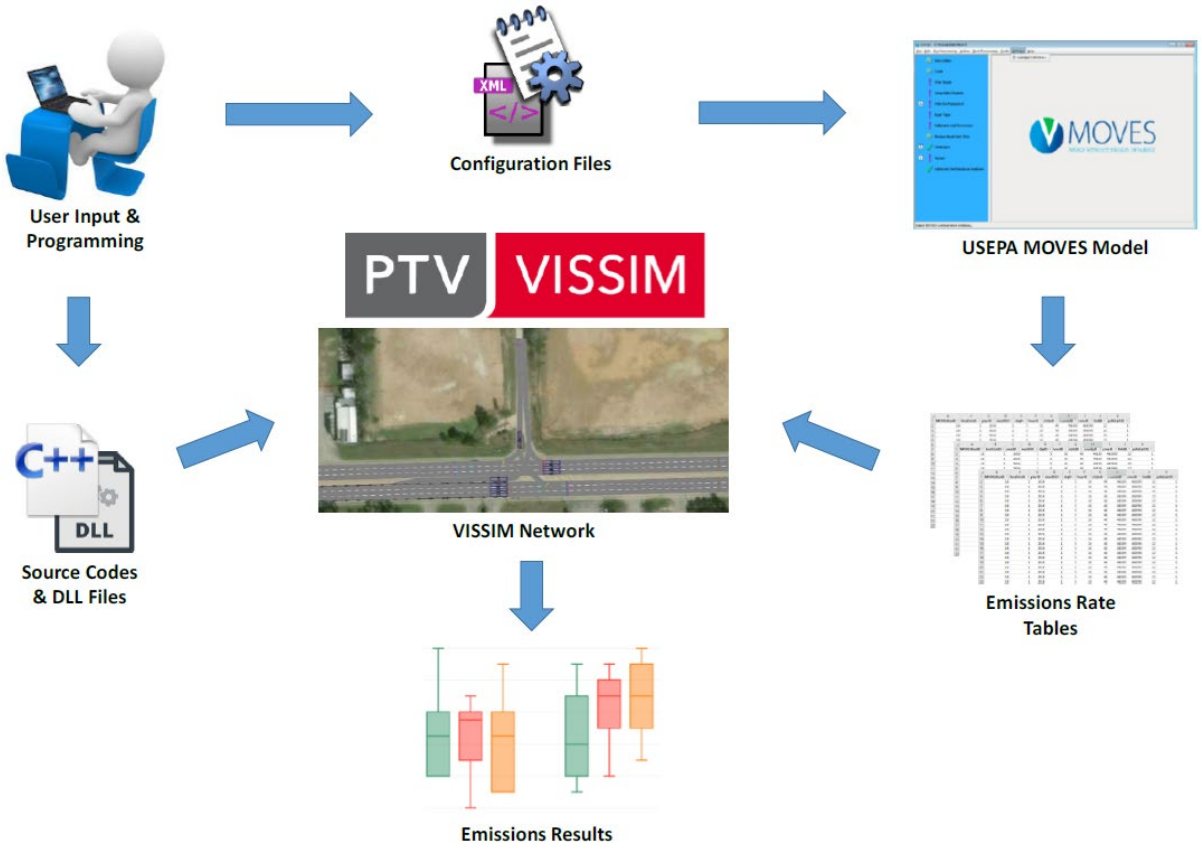


Figure 4.1.3-3: Work flow of MOVES plug-in development for PTV VISSIM.

1. **Acquire Emission Rate Tables from MOVES**. To retrieve the customized (for specific projects) emission rate tables (from MOVES) for microscopic simulation, the user need to first install the MOVES model (e.g., MOVES2014a). After the installation, the user is required to input the network model information, such as geographic region (e.g., Santa Clara County in California), calendar month and year[3] (e.g., July 2005). This configuration will be coded into settings files (e.g., in the XML format) and will be link to the MOVES database to obtain the data files (e.g., in the Excel format) on vehicle population/activity, fuel type/engine technology, vehicle inspection/maintenance program and meteorological statistics. Once all the input data files are ready, MOVES can be executed and output emission rate tables for different source types (e.g., passenger car, transit bus) defined by U.S. EPA, taking into account various factors, such as vehicle model year distribution, fuel type/engine technology market share, and temperature and/or humidity adjustment. All these emission rate tables will be coded as the configuration files for the microscopic simulation tool.

[3] The calendar year could be some time in the future, since the MOVES database also contains the projected vehicle fleet's information.

2. **Develop Code to Calculate Operating Mode (OpMode) in Simulation.** In the microscopic simulation, application programming interfaces (APIs) need to be developed to access the second-by-second vehicle trajectories (including both speed and acceleration) and road grade information. With these activity data for each vehicle as well as the information on vehicle class and weight, the vehicle specific power[4] (VSP) characteristics (in kWatt/tonne) can be calculated using the following formula.

$$VSP = \left(A/M\right) \cdot v + \left(B/M\right) \cdot v^2 + \left(C/M\right) \cdot v^3 + (a + g \cdot sin\theta) \cdot v$$

where A, B and C are the road-load related coefficients for rolling resistance ($kW{\cdot}sec/m$), rotating resistance ($kW{\cdot}sec2/m2$) and aerodynamic drag ($kW{\cdot}sec3/m3$), respectively; v is the vehicle speed (m/sec); M is the mass of vehicle (metric ton); g is the acceleration due to gravity (9.8 $m/sec2$); a is the vehicle acceleration (meter/sec2); and θ is the (fractional) road grade. Default values of these parameters are provided in Table 1. After the VSP values are calculated, they are binned according to the MOVES' vehicle operating mode (OpMode) bin definition given in Figure 4.1.3-4**.** *Vehicle operating mode bin definition in MOVES [7]*. With the emission rate tables coded in the configuration files (Step 1), the energy consumption and pollutant emissions can be estimated in a disaggregate (e.g., second-by-second for each vehicle) or aggregate (in both space and time) manner.

Table 4.1.3-1: Recommended Coefficients for Each Source Type Defined in MOVES [7]

Source TypeID	HPMS Vtype ID	SourceType Name	Rolling TermA (kW-s/m)	Rotating TermB (kW-s^2/m^2)	Drag TermC (kW-s^3/m^3)	Source Mass (metric tons)	FixedMass Factor (metric tons)
11	10	Motorcycle	0.0251	0	0.000315	0.285	0.285
21	20	Passenger Car	0.156461	0.002002	0.000493	1.4788	1.4788
31	30	Passenger Truck	0.22112	0.002838	0.000698	1.86686	1.86686
32	30	Light Commercial Truck	0.235008	0.003039	0.000748	2.05979	2.05979
41	40	Intercity Bus	1.29515	0	0.003715	19.5937	17.1
42	40	Transit Bus	1.0944	0	0.003587	16.556	17.1
43	40	School Bus	0.746718	0	0.002176	9.06989	17.1
51	50	Refuse Truck	1.41705	0	0.003572	20.6845	17.1
52	50	Single Unit Short-haul Truck	0.561933	0	0.001603	7.64159	17.1
53	50	Single Unit Long-haul Truck	0.498699	0	0.001474	6.25047	17.1
54	50	Motor Home	0.617371	0	0.002105	6.73483	17.1
61	60	Combination Short-haul Truck	1.96354	0	0.004031	29.3275	17.1
62	60	Combination Long-haul Truck	2.08126	0	0.004188	31.4038	17.1

[4] For medium- and heavy-duty vehicles, the Scaled Tractive Power (STP) concept will be used instead of VSP where a fixed mass factor, f, is adopted.

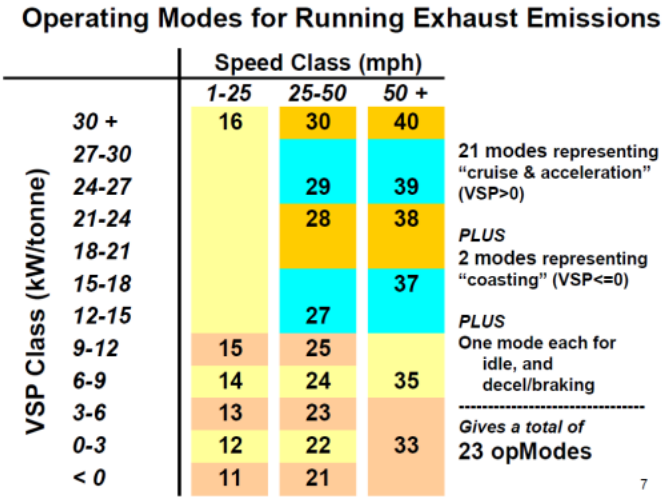


Figure 4.1.3-4: Vehicle operating mode bin definition in MOVES [7].

4.1.3.3 Data

4.1.3.3.1 Data Overview

The generic data that flow into and out of PTV VISSIM have been well described in previous sections, which may include: 1) geometric features for simulation network construction; 2) pedestrian and traffic data such as volume, turning ratio, vehicle type mix, and technology market penetration rate; 3) information related to traffic control device, e.g., stop sign, signal phase and timing, roundabout; and 4) system performance metric that can be used to evaluate the safety, mobility, and environmental sustainability.

4.1.3.3.2 Energy/Emission Data (Optional)

As to the environment-focused application, energy and emission data may be preferable to better calibrate (or customize) the energy or emission estimation model beyond the MOVES model.

4.1.3.4 Simulation Setup

4.1.3.4.1 Network Coding

The basic element of a road network in PTV VISSIM is *link*. Links can run in one direction over one or more lanes, and are connected via connectors to construct a network. The traffic can only flow via connectors from one link to another. For public transportation, a line network needs to be created by using links and connectors. Once the network is constructed, other required network objects (see Figure 4.1.3-5) can be added and their associated attributes can be defined. For more detailed information on how to create or edit a PTV VISSIM network, please refer to Chapter 6 in the manual.

Icon	Network object type
	Links and Connectors (see "Modeling links for vehicles and pedestrians" on page 356), (see "Modeling connectors" on page 369)
	Desired Speed Decisions (see "Modeling links for vehicles and pedestrians" on page 356)
	Reduced Speed Areas (see "Using reduced speed areas to modify desired speed" on page 381)
	Conflict Areas (see "Modeling conflict areas" on page 490)
	Priority Rules (see "Modeling priority rules" on page 472)
	Stop Signs (see "Modeling stop signs and toll counters" on page 501)
	Signal Heads (see "Modeling signal groups and signal heads" on page 509)
	Detectors (see "Using detectors" on page 526)
	Vehicle Inputs (see "Modeling vehicle inputs for private transportation" on page 399)
	Vehicle Routes (see "Modeling vehicle routes, partial vehicle routes, and routing decisions" on page 403)
	Parking Lots (see "Modeling parking lots" on page 431)
	Public Transport Stops (see "Modeling PT stops" on page 445)
	Public Transport Lines (see "Modeling PT lines" on page 451)
	Nodes (see "Modeling nodes" on page 619)
	Data Collection Points (see "Defining data collection points" on page 391)
	Vehicle Travel Times (see "Defining vehicle travel time measurement" on page 392)
	Queue Counters (see "Modeling queue counters" on page 395)
	Sections (see "Modeling sections" on page 604)
	Background Images (see "Inserting a background image" on page 345)
	Pavement Markings (see "Modeling pavement markings" on page 389)
	3D Traffic Signals (see "Modeling 3D signal heads" on page 514)
	Static 3D Models (see "Using static 3D models" on page 601)
	Vehicles in the network are the result of simulation and cannot be inserted as network

Figure 4.1.3-5: The list of network objects defined in PTV VISSIM [2].

4.1.3.4.2 Enabling SPaT Information Delivery

The coding of signalized intersections is critical for the EAD application scenario. As described in previous sections, PTV VISSIM provides a very flexible interface for various signal control procedures (from fixed time control to more advanced adaptive signal control). No matter which signal control strategy is selected, a key question for the EAD application is how to deliver the signal phase and timing (SPaT) information to the equipped vehicle. If the signal is running fixed time control, then the future SPaT can be well determined with the preview knowledge of cycle length, phase sequence and duration. It would be very simple to get the equipped vehicle aware of the upcoming SPaT even by hard coding. For the traffic responsive signal control (e.g., with the Econolite ASC/3 add-on module), SPaT information broadcasting function needs to be activated and some communication module (e.g., socket) should be set up (via APIs) to enable the access of such information by the equipped vehicle in the traffic simulation environment.

4.1.3.4.3 Vehicle Type Definition

To differentiate the EAD-equipped vehicles from the legacy vehicles in PTV VISSIM, different vehicle types (e.g., legacy vehicle vs. eco-vehicle) should be defined. As described in previous section, to activate the external driver model for the EAD equipped vehicles, the DLL file for modeling the eco-driving behavior should be linked with the right vehicle type (see Figure 4.1.3-6). In addition, to evaluate the system performance under different market penetration rate (MPR) of eco-friendly technologies, the ratio between legacy vehicle and eco-vehicle should be changed accordingly.

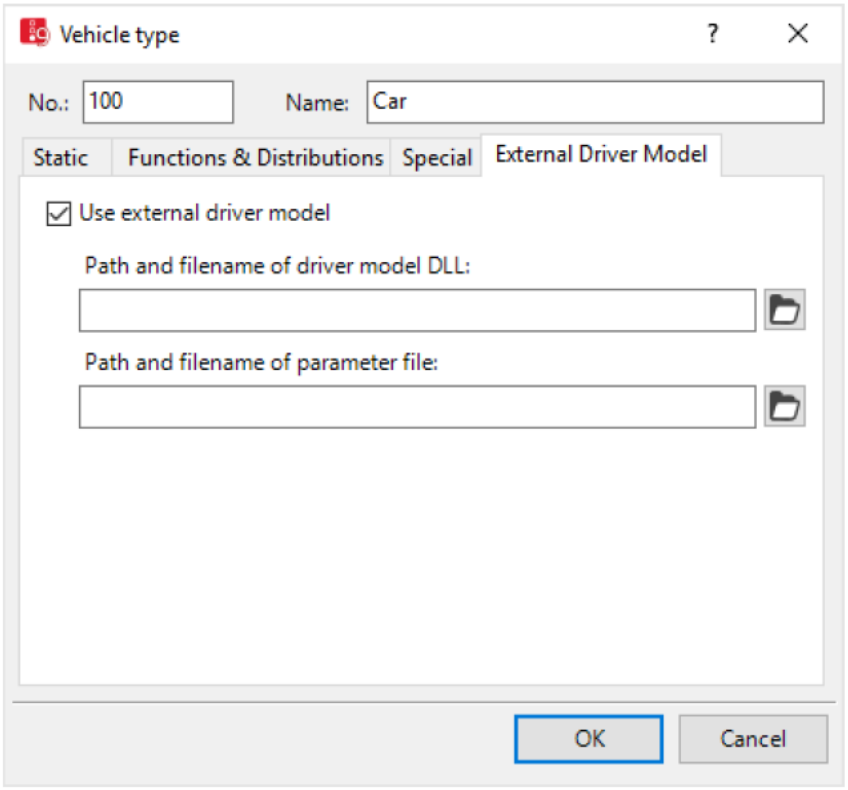


Figure 4.1.3-6: Screenshot for opening the "Vehicle Type" window in PTV VISSIM [2].

4.1.3.5 Simulation Calibration

4.1.3.5.1 Vehicle Type Definition

The purpose of simulation calibration is to replicate the real-world traffic characteristics in the simulation network by adjusting various parameters to achieve (both qualitative and quantitative) consistency between the simulation and the field data. In the whole process of model calibration, a key step is to estimate the existing origin-destination patterns or O-D tables. The fundamental problem of estimating O-D tables from traffic counts is that there are an infinite number of solutions (in most cases) in theory that can match the given set of observations, because the O-D tables usually have more degrees of freedom than the number of links to constrain them. One potential solution is to apply the gravity model that is used in regional transportation demand estimation to obtain the "seed" O-D tables, and to employ the Furness process and iterative

approach to converge to desired O-D tables [8]. Figure 4.1.3-7 lists a set of suggested criteria for microscopic simulation network calibration.

Criteria & Measures	Acceptability Targets
Hourly Flows, Model vs. Observed	
Individual Link Flows	
Within 15%, for 700 vph < Flow <2700 vph	> 85% of cases
Within 100 vph, for Flow < 700 vph	> 85% of cases
Within 400 vph, for Flow > 2700 vph	> 85% of cases
Total Link Flows	
Within 5%	All Accepting Links
GEH Statistic – Individual Link Flows	
GEH < 5	> 85% of cases
GEH Statistic – Total Link Flows	
GEH < 4	All Accepting Links
Travel Times, Model vs. Observed	
Journey Times Network	
Within 15% (or one minute, if higher)	> 85% of cases
Visual Audits	
Individual Link Speeds	
Visually acceptable Speed-Flow relationship	To analyst's satisfaction
Bottlenecks	
Visually acceptable Queuing	To analyst's satisfaction

Figure 7. A set of recommended criteria for simulation network model calibration [8].

The *GEH Statistic* is a formula used in traffic engineering to compare two set of traffic volumes and more specifically.

$$\text{GEH} = \frac{\sqrt{(M-C)^2}}{\sqrt{(M+C)/2}}$$

where M is the estimated directional hourly volume at a location (output from the simulation); and C is the directional hourly count at the same location (observed from the field).

4.1.3.5.2 Speed-Acceleration Profile

Besides the generic data, information about the vehicle dynamics such as speed-acceleration profile is preferable for better calibrating the dynamics model for the equipped vehicle as well as other traffic, in order to obtain results with higher fidelity. In PTV VISSIM, for each vehicle type, the associated acceleration or deceleration (maximum, desired, and minimum) can be defined as a function of speed. With the appropriate settings, speed-acceleration or speed-deceleration profiles output from the simulation model can be calibrated against data collected from the real-world (e.g., via OBD or GPS), as shown in Figure 4.1.3-8.

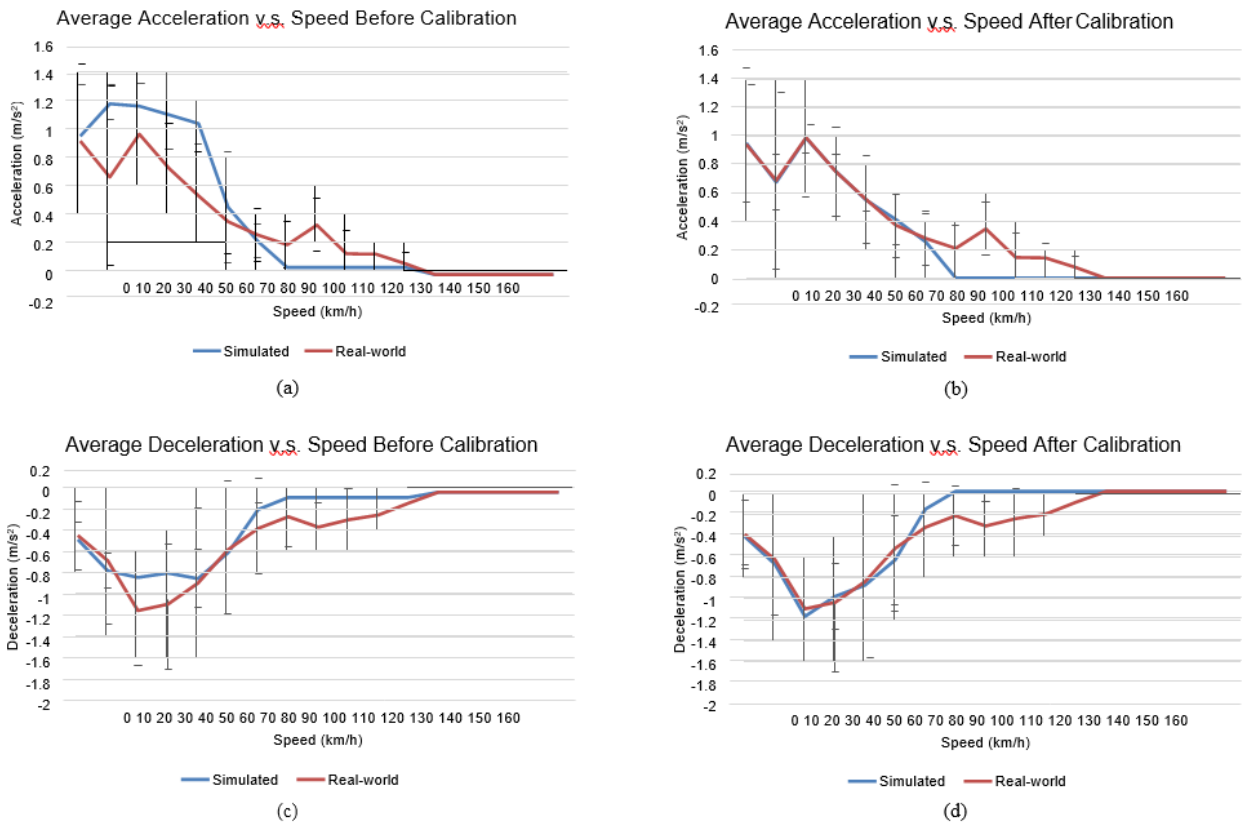


Figure 4.1.3-8: An example of speed-acceleration/deceleration profile calibration in PTV VISSIM [9].

4.1.3.6 Appendix – Sample Codes for MOVES Model Development

As aforementioned, the development of API for MOVES model requires: 1) the acquisition of emission rates from MOVES; and 2) the calculation of operating mode (OpMode) for each vehicle at each second. The first step needs the running of MOVES model offline, while the second step needs some coding efforts with API (e.g., some modifications on the EmissionModel.DLL). In the following, some sample codes for calculating OpMode are presented for reference.

```
#include <vector>
#include <cmath>
using namespace std;

//Calculate VSP function
double calculateVSP(long type, double velocity, double acceleration, double slope)
{
	vector<double> vehicleAttributes;
	if (type == 100)			// Passenger Car whose SourceTypeID is 21
	{
		vehicleAttributes = { 0.156461, 0.002002, 0.000493, 1.4788, 1.4788 };
	}
	else if (type == 200)	// Transit Bus whose SourceTypeID is 42
	{
		vehicleAttributes = { 1.0944, 0.0, 0.003587, 16.556, 17.1 };
```

```
	}


	else if (type == 300) // Combination Short-haul Truck whose SourceTypeID is 62
	{
	vehicleAttributes = { 1.96354, 0.0, 0.004031, 29.3275, 17.1 };
	}
	double A = vehicleAttributes.at(0); //Rolling (kW-s/m)
	double B = vehicleAttributes.at(1); //Rotating (Kw-s**2/m**2)
	double C = vehicleAttributes.at(2); //Drag (kW-s**3/m**3)
	double M = vehicleAttributes.at(3); //Source Mass (metric tons)
	double F = vehicleAttributes.at(4); //Fixed Mass Factor (metric tons)
	double VSP = ((A * velocity) + (B * pow(velocity, 2)) + (C * pow(velocity, 3)) +
		M * (acceleration + ( 9.8 * sin(slope) ) )* velocity) / F;
	return VSP;
}
//Determine and return the OpMode
int getOpmode(vector<double> accelerations, double velocity, double VSP)
{
	int OpMode = -99999;
	if (accelerations.size() > 0)
	{
		if ((accelerations.at(accelerations.size() - 1)*2.23694) <= -2.0)
		{
			OpMode = 0;
			return OpMode;
		}
	}
	if (accelerations.size() == 3)
	{
		if ((accelerations.at(2)*2.23694) < -1.0 && (accelerations.at(1)*2.23694) < -1.0
	&& (accelerations.at(0)*2.23694) < -1.0)
		{
			OpMode = 0;
			return OpMode;
		}
	}
	if (-1.0 <= (velocity*2.23694) && (velocity*2.23694) < 1.0)
	{
		OpMode = 1;
	}
	else if ((velocity*2.23694) >= 50)
	{
		if (VSP >= 30)
		{
		OpMode = 40;
		}
```

```
        else if (VSP >= 24)
        {
        OpMode = 39;
        }
        else if (VSP >= 18)

        {

                OpMode = 38;
        }
        else if (VSP >= 12)
        {
                OpMode = 37;
        }
        else if (VSP >= 6)
        {
                OpMode = 35;
        }
        else
        {
                OpMode = 33;
        }
}
else if ((velocity*2.23694) >= 25)
{
        if (VSP >= 30)
        {
        OpMode = 30;
        }
        else if (VSP >= 24)
        {
        OpMode = 29;
        }
        else if (VSP >= 18)
        {
        OpMode = 28;
        }
        else if (VSP >= 12)
        {
        OpMode = 27;
        }
        else if (VSP >= 9)
        {
        OpMode = 25;
        }
        else if (VSP >= 6)
```

```
{
OpMode = 24;
}
else if (VSP >= 3)
{
OpMode = 23;
}
else if (VSP >= 0)
{
OpMode = 22;
}
else
{
OpMode = 21;
}
}
else if ((velocity*2.23694) >= 0)
{
if (VSP >= 12)
{
OpMode = 16;
}
else if (VSP >= 9)
{
OpMode = 15;
}
else if (VSP >= 6)
{
OpMode = 14;
}
else if (VSP >= 3)
{
OpMode = 13;
}
else if (VSP >= 0)
{
OpMode = 12;
}
else
{
OpMode = 11;
}
}
return OpMode;
};
```

4.1.3.7 Reference

[1] M. Li, K. Boriboonsomsin, G. Wu, W. Zhang, and M. Barth. *Traffic Energy and Emission Reductions at Signalized Intersections: A Study of the Benefits of Advanced Driver Information*. International Journal of ITS Research, 2009

[2] PTV Group. *PTV VISSIM*. http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/

[3] U.S. Environmental Protection Agency. *MOtor Vehicle Emission Simulator (MOVES)*. https://www.epa.gov/moves

[4] U.S. Department of Transportation. *Surrogate Safety Assessment Model (SSAM)*. https://www.fhwa.dot.gov/publications/research/safety/08049/

[5] O. Altan, G. Wu, M. Barth, K. Boriboonsomsin, and J. Stark. *GlidePath: Eco-Friendly Automated Approach and Departure at Signalized Intersections*. IEEE Transactions on Intelligent Vehicles, 2(4), 2017, pp. 266 – 277

[6] Q. Jin, G. Wu, K. Boriboonsomsin, and M. Barth. *Power-Based Optimal Longitudinal Control for a Connected Eco-Driving System*. IEEE Transactions on Intelligent Transportation Systems, Vol. 17, No. 10, 2016, pp. 2900 – 2910

[7] U.S. EPA. *MOVES2010 Highway Vehicle Population and Activity Data*. EPA-420-R-10-026, 128 p, November 2010

[8] Dowling, R. et al. (2002). *Guidelines for Applying Traffic Microsimulation Modeling Software*. Report for Caltrans

[9] F. Ye, P. Hao, G. Wu, D. Esaid, K. Boriboonsomsin, and M. Barth. *An Advanced Simulation Framework of an Integrated Vehicle-Powertrain Eco-Operation System for Electric Buses*. 2019 IEEE on Intelligent Vehicles Symposium

4.2 Control for Mixed Traffic

4.2.1 Using Vissim External Driver Model DLLfor Modeling Mixed Traffic with both Automated Vehicles and Human Driven Vehicles

Contributor(s): Yunpeng Shi, Qing He

4.2.1.1 Introduction

PTV Vissim is a traffic simulation software used for microscopic simulation. The default car-following model types installed in VISSIM are Wiedemann 74 and Wiedemann 99 which are popular in simulating human driving behaviors. If one wants to use other behavior models such as adaptive cruise control (ACC) or intelligent driver model (IDM) for connected and autonomous vehicles, a function called Driver Model DLL can be used to modified the vehicles' behaviors in simulation. The External Driver Model allows users to replace various vehicle-related information such as velocity, acceleration, location, lane-changing signal, and intersection signals for vehicles in the simulation runs.

This tutorial will explain how to write into the External Driver Model DLL interface and use it in Vissim. Towards the end, a sample code is provided. Additionally, a Vissim sample model is attached for running the sample code. This tutorial is constructed based on Microsoft Windows 10 pro, PTV Vissim 7.00-13, and Microsoft Visual Studio Community 2017 Version 15.4.0.

4.2.1.2 Interface

This section introduces the basics of the driver model DLL interface.

Generally, the External Driver Model DLL interface file can be found in the following folder: Local Disk (C) / Program File/ PTV Vision/ PTV Vissim/ API/ DriverModel_DLL. There are several files in the folder:

- DriverModel.h: Header file for a driver model DLL.
- DriverModel.cpp: Main source file of a driver model DLL.
- DriverModel.vcxproj: Visual C++ 2010 project file for a driver model DLL. This file can be used if the DLL is to be created with Microsoft Visual C++.
- Interface Description: An explanatory PDF introducing the content in driver model DLL.
- DriverModel.Changes: A text file explaining the changes in driver model DLL

Some contents from this tutorial are taken from the Interface Description. Visual C++ is required to use driver model DLL.

Open DirverModel.vcproj with Visual C++, a Review Solution Action message might show. The settings in the figure below works.

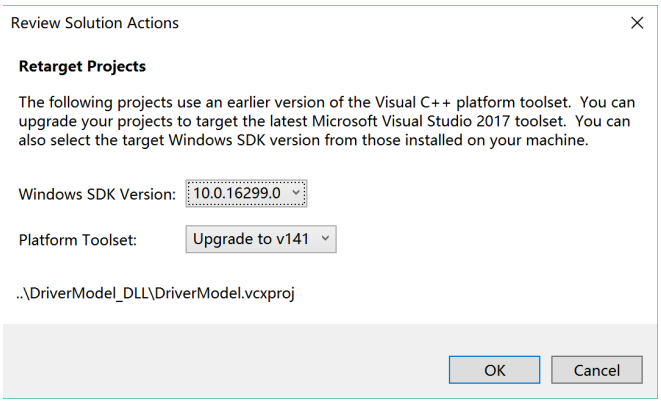


Figure 4.2.1-1: Review Solution Actions

After opening the project file, find "Solution Explorer" (usually on the left of the window), click on **DriverModel**, and then click on **DriverModel.cpp**, the default DLL code will appear. All codes will run once per vehicle per time step, meaning that the calculated values in DLL will be update for each vehicle in each time step in Vissim. There are three main functions: DriverModelSetValue, DriverModelGetValue, DriverModelExecuteCommand. The format and meaning of the functions are as follows:

```
DRIVERMODEL_API  int  DriverModelSetValue (long   type,
                      long   index1,
                      long   index2,
                      long   long_value,
                      double double_value,
                      char   *string_value)

{
 switch (type) {
 All the cases
 }
}
```

The function for Set Value is to extract data from Vissim. As defined in the Interface Description, VISSIM passes the current value of the data item indicated by type and (for most types) indexed by index1 and sometimes index2, the meaning of these values will be explained later in the tutorial. The value is passed in long_value, double_value or string_value, depending on type. Under the switch function are all the cases or types of data you can get from Vissim to simulate the vehicle behavior. The specification of the cases will be explained later in the tutorial.

```
DRIVERMODEL_API  int  DriverModelGetValue (long   type,
                      long   index1,
                      long   index2,
                      long   *long_value,
```

```
                                    double *double_value,
                                        char   **string_value)
{
 switch (type) {
 All the cases
 }
}
```

The Get Value function has the same format as Set Value, but can send data back to Vissim.

```
DRIVERMODEL_API  int  DriverModelExecuteCommand (long number)
{
switch (number) {
   case DRIVER_COMMAND_INIT :
     return 1;
   case DRIVER_COMMAND_CREATE_DRIVER :
     return 1;
   case DRIVER_COMMAND_KILL_DRIVER :
     return 1;
   case DRIVER_COMMAND_MOVE_DRIVER :
     return 1;
 }
}
```

The Execute Command is the core function in DLL for calculating vehicle behavior. The commands mean as follows:

- Init is called at the start of a VISSIM simulation run to initialize the driver model DLL.
- Create driver is called from VISSIM during the simulation run whenever a new vehicle is set into the network in VISSIM at the start of each time step.
- Kill driver is called from VISSIM when a vehicle reaches its destination and thus leaves the network to free memories.
- Move driver is called from VISSIM during the simulation run once per time step for each vehicle of a vehicle type which uses this driver model DLL. All vehicle behavior calculations should be written under this command.

All of the four commands have to return 1 as shown on the top example for Vissim to stop the simulation run. Create driver and kill driver commands can usually be controlled by Vissim if no special requests are needed.

The order of the driver model DLL commands being called at each time step is: Set Value to extract data needed from Vissim, Execute Command (Move Driver) to calculate driver behavior inserted in the command, then Get Value to send calculated parameters back to Vissim.

4.2.1.3 Data

This section introduces the data that can be extracted from and sent to Vissim.

4.2.1.3.1 Data Overview

As explained in the previous section, Set Value and Get Value are used to exchange parameter data with Vissim, the data includes:

- Subject vehicle data: physical characteristics of the current vehicle such as velocity, acceleration, location, and size; decision parameters such as turning indicator and desired velocity
- Nearby vehicle data: Information about nearby vehicles such as ID, velocity, acceleration, position, distance from the current vehicle. The detection range is two lanes to both sides, and two vehicles upstream and downstream.
- Link information data: Lane width and ending distance from current location.
- Current and upcoming environment data: Information about current slopes or upcoming curve.
- Next signal head data: Information about distance and state of the upcoming signal head.
- Speed decision data: Parameters related to speed limit sign.
- Behavior suggestion data: The behavior parameters that will influence decisions in the next time step, such as desired acceleration and lane change.

There are two ways to see the full list and explanation of the data.

Option 1: Interface Description. PDF, page 7-12.

Option 2: In **DriverModel.vcproj**, go to the Solution Explorer on the left, open **DriverModel** ➔ **DriverModel.h**. The window will look like the figure below; all the parameters are explained in green words below the purple parameter names.

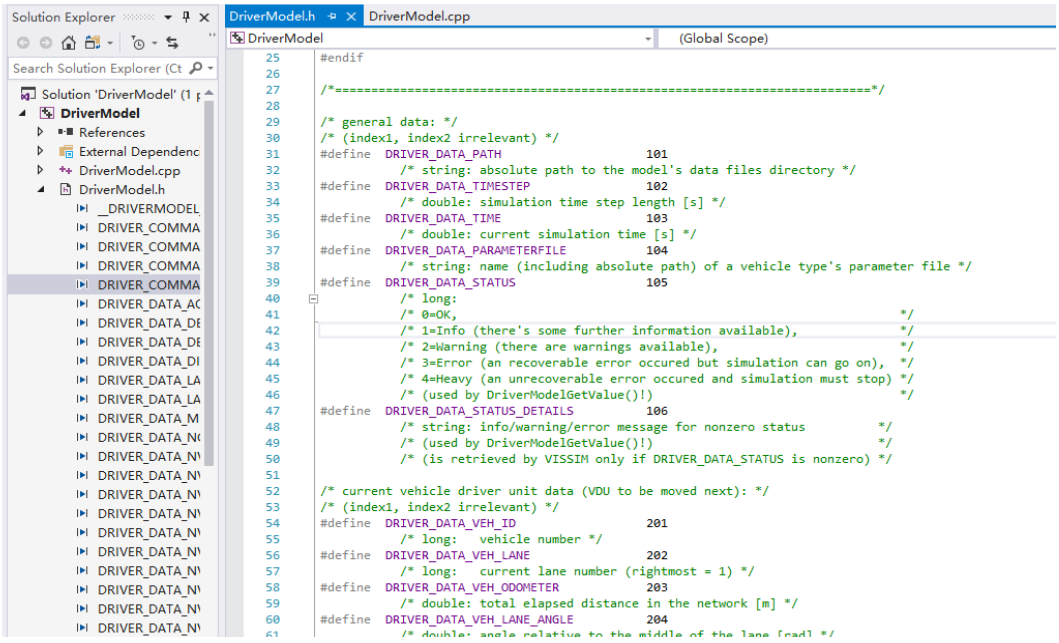


Figure 4.2.1-2: List and explanation of data

4.2.1.3.2 Create Variables

When data information is read from Vissim by Set Value, variables can be created to save the information. In order to do that, go to **DriverModel.cpp**, create variables in the blank space prior to Set Value function, make sure that these codes are not in any of the three main functions. The format should be (data format) + space + variable name you create. An example is shown in the figure below.

```
45  /*Create Variables*/
46
47  double current_time;
48  long current_vehicle;
49  double  current_velocity;
50  double  current_acceleration;
51  double  vehicle_length;
52  double  vehicle_x_coord;
53  double  vehicle_y_coord;
54  long  vehicle_type;
55  int vehicle_current_link;
```

Figure 4.2.1-3: Create Variables

Parameters in Vissim have default formats, format of the variables should match the default format in order to avoid errors. The default formats are shown in the parameter explanation in **DriverModel.h**. Figure below is an example, DRIVER_DATA_TIME has explanation: double: current simulation time [s], indicating that parameter "current simulation time" should be saved in a variable with format double. Thus, the variable can be created as: double current_sim_time.

```
31  #define  DRIVER_DATA_PATH                  101
32             /* string: absolute path to the model's data files directory */
33  #define  DRIVER_DATA_TIMESTEP              102
34             /* double: simulation time step length [s] */
35  #define  DRIVER_DATA_TIME                  103
36             /* double: current simulation time [s] */
```

4.2.1-4: Format types

In addition, there is a faster way in **DriverModel.cpp** to check the default data format. In Set Value function, move the cursor to the parameter below the desired one, a message box will appear, the second line in the box indicates the format and explanation of the desired parameter. For example, as shown in the figure below, the cursor is on Driver_DATA_VEH_LANE, but the message box appeared shows explanation describing the previous parameter: DRIVER_DATA_VEH_ID. The second line in the box shows that the format to save vehicle number should be long. Thus, variable saving vehicle number ("current_vehicle" in this example) can be created by: long current_vehicle.

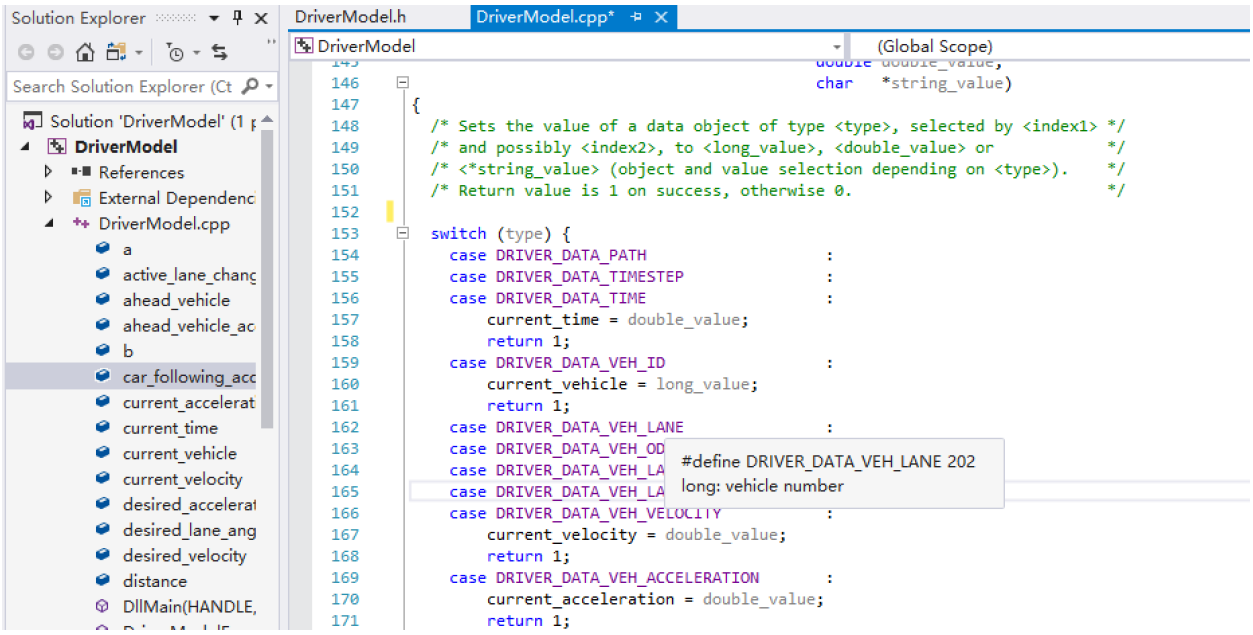


Figure 4.2.1-5: Default data type

4.2.1.3.3 Extract and Replace

After creating variables to save data, use Set Value function to exact data from Vissim. The usual steps are:

1. Find the desired parameter (case) in Set Value function, press Enter to create a blank line.
2. Type in: "created variable = format_value;". The format should match the default format mentioned previously.
3. In the next line, type in "return 1;".

An example is shown below extracting current vehicle acceleration from Vissim in Set Value function and saving to the variable current_acceleration.

```
case DRIVER_DATA_VEH_ACCELERATION:
        current_acceleration = double_value;
        return 1;
```

After running Execute Commands, calculated parameters can be sent back to Vissim using Get Value function. The steps are similar to the Set Value function:

1. Find the desired parameter (case) in Get Value function, if the desired parameter is not found, copy the case from Set Value function, then press Enter to create a blank line.
2. Type in: "*format_value = created variable;". The format should match the default format mentioned previously.
3. In the next line, type in "return 1;".

An example is shown below using the same variable as the previous example.

```
case DRIVER_DATA_VEH_ACCELERATION:
        *double_value = current_acceleration;
```

```
return 1;
```

4.2.1.3.4 Signal Data

The data describing intersection signal state is discrete. If the intersection is signalized, the value means as follows:

```
Set DRIVER_DATA_SIGNAL_STATE =
    red = 1, amber = 2, green = 3, red/amber = 4, amber flashing = 5, off = 6,
    other = 0
```

If the intersection is yield sign, the value means as follows:

```
Set DRIVER_DATA_SIGNAL_STATE =
    red = 1, green = 3
```

4.2.1.3.5 Data of Nearby Vehicles

Data of nearby vehicles can be obtained in Set Value function, including vehicle information, turning indicator, category information, and relative position to current vehicle. In Set Value, nearby vehicle data starts with DRIVER_DATA_NVEH. The range is up to two vehicles each downstream and upstream, on up to 2 lanes on both sides and the current lane. Index 1 and index 2 mentioned in section 2 are used to specify the chosen vehicle as follows:

```
For DRIVER_DATA_NVEH_* index1 and index2 are used as follows:
index1 = relative lane:    +2 = second lane to the left, +1 = next lane to the left,
                            0 = current lane,
                           -1 = next lane to the right, -2 = second lane to the right
index2 = relative position:    positive = downstream (+1 next, +2 second next)
                               negative = upstream (-1 next, -2 second next)
```

For example, if one wants to get the distance between current vehicle and the front vehicle on the next lane to the left, the code can be as follows:

```
case DRIVER_DATA_NVEH_DISTANCE        :
        if (index1 == 1 && index2 == 1)
        {
            gross_distance = double_value;
        }
        return 1;
```

4.2.1.3.6 Lane Change

The external driver model DLL allows current vehicle to perform lane change in the simulation. There are two ways to achieve that:

1. Simple lane change: Vissim will have control over the lane change process, a lane changing signal needs to be sent to Vissim by driver model DLL. In Get Value function, find simple_lanechange case and make it as the following format:

```
case DRIVER_DATA_SIMPLE_LANECHANGE :
        *long_value = 1;
            return 1;
```

   Then find DRIVER_DATA_ACTIVE_LANE_CHANGE, set the value to 1 to change to the left or -1 to change to the right. Vissim will make the lane change when default safety conditions are met.

2. Full control lane change: The driver model DLL will have full control in the lane changing process. For details, please look at Interface Description PDF Page 15.

4.2.1.4 Execute Command

This section introduces Execute Command.

As introduced in Section 2, calculation codes should be included in Execute Command. Other than create or delete drivers (vehicles), codes should be under case DRIVER_COMMAND_MOVE_DRIVER. A complete process of creating an IDM driver model using Driver Model DLL is in Appendix.

4.2.1.5 Build Solution

This section explains how to build the DLL file.

After finish building the model in DriverModel.cpp, select top task bar **Build ➔ Build Solution** (or press F7). The output window on the bottom will show the build outcome. If failed, please debug the driver model. If succeed, a path showing where the DriverMode.dll file is stored will appear like the figure below. Save the path for future use (inputting external driver model into Vissim).

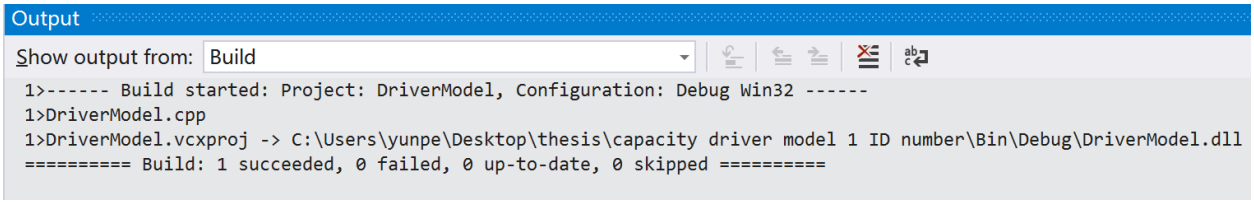


Figure 4.2.1-6: Building Solution

4.2.1.6 Vissim Setup

The section will explain how to insert the Driver Model DLL into Vissim.

4.2.1.6.1 Assign Vehicle Type

After building the Driver Model DLL, open Vissim. Click on the top task bar **Base Data** then **Vehicle Types**. The driver model DLL can either be applied on current vehicle types or new vehicle types created. To create a new vehicle type, right click on blank space inside the Vehicle Types window, and click on **Add…**

A vehicle type setting will appear after creating a new vehicle type or if use a current vehicle type, right click on the designated vehicle type and click **Edit…**

In vehicle type settings, click on **External Driver Model** tab, check the box "**Use external driver model**". Then type in or browse the path of the DriverModel.dll file. Then click **OK.** An example is shown below.

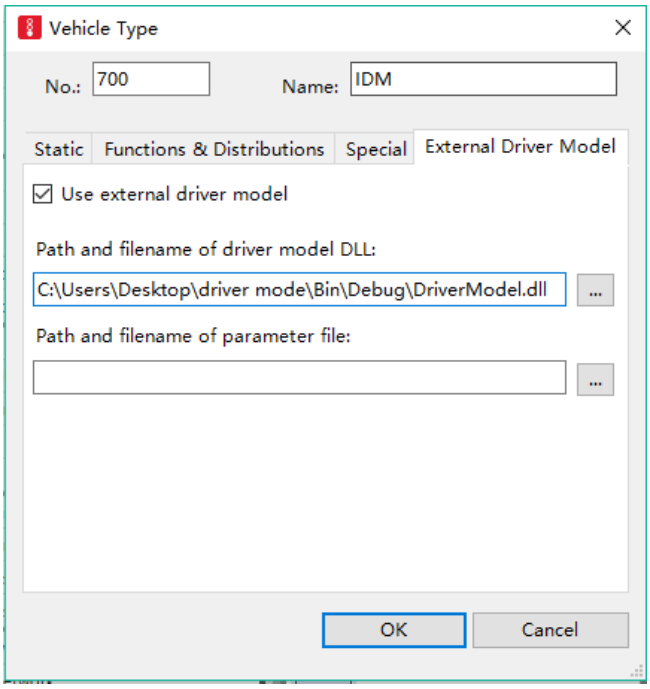


Figure 4.2.1-7: Vehicle Type

4.2.1.6.2 Assign Vehicle Input

After entering the vehicle type, click on the top task bar **Traffic** then **Vehicle Composition**. The Figure below is an example of vehicle composition window. The left window contains different vehicle compositions. Use an existing one by left click it or create a new one by right click in the blank space and then choose **Add…** Then move to the window on the right which shows relative flows between different vehicle types. If only one type of vehicle is used, click on the second column **VehType** and choose the desired vehicle type. If more than one type is used, right click in the blank space and choose Add… to create as many types as need, then choose desired vehicle types in the second column **VehType**. The relative flow can be modified in the fourth column **RelFlow**, inserting either the percentage of each vehicle type or the desired volume of each vehicle type will be acceptable. Vissim will automatically transform this relative flow to percentages. The desired speed distribution can be edited in the third column, to create new desired speed distributions, go to top task bar **Base Data ➔ Distributions ➔ Desired Speed**.

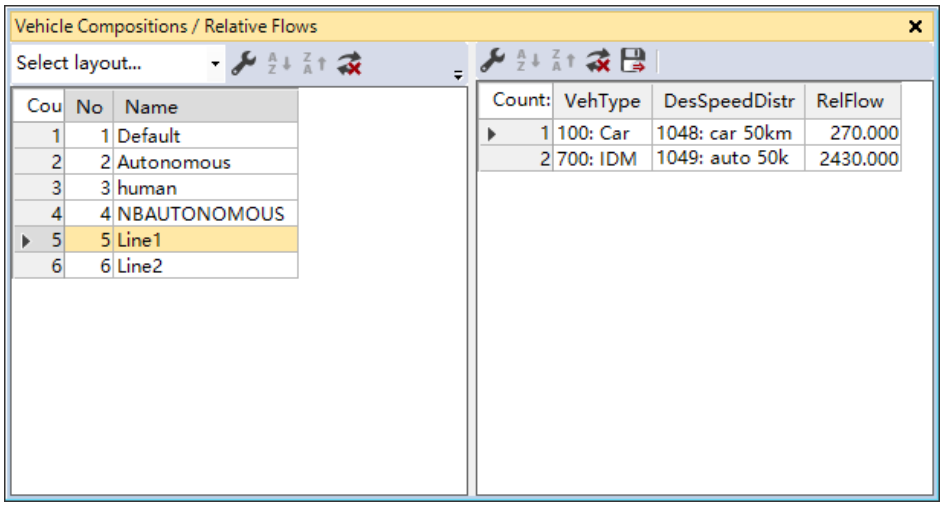


Figure 4.2.1-8: Vehicle Compositions

After creating the desired vehicle compositions, click on the top task bar **Lists ➔ Private Transport ➔ Input** to enter the desired volume. Choose the desired link on the fourth column, type in desired volume in the fifth, and choose the vehicle composition just created in the last column.

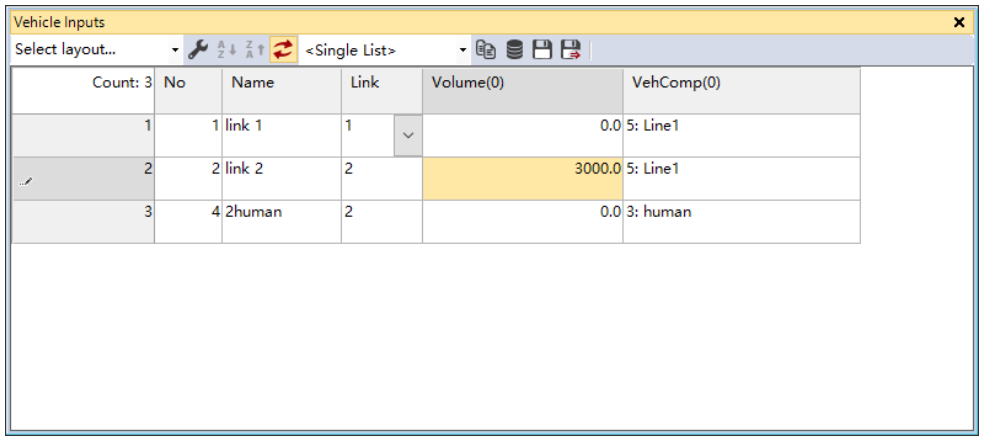


Figure 4.2.1-9: Vehicle Inputs

4.2.1.6.3 Before Simulation

Before simulation, go to top task bar **Simulation** > **Parameters**. As shown in the figure below, if any vehicle in the simulation use driver model DLL, the last option **Number of cores** needs to be 1 Core to avoid errors in the simulation. After done all the steps above, simulation can be started.

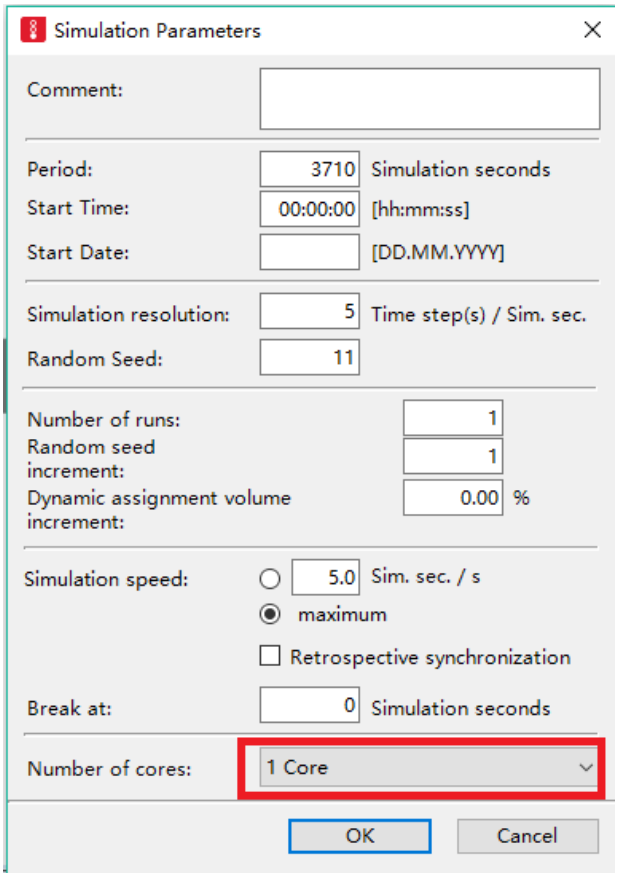


Figure 4.2.1-10: Simulation Parameters

4.2.1.7 Run Sample

This section will guide you to run the Vissim sample.

The sample contains two parts, sample codes and sample Vissim model.

The sample dll model is an IDM model similar to the one in Appendix. To make sure that the dll file works, it needs to be rebuilt. Open **DriverModel.vcxproj** in **Sample Code** file, select top task bar **Build ➔ Build Solution** (or press F7). The output window will show a succeed build with a path of the dll file. Copy the path.

Next, open **Capacity Analysis** (Vissim Input Model) in **Sample Vissim Model** file. Select top task bar **Base Data ➔ Vehicle Types**. Right click on No 700 IDM, and select **Edit…** A vehicle type window will appear. Choose **External Driver Model**, and update the box below "**Path and filename of driver model DLL:**" with the path you just copied.

Then choose top task bar **Simulation ➔ Continuous** to start the simulation. This sample Vissim model is a two-lane throughway network with length of 1,000 meters. Without changing any other parameters, the simulation will run for one hour. Both lanes will have desired volume of 2000

vehicle/hour/lane, and vehicle composition of 50% Vissim default human driven vehicles (Wiedemann 99), and 50% vehicles with IDM behavior.

* Author used Vissim 7 and Visual Studio (C++) 2017. Other versions might require minor changes to make the sample work.

4.2.1.8 Appendix

Intelligent Driver Model (IDM)

The IDM acceleration calculation is as follows:

$$a_{IDM}(S, v, \Delta v) = a[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{S^*(v, \Delta v)}{S}\right)^2]$$

And

$$S^*(v, \Delta v) = S_0 + vT + \frac{v\Delta v}{2\sqrt{ab}}$$

S: bumper to bumper distance to the leading vehicle

v: actual speed

Δv: velocity difference from the leading vehicle.

The values of the three parameters above will use values in Vissim simulations, other parameter values are fixed in this example, use values shown as follows:

Maximum Acceleration a	2 m/s^2
Maximum Deceleration b	3 m/s^2
Desired Speed v0	15 m/s
Free Acceleration Component δ	4
Jam Distance S0	1.5 m
Safe-time Headway T	0.6 s

Codes:

Default Parameters (by Vissim)

```
/*Default Parameters*/

double  desired_acceleration = 3.5;
double  desired_lane_angle   = 0.0;
long    active_lane_change   = 0;
long    rel_target_lane      = 0;
double  desired_velocity     = 13.9;
long    turning_indicator    = 0;
long    vehicle_color        = RGB(225,225,225);
```

Create Variables

```
/*================================================================================*/
/*Create Variables to store data*/

long current_vehicle;
double  current_velocity;
double  vehicle_length;
double  gross_distance;
double  speed_difference;
double  s_star;
double car_following_acceleration;
double ahead_vehicle;
static int vehicle_to_stop = 0;
```

IDM Parameters and Equations

```
/*================================================================================*/
/* IDM parameters and equation*/

double a = 2;                              // maximum acceleration, constant a (m/s^2)
double b = 3;                              // maximum deceleration, constant b (m/s^2)
double IDM_desired_velocity = 15;          // desired velocity uesd in IDM (m/s)
double v0 = IDM_desired_velocity;
double jam_distance = 0.6;                   // minimum gross distance (m)
double S0 = jam_distance;
double T = 0.6;                              // safe-time headway (s)

double S_star(double v, double v_delta)
{
       double S_star = S0 + (v * T) + ((v * v_delta) / (float)(2 * sqrt(a*b)));
       return S_star;
}
```

Set Value (Unused cases are deleted to avoid confusion)

```
DRIVERMODEL_API  int  DriverModelSetValue (long   type,
                                           long   index1,
                                           long   index2,
                                           long   long_value,
                                           double double_value,
                                           char   *string_value)
{

  switch (type) {
    case DRIVER_DATA_VEH_ID                :
              current_vehicle = long_value;
              return 1;
    case DRIVER_DATA_VEH_VELOCITY          :
              current_velocity = double_value;
              return 1;
    case DRIVER_DATA_VEH_ACCELERATION      :
              current_acceleration = double_value;
              return 1;
    case DRIVER_DATA_VEH_LENGTH            :
              vehicle_length = double_value;
```

```
            return 1;
    case DRIVER_DATA_NVEH_ID                :
            if (index1 == 0 && index2 == 1) {
                  ahead_vehicle = long_value;
            }
            return 1;
    case DRIVER_DATA_NVEH_DISTANCE          :
            if (index1 == 0 && index2 == 1)
            {
                  gross_distance = double_value;
            }
            return 1;
    case DRIVER_DATA_NVEH_REL_VELOCITY      :
            if (index1 == 0 && index2 == 1)
            {
                  speed_difference = double_value;
            }
            return 1;
    case DRIVER_DATA_DESIRED_ACCELERATION :
            desired_acceleration = double_value;
            return 1;
    case DRIVER_DATA_DESIRED_LANE_ANGLE :
            desired_lane_angle = double_value;
            return 1;
    case DRIVER_DATA_ACTIVE_LANE_CHANGE :
            active_lane_change = long_value;
            return 1;
    case DRIVER_DATA_REL_TARGET_LANE :
            rel_target_lane = long_value;
            return 1;
    default :
      return 0;
  }
}
```

Get Value (Unused cases are deleted to avoid confusion)

```
DRIVERMODEL_API  int  DriverModelGetValue (long   type,
                                           long   index1,
                                           long   index2,
                                           long   *long_value,
                                           double *double_value,
                                           char   **string_value)
{

  switch (type) {
    case DRIVER_DATA_DESIRED_ACCELERATION :
      *double_value = desired_acceleration;
    default :
      return 0;
  }
}
```

Execute Command

```
DRIVERMODEL_API  int  DriverModelExecuteCommand (long number)
{


  switch (number) {
    case DRIVER_COMMAND_INIT :
      return 1;
    case DRIVER_COMMAND_CREATE_DRIVER :
      return 1;
    case DRIVER_COMMAND_KILL_DRIVER :
      return 1;
    case DRIVER_COMMAND_MOVE_DRIVER :

       if (ahead_vehicle < 0) {
              vehicle_to_stop = current_vehicle;
       }

//In Vissim, if a vehicle passes the destination and is deleted from the network, the
vehicle number will show as -1.
//Since IDM is a car-following model, it cannot control the first vehicle of a stream
properly. Therefore, the control of the first vehicle in stream will be handled by
Vissim.

       if(current_vehicle != vehicle_to_stop ){

              double S = gross_distance - vehicle_length;
              double v = current_velocity;
              double v_delta = speed_difference;
              double s_star = S_star(current_velocity, speed_difference);

              car_following_acceleration = a * (1 - pow((v / (float)(v0)), 4) -
pow((s_star / (float)S), 2));


              if (gross_distance > jam_distance){
                     desired_acceleration = car_following_acceleration;
// setting the IDM acceleration as the desired acceleration if gross distance is greater
than the minimum gross distance required.
              }

       }
      return 1;
    default :
      return 0;
  }
}
```

4.2.1.9 References

Interface Description.pdf, PTV Vissim