Czech Technical University of Prague
Faculty of Transportation Sciences
Department of Transport Telematics

Control System Development of Traffic Signal Control in MATLAB for PTV VISSIM

MASTER´S THESIS

Author: Bc. **Filip Skružný**
Leader: **Ing. Milan Koukol, Ph.D.**
Year: **2017**

CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Transportation Sciences
Dean's office
Konviktská 20, 110 00 Prague 1, Czech Republic

K620..Department of Transport Telematics

MASTER'S THESIS ASSIGNMENT

(PROJECT, WORK OF ART)

Student's name and surname (including degrees):

Bc. Filip Skružný

Code of study programme code and study field of the student:

N 3710 – IS – Intelligent Transport Systems

Theme title (in Czech): **Vývoj systému řízení SSZ v aplikaci MATLAB pro mikrosimulační model PVT VISSIM**

Theme title (in English): Control System Development of Traffic Signal Control in MATLAB for PTV VISSIM

Guides for elaboration

During the elaboration of the master's thesis follow the outline below:

- VISSIM + MATLAB connection through COM interface
- Interface creation for setting parameters of external signal plan
- Testing of different options in terms of load, travel times, directionality and composition of traffic flow
- Finding of optimization technique for model testing through Matlab
- Analysis of the results

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Graphical work range: according to supervisor's recommendations

Accompanying report length: min. 55 pages including figures, graphs and tables

Bibliography:

MATHWORKS, INC. MATLAB Programming Fundamentals [online]. U.S., 2015.

MATHWORKS, INC. MATLAB External Interfaces [online]. U.S., 2015.

PTV AG. PTV Vissim 8: User Manual [online]. Germany: PTV AG

Master's thesis supervisor: **Ing. Milan Koukol, Ph.D.**

Date of master's thesis assignment: **August 28, 2015**
(date of the first assignment of this work, that has be minimum of 10 months before the deadline of the theses submission based on the standard duration of the study)

Date of master's thesis submission: **May 30, 2017**

a) date of first anticipated submission of the thesis based on the standard study duration and the recommended study time schedule
b) in case of postponing the submission of the thesis, next submission date results from the recommended time schedule

L. S.

....	....
doc. Ing. Pavel Hrubeš, Ph.D.	prof. Dr. Ing. Miroslav Svítek, dr. h. c.
head of the Department of Transport Telematics	dean of the faculty

I confirm assumption of master's thesis assignment.

....
Bc. Filip Skružný
Student's name and signature

Prague ....November 30, 2016

Declaration

I declare that I have accomplished my final thesis by myself and I have named all the sources I had used in accordance with the guideline about the ethical rules during preparation of university final theses.

I have no relevant reason against using this schoolwork in the sense of § 60 of Act No121/2000 concerning the authorial law.

Prague 26.5.2017
....
Place, date

....
Filip Skružný

Acknowledgement

I would like to express my respect and gratitude to Ing. Milan Koukol, Ph.D. for the supervision of this thesis, his immense knowledge and kindly continuous support.

With my deepest thanks and appreciation to my parents for letting me study at the University.

…………………………

Filip Skružný

Title:	Control System Development of Traffic Signal Control in MATLAB for PTV VISSIM
Author:	Bc. Filip Skružný
Field:	Intelligent Transport Systems
Project:	Traffic Models and Traffic Control
Type of Thesis:	Master's Thesis

Abstract:

This master's thesis is a*i*med to control system development of traffic signal control for microsimulation models in PTV Vissim. The control system is being developed in Matlab program and it controls models in PTV Vissim through COM (Component Object Model) interface.

Papers are also dedicated to optim*ization in model testing in a sens*e of simplify the work during testing several scenarios within one model.

The ove*r*all aim is to present new possibilities in controlling and testing microsimulation models via Matlab. It is considered to be a quite new, not very well-known and used approach which could help to b*roa*den horizons and make working with traffic models more efficient. At least for the people from the faculty.

Key Words: Traffic simulations, PTV Vissim COM interface, Control system of traffic signal control, Optimization in model testing, Matlab programming

Abstrakt:

Tato diplomová práce je věnována vývoji systému řízení světelného signalizačního zabezpečení pro mikrosimulační modely v PTV Vissim. Systém řízení je vyvíjen v programu Matlab a modely v PTV Vissim ovládá skrze COM (komponentový objektový model) rozhraní.

Dále se práce věnuje optimalizaci v testování modelů ve smyslu zjednodušení práce při testování různých scénářů na jednom modelu.

Celkově jde hlavně o představení nových možností v řízení a testování mikrosimulačních modelů pomocí Matlabu. Jedná se o poměrně novou, ne příliš zmapovanou cestu, která by mohla alespoň lidem na fakultě pomoci rozšířit obzory a zefektivnit práci s modely.

Klíčová slova:

Dopravní simulace, PTV Vissim COM rozhraní, Systém řízení světelného signalizačního zabezpečení, Optimalizace v testování modelů, Programování v Matlabu

Used abbreviations

COM	Component Object Model
GUI	Graphical User Interface
VAP	Vehicle Actuated Programming
ID	Identifier

Contents

Introduction

Since every module for PTV Vissim is offered separately and its functionality is very specific and limited, there is a huge opportunity in obtaining those modules with some additional software through the COM interface.

One of the best software for that is Matlab. The biggest opportunity of using external software is the possibility of controlling Vissim model and simulation in a real time. It can save a lot of time if the code is well build and changing of parameters is done automatically between each simulation for example. It can bring an opportunity to test several scenarios within one model without changing all parameters manually in separated file for each scenario. Since Matlab is mostly designed for mathematical computation and simulation, it can also evaluate simulation results for better performance.

Even though the real situation is used in following chapters, many parameters are not properly calculated or measured. Some data comes from proper traffic survey but some data is only estimated from other values so that the whole system would give sense (speaking mainly of some vehicle inputs and vehicle composition from not very busy inputs to network where the traffic survey did not take place as well as some parameters in control signalization system).

The reason is that the aim of this work is definitely not to solve some exact situation but more likely to show the way of what is possible to do using Vissim and Matlab together. This could work as some kind of manual. Some specific situation is used only for better understanding the problem and for better imagination.

Advantages of the thesis should be providing of some advanced skills in controlling Vissim simulations via Matlab, regarding mainly signal control systems and testing of models. The assignment rose at the Faculty of Transportation Sciences mainly for the purposes of people from the project Traffic Models and Traffic Control, because this area has not been significantly examined yet there. They can use it as an overview of possibilities in this area or use it as a base for their future work with these systems.

The thesis starts with the introduction of used software and the COM interface. It continues by simulations, control systems and evaluation, all controlled by Matlab. Then there is the already mentioned more complex control system with a model based on a real area used. It is followed by a GUI to control such system. It contains several scenarios to demonstrate optimization in model testing. It is followed by outputs of simulations. The last chapter is dedicated to a possibility of using Vissim with Matlab to simulate potential future scenarios. After the conclusion, there are source codes in appendix.

Chapter 1

Vissim and Matlab

In following subchapters used software is presented as well as the COM interface and some practical sample of obtaining and changing values of attributes within Vissim and Matlab.

1.1 Software

The default program that is being used for simulations is PTV Vissim. It is the leading microscopic simulation program for modelling multimodal transport operations. It displays all road users and their interactions in one model. Scientifically sound motion models provide a realistic modelling of all road users. [1]

All models for this thesis were developed and run in Vissim 7.00 – 15. PTV provided External, VisVAP and Vissig signal controllers and the most important module COM Interface.

The default program for controlling Vissim models is Matlab (matrix laboratory) developed by MathWorks. It is a programming language with emphasis on matrix operations and numerical calculations in general.

Version R2015b and R2017a were mainly used. But so were some other older versions. COM interface was introduced before version R2006a.

For the operation software, which allows these programs to exchange data, is being used Microsoft Windows 10.

1.2 COM Interface

The Component Object Model describes how binary components of different programs collaborate. COM gives access to data and functions contained in other programs. Since Vissim version 4.0, data contained in Vissim can be accessed via the COM interface using Vissim as automation server. Since Vissim Version 4.30, COM scripts can be called directly from the Vissim main menu.

COM does not depend on a certain programming language. COM Objects can be used in a wide range of programming and scripting languages, including VBA, VBS, Python, C, C++, C#, Delphi and MATLAB.

Important note is that COM Interface is not available in any version prior to Vissim 4.0 as well as in student and demo versions. A complete version needs to be installed.

The Vissim COM Model is subjected to a strict object hierarchy. It is visible in Figure 1 below. IVissim is the highest-ranking object. To access a sub-object, e.g. a link in the network, one must follow the hierarchy.

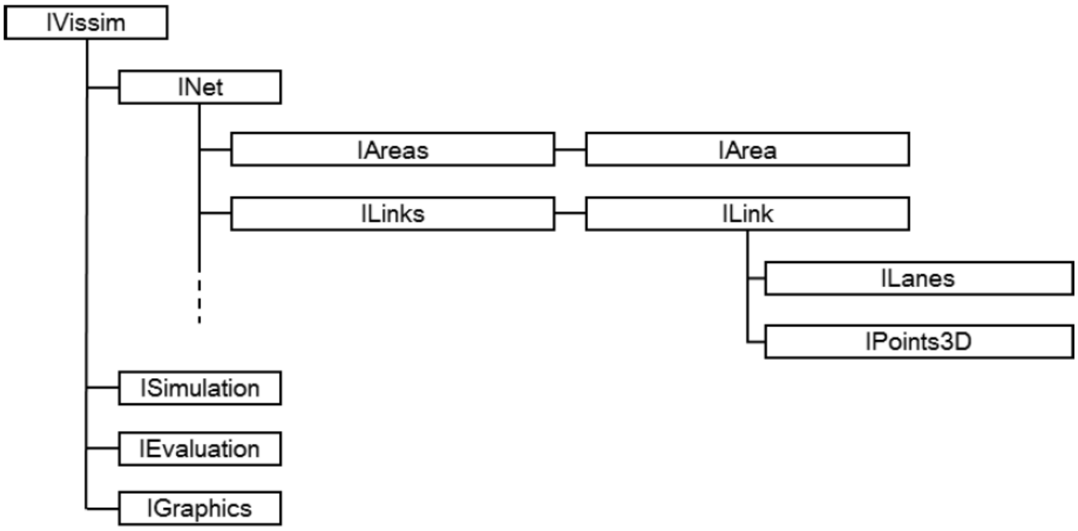

Figure 1: Vissim object hierarchy (source: [2])

The leading "I" of an object (e.g. at IVissim) stands for Interface. The name of the COM-Interface is the same as the Vissim class name plus the leading "I". [2]

1.3 Connection

Firs of all, a model in Vissim has to be created and saved. It is better if the path excludes diacritics (the same applies later to Matlab file).

Many things can be done via COM, such as editing of links, but it is much easier to be done in Vissim. The requirement is to do as much as possible in Matlab, but the whole infrastructure is better to be done in Vissim. Thanks to the graphical interface and the ability to set a real situation in the background, it is fast and precise.

When the infrastructure is ready (or whenever during the process), it is necessary to click on *Help -> Register COM Server* and if some additional authorization question from operation system appears, it has to be confirmed.

After that, Matlab should be started. It is better to use M-Files, but Command Window works as well, at least for some simple demonstrational commands. Starting with *File -> New -> Blank M-File*, the script should be saved with same rule specified above. Regarding M-Files locations, the path should also exclude spaces.

The command for creating COM server is:

`Vissim = actxserver('Vissim.Vissim')`, for some specific Vissim version can

be used: `Vissim = actxserver('Vissim.Vissim.700')`.
`700` is for version 7.00, which is being used for these papers.

The first *Vissim* is assigned variable which now represents the IVissim from the hierarchy. It can be called differently of course, this is just for better overview.

Actxserver creates an in-process server for a dynamic link library (DLL) component or an out-of-process server for an executable (EXE) component. The following Figure 2 shows the basic steps in creating the server process.

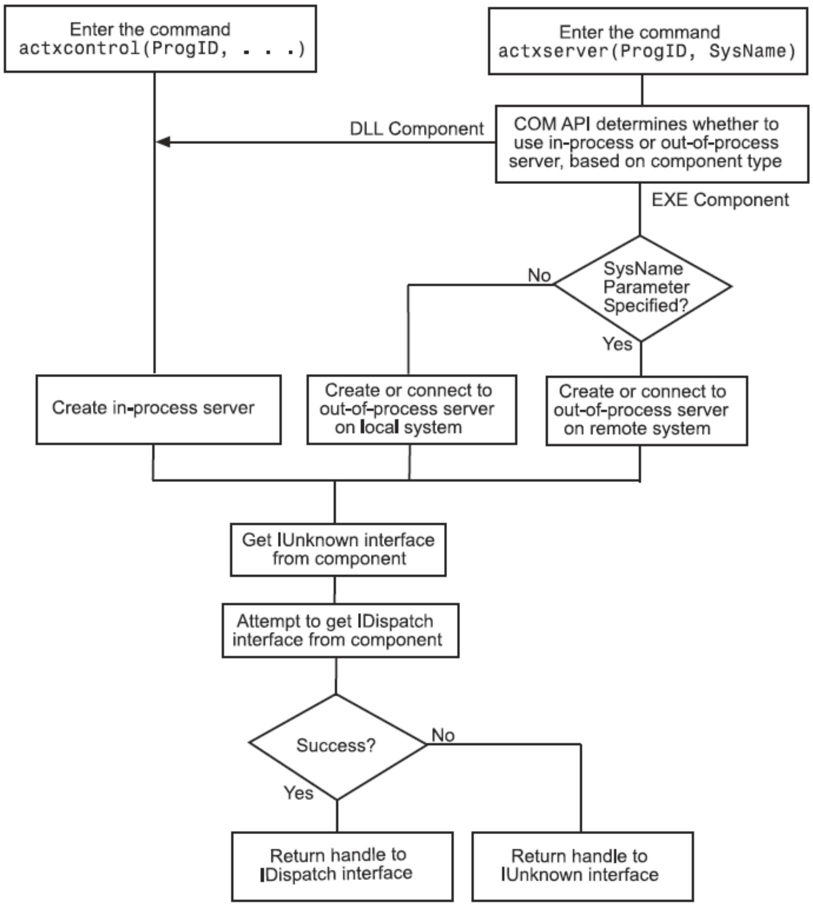

Figure 2: Diagram of creating COM server (source: [3])

Nevertheless, the first command in each M-File should be `clear all`. It clears all objects in the Matlab workspace. Without any cleaning command, old variables could cause troubles if the new ones would not have been specified well. There should be as well the command `clc`, which clears only the Command Window. [4]

After starting the COM server, access to the Vissim files should follow. Vissim creates at least two files, '*.inpx' is a file with Network and '*.layx' is for layout, the '*' is for file name. As the variable *Vissim* already represents the IVissim, working with files is on the next level. It is reachable by adding dot followed by path in brackets and quotation marks:
`Vissim.LoadLayout('c:\Users\...*.layx')`
and `Vissim.LoadNet('c:\Users\...*.inpx')`.

The `c:\Users\` followed by name of the user is a default Windows location, but files can be stored anywhere on hard drive. If the path is not specified, file explorer will appear and let the user to find the desired file.

Now, the connection is ready and Vissim files can be edited and controlled. To reach specific attributes, the hierarchy must be followed. For example, to reach objects from INet, `Vissim.Net` command must be written. To make the writing easier for the future, this hierarchy way can be stored in a different variable: `vnet = Vissim.Net;`. For obtaining specific attribute, function `get` and `ItemByKey` need to be used. For setting some attribute, function `set` needs to be used:

```
FirstLinkName = get(vnet.Links.ItemByKey(1),'AttValue','Name');
NewName = 'newstreet';
set(vnet.Links.ItemByKey(1),'AttValue','Name',NewName);
```

The old name of Link number 1 is stored in `FirstLinkName`, but in Vissim it has a new name (newstreet). Setting parameters can be done without previous getting them. Only the access must be correctly specified. The number in bracket of `ItemByKey` works also as variable. Instead of `1` it can contain for example `Link_number`, but it needs to be specified above:

```
Link_number = 1;
set(vnet.Links.ItemByKey(Link_number),'AttValue','Name',NewName);
```

This number corresponds to a number in Vissim network. After double left clicking on desired link, the number is in upper left box (*No.:*).

Another way of getting to attribute is:

```
Attribute = 'name';
Name_of_Links = vnet.Links.GetMultiAttValues(Attribute);
```

If there are any sub attributes, it can be reached by `GetAll`. It will be used later. All these basic approaches can be found after Vissim installation in *c:\Users\Public\Documents\PTV Vision\PTV Vissim 7\Examples Training\COM\Basic Commands\COM_examples.m* [2]

For details about the various objects and their methods and properties, the COM interface reference is included in the Online Help. The public methods are shoved in Figure 3 below. As seen in the figure, it is possible to save changes in the layout or network as well as to save them in a new files with arbitrary path.

To find out more about objects that are important for controlling and editing models, the access is throw *Help -> Online Help -> Vissim - COM -> Objects*. Each object starts with the "I".

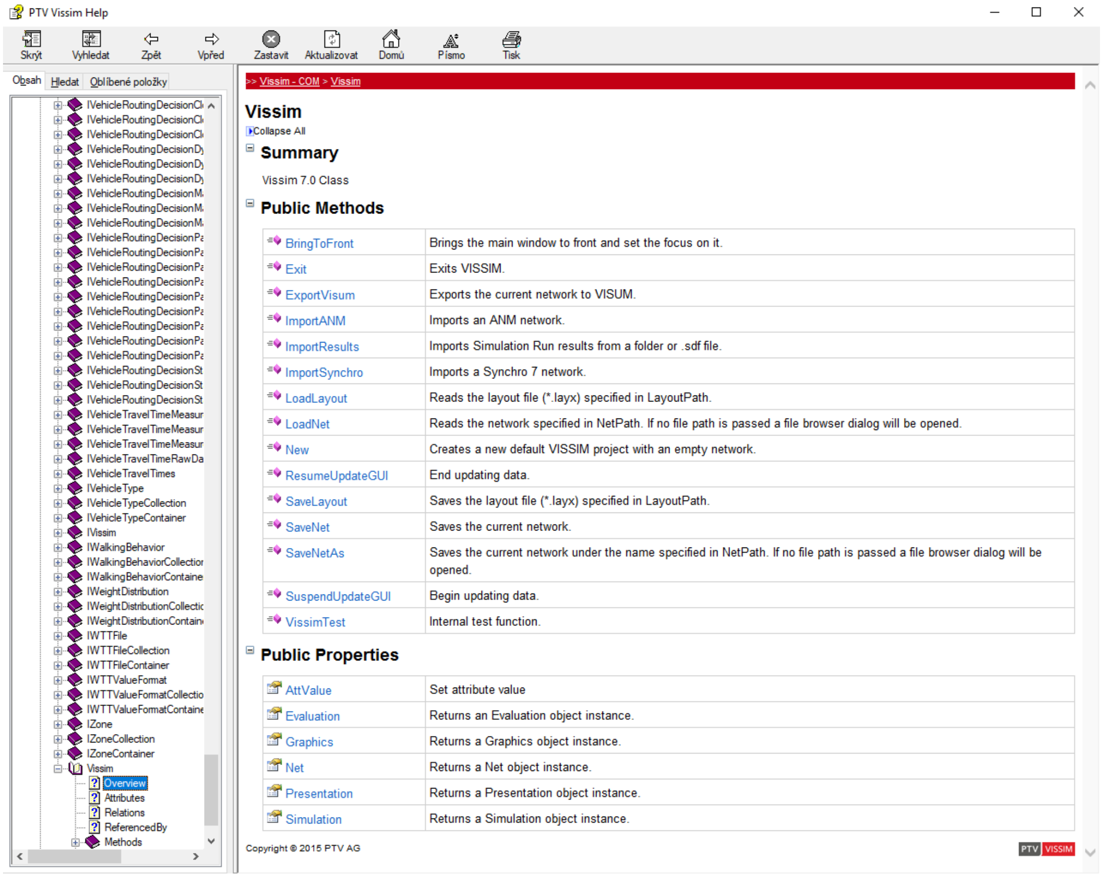

Figure 3: Public Methods - Online help (source: [1])

Some basic demonstrative program for changing a link name could look like this:

```
clc;
clear all;
Vissim = actxserver('Vissim.Vissim.700');
Vissim.LoadLayout('c:\Users\...*.layx');
Vissim.LoadNet('c:\Users\...*.inpx');
vnet = Vissim.Net;
NewName = 'newstreet';
set(vnet.Links.ItemByKey(1),'AttValue','Name',NewName);
Vissim.SaveNetAs('e:\…*.inpx');
Vissim.SaveLayout(('e:\…*.layx');
Vissim.Exit;
Vissim.release;
```

For execution of the M-File, the *Run* button needs to be pushed. In several seconds, the Vissim will start desired file, change the link name and thanks to the `Vissim.Exit` it will close the Vissim application when changing and saving have been done. The last function sets the COM server free and the status bar gets from Busy to Ready or blank space again.

Chapter 2

Control Systems

In this chapter, ways of running simulations are presented as well as possibilities of putting vehicles into the network. Also, some interface of external signal plan is shown.

2.1 Simulations

2.1.1 Random Seed

The hierarchy for controlling simulation leads to the second level: `sim = Vissim.Simulation`. On this level, one important parameter can be set. It is a value of random seed. This value initializes a random number generator. Two simulation runs, using the same network file and random start number, look the same. If the random seed is changed, the stochastic functions in Vissim are assigned a different value sequence and the traffic flow changes. This, e.g., allows to simulate stochastic variations of vehicle arrivals in the network. [1]

This function is set by:

```
Random_Seed = 42;
set(sim, 'AttValue', 'RandSeed', Random_Seed);
```

As for the syntax, `sim.set('AttValue', 'RandSeed', Random_Seed);` has the same meaning.

Important note is that directly in Vissim, the maximal value for random seed is *4294967295*. But when setting in M-File, this number would exceed value range of attribute. Here the maximal value can be *2147483647*. Also, only natural number can be used.

2.1.2 Period Time and Cores

Other parameter to be set are the period time and number of cores:

```
period_time = 3600;
sim.set('AttValue', 'SimPeriod', period_time);
```

The number is a simulation time in simulation seconds.

```
max_cores = 4;
sim.set('AttValue', 'NumCores', max_cores);
```

It is a number of processor cores used during simulation. The maximum number of cores used depends on the computer.

2.1.3 Simulation Resolution:

```
step_time = 10;
sim.set('AttValue', 'SimRes', step_time);
```

Simulation resolution is a number of time steps per simulation second. It specifies how often vehicles and pedestrian are move in a simulation second. The position of vehicles is recalculated in a simulation second with each time step. The simulation resolution specifies the number of time steps.

The value can be a natural number from 1 to 20. Values < 5 lead to jerky movements. This is why this value range is less suitable for production of the final simulation results. As lower values accelerate the simulation, the use of lower values during setup of the network model can be helpful.

Values between 5 and 10 lead to a more realistic demonstration. This value range is suitable for the production of the final simulation results.

Values between 10 and 20 lead to smoother movements. This value range is suitable for high-quality simulation animations.动画

The simulation resolution has an impact on the behaviour of vehicles, pedestrians, and the way they interact. This is why simulations, using different simulation resolutions, produce different results. [1]

2.1.4 Simulation Run

For the run of the simulation, there are two possibilities depending mainly on the control. If the aim is to set parameters, run simulation and get results, and there are no traffic lights, or there are some, but the signal program is specified in some different program (Vissig, VisVAP,..), continuous run can be used:

```
sim.RunContinuous;
```

If there is any need to interrupt the simulation in a specific time, `set(sim, 'AttValue', 'SimBreakAt', Sim_break_at);` shall be specified above. The `Sim_break_at` should include the desired time in simulation seconds of course. Then, some parameters changes can follow and next `sim.RunContinuous` will let the simulation to continue. If a value of signal head/group was changed manually during the stop, and there is a necessity to give the control back, it can be specified in the next break as `set(SignalController.SGs.ItemByKey(1), 'AttValue', 'ContrByCOM', false);`.

However, if the signal program is specified only in M-File, there is a necessity to check some parameters (especially the simulation seconds and detectors states) as frequently as possible. In this case, the simulation has to be processed by single steps: `sim.RunSingleStep;`

To use it in practise, a Matlab function should be put together. Best working is the *for* cycle:

```
for i=0:(period_time*step_time)
```

```
sim.RunSingleStep;
end
```

The multiplication `period_time*step_time` delivers as many single steps as it is required to cover all period. The `i` rises by one during each cycle. Before the end of cycle comes, commands and verifying, regarding signal plan and setting parameters and so on, can be putted.

To be able to observe the vehicle to vehicle interactions, the simulation speed is possible to be set:

```
set(sim, 'AttValue', 'SimSpeed', 1);
```

The value indicates simulation seconds per real-time second. When there is the `1`, the simulation is run in real-time, `10` would mean ten times faster than real-time. It can be editable during the simulation. To set the speed to maximum, there is a special command:

```
set(sim, 'AttValue', 'UseMaxSimSpeed', true);
```

2.1.5 Quick Mode

When the simulation speed is preferred to observation, there is the Quick Mode in Vissim. In the Quick Mode, all dynamic objects (e.g. vehicles, pedestrians, dynamic labels, and colours) are hidden in all network editors. In addition, in the Quick Mode, list windows and the Quick view are only then updated when it is scrolling or clicking in them. This allows for a maximum simulation speed. The difference in time duration is really perceptible. [1]

Inside the Vissim environment, there is a simple button to activate or deactivate this mode whenever, even during the simulation. For the Matlab, there is a formula:

```
set(Vissim.Graphics.CurrentNetworkWindow, 'AttValue', 'QuickMode',
1);
```

`1` is to activate and `0` to deactivate the Quick Mode. It can be done any time as well.

2.2 Vehicle Inputs and Vehicle Routing

Inputs of vehicles are the basis of every simulation. They should come from a traffic survey or from some prediction. In the following example, inputs are artificially set. The same applies to vehicle routings. Vehicle inputs can be easily set in Vissim. For better orientation in assigning volume for different classes of vehicles it is useful to create at every entry link as many inputs as the number of vehicle composition is (in these papers, only three classes are considered – cars, buses and trucks). The problem stars when the data is coming from a survey, it is measured for example in five minutes lasting intervals and it is desirable to change it every five minutes during the simulation as well. Then a lot of clicking and filling numbers would follow. Again, the same applies to vehicle routings. It can be really confusing.

To let this setting for Matlab, vehicle inputs should be created for every vehicle class separately. Vissim hides more black stripes in one link into one stripe, but after clicking that stripe, a table with all vehicle inputs appears. The example is visible in Figure 4.

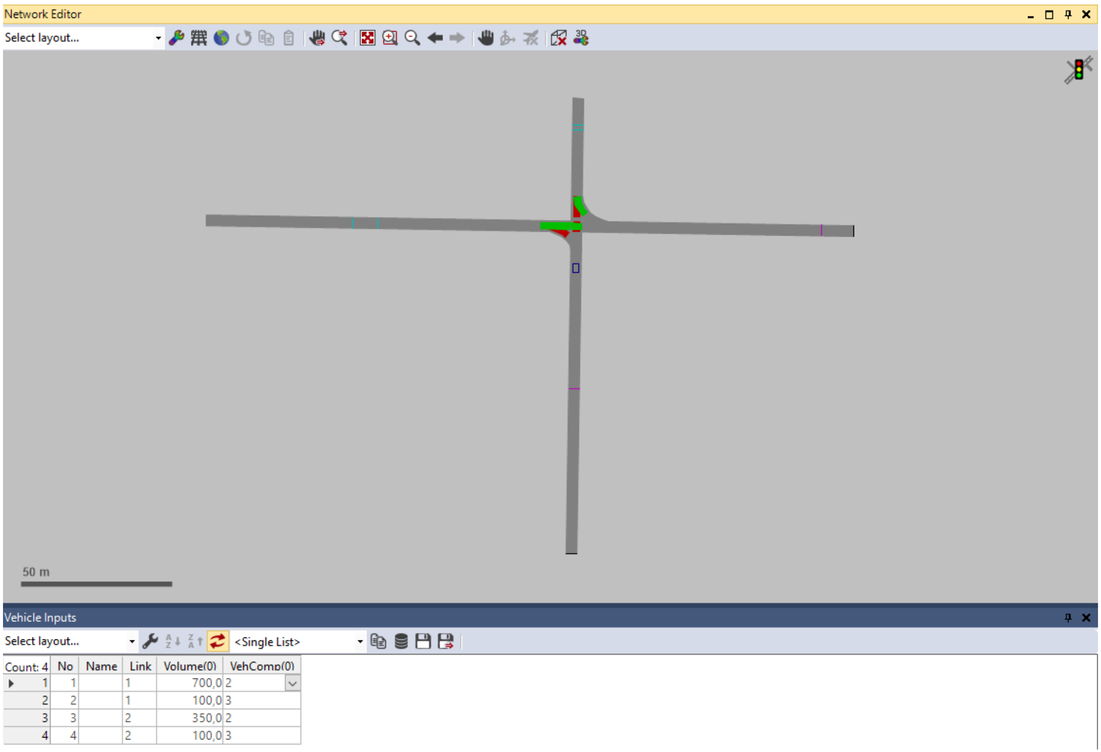

Figure 4: Vehicle inputs

There are four inputs, first two in the bottom link and the second two in the right link. Just two in each entrance, because only cars and trucks are considered in this example. All links are one-ways. No volume values are necessary to be filled in Vissim, only in the right column, two new vehicle compositions should be created (by simple clicking on *New…*). Then, also vehicle routs should be created. In this case, each route is defined for both vehicle types. To divide them and set them to different values, each vehicle class would need its own route. So, instead of four (from each entrance to each exit) there would be eight of them. This kind of solution is described and shown in practise in the next chapter.

The idea is that vehicle intensities are stored in some text file and Matlab will get them directly from the file (values in time intervals should be recalculated to hourly intensities first – it can be easily done also in Matlab). Regarding the *.txt file, data should be stored in rows for relevant vehicle inputs numbers (the first row for the first vehicle input – cars) and volumes should be divided by spaces. Demonstrative text file with data for this example is in Figure 5. Some better formatting with title row is possible, but it has to be renumbered well in Matlab.

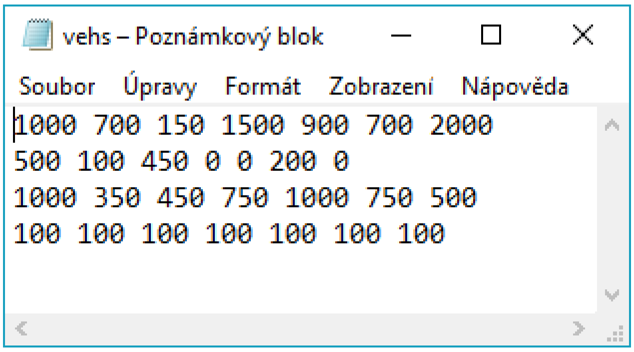

Figure 5: Intensities for vehicle inputs

To load the file to Matlab, it should be in the same directory, then simple `load vehs.txt` needs to be written. It contains the file name. Then some access should be specified:

```
vehins=vnet.VehicleInputs;
vehin(1)=vehins.ItemByKey(1);
vehin(2)=vehins.ItemByKey(2);
vehin(3)=vehins.ItemByKey(3);
vehin(4)=vehins.ItemByKey(4);
```

Then it is essential to set the vehicle composition, because in Vissim only new empty classes were created:

```
Veh_composition_number = 2;
Rel_Flows =
vnet.VehicleCompositions.ItemByKey(Veh_composition_number).VehCompRe
lFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        100);
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   50);
set(Rel_Flows{1}, 'AttValue', 'RelFlow',        1);

Veh_composition_number = 3;
Rel_Flows =
vnet.VehicleCompositions.ItemByKey(Veh_composition_number).VehCompRe
lFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        200);
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   50);
set(Rel_Flows{1}, 'AttValue', 'RelFlow',        1);
```

The vehicle composition number starts from two, because the first one was already formed by Vissim, but it is the shared one for cars and trucks. Here the vehicle type is defined (100 is for cars and 200 for trucks, 300 would be for buses), then the desired speed distribution is set – both classes for 50 km/h. Also, the relative flow can be set here.

The next procedure is to connect these preset classes to corresponding inputs:

```
vehin(1).set('AttValue', 'VehComp(1)', 2);
vehin(2).set('AttValue', 'VehComp(1)', 3);
vehin(3).set('AttValue', 'VehComp(1)', 2);
vehin(4).set('AttValue', 'VehComp(1)', 3);

veh_id = 1;
vehin(1).set('AttValue', 'Volume(1)', vehs(1,veh_id));
vehin(2).set('AttValue', 'Volume(1)', vehs(2,veh_id));
vehin(3).set('AttValue', 'Volume(1)', vehs(3,veh_id));
vehin(4).set('AttValue', 'Volume(1)', vehs(4,veh_id));
```

Vehicle composition two stands for cars specified above and three is for trucks. In the next section, first values from loaded file are set to vehicle inputs before the start of simulation. The same procedure (the second section with volumes) later appears in the simulation with time interval verification for volume change. The verification of time intervals is shown in the next section with signal plan. With each change, small function appears to load next column:

```
if veh_id < length(vehs)
     veh_id = veh_id + 1;
end
```

When the last column is reached but it is not the end of simulation, thanks to the condition, the last value will remain in vehicle inputs until the end.

The routing could be set like this:

```
routing(1,2)=vnet.VehicleRoutingDecisionsStatic.ItemByKey(1).VehRout
Sta.ItemByKey(2);
routing(1,2).set('AttValue', 'RelFlow(1)', 1);
```

The first `ItemByKey` specifies the starting link and the second one the destination link. So, this is for vehicles traveling from bottom which then turn. The `1` for real flow does not mean much. Now, everything depends on the value in the second route from the same starting link. Considering the same value, 50% of vehicles would turn left and 50% would go straight up. With a zero, no vehicle would go this way and 100% would turn left. If there was a ten, ten times more vehicles would go this way and so on.

It can be set for each route according to the time interval from a file as well.

2.3 Creation of Signal Plan

Simple static fixed time control is being developed even with some basic kind of dynamic control in this subchapter.

It is necessary to put signal heads to links in Vissim first. But the program gives warning that *"A signal controller with a signal group must exist before a signal head can be created."*. So, it is just enough to click on *Signal Control* -> *Signal Controllers* and in the new field right click and *Add…* -> *Edit Signal Control*, right click and add new signal group in the new window. If more than one signal group is required, it needs to be declared here. Then it should be saved and that is all. It will create a *.sig file in the same directory as the model is. Signal heads can be installed into lanes after this declaration.

Signal heads can be easily assigned to signal controller - SC (it can be more of them by adding them into the lower field as specified above) and signal group in option window of each signal head in Vissim. But the same work can be done in the M-File. When choosing this way, it is necessary to keep on mind the order of planting them. Assigning in M-File follows their IDs – it does with all objects.

For example, if there are four signal heads and two signal controllers and the aim is to set them for the second controller and divide into two signal groups, the procedure could look like this:

```
headl_1=vnet.SignalHeads.ItemByKey(1);
new1_sg='2-2';
headl_1.set('AttValue', 'sg', new1_sg);
headl_3=vnet.SignalHeads.ItemByKey(3);
new2_sg='2-2';
headl_3.set('AttValue', 'sg', new2_sg);
headl_2=vnet.SignalHeads.ItemByKey(2);
new3_sg='2-1';
headl_2.set('AttValue', 'sg', new3_sg);
headl_4=vnet.SignalHeads.ItemByKey(4);
new4_sg='2-1';
headl_4.set('AttValue', 'sg', new4_sg);
```

The first number specifies the signal controller and the second one the signal group. The exact formula with hyphen goes for COM but later in the source code in *.inpx file appears as *sg="2 1"* and so on. It is stored inside a *<signalHeads>* tag.

To set exact signal group to some specific state, it has to be reached by the hierarchy:

```
SignalController = vnet.SignalControllers.ItemByKey(1);
SG(1)=SignalController.SGs.ItemByKey(1);
```

Then, the state of signal group can be changed (the first one because of the `ItemByKey(1)` in the second row):

```
SG(1).set('AttValue', 'State', 3);
```

Several values can be filled in the brackets. `3` stands for green – vehicles can move. Possible states are specified in Table 1.

Table 1: State values of signal groups (source: [1])

Member	Value
SignalizationStateAlternatingRedGreen	10
SignalizationStateAmber	4
SignalizationStateFlashingAmber	7
SignalizationStateFlashingGreen	9
SignalizationStateFlashingRed	8
SignalizationStateGreen	3
SignalizationStateGreenAmber	11
SignalizationStateOff	5
SignalizationStateRed	1
SignalizationStateRedAmber	2
SignalizationStateUndefined	6

It is also possible to type `'GREEN'` and so on instead of numbers in the brackets. Unfortunately, this information can be found in the official example M-File, laying on the hard drive after installation of Vissim - *COM_examples.m*, but not in the product help. There is only the table shown above with numbers.

2.3.1 Fixed Time

If it is not necessary to create own signal control in Matlab, there is a powerful software for creating signal control with fixed time (cycle time is determined and it does not fluctuate), it is the Vissig module for Vissim. It complements the phase-based fixed time control by additionally providing stage-based fixed time signal control. Vissig contains a graphical editor for defining stages and interstages. Signal program creation in Vissig can be found in Figure 6 for illustration. Even if the control would be done by Matlab, the second item in the left list (*Signal group*) needs to be set as it was commented above.

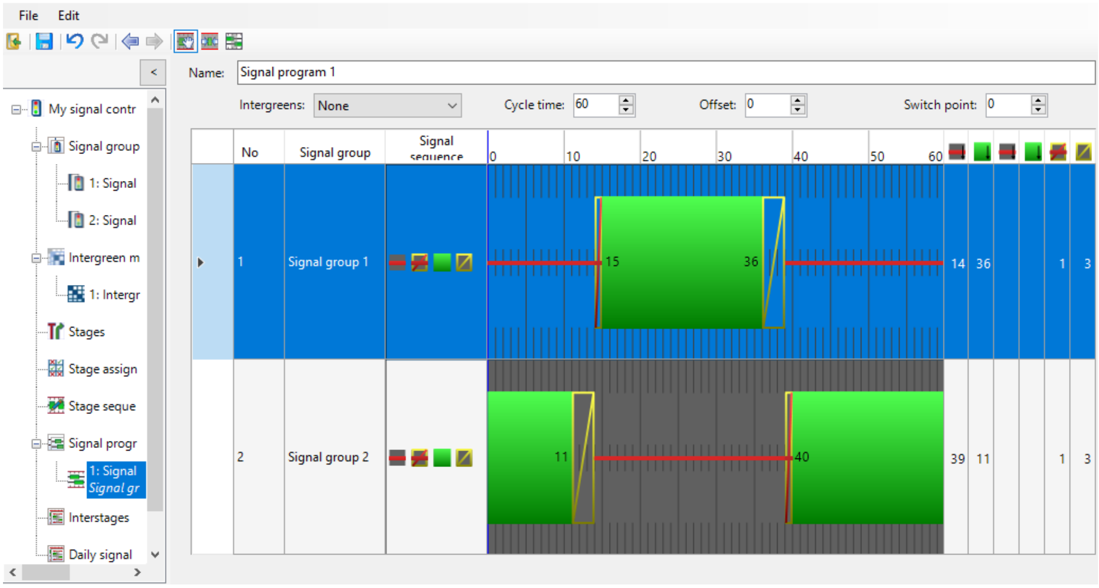

Figure 6: Vissig environment

说明书

If the aim is to run own signal program from Matlab, instructions are here. Most importantly, some time verification must exist. Signal programs are managed to work with whole seconds. Best Matlab function for such a verification is *rem* as a condition for *if* function: 整秒

```
verify = 1;
for i=0:(period_time*step_time)
sim.RunSingleStep;
if rem(i/step_time, verify)==0
% commands here
end
end
```

The `rem` stands for remainder after division. Here, the division `i/step_time` delivers actual simulation second with one decimal digit number. And if the reminder after dividing this number by 1 (variable `verify` is set to `1`) is equal to zero, the whole second is verified and can be added to a current length of stage. Inside of the *if* cycle (ending by the first *end*), some commands should follow. In the Matlab environment, when *%* is being typed, comment can follow with no influence to the program. It also turns the colour to green for better view.

For an example of own signal plan with fixed time, a simple four-way intersection was created. The model created in Vissim can be found in Figure 7. As it is clear from the picture, the net is created with the ability to travel within all directions. Signal heads are planted in the network as well as vehicle routing. Violet stripes represent starts of routing and targets are turquoise. Red and green areas in the middle are conflict areas. Those are better to set in Vissim as well. Signal heads are set in order: bottom-top-left-right. Numbers of Signal heads correspond to this order. Signal groups are also assigned in Vissim, but it can be easily done as it is specified above. Vehicle inputs are not visible in the picture, those are black stripes at the beginning of links behind edge of the image. But their numbering correlates with numbering of signal heads.

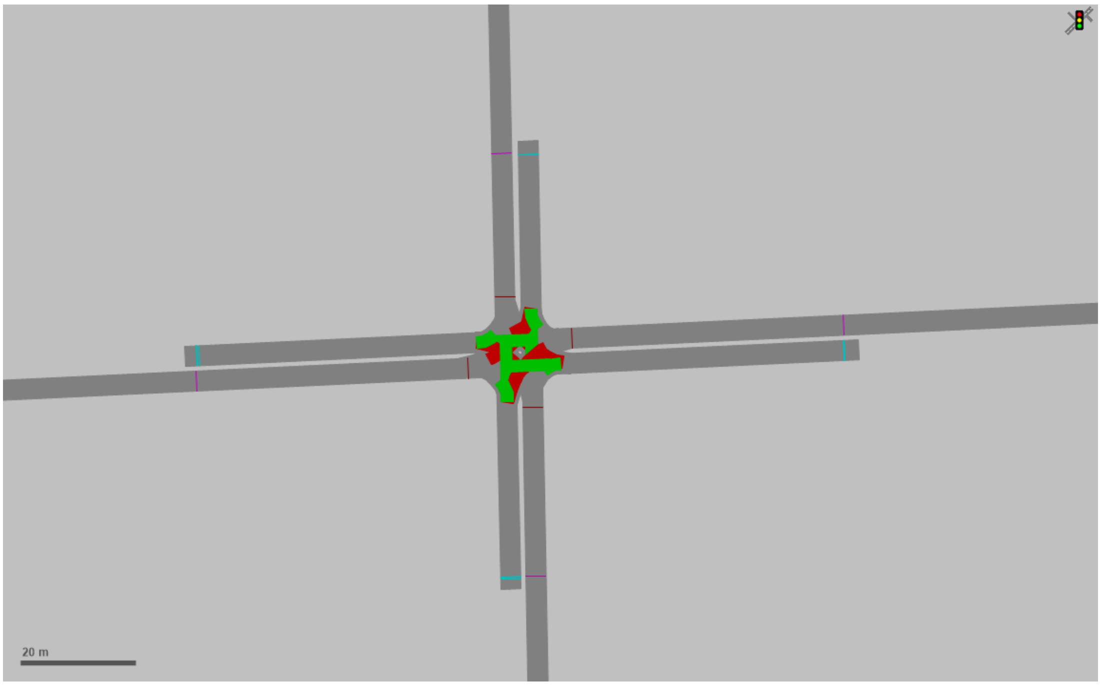

Figure 7: Vissim model for fixed time signal plan

In the M-File, there are specifications such as model placing and clearing the environment first, following by setting shortcuts (`vnet` and `sim`). Then there are named signal groups:

```
SignalController = vnet.SignalControllers.ItemByKey(SC_number);
SG(1)=SignalController.SGs.ItemByKey(1);
SG(2)=SignalController.SGs.ItemByKey(2);
SG(3)=SignalController.SGs.ItemByKey(3);
SG(4)=SignalController.SGs.ItemByKey(4);
```

Next rows belong to simulation settings:

```
period_time=3600;
sim.set('AttValue', 'SimPeriod', period_time);
step_time=10;
sim.set('AttValue', 'SimRes', step_time);
max_cores=4;
sim.set('AttValue', 'NumCores', max_cores);
```

Then some variables are declared:

```
verify = 1;
tsg1 = 17;
tsg2 = 17;
tsg3 = 17;
sg1_time = 0;
sg2_time = 0;
sg3_time = 0;
t12 = 0;
t23 = 0;
t31 = 0;
stage = 1;
```

The first variable is for the verification of whole seconds, three next specify lengths of stages. There are three of them, signal groups *3* and *4* on side roads could have been merged, because they fulfil the definition of showing exactly the same state in each moment (the same switching on time, duration and interstage), but it should be always related only to one exact intersection entrance, which it does not. In this case, they are in the opposite directions. Other six variables serve to counting time of stage (e.g. `sg1_time`) or interstage (e.g. `t12`). In the last variable (`stage`) there is stored the starting stage and later, as the program starts, the current stage/interstage.

The next part of the program is simulation with the signal plan itself, it is stored in Table 2. Numbers on the left side are there only for better orientation.

Table 2: Signal plan with fixed time

```
for i=0:(period_time*step_time)
sim.RunSingleStep;
if rem(i/step_time, verify)==0
       if stage == 1
       SG(1).set('AttValue', 'State', 3);
       SG(2).set('AttValue', 'State', 1);
       SG(3).set('AttValue', 'State', 1);
       SG(4).set('AttValue', 'State', 1);
       sg1_time = sg1_time+1;
       end
       if stage == 2
       SG(1).set('AttValue', 'State', 1);
       SG(2).set('AttValue', 'State', 3);
       SG(3).set('AttValue', 'State', 1);
       SG(4).set('AttValue', 'State', 1);
       sg2_time = sg2_time+1;
       end
       if stage == 3
       SG(1).set('AttValue', 'State', 1);
       SG(2).set('AttValue', 'State', 1);
       SG(3).set('AttValue', 'State', 3);
       SG(4).set('AttValue', 'State', 3);
       sg3_time = sg3_time+1;
       end
       if stage == 12
           if (t12 == 0) || (t12 == 1)
           SG(1).set('AttValue', 'State', 4);
           SG(2).set('AttValue', 'State', 1);
           end
           if t12 == 2
           SG(1).set('AttValue', 'State', 1);
           SG(2).set('AttValue', 'State', 2);
           end
           if t12 > 2
               stage = 2;
               sg2_time = 0;
           end
        t12 = t12 + 1;
       end
       if stage == 23
           if (t23 == 0) || (t23 == 1)
           SG(2).set('AttValue', 'State', 4);
           SG(3).set('AttValue', 'State', 1);
           SG(4).set('AttValue', 'State', 1);
           end
           if t23 == 2
           SG(2).set('AttValue', 'State', 1);
           SG(3).set('AttValue', 'State', 2);
```

```
49            SG(4).set('AttValue', 'State', 2);
50            end
51            if t23 > 2
52                stage = 3;
53                sg3_time = 0;
54            end
55        t23 = t23 + 1;
56       end
57       if stage == 31
58            if (t31 == 0) || (t31 == 1)
59            SG(3).set('AttValue', 'State', 4);
60            SG(4).set('AttValue', 'State', 4);
61            SG(1).set('AttValue', 'State', 1);
62            end
63            if t31 == 2
64            SG(3).set('AttValue', 'State', 1);
65            SG(4).set('AttValue', 'State', 1);
66            SG(1).set('AttValue', 'State', 2);
67            end
68            if t31 > 2
69                stage = 1;
70                sg1_time = 0;
71            end
72        t31 = t31 + 1;
73       end
74       if sg1_time == tsg1
75            stage = 12;
76            sg1_time = 0;
77            t12 = 0;
78       end
79       if sg2_time == tsg2
80            stage = 23;
81            sg2_time = 0;
82            t23 = 0;
83       end
84       if sg3_time == tsg3
85            stage = 31;
86            sg3_time = 0;
87            t31 = 0;
88       end
89 end
90 end
```

In the beginning, when the simulation is started, first simulation step happens with no light at signal heads. The reason is that a single step is processed before their states are set. But it is just a one tenth of the first second and no vehicle can reach signal heads that fast. This one step shift appears during the whole simulation, but it has no impact on simulation course and results, because cycle time and stages last exactly the same time as it was designed. Simple shifting the simulation step command after the whole second verification function (including setting of signal heads states) cannot solve this issue, because stages cannot be set before the simulation starts.

As the program continues, signal groups are set for each state. Inside of the cycle that is active, every simulation second, one second is added. Until it gets the same length as the total length of stage is. It is being verified in the bottom of the program (row 74 to 88). After verification, current stage is set to interstage and used variables are set to zero, so they could be used again. As this part lays at the end of the program, one simulation second happens before it has impact to signal heads. When it comes to interstage (row 25), it sets previous signal group with green signal to red-amber and it remains for two seconds. For the third second it turns red and next signal group turns amber. As soon as this procedure is done, next stage is called in which the signal group turns green from amber. Since this moment, the whole process repeats, just for different signal group. The green signal changes from buttom to top road and then side roads left and right get the signal at the same time. The order of stages is clear in Figure 8. The *F* stands for stage, the first signal group is *VA*, second one *VC*, then *VB* and the fourth is *VD*.

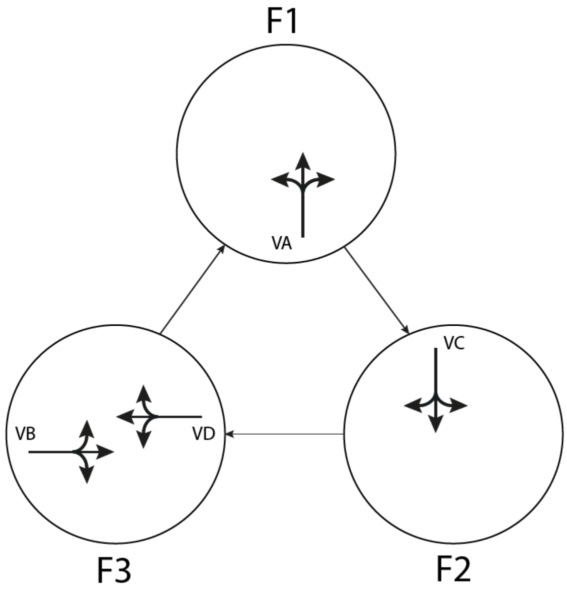

Figure 8: Stages of fixed time signal plan

Durations of interstages are not calculated from any terrestrial conditions, maximum vehicle speed and so on. Values are only sudgested for demonstrative purposes as it is typical for these papers. The cycle lasts for 60 seconds. It is the summary of stages durations and all interstages, hence (17*3)+(3*3)=60.

As it was mentioned before, some controller has to be present to be able to use signal heads. When the talk is about Vissig with its own signal plan created, it can still be used more powerfully with the COM interface. That is thanks to the ability to create more signal programs within one signal controller.

It could work for example in a respond to increased volume of vehicles in one direction. The second program would contain longer green signal for this way. Some detector would check the density or intensity of traffic regularly and when some threshold value is exceeded, signal program would be changed. There is a simple procedure to do that:

```
SignalController = vnet.SignalControllers.ItemByKey(1);
new_signal_program = 2;
set(SignalController, 'AttValue', 'ProgNo', new_signal_program);
```

2.3.2 Dynamic Control

There are many ways of using the dynamic control. It can be done by changing length of the green signal, by changing order of phases or by adding phase by call and so on. When regarding the changing order or adding extra phase, these could be probably created using Vissig and the COM. Vissig would contain the special phase or different order inside another one signal program, that can be changed as shown. And the changing could be dependent on some special traffic situation. Getting the detector state thanks to the COM will guarantee an impulse for the change.

But when the changing length of the green signal is being considered, there is only minimal and maximal length of the green signal specified. So, it cannot be done by changing the signal program. Even creating more signal controllers would not work here. One possibility is to use VisVAP to create such a control.

VisVAP (Visual VAP) is an easy to use tool for defining the program logic of VAP signal controllers as a flow chart. All VAP commands are listed in a function library. The export function allows users to generate *.*VAP* files, where the control logic is saved. During simulation runs, actual detector variables are retrieved from the simulation and processed in the logic, thanks to that, the dynamic control can exist. An example of dynamic control developed in VisVAP environment is in Figure 9. Conditions based on detector values are specified in the right box called *Expressions*. Parameters as maximal lengths of stages and detector threshold value are stored in the top right box.

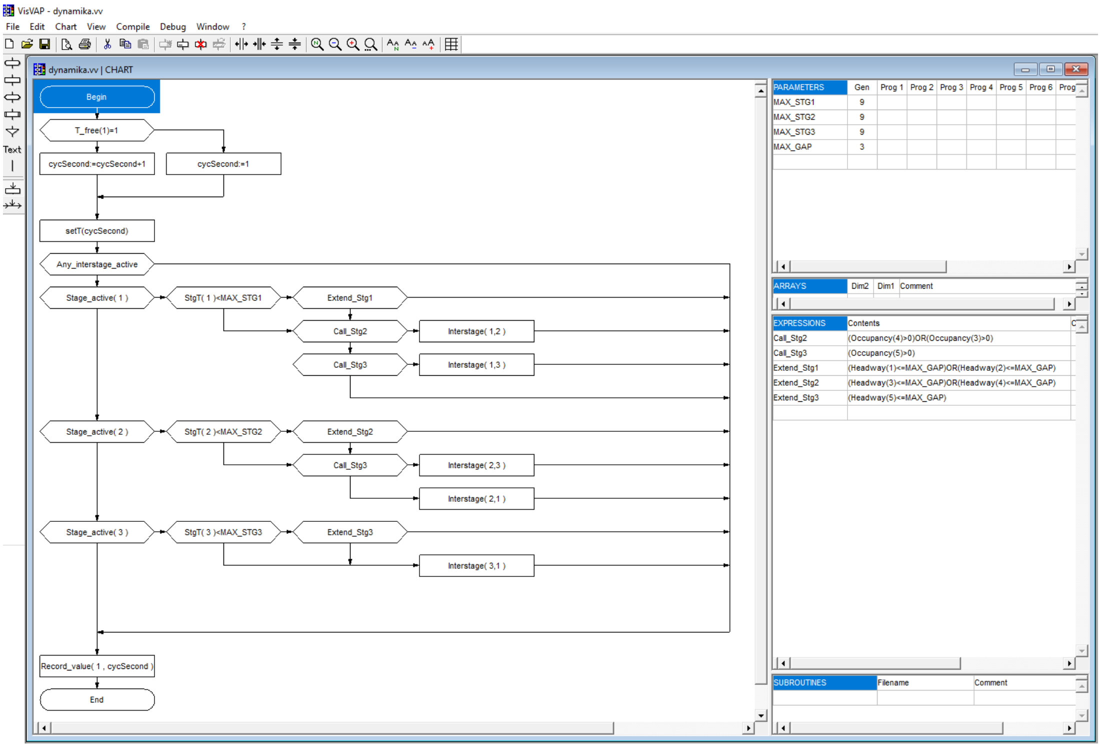

Figure 9: Dynamic control in VisVAP

But this module is not delivered with Vissim automatically and there is a possibility to avoid it. Again, via COM interface. First of all, it is important to know how to get data from detectors. After planting the detector into some link, it can be accessed by:

```
SignalController = vnet.SignalControllers.ItemByKey(1)
dets=SignalController.Detectors;
det_all=dets.GetAll;
det_1=det_all{1};
```

Detectors are also paired with signal controllers, it can be done in Vissim or as well in Matlab. If there are more than one detector, function `GetAll` can be used. It can be used also for assigning signal heads and so on. Just the type of brackets is different here. Specific data can be obtaining from detector for example by:

```
det_1.get('AttValue', 'GapTm')
det_1.get('AttValue', 'Detection')
det_1.get('AttValue', 'Occup')
det_1.get('AttValue', 'Presence')
det_1.get('AttValue', 'Impulse')
det_1.get('AttValue', 'VehSpeed')
```

The control based on gap time is used in a practical example. Vissim model of this example is in Figure 10. The `GapTm` gives a number, which corresponds with the time that was spent between the first vehicle left the detector and the following vehicle entered the detector.

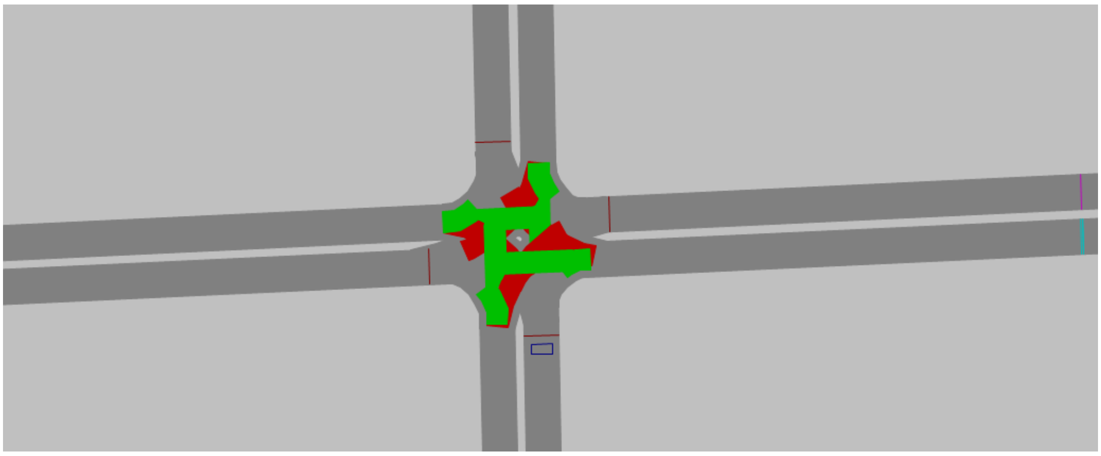

Figure 10: Vissim model for dynamic control

It is the same layout as the example for fixed time with one difference. In front of the first signal head, there is a detector placed. The start of the program is also the same, only extended by some variables for using the detector:

```
tsg1max = 37;
claim1 = 1;
maxgap = 1.8;
```

The variable `tsg1max` sets the maximal length of green signal for stage 1, it is 37 seconds. Next variable is for providing claim of the detector to prolong stage 1. The last one value specifies the threshold value for the detector to decide if the stage will be prolonged. The time gap has to be 1.8 second or less to be able to prolong the stage. Also, there are some changes inside of the simulation:

```
for i=0:(period_time*step_time)
sim.RunSingleStep;
```

```
if (stage == 1) && (sg1_time >= (tsg1 - 3))
 gap1=det_1.get('AttValue', 'GapTm');
 if gap1 >= maxgap
     claim1 = 0;
 end
end
if rem(i/step_time, verify)==0
% commands here
end
end
```

This procedure guarantees that the state of the detector (gap time between vehicles) starts to be checked three seconds before the standard duration of stage 1 is over. Since this time, whenever the value exceeds the maximum gap time, the claim for prolonging the state expires. It is checked every simulation step – 10 times per simulation second, when it is relevant. The program follows the same way as the previous for fixed time does. Next change is at the end of *interstage 31* (from stage 3 to 1), right before the stage 1 starts again, the variable for claim is set back to 1, to be able to prolong the stage 1 again: `claim1 = 1`.

The last difference in the program is at the end, where the duration of current stage is checked in behalf of interstage:

```
if (sg1_time >= tsg1) && ((claim1 == 0) || (sg1_time >= tsg1max))
     stage = 12;
     sg1_time = 0;
     t12 = 0;
end
```

When the maximal length of stage 1 is achieved or the claim for prolonging expires, *interstage 12* is called.

2.3.3 Evaluation of Signal Plan Creation

In the following chapter, a little bit different approach is chosen for the signal control, so that states are not being overwitted each second with the same value but they persist until the interstage is required. It has no influence on driving behaviour only the code is more elegant. But in these papers, it is not delivered in any modular way. It can be done partially in order that some parameters, such a length of cycle and lengths of green signal and so on, could be inputted. But for different number of signal groups and phases, signal heads would have to be assigned and it would be difficult to create such interface when there is the Vissig interface working well.

Unfortunately, just for assigning signal heads to lanes, Vissig or VisVAP have to be present. Vissim does not consider controlling signal heads over COM as an external signal controller. It would have to contain a *.dll library on hard drive.

Chapter 3

Optimization in Model Testing and Control System Development

In this chapter, a representative situation is chosen to demonstrate both, more complex control system and optimization in model testing. The control system includes two intersections with coordination and the optimization lies in effective testing of several scenarios for one situation within one program execution. The benefit is in getting of results at one time without the necessity of adjusting the model between each scenario or without the need of having several versions of one model and executing them separately one after another.

3.1 Control System Development

As it was mentioned before, a concrete situation is chosen to represent more complex control system. Even though it is a real situation, many elements are let out for simplification. And some data is estimated and fictional. The aim is not to demonstrate in detail current situation or to come up with concrete improvement of a traffic situation in the area. It is about showing possibilities of using Matlab to control Vissim on not only theoretical level.

3.1.1 Concrete Situation

The situation was selected on the basis of having some traffic data available. It is a part of city Děčín (Czechia). Wider relations are shown in Figure 11. The city is situated in a district of the same name (inside the red shape). There are more than fifty thousand inhabitants and it is located close to the border with Germany on the north (the map is north oriented). It connects several villages and towns from east to a highway (D8) with continuity to Prague or Dresden (Germany). It also links northern and eastern places with Ústí nad Labem. As it is the last city on the river Labe in Czechia, it dominates with river harbour. The city is an important rail intersection as well. All together generates significant number of trucks.

Figure 11: Situation with wider relations (source: https://mapy.cz)

A closer detail of processed area is visible in Figure 12. Teplická street in the top left corner continues to highway and Teplice. On the right (eastern) side, street Ústecká meets road marked as E442. It leads from Ústí nad Labem through Nový Bor to Liberec. It is noticeable from the previous figure (except for the city Liberec, it lies too far from Děčín).

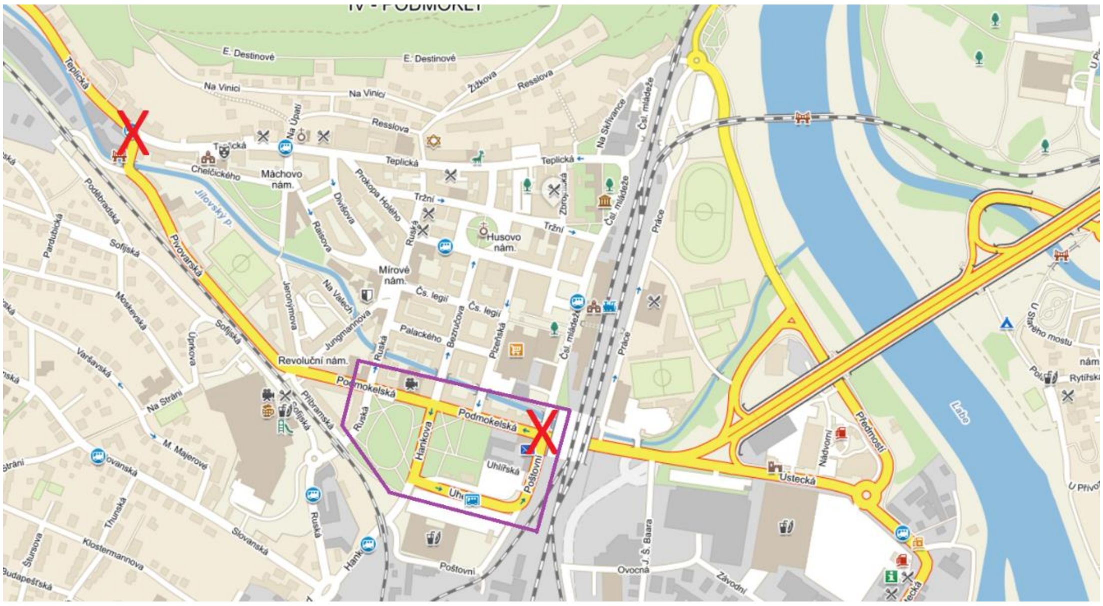

Figure 12: Closer detail of the situation (source: https://mapy.cz, edited)

The Vissim model is constructed from a smaller area than it is pictured in the Figure 12. But the figure shows main direction from the area and it contains intersections where a traffic survey took part recently from which some data was used. The survey relates to intersections marked by red X. The model consists of roads inside the violet shape.

3.1.2 Model and Input Data

3.1.2.1 Input Data

There are four crossings controlled by traffic lights in the selected area. Three of them are there only for pedestrian crossing. But for the simplicity of demonstrative control, pedestrians were left out. That is why only two traffic light controlled intersections are created in the Vissim model below. For better observation of junctions and traffic lanes divide, there is a satellite shot in Figure 13. It consists of several shots merged together to achieve a better resolution background for creating the model in Vissim.

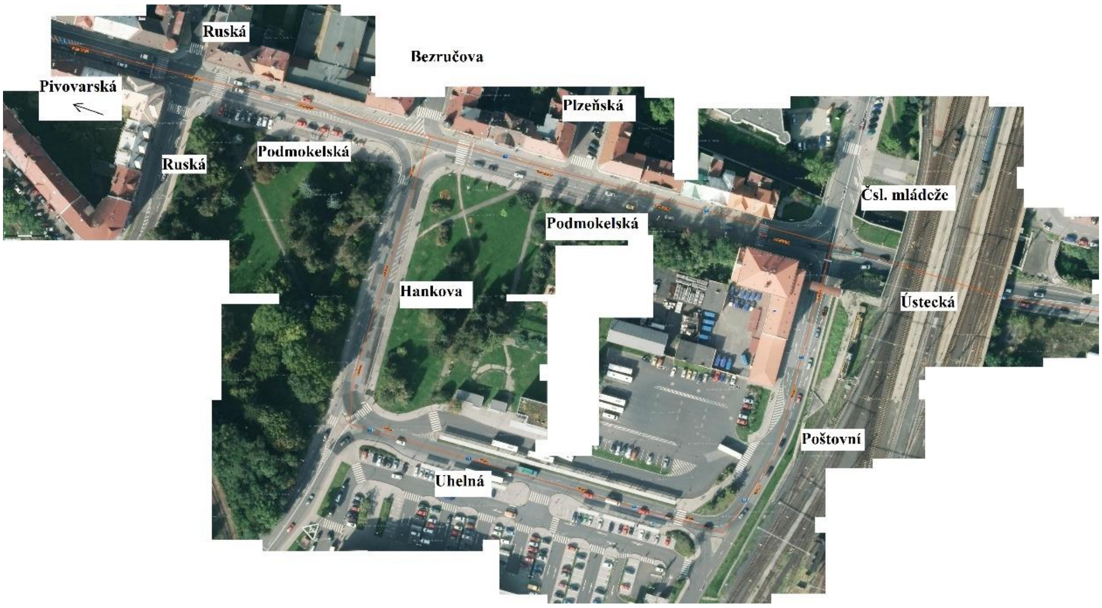

Figure 13: Background for the model (source: https://mapy.cz, edited)

In the direction from street Ústecká to street Pivovarská, there is a coordination of green signals. After the simplification, it is regarding only the intersection Ústecká-Poštovní-Podmokelská-Čsl. mládeže and the intersection Podmokeská-Ruská (both streets do not change name behind the intersection, only Podmokelská turns Pivovarská later). For the purposes of this thesis, the Department of Transport Telematics provided the survey from the first intersection (Ústecká-…) and intersection Teplická-Pivovarská (Figure 12). So there is a real input data available for the first junction. The input data for the last junction had to be adjusted from the survey of Teplická-Pivovarská. Other inputs to the area were estimated with respect to traffic relevance of regarded zones. A brief inspection of public transport routes took part in the estimation as well.

The survey took time from seven p.m. till eleven p.m. and numbers of vehicles were counted in five minute intervals. Every estimated input was filled by values with respect to these intervals. Since the street Bezručova and Ruská behind the junction leading north are one-ways from the area out, these took part only in the vehicle routings (will be discussed later). And the street in bottom right corner (Figure 13) was left out during building the model, because it serves only for the shopping centre attached to the parking lot, for vehicles to have the ability of getting back south without participating in the Podmokelská street. As it is clear from the Figure 12, the first section of Podmokelská street is a one-way from east to west and the opposite direction is provided by one-ways Hankova-Uhelná-Poštovní (in this direction). Because of these facts, estimated inputs were set only for streets Hankova, Ruská (both from bottom left in the previous figure) and Plzeňská. All of these two streets associate mainly local, personal together with public, transportation (local and from neighbouring villages).

There were eight categories specified for the survey, it was: personal vehicles, vans, light trucks, heavy trucks, trucks with trailer, buses, public transportation buses and motorcycles. Again, for the simplicity, they were merged into three categories: personal vehicles (personal vehicles with motorcycles), trucks (vans, light and heavy trucks together with trucks with trailer) and buses (buses with public transportation vehicles).

These inputs are preserved in five minute intervals, just simply multiplied to get hourly intensities for Vissim. Values are stored in the text file *vehinsmatrix.txt*, each column represents one time interval and each row represents inputs of one vehicle class to a concrete source link. The exact order will be specified in the next subchapter.

Unfortunately, the traffic survey was not the areal one. It means that routing is available always only within one concrete junction. For example, licence plates would have to be registered to be able to pair them with other junctions or links for better view of how do the vehicles behave in the network.

In this case, routing was set for each possible path with divided vehicle classes (some paths were set only for one or two categories). For example, it was supposed that no vehicle would travel from the source of Ruská street to Hankova street destination, or that there would be no trucks coming from Čsl. mládeže to Ruská south and so on. The traffic survey was used as much as possible in setting at least proportion of vehicle classes within one path. Then each input value was divided into percentage proportion for each possible path from this particular source. The percentage proportions were estimated.

These routing values are stored in the text file *roumatrix.txt*. It also respects specified time intervals, so it can be changed with the same period as vehicle inputs. And each row in this file represents a proportion of one vehicle input for one particular path for one class.

3.1.2.2 Model Construction

Background

For the construction of the model, first the background image was loaded to a new Vissim file. To set own image, there is a *Background Images* item in the *Network Objects* section. After loading the image, there is an option to set scale, when user right clicks into the image. Then by left click and hold during movement, some distance can be chosen and after releasing the left button, a scale window appears. There should be specified the real distance in meters of the selected distance.

Infrastructure

The next step is to draw the infrastructure. All links and connectors. Just before planting signal heads, signal controllers have to be specified. There should be created two empty signal groups in the first signal controller. Then the second signal controller needs to be set and it will contain two signal groups as well (it should contain three, but during the implementation, opposite directions were set to the same signal group, because they do not differ in any second, even though it is not exactly a correct practise). The control will be discussed later. The first signal head belongs to the end of Ústecká street. The second one to the end of Poštovní street. Both of them belong to the first signal controller and it should be set to the first and second signal group in the same order during setting them to the network. The last crossing has two signal heads in each direction of the street Podmokelská, set to the second signal controller and the first signal group (should be two different, but it would have to be implemented as well in the Matlab program). The last two signal heads go to Ruská street from south to north and east way. Here, the signal group is set to the second one within the second signal controller. The whole model is visible in the Figure 14. The exact position of each signal head is there noticeable as well.

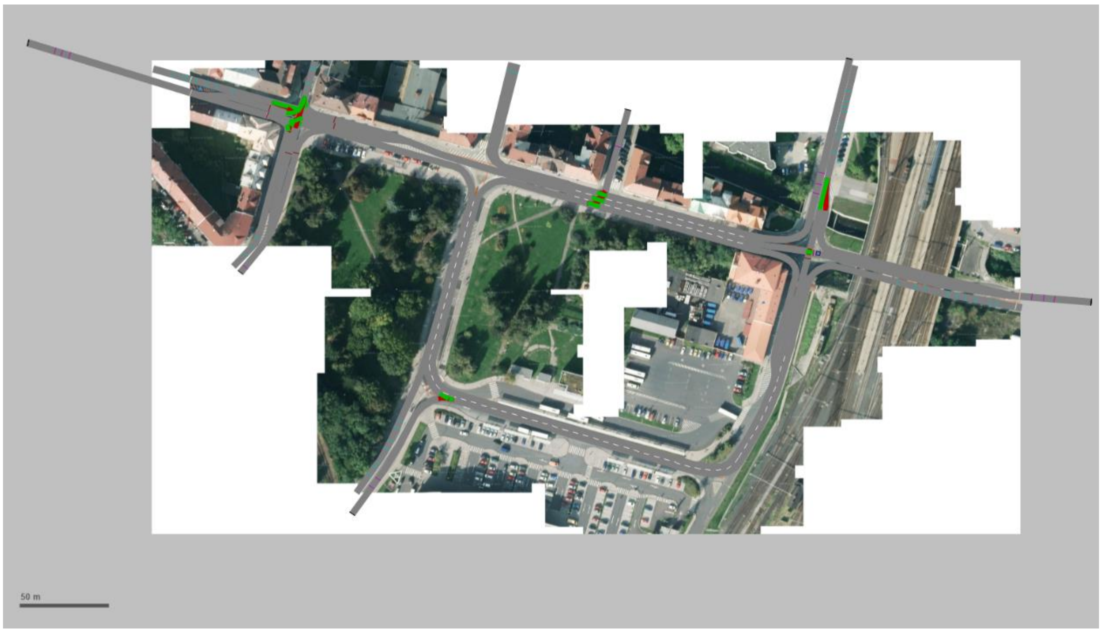

Figure 14: Vissim model of a specific area in Děčín

Vehicle inputs

Vehicle inputs start at Ústecká, there are the first three of them (in the relevant order to personal vehicles, trucks and buses). There are always starts of vehicle routings in the same amount and for the same vehicle classes as the inputs are. Next three inputs are at the start of Podmokelská street. Every input corresponds to a row in the *vehinsmatrix.txt* file, it means, that the first row is for personal vehicles from Ústecká street, the second row is for trucks from the same direction and the third one is for buses. Then again, the fourth row is for personal vehicles, but from the opposite side of the area. Inputs seven to nine are situated at the start of Čsl. mládeže street. Then from the Hankova street, there are only inputs for personal vehicles and buses. Input number twelve is the only one from the street Ruská and it belongs to personal vehicles. The same applies for input number thirteen in the street Plzeňská.

Vehicle compositions

Then there are three new vehicle compositions supposed to be created in Vissim. It is all right to let them filled by default, it will be set properly in Matlab.

Conflict areas

It is clear from the picture, that there are conflict areas set with respect to traffic signs setting preferences in the area.

Routing

The complete routing is better to finish in Vissim as well. There have to be a set of paths for every input (for separated vehicle classes). In the total, there are 38 of them. The order is also important, because later the rows from *roumatrix.txt* file will be assigned. It can be checked in *Static Vehicle Routing Decision / Static Vehicle Routs* tables in Vissim. After clicking at any, it shows the path in the model.

Evaluation

There was nothing said about the evaluation yet. To get some evaluation is a purpose of most simulations. For these papers, evaluation of travel times, queues and delays are taking place. A great advantage of Vissim is that it can make evaluations even for specific vehicle classes.

To activate these evaluations, it has to be specified in Vissim first. By clicking the *Evaluation* -> *Configuration…*, new window appears. It is clear in the Figure 15. Desired vehicle classes have to be selected here to get the data later. Boxes next to required functions have to be checked and the last column with the interval is also very important. If the number stays unchanged as *99999*, it means that the result will be one average value. Vissim makes always averages, number of results depends on how big the interval is. In this thesis, thirty second intervals were chosen. It means that there will be always *period time/30* values for each function. By clicking the *More…* button, regarding for example queue counters, another window appears and it can be set which speed interval will be counted as a queue. The default beginning is if the speed drops below 5 km/h and the default end of counting is when the vehicle accelerates above 10 km/h.

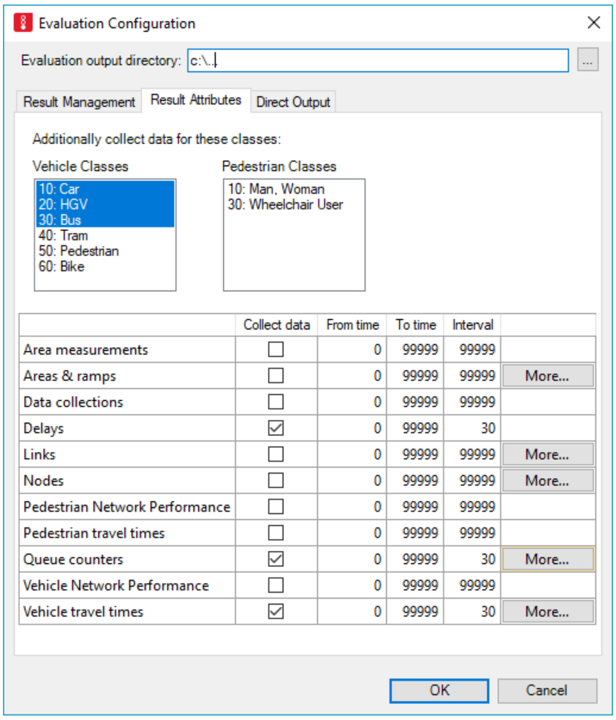

Figure 15: Evaluation configuration

When this is specified, there should be vehicle travel time measurements added to the network. There are two of them used in this model. First of them starts on the east side, where Ústecká meets the background image and ends on the west side (the same situation for Podmokelská street). And the second one occupies the same spots but in opposite direction. It will measure vehicles traveling throw the network from east to west and from west to east.

For the delay measurements, there is not the same procedure, because the delay is counted from the same section as the travel time. The procedure is to click on *Lists* -> *Measurements* -> *Delay Measurements*, then by adding there should be two items in the table and in the right column (*VehTravTmMeas*) it should be assigned to those sections.

When the talk is about the queue counter, there is a simple tool in the left list again. It is placed at the stop line at the street Ústecká and at the stop line at the street Podmokelská. It will measure queues from the start of those two streets till the first traffic lights.

M-File

The first several rows are still almost the same. There must be clearing of the Matlab environment. Then specifying the location of Vissim files and defining access on lower levels of the hierarchy for easier future declaration (*sim*, *vnet*,…). Then there should be some simulation setting and as mentioned before, defining of the vehicle composition:

```
Composs= vnet.VehicleCompositions.GetAll;
Rel_Flows=Composs{2}.VehCompRelFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        100);
```

```
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   50);
Rel_Flows=Composs{3}.VehCompRelFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        200);
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   50);
Rel_Flows=Composs{4}.VehCompRelFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        300);
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   50);
```

The first command load all compositions (three new were added manually in Vissim). There are four of them in total, when counting the first – default. Since the default one is mixture of two classes, it will be always skipped by starting with number two. So, the first new category consists only of personal vehicles (attribute `VehType` is `100`). `200` stands for trucks and `300` for buses. The second attribute (`DesSpeedDistr`) means desired speed distribution, which is set to fifty kilometres per hour for all categories.

Continuing in the M-File, these compositions are assigned to concrete vehicle inputs:

```
vehins=vnet.VehicleInputs.GetAll;
vehins{1}.set('AttValue', 'VehComp(1)', 2);
vehins{2}.set('AttValue', 'VehComp(1)', 3);
vehins{3}.set('AttValue', 'VehComp(1)', 4);
vehins{4}.set('AttValue', 'VehComp(1)', 2);
vehins{5}.set('AttValue', 'VehComp(1)', 3);
vehins{6}.set('AttValue', 'VehComp(1)', 4);
vehins{7}.set('AttValue', 'VehComp(1)', 2);
vehins{8}.set('AttValue', 'VehComp(1)', 3);
vehins{9}.set('AttValue', 'VehComp(1)', 4);
vehins{10}.set('AttValue', 'VehComp(1)', 2);
vehins{11}.set('AttValue', 'VehComp(1)', 4);
vehins{12}.set('AttValue', 'VehComp(1)', 2);
vehins{13}.set('AttValue', 'VehComp(1)', 2);
```

There are 13 of them in total and this assigning corresponds with the previous section called *Vehicle inputs*. In case that there were many inputs, it would be possible to fill them by function in a *for* cycle. Values would loaded and assigned prom a text file for example.

Next part of the program belongs to vehicle routing. If it is not manually assigned in the Vissim during the process of creating all paths for each vehicle input (and category), it needs to be done here:

```
routingsource=vnet.VehicleRoutingDecisionsStatic.GetAll;
routingsource{1}.set('AttValue', 'AllVehTypes', 'false');
routingsource{1}.set('AttValue', 'VehClasses', 10);
routingsource{2}.set('AttValue', 'AllVehTypes', 'false');
routingsource{2}.set('AttValue', 'VehClasses', 20);
routingsource{3}.set('AttValue', 'AllVehTypes', 'false');
routingsource{3}.set('AttValue', 'VehClasses', 30);
routingsource{4}.set('AttValue', 'AllVehTypes', 'false');
routingsource{4}.set('AttValue', 'VehClasses', 10);
routingsource{5}.set('AttValue', 'AllVehTypes', 'false');
routingsource{5}.set('AttValue', 'VehClasses', 20);
routingsource{6}.set('AttValue', 'AllVehTypes', 'false');
routingsource{6}.set('AttValue', 'VehClasses', 30);
routingsource{7}.set('AttValue', 'AllVehTypes', 'false');
routingsource{7}.set('AttValue', 'VehClasses', 10);
routingsource{8}.set('AttValue', 'AllVehTypes', 'false');
routingsource{8}.set('AttValue', 'VehClasses', 20);
routingsource{9}.set('AttValue', 'AllVehTypes', 'false');
```

```
routingsource{9}.set('AttValue', 'VehClasses', 30);
routingsource{10}.set('AttValue', 'AllVehTypes', 'false');
routingsource{10}.set('AttValue', 'VehClasses', 10);
routingsource{11}.set('AttValue', 'AllVehTypes', 'false');
routingsource{11}.set('AttValue', 'VehClasses', 30);
routingsource{12}.set('AttValue', 'AllVehTypes', 'false');
routingsource{12}.set('AttValue', 'VehClasses', 10);
routingsource{13}.set('AttValue', 'AllVehTypes', 'false');
routingsource{13}.set('AttValue', 'VehClasses', 10);
```

Every several paths have the same source, that is why only 13 needs to be set. These vehicle classes correspond to vehicle types, they are just a one-tenth lower. This procedure enables to route each vehicle class separately. Before specifying vehicle classes, checkbox for all vehicle types needs to be unchecked. Since the unchecking (setting value to *false*) regards each source, it can be done by setting multiple attribute value instead of doing it separately as shown:

```
vnet.VehicleRoutingDecisionsStatic.SetMultiAttValues('AllVehTypes',
'false');
```

Then again, some simulation settings take part. It is in the same form as in the previous program.

For the evaluation, some access throw the hierarchy should be declared:

```
vehTTs1 = vnet.VehicleTravelTimeMeasurements.ItemByKey(1);
vehTTs2 = vnet.VehicleTravelTimeMeasurements.ItemByKey(2);
queue1 = vnet.QueueCounters.ItemByKey(1);
queue2 = vnet.QueueCounters.ItemByKey(2);
del1 = vnet.DelayMeasurement.ItemByKey(1);
del2 = vnet.DelayMeasurement.ItemByKey(2);
```

Some variables for storing of obtained values need to be specified. And the best way it to prepare matrixes or vectors of the exact needful sizes:

```
period_meas = 30;
x=period_meas:period_meas:period_time;
if rem(period_time, period_meas) ~= 0
   x(length(x)+1)=period_time;
end
DelayA=zeros(length(x));
DelayB=zeros(length(x));
TTA=zeros(length(x));
TTB=zeros(length(x));
QA=zeros(length(x));
QB=zeros(length(x));
```

The variable `period_meas` has to contain the same number as it is set in Vissim for the measure interval. The *if* function ensures that there will be enough space, even when the last interval is shorter than others. For example, if the period time had been set to 3550 s, the variables would have contained 119 zeros.

The filling by real values happens inside of the simulation, after a measure interval verification:

```
if (i~=0) && (rem((i)/step_time, period_meas)==0) &&
(i<((period_time*step_time)-1))
    TTactual(1,1) = get(vehTTs1,'AttValue',
'TravTm(Current,Total,All)');
    TTactual(2,1) = get(vehTTs2,'AttValue',
'TravTm(Current,Total,All)');
```

```
    Qlenactual(1,1) = get(queue1,'AttValue', 'QLen(Current,Total)');
    Qlenactual(2,1) = get(queue2,'AttValue', 'QLen(Current,Total)');
    DLactual(1,1) = get(del1, 'AttValue',
'VehDelay(Current,Total,All)');
    DLactual(2,1) = get(del2, 'AttValue',
'VehDelay(Current,Total,All)');
```

The first sub attribute defines the specific simulation. Here it is being used only as the current. The second one specifies the value of a specific time interval. It can be set to a value (*1, 2,…*) or an aggregated value of all time intervals of one simulation (*Avg*, *StdDev*, *Min*, *Max*). The last possibility is to fill *Total* as it is used above. It summarizes values from all time intervals till the current one. To get the data for each time interval separately, there is procedure developed for this purpose below. The last sub attribute (if it is available) specifies which vehicle class to show data from. It is set to *All* to get average number through all classes. For example, *10* would get data only for personal vehicles. During the implementation of the program, it seemed that it is impossible to get data from the last interval, because the Vissim alwas prepared new simulation run at the end and there were empty cells. That is why the data from the last interval is being get one simulation step before the end of simulation run (the last condition in the *if* function). But it showed up that this issue could be eliminated in the same window where the evaluation is being set (Figure 15). There is another card called *Result Management* and there is a check box *Automatically add new columns in lists*, it should be unchecked manually. There is probably a way to do it via Matlab, but it is quite difficult to get to all functions just with the *Online help*.

The measure interval verification function continues by getting the real values and setting them into the final vectors:

```
if c ~= 1
    if isnan(TTactual(1,1))
        TT(1,c)=0;
    else
        TTsum(1,1) = TTsum(1,1) + TT(1,c-1);
        TT(1,c) = (TTactual(1,1) - TTsum(1,1));
    end
    if isnan(DLactual(1,1))
        DL(1,c)=0;
    else
        DLsum(1,1) = DLsum(1,1) + DL(1,c-1);
        DL(1,c) = (DLactual(1,1) - DLsum(1,1));
    end
    if isnan(Qlenactual(1,1))
        Q(1,c)=0;
    else
        Qsum(1,1) = Qsum(1,1) + Q(1,c-1);
        Q(1,c) = (Qlenactual(1,1) - Qsum(1,1));
    end
    c=c+1;
else
    if isnan(TTactual(1,1))
        TT(1,c)=0;
    else
        TT(1,c) = TTactual(1,1);
    end
    if isnan(DLactual(1,1))
        DL(1,c)=0;
```

```
    else
        DL(1,c) = DLactual(1,1);
    end
    if isnan(Qlenactual(1,1))
        Q(1,c)=0;
    else
        Q(1,c) = Qlenactual(1,1);
    end
    c=c+1;
end
```

There is the same procedure inside the main cycle for the second direction (variables with *2,1* inside brackets). Here it is left out to save space and keep clarity. The complete procedure is present in the main program in attachments.

This procedure not only fills variables by data from each interval, it also turns non-numerical values to zeros to get homogeneous output. These values appear when there is no data to forward, it happens for example when no vehicle starts a journey through the measure section within some concrete measure interval. These values are originally stored as a *NAN* (not a number). To deal with the first interval, there must exist a declaration of `c=1;` above the simulation start. The second section serves to the first filling. There is no addition of previous state.

The representation of results is done majorly in graphs, using these filled variables and the variable *x* for time sampling. The output representing is shown below, in the section 3.2.3.

There are more interesting sections in the program. The talk is about loading data from text files for vehicle inputs and routings. First columns must be loaded before the simulation start:

```
load vehinsmatrix.txt;
row2 = 1;
column = 1;
for var1 = 1:length(vehins)
    vehins{var1}.set('AttValue', 'Volume(1)',
vehinsmatrix(row2,column));
    if row2 < size(vehinsmatrix,1)
    row2=row2+1;
    end
end

load routmatrix.txt
row = 1;
routing=vnet.VehicleRoutingDecisionsStatic.GetAll;
for var1 = 1:length(routing)
routingx=routing{var1}.VehRoutSta.GetAll;
    for var2 = 1:length(routingx)
    routingx{var2}.set('AttValue', 'RelFlow(1)',
routmatrix(row,column));
        if row < size(routmatrix,1)
        row=row+1;
        end
    end
end
```

These functions fill each vehicle input and each relative routing value. Later, in the simulation section, these functions are repeated, but they lie inside of the time verification which corresponds to the survey intervals (5 minutes). The variable *column* inside of setting commands is increased by one before the assigning. Also, an added condition keep these procedures from overflowing the file dimension. It can be found in the appendix as well.

3.1.3 Default Control

Besides the traffic survey, the faculty department provided a documentation regarding a change of interstage duration for the first interstage (Ústecká - Poštovní). The requirement came from police of Czechia and the aim was to prolong the interstage when vehicles from Ústecká are clearing out the conflict area and vehicles from Poštovní are arriving there. The time from red signal in Ústecká till the green signal in Poštovní was set to four seconds instead of two.

Thanks to this change, the documentation includes the whole signal plan with stages and interstages. Because of the fact that this thesis is not aimed to pedestrians, their stage was simply left out with preserving of vehicle regarding parameters. This stage was integrated only on request. The truth is that more vehicles can go throw the net during one cycle when there are no pedestrians demanding their stage. So it is clear that no results from this model can be used to testify in real about the traffic situation in this area. The only case would be, for example, a study of the traffic situation if the pedestrian crossings were replaced by underpasses or footbridges.

The stage schema for this specific model for the first intersection is in Figure 16.

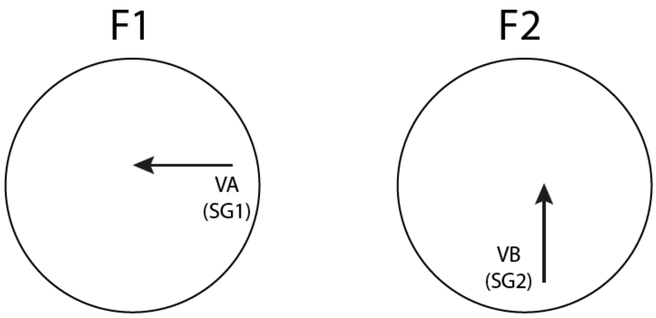

Figure 16: Stages of the first intersection

The cycle time is 60 seconds due to the documentation. The interstage 1.2 takes six seconds and the second one only four. The interstage parts of signal program are visible in Figure 17.

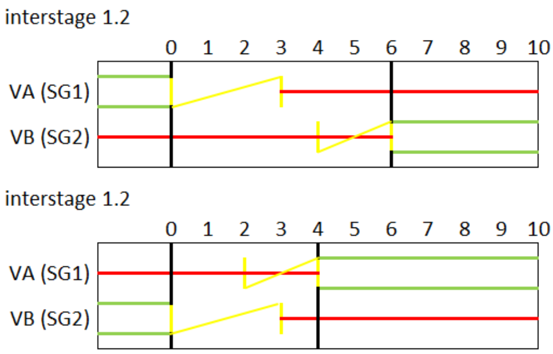

Figure 17: Interstages of signal plan

In this case, the second controlled interstage does not meet with real parameters, because any documentation was available. But since it is sure thing, that these two interstages are in coordination in the east-west direction, it is clear that the cycle must be of the same length as well as the green signal. It is just shifted in time. Other parameters such as signal groups, interstage time table, offset and so on were estimated. The final stage schema for this junction (Podmokelská - Ruská) is in Figure 18.

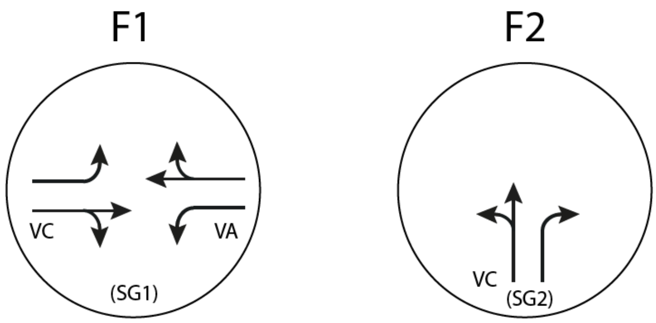

Figure 18: Stages of the second intersection

Here, the Table 3 with interstage time follows:

Table 3: Interstage time

		coming		
	SG	SG1 VA	SG2 VB	SG1 VC
leaving	SG1 VA	x	8	0
	SG2 VB	8	x	8
	SG1 VC	0	8	x

The offset was set to 22s and the interstage parts of signal program for the second controlled interstage are visible in Figure 19.

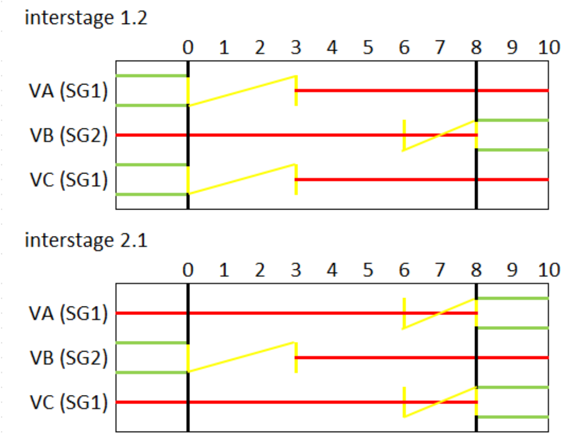

Figure 19: Interstages of the second signal plan

In the M-File, the start of the control looks like this:

```
cycle = 60;
offset = 22;
green(1) = 35;
green(2) = green(1);
stage = [0 0];
change = [0 0;0 0];
terminate = [0 0];
start_time = [0 0];
sim_sec = 0;
for i=0:(period_time*step_time)
    sim.RunSingleStep;
if rem(i/step_time, verify(1))==0
    if stage(1) == 0
        if (start_time(1) ~= 0) && (start_time(1) == terminate(1))
            stage(1) = 21;
        end
        if start_time(1) == 0
            SG(1,1).set('AttValue', 'State', 1);
            SG(1,2).set('AttValue', 'State', 3);
            SG(2,1).set('AttValue', 'State', 1);
            SG(2,2).set('AttValue', 'State', 3);
            terminate(1) = 3;
        end
        start_time(1) = start_time(1) + 1;
        if stage(1) == 21
            start_time(1) = 0;
            terminate(1) = 0;
        end
    end
```

After the variables declaration, the first stage runs. It is a starting stage (stage zero). It takes three seconds and then it continues to interstage 2.1 in the first intersection. The second one remains in this stage for the offset time. The time for the first intersection to remain in this stage should correspond to a value of minimal green signal length. It is more likely being set to five seconds. In that case, it would be just enough to change the value of *terminate(1)* inside of the function from *3* to *5*.

The interstage 2.1 follows right after the zero stage. All values in brackets represent the signal controller by the first number and the signal group by the second number.

```
if stage(1) == 21
   if (start_time(1) ~= 0) && (start_time(1) == change(1,1))
       SG(1,1).set('AttValue', 'State', 2);
   end
   if (start_time(1) ~= 0) && (start_time(1) == change(1,2))
       SG(1,2).set('AttValue', 'State', 1);
   end
   if (start_time(1) ~= 0) && (start_time(1) == terminate(1))
       stage(1) = 1;
   end
   if start_time(1) == 0
       SG(1,2).set('AttValue', 'State', 4);
       terminate(1) = 4;
       change(1,1) = 2;
       change(1,2) = 3;
   end
   start_time(1) = start_time(1) + 1;
   if stage(1) == 1
       start_time(1) = 0;
       terminate(1) = 0;
       change(1,:) = 0;
   end
end
```

Interstage 1.2 looks very similar, there are only twisted signal groups and terminate time and change time are different. The stage 1 is quite easily defined:

```
if stage(1) == 1
   if (start_time(1)~=0) && (start_time(1) == (terminate(1)-1))
       stage(1) = 12;
   end
   if start_time(1) == 0
      SG(1,1).set('AttValue', 'State', 3);
      terminate(1) = green(1);
   end
   start_time(1) = start_time(1) + 1;
   if stage(1) == 12
       start_time(1) = 0;
       terminate(1) = 0;
       startshift = 1;
      end
end
```

The condition to start interstage is reduced by 1, because interstages are defined above stages and so one simulation second must happen with the same state before the interstage starts. The *startshift* has an influence on stage 1 in the second signal controller. It ensures that the end of the stage comes with compliance to the end of stage 1 in the first signal controller with the offset.

```
if stage(2) == 1
   if startshift == 1
       terminate(2) = start_time(2) + offset;
       startshift = 0;
   end
   if (start_time(2)~=0) && (start_time(2) == (terminate(2)-1))
       stage(2) = 12;
   end
   if start_time(2) == 0
       SG(2,1).set('AttValue', 'State', 3);
   end
   start_time(2) = start_time(2) + 1;
```

```
        if stage(2) == 12
            start_time(2) = 0;
            terminate(2) = 0;
            if (maxgreen ~= 0)
                claim1 = 1;
            end
        end
    end
```

The whole signal program is attached as a part of the main program in attachments. This kind of controlling is a bit different from the presented one in previous chapter. The advantage here is that states are not overwritten by the same value but the state remains untouched till it is time to change it.

3.2 Optimization

The optimization in model testing takes part here. It is based on the idea that there is a requirement to test several signal plans within one model. The optimization here lies in the fact, that it can be done with no network and parameters editing between simulation runs. As well as the simulation runs, it can be started just once. Every change or different signal plan can be specified in advance in the M-File and Vissim model can remain in the original form. The results can be then represented in the end for each scenario together in one graph.

3.2.1 Alternative Signal Plan

To test a different scenario within the same model, some alternative signal plan should be created. For these purposes the first alternative scenario has a 70s cycle in behalf of the coordinated direction. Other parameters are the same as in previous signal plan. So only two rows differ:

```
cycle = 70;
green(1) = 45;
```

The third signal plan is based on both of the previous. It includes a dynamic control which is based on time space between vehicles. For this purpose, an inductive loop must be present. It is clear in Figure 14: Vissim model of a specific area in Děčín, that it is already planted in the street Ústecká. Based on the time space, the cycle time can be increased from 60s to 70s. Again, it is regarding only the coordinated direction. The maximal time space can be set by the user, it is shown in the following section. The verification happens always during the stage 1, three seconds before the end of original length (35s). Once the threshold is exceeded, the claim disappears and appears again in the next cycle. When the claim disappears several seconds before the end of original stage length, the interstage starts after the whole original stage terminates. When vehicles are close to each other and the claim does not disappear till the end of prolonged stage (45s), it terminates by this second automatically. During the prolonging, it is always done by adding one second:

```
if (maxgreen ~= 0) && (stage(1) == 1) && (start_time(1) >=
(green(1)-3)) && (start_time(1) < maxgreen)
```

```
    if claim1 == 1
        gap1=det1.get('AttValue', 'GapTm');
        if gap1 > maxgap
            claim1 = 0;
        end
    end
end
if rem(i/step_time, verify(1))==0
    if stage(1) == 1
        if (maxgreen ~= 0) && (start_time(1) >= (terminate(1)-1)) &&
(start_time(1) < (maxgreen-1))
            if claim1 == 1
                terminate(1)=terminate(1)+1;
            end
        end
        if (start_time(1)~=0) && (start_time(1) == (terminate(1)-1))
            stage(1) = 12;
        end
        if start_time(1) == 0
            SG(1,1).set('AttValue', 'State', 3);
            terminate(1) = green(1);
        end
        start_time(1) = start_time(1) + 1;
        if stage(1) == 12
            start_time(1) = 0;
            terminate(1) = 0;
            startshift = 1;
        end
    end
```

For a better performance of this optimized model testing, there is a possibility to increase the traffic by some percentage value for each scenario and run it again:

```
increase = (200/100)+1;
load vehinsmatrix.txt;
vehinsmatrix=vehinsmatrix*increase;
```

Instead of *300*, there is a variable in the program. This will increase the traffic about 200% (to get to total 300%).

As it was already discussed, the optimization lies in the fact, that all of these scenarios are executable by one click with no further setting. There is a simple procedure to run these simulations one after another. The user only needs to fill a matrix `scenario` with desired scenarios.

```
EOS=sum(scenario(:,1))+sum(scenario(:,2));
for j=1:EOS
if scenario(1,1)==1
%...
elseif scenario(1,2)==1
%...
elseif scenario(2,1)==1
%...
elseif scenario(2,2)==1
%...
elseif scenario(3,1)==1
%...
elseif scenario(3,2)==1
%...
end
%...
```

The program gets into the first desired scenario, get parameters and thanks to the *elseif* it will come again to get parameters from different scenario after the simulation happens. When this happens, the value inside relevant *scenario* is increased for not to stuck in one cycle forever.

If there were more different signal groups (they would differ in signal groups for example), signal heads could be easily reassigned to signal groups between simulation runs by Matlab and there is still no necessity to have separated Vissim models and to test them apart.

The most easily way of using this optimization would be if the signal programs were created in VisVap or Vissig. The main program would just reassign signal groups, signal controllers and change some more required data between simulation runs.

3.2.2 Graphical User Interface

When there is a program developed and it is supposed to be used by some user, it is the best solution to create a graphical user interface so that he cannot make changes in the source code. In this case, it is not really necessary, but when considering some extensive testing, it can be useful.

Matlab includes GUI, it can be called by typing *guide* in the command window. It opens a file explorer with a possibility to open blank window or some template. After selecting the default blank window a program for creating graphical components appears. It is quite easy here to put buttons, check boxes, list boxes and so on into the GUI. After saving the file, it creates *.fig. It contains all graphical layout, but it is not executable itself. Respectively it is, but it lets only the user to push buttons but no function proceeds. There is a *.m file created together with the graphical one. It is related to that file and it contains all functions to run the GUI. Each graphical component added into the GUI creates a section in the M-File, where the corresponding function should be specified.

The GUI created for this model is in Figure 20. This is how the GUI looks by default. In the *Scenarios* section, only the first one is picked. User can pick an arbitrary combination or all of them. When he picks also the function for the second run for each scenario in *Increased vehicle inputs* sub section, white field appears between *Set the increase* and *%* with some preset value. The same happens in the sub section *Max gap for detector* when the third scenario (*60-70s*) is picked. It is visible in Figure 21 together with other picked functions. In the *Quick mode* section, user can decide whether he want to observe vehicles in the network or to get results as soon as possible.

In the next section, period time can be edited. The default value is 3600s. Vehicle inputs start time can be set here. It is divided into 5 minute intervals, that is why the unit is in minutes as well. If the value lies somewhere inside the interval, it is always divided by 5 and then rounded to the nearest integer towards infinity. So, when tipping 4 minutes, it will start with the first interval. Number 6 would change it to the second interval and so on.

The random seed value was already discussed, in the last section it can be set. Possibilities of the list box, defining how often to change the random seed, are visible in Figure 21.

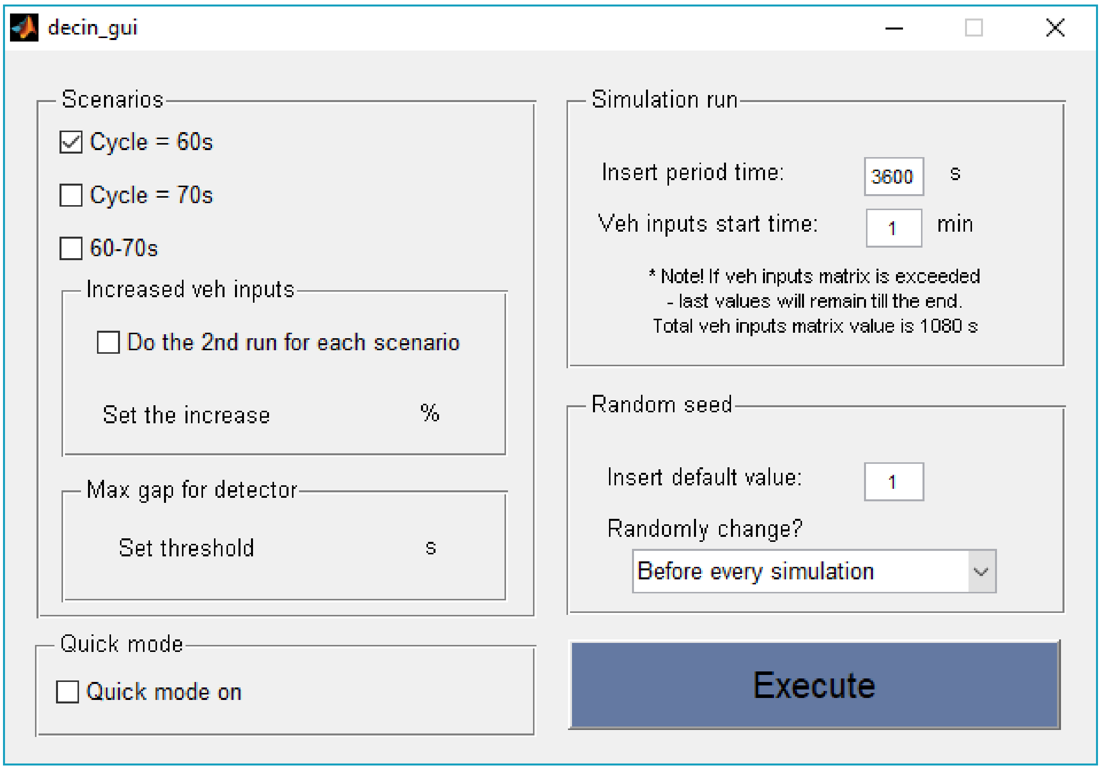

Figure 20: GUI for the Děčín model with default values

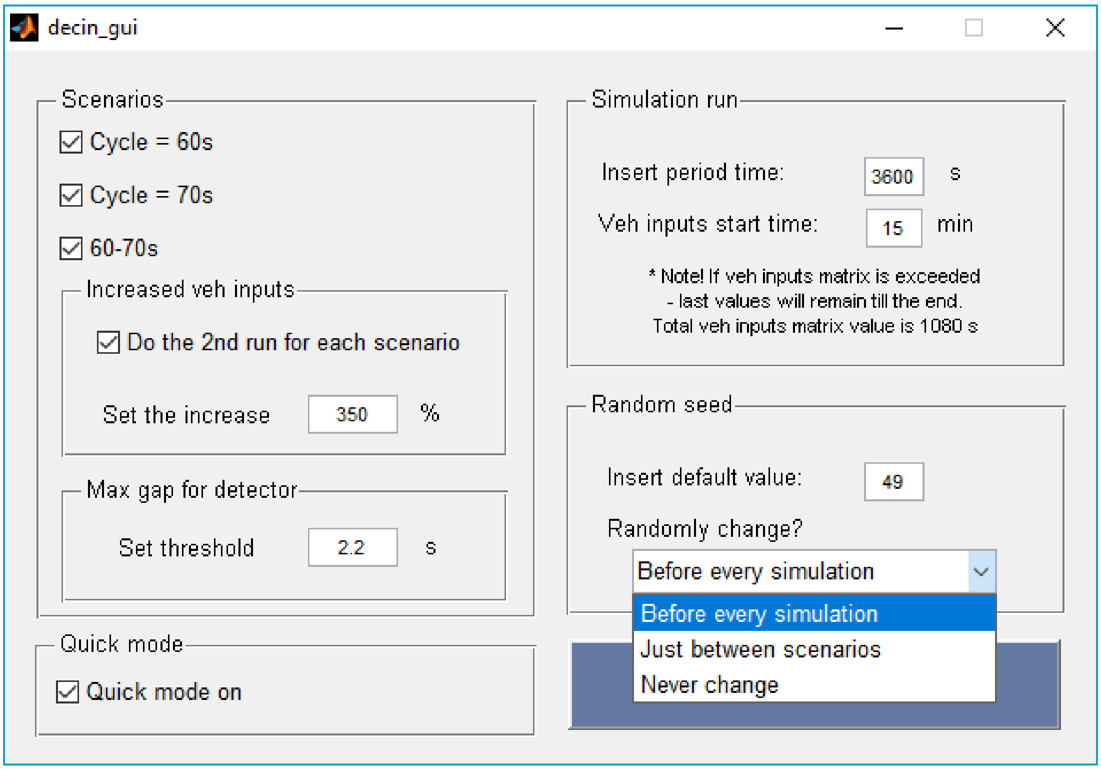

Figure 21: Filling the GUI

The increase value in the first sub section is set to 350% by default, it can be changed of course. Together with the threshold 2.2s for maximal gap it creates environment where the dynamic control is being frequently used. Because of the lack of pedestrian crossing stages and the fact, that the measured intensity is not very high, it takes part only a few times even with the threshold set to 3s with the original intensity. That is why the default increase of vehicle inputs is such a big number.

Every editable field is protected by verification functions to prevent setting inappropriate value. It shows an error message with a specification how the string or value should look like every time, when a wrong format of number is being inputted. For example, the threshold value verification functions look like this:

```
ThresHold = str2double(get(hObject, 'String'));
if isnan(ThresHold)
    set(hObject, 'String', 2.2);
    errordlg('Input must be a number','Error');
end
ThresHold=str2num(get(hObject, 'String'));
if sum(size(ThresHold)) > 2
    set(hObject, 'String', 2.2);
    errordlg('Input must be a one dimensional number','Error');
end
if ThresHold < 0
    set(hObject, 'String', 2.2);
    errordlg('Input must be a possitive number','Error');
end
```

The *size* generates two values for one dimensional element *[1 1]*, sum of it makes *2*. That is why the condition is *>2*.

The output values are stored in application-defined data. This data is visible within the Matlab program so it does not disappear after closing the GUI. Unfortunately, when the aim was to start the main program automatically from the GUI, this data was not transferred properly and the main program could not operate. For this reason, only information message appears after clicking on the button *Execute*. The message includes the information that the GUI can be closed and to execute the main program, *vis_decin* should be typed in the command window. It can work only if these files are in the same folder. The *vis_decin* is a name of the main program.

For example, to set a value to the application-defined data can look like this:

```
setappdata(0,'Quickmode',get(handles.Quickmode,'Value'));
```

It needs to be specified inside of the function for the *Execute* button. Then in the main M-File it can be called and stored in a new variable:

```
qm = getappdata(0,'Quickmode');
```

After that, it needs to be cleared from the memory so that it will not interfere in a case that the program is launched again:

```
rmappdata(0,'Quickmode')
```

The whole source code can be found in the appendix section and in electronic attachments together with the *.fig file.

3.2.3 Outputs

In this subchapter, analysis of results can be found here together with the interpretation. Results are stored in matrixes and variables and Matlab provides several filtering and data processing methods as well as powerful graph printing tools. So, in this situation, there is no necessity to transmit results for analysis and representation to a different programs or files. Results in figures and tables in this subchapter come from the main program with a specific setting.

The GUI was set to all scenarios including increased vehicle inputs, the increase was set to 350% and the threshold value for the detector was set to 2.2s. It was run using the quick mode and the period time took 3600s. Vehicle input start time was set to 60^{th} minute and the random seed was 17 with the never change choice.

There are together three figures containing 12 graphs in total and the stand-alone results are in tables in command window.

3.2.3.1 Travel Time

Very important quantity from traffic simulation is the travel time. As it was specified in the model creating section, there are two measured sections in opposite direction. The average values for every measure interval are collected for all vehicle classes together in a matrix. For all scenarios with the increase function, there are six sets of data. A logical fact is, that the travel time cannot be equal to zero, but there are zeros in matrix coming from intervals where no vehicle took this path within the specified time. To get rid of these zeros to avoid having corrupted data, there is a possibility to replace them by the closest neighbour:

```
prov = TT(1,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
           prov =
interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap
');
        end
        TTA(1,:)=prov;
```

The `prov` and `pr` are just auxiliary variables, because in this form, it cannot work with matrix. The condition is there because it works only when there are at least two non-zero values. The result values are displayed in Figure 22. Two top graphs represent the direction of Ústecká – Podmokelská, lower graphs represent the opposite direction. The left side represents standard vehicle inputs and on the right side, there are increased values. The red line is for the standard 60s cycle, green represents 70s and the dynamic control is displayed by blue. It is in the legend box within the last graph. The same key applies to all other graphs. In the Matlab figures it is possible to zoom in each graph for better understanding. If there will be straight lines in the graphs, equal to zero in whole length, it means that specific scenario, corresponding to the colour, was not selected in the GUI. Graphs are generated automatically and there is no function to eliminate printing such scenarios. But it is not very difficult to do so.

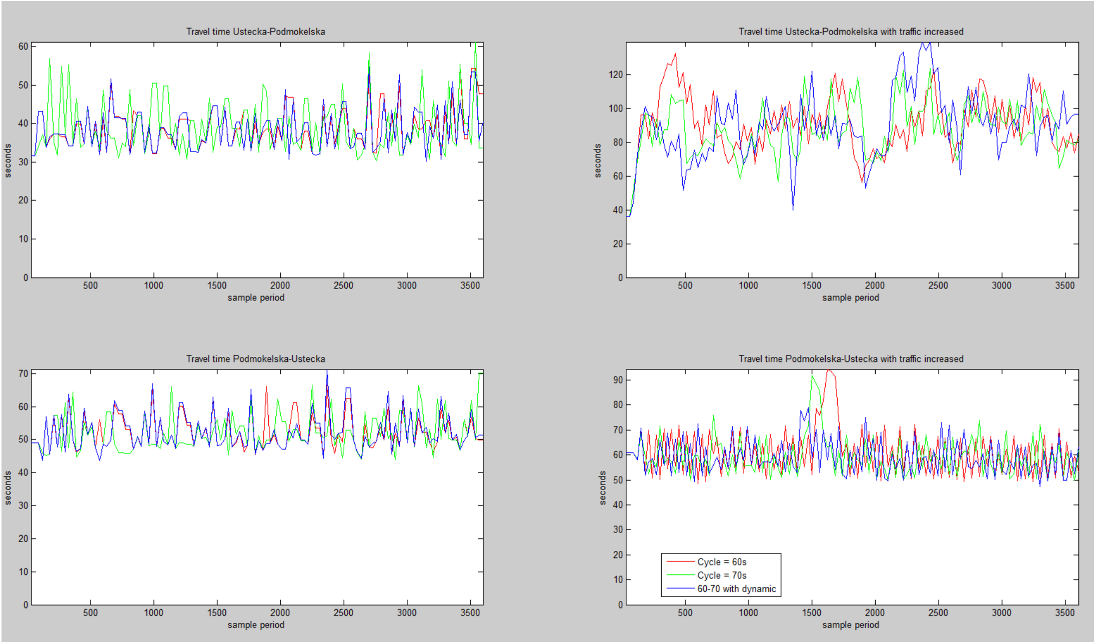

Figure 22: Results of travel time measurement

The code belonging to graphs representation is visible in appendix section. It was necessary to edit axes a bit to represent the whole period time and nothing else.

During the analysis of results, sometimes it is very important to get the data smoother when there is too much of it and it is difficult to understand it. There is a handful function in Matlab available from version R2016a called *movmean*. It works on a principle of floating window. The size of the window can be very easily set. For these results value 5 was used. It is specified in the beginning of the main program and can be edited. It could be also very easily planted to the GUI to set this value by user. The biggest advantage of this function is a fact, that it fills the edge spaces, so it has the same length as before. The principle of floating window is that it makes mean of desired number of values and shift by one and do the same. After whole array is done, it is replaced by those new values. Other functions working with the floating mean create empty edges and the generated structure do not correspond to the axis values then. The previous graph smothered by this moving mean function is visible in Figure 23. Other graphs are automatically smothered by this function as well.

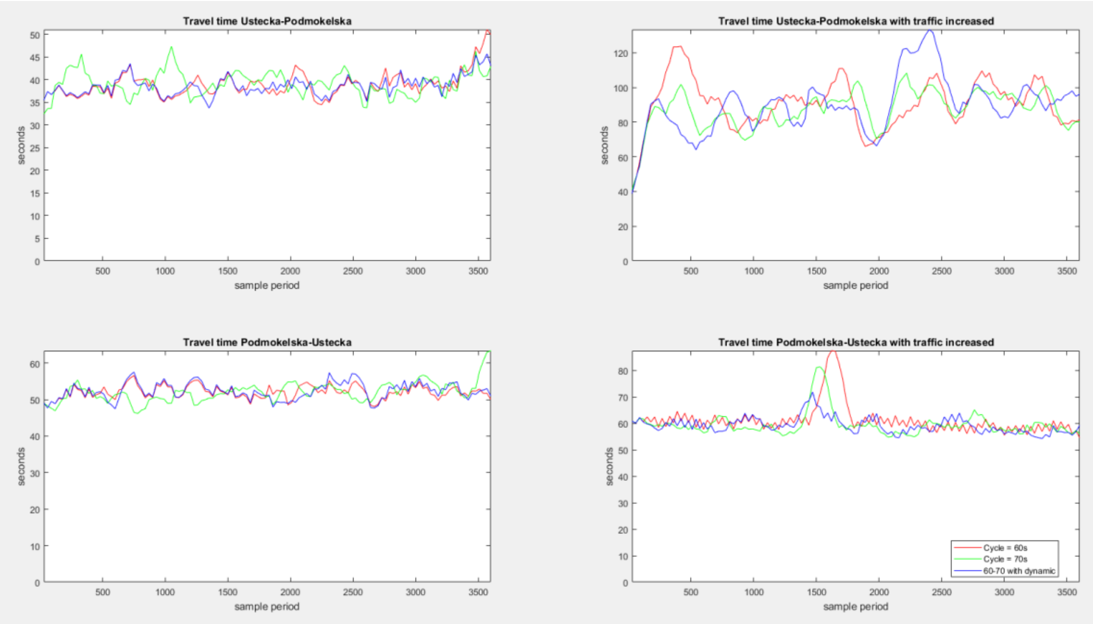

Figure 23: Smooth results of travel time measurement

Next to the graph of data from measure intervals, another output is a table with one dimensional values. Those are overall results from the simulation. Maximal and average travel times are calculated from all vehicle classes, but then average values are also separated to each vehicle class. At the end of the program, data is collected together in matrixes and those are simply represented in the command window by using *table* function, which is available in Matlab since version R2013b. For older versions, there is an alternative function *printmat*. Unfortunately, this alternative function does not work in the newest versions.

Results from the specified testing are shown in Table 4 and 5. The first row and column are used from the old *printmat* function in all tables. In the command window, there are names of columns separated by short horizontal lines with the *table* function. It is better for orientation but the old form is better usable here in printing results. So, after running the program, tables will slightly differ in heading from these.

Table 4: Travel time Ústecká-Podmokelská [s]

	personal_veh	trucks	buses	all	max_all
60s	38.48411	39.38676	NaN	38.45934	55.33964
increase	90.97753	94.31275	NaN	91.04598	132.78169
70s	38.88332	36.33027	NaN	38.79194	61.21064
increase	87.53287	86.91873	NaN	87.59668	123.99445
60to70s	38.28510	40.34234	NaN	38.27074	54.69068
increase	90.19942	91.76606	NaN	90.16227	139.61833

Table 5: Travel time Podmokelská-Ústecká [s]

	personal_veh	trucks	buses	all	max_all
60s	51.96274	50.27360	NaN	51.87680	66.45248
increase	60.85013	60.46217	NaN	61.15895	94.37318
70s	52.15607	49.88429	NaN	51.99071	70.13293
increase	59.58591	58.23984	NaN	59.55893	91.90840
60to70s	52.24057	50.96316	NaN	52.17678	71.33331
increase	59.33382	54.86031	NaN	59.19931	78.88195

Those *NAN* values in bus sections mean that no buses took this path during the tested time. In this case, they could have been left out from the evaluation. But it is about showing possibilities, so this category remains. The first column represents the average travel time for personal vehicle, then for trucks in the second column followed by buses. In the fourth column, there is average for all vehicle classes and in the last one there maximum of all measured intervals for all vehicle classes.

Next to the travel times, interesting quantity is vehicle delay. In Vissim it is calculated from travel times. It is obtained from the same measure points. Here, some values can be equal to zero, when vehicles catch the green signal and there is nothing in front of them slowing them down. The data is smothered at least. It is visible in Figure 24. Separated results are again in tables. Concretely Table 6 and 7.

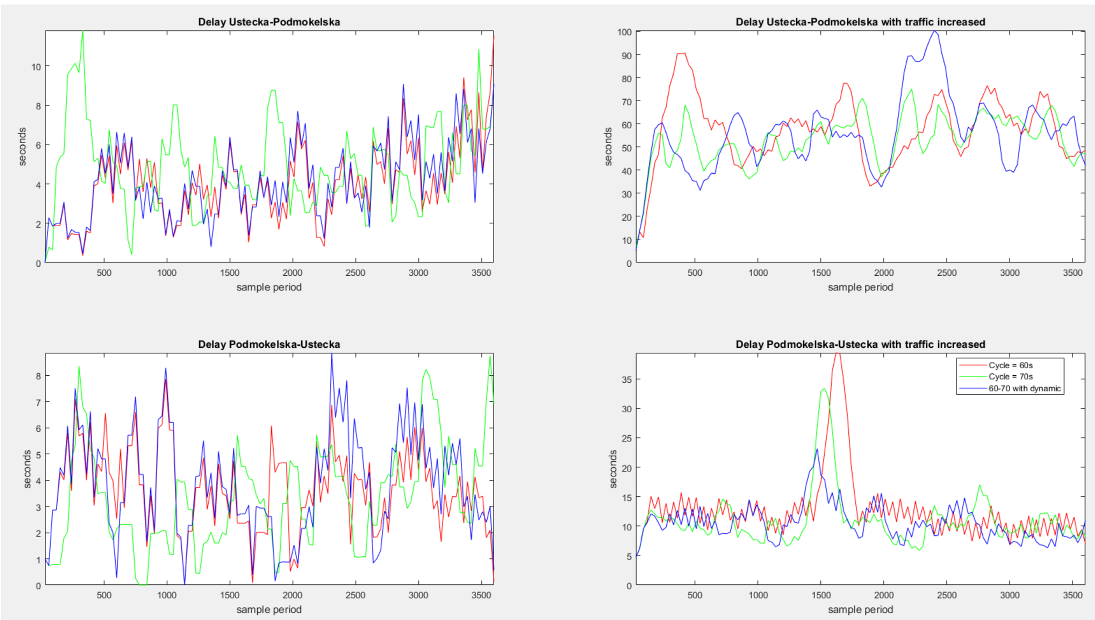

Figure 24: Smooth results of delay measurement

Table 6: Delay Ústecká-Podmokelská [s]

	personal_veh	trucks	buses	all	max_all
60s	5.29518	6.80330	NaN	5.27240	20.41135
increase	57.80178	61.24202	NaN	57.86141	98.44872
70s	5.73458	3.88357	NaN	5.69350	26.99608
increase	54.21337	53.78643	NaN	54.33575	90.30011
60to70s	5.11052	7.68310	NaN	5.10712	21.82998
increase	56.99272	58.59539	NaN	56.97582	106.21658

Table 7: Delay Podmokelská- Ústecká [s]

	personal_veh	trucks	buses	all	max_all
60s	4.07320	2.13421	NaN	4.00311	20.73937
increase	12.91136	12.33276	NaN	12.91136	46.39114
70s	4.19253	2.02754	NaN	4.03900	21.00708
increase	11.32182	10.26869	NaN	11.30295	41.94432
60to70s	4.31165	2.88370	NaN	4.26698	22.45904
increase	10.95791	6.34334	NaN	10.82042	29.53059

3.2.3.2 Queue Length

Queue length can be counted only for all vehicle classes. That is why results are in smaller tables (Table 8 and 9). Values are counted in meters. In this section, data can contain zero values again. That is why those are not replaced. It is clear in Figure 25, where all values from measured intervals are displayed.

Table 8: Queue Ústecká-Podmokelská [m]

	all	max_all
60s	2.63293	17.57276
increase	77.10625	154.94151
70s	2.28999	15.14649
increase	73.26252	155.98601
60to70s	2.53271	22.69647
increase	72.28373	149.66903

Table 9: Queue Podmokelská-Ústecká [m]

	all	max_all
60s	1.97934	13.60532
increase	57.12998	98.31237
70s	1.84084	15.48875
increase	49.90600	128.80008
60to70s	2.10707	13.46977
increase	52.17921	116.92002

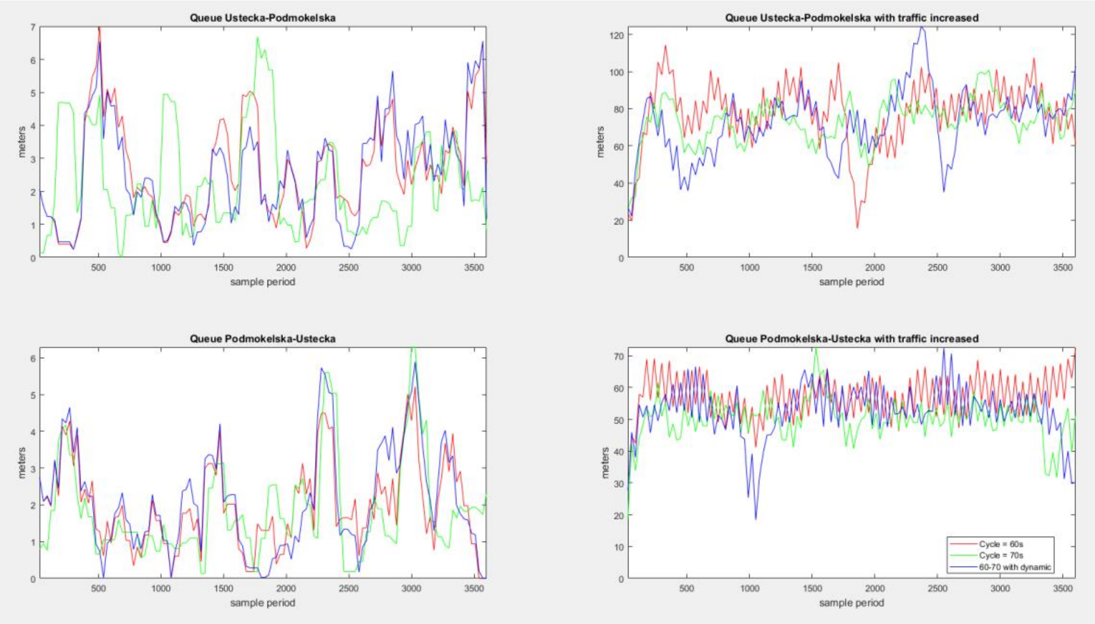

Figure 25: Smooth results of queue measurement

3.2.3.3 Evaluation

For queue and delay, wider floating window could have brought even smoother results. There is the first graph in Figure 26 for better observation. It is clear from all graphs and tables, that there is not very big difference between the original signal plan and the dynamic control. The dynamic control looks a little bit better, but to think about implementing this system based on these results would not be reasonable for such network. The only possible outcome could be that the cycle of 70s would not work well here. And to get better data for the dynamic control, some more testing should have been done. It was done only as an example for 3600s even though there is more vehicle inputs data. Also, some more threshold values could be tested.

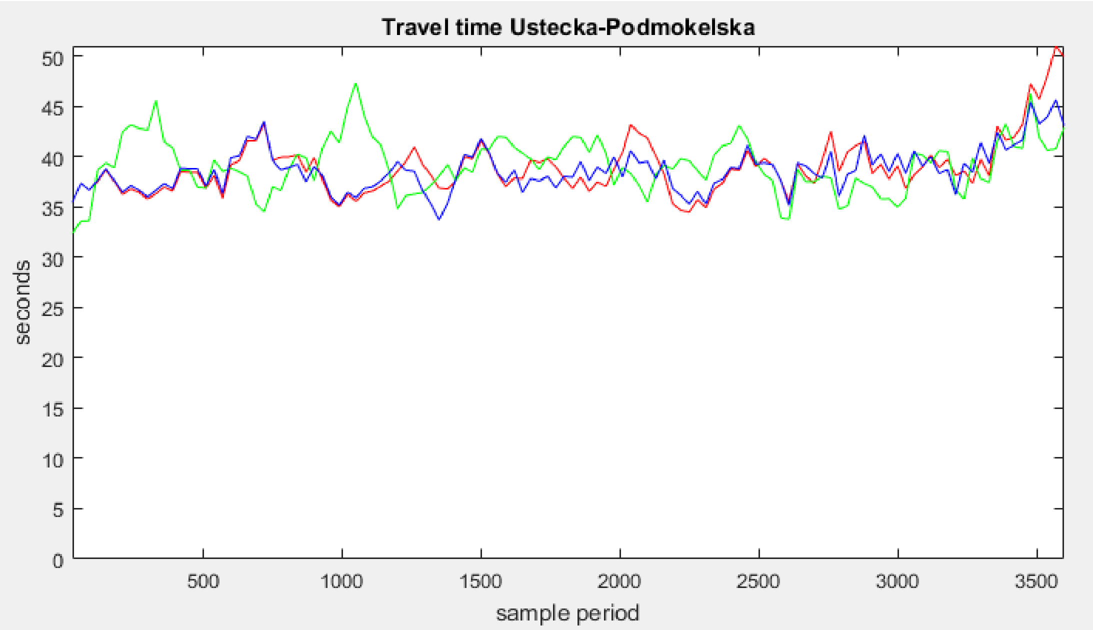

Figure 26: Detail of the first graph

There are many other possibilities to evaluate model in Vissim. It allows for example to store in a log file all events when public transportation stopped. There is also a possibility to measure vehicle speed by Matlab and so on.

Chapter 4

Future Vissim COM Usage

PTV Vissim is designed to cover all traffic situations that can occur during the present state of infrastructure and fleet. Regarding mainly the Europe.

Thanks to the COM interface and the number of available detectors, it can be simulated many situations regarding future trends of the traffic. It can be in a sense of intelligent infrastructure together with intelligent vehicles or autonomous vehicles and so on.

4.1 Intelligent Infrastructure

Thanks to the detectors and information about all vehicles in the network, that Vissim provides over the COM, it can be simulated such conditions that respond to the advanced development of communication between vehicles and infrastructure. For example, in dependence on formation of some traffic excess, the infrastructure will be able to transfer such information with vehicles and find out an alternative way for continuous driving or at least minimal delay. Closing of arbitrary link can be reached for example by changing of routing together with the signal plan in M-File. It can be changed whenever during the simulation.

Many other similar or even more complex situations can be simulated. A practical example is created just for the following subchapter.

4.2 Autonomous Vehicles

Likely progression in the area of autonomous vehicles is their platooning into clusters depending on their speed and routing. All thanks to the vehicle to vehicle communication and included sensors. Benefits could be reduced fuel or battery consumption thanks to the air flow (especially for trucks) and the increased capacity of communications. [5]

Such conditions are not easy to simulate in Vissim. But there is a possibility to create these platoons directly in vehicle inputs. A simplified example is concretely designed below.

In the first place, it is necessary to create the infrastructure in Vissim. For this case, a one-way, single lane communication is used. The road is crossed by pedestrian path. The crossing is secured by traffic lights. An inductive loop is present approximately one meter before the stop line. The working length is two meters back. The second inductive loop is situated near the vehicle input spot. This one is ten metres long. The reason is getting states in whole seconds. There is a possibility to shorten the loop, but the state would have to be checked several times per second so that no vehicle would pass without detection. In real, this loop would be replaced by shorter one or it would disappear at all, because vehicles might communicate directly with some part of signal controller in the future. The whole idea is to prepare a free way for an approaching cluster of vehicle through an intersection in a desired way, so that other ways would provide a green signal for pedestrians. The routing preferences from the cluster can be obtained by functions in M-File. Here it is just simplified to the crossing. The modelled situation is in Figure 27.

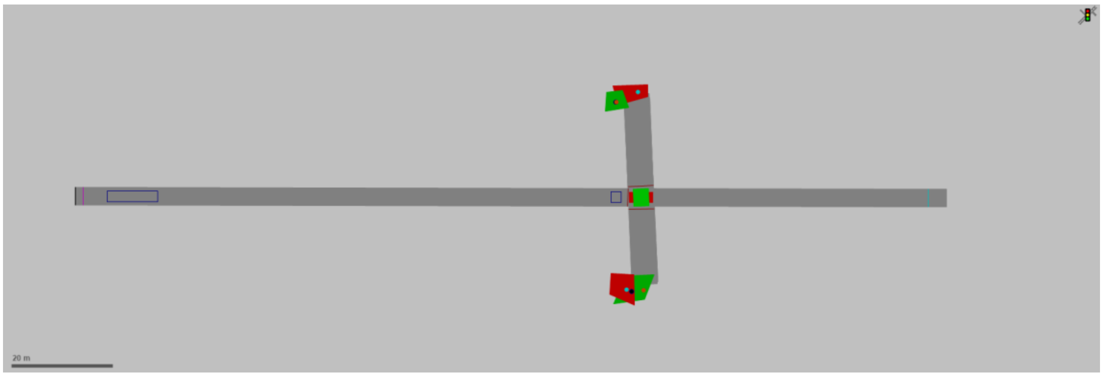

Figure 27: Vissim model for clusters of vehicles

When the infrastructure is ready, vehicle and pedestrian inputs should follow as well as routing. The future vision could also contain an assumption that the leading vehicle would go by a homogenous speed in straight sections. The speed could be probably even higher in cities where only electrical autonomous vehicle could attend the infrastructure. In this example, the speed of 58km/h is used. There is a necessity to create a new speed class in Vissim. The access is through *Base Data -> Distributions -> Desired Speed*. Here *add…* and set the *LowerBound* and *UpperBound*. Both are read only, so it cannot be done via Matlab. Important note is that both values cannot be set to the same number. But even the difference of 0.01km/h satisfies the condition. In the model example a new speed class id104 with 58.00-58.01km/h is created.

Platooning or clustering is achieved by vehicle input which is being changed every second. For a few seconds, there is a demand of extreme amount of vehicles. It is followed by multiple longer time of zero vehicle input. Values are loaded from a text file. Regarding vehicle composition, there is one new category necessary to be created in Vissim. It is then set in M-File to only personal vehicles.

For a more homogenous cluster in a sense of lowered spaces between vehicles, a *Driving behaviour* would need to be adjusted. A major influence on that has the value *max. look ahead distance*, it can be lowered, but it dramatically increases the probability of vehicle indifference towards the traffic lights. When using a different *Car following model – no interaction*, it has the same result. But the probability here is equal to one. So it is completely useless to install signal heads into such network. The alternative could be using parking lot to generate vehicles or traffic lights at the beginning of the network to cumulate them. It would take over the function of specified vehicle input. But there would be necessary to set the acceleration of vehicles to the same parameters. But this example is not based on that solution.

The control of this model works on the principle of setting the pedestrians green states all the time when there are no vehicles. When a cluster approaches, it is interrupted for the minimal possible time so that vehicles can go through the crossing without reducing their speed. And just after that, the green state is again returned to pedestrians. This necessary time is calculated on the basis of vehicle speed (specified for all vehicles, but can be obtained from the detector as well) and distance of the first detector from the stop line. An interstage is started after subtraction of interstage time and verification of safe escaping the crossing from pedestrians and the minimal state duration. After the cluster leaves, there is again the green state for pedestrians after another interstage and red stage for vehicle.

If there are many clusters tightly in a row or a really long one, the minimal stage is prolonged, but only to the maximal value. If that happens and not all vehicles came through the crossing, pedestrians get their stage but only for the minimal time and then vehicles are free to drive again. When there are no others on the horizon, the stage ends quickly and pedestrians are free to go and the cycle repeats from the beginning. The program can be found in the appendix section. The electronic attachment includes the Vissim model as well.

Conclusion

It is clear that connecting these two programs can bring new possibilities in simulations running as well as in results evaluating. The most significant advantage lies in the model testing of scenarios based on signal plans created in Vissig. The algorithm can be extended by multiple running of each scenario just with different random seed, averaged together to get more objective results.

To create fixed time signal plan, it is much easier to do it in Vissig. It showed up that developing of some interface for signal plans creating by just setting parameters would be pointless in a share of the Vissig, since this or some other module is obligatory to be present for using signal heads. But when the aim is to avoid the VisVap module to create some dynamic signal plan or to create more complex solution with changing some parameters inside of the signal plan, it can be all perfectly done by Matlab M-File.

The opportunity of continuing with these papers lies in a possibility to extend the model testing algorithm by mentioned multiple runs and to deal with the issue of providing data from the GUI to a different M-File. Also, a dynamic assignment within a wider network (an area with more intersections and several possibilities of choosing different paths) can be examined for possibilities of utilization the COM interface with Matlab. Last but not least, some more data evaluation can be done in Matlab to show more possibilities in filtering and representing of results.

Bibliography

1. *PTV Vissim Help: Product help* [online]. PTV, 2015 [cit. 2017-05].
2. *PTV Vissim 7 - Introduction to the COM API* [online]. PTV AG. D-76131 Karlsruhe, Germany, 2015 [cit. 2017-05].
3. *External Interfaces* [online]. MATHWORKS, INC. MATLAB. U.S., 2015 [cit. 2017-05].
4. *Programming Fundamentals* [online]. MATHWORKS, INC. MATLAB. U.S., 2015 [cit. 2017-05].
5. FERNANDES, Pedro and Urbano NUNES. *Platooning With DSRC-Based IVC-Enabled Autonomous Vehicles: Adding Infrared Communications for IVC Reliability Improvement. 2012 Intelligent Vehicles Symposium* [online]. Spain, 2012, 517-522 [cit. 2017-05].

ANNEXES

Appendix A

Source codes

The electronical attachment is divided into two folders. The first one contains the Vissim and Matlab files for the Děčín model as well as some other important files (vehicle inputs, background and so on). The second folder contains files for the Autonomous Vehicles chapter. Here, in the offline version of attachments, three M-Files are printed.

GUI

The following Table 10 contains the source code of the GUI for the main program (*decin_gui.m*).

Table 10: GUI source code

```
function varargout = decin_gui(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @decin_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @decin_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
function decin_gui_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
function varargout = decin_gui_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
set(handles.scenario1,'value',1)
if get(handles.Run2nd,'Value') == 0
    set(handles.Increase,'Visible','off');
end
if get(handles.scenario3,'Value') == 0
    set(handles.ThresHold,'Visible','off');
end
function execute_Callback(hObject, eventdata, handles)
setappdata(0,'scenario1',get(handles.scenario1,'Value'));
setappdata(0,'scenario2',get(handles.scenario2,'Value'));
setappdata(0,'scenario3',get(handles.scenario3,'Value'));
setappdata(0,'Run2nd',get(handles.Run2nd,'Value'));
setappdata(0,'ThresHold',str2num(get(handles.ThresHold,'String')));
setappdata(0,'Increase',str2num(get(handles.Increase,'String')));
setappdata(0,'PeriodTime',str2num(get(handles.PeriodTime,'String')));
setappdata(0,'RandomSeed',str2num(get(handles.RandomSeed,'String')));
setappdata(0,'VehinStart',str2num(get(handles.VehinStart,'String')));
setappdata(0,'RandomChange',get(handles.RandomChange,'Value'));
setappdata(0,'Quickmode',get(handles.Quickmode,'Value'));
msgbox('Please close the GUI and type vis_decin in Command Window to execute the main program','Note','Help');
```

```
function RandomSeed_Callback(hObject, eventdata, handles)
RandomSeed = str2double(get(hObject, 'String'));
if isnan(RandomSeed)
    set(hObject, 'String', 1);
    errordlg('Input must be a number','Error');
end
RandomSeed=str2num(get(hObject, 'String'));
if sum(size(RandomSeed)) > 2
    set(hObject, 'String', 1);
    errordlg('Input must be a one dimensional number','Error');
end
if (RandomSeed < 0) || (RandomSeed > 2147483647) || (rem(RandomSeed, 1)>0)
    set(hObject, 'String', 1);
    errordlg('Input must be a Natural number between 1 to 2147483647','Error');
end
function RandomSeed_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function RandomChange_Callback(hObject, eventdata, handles)
function RandomChange_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function PeriodTime_Callback(hObject, eventdata, handles)
PeriodTime = str2double(get(hObject, 'String'));
if isnan(PeriodTime)
    set(hObject, 'String', 3600);
    errordlg('Input must be a number','Error');
end
PeriodTime=str2num(get(hObject, 'String'));
if sum(size(PeriodTime)) > 2
    set(hObject, 'String', 3600);
    errordlg('Input must be a one dimensional number','Error');
end
if PeriodTime < 60
    set(hObject, 'String', 3600);
    errordlg('Too low value! (At least 60)','Error');
end
if rem(PeriodTime, 1)>0
    set(hObject, 'String', 3600);
    errordlg('Input must be a Natural number!','Error');
end
function PeriodTime_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function VehinStart_Callback(hObject, eventdata, handles)
VehinStart = str2double(get(hObject, 'String'));
if isnan(VehinStart)
    set(hObject, 'String', 1);
    errordlg('Input must be a number','Error');
end
VehinStart=str2num(get(hObject, 'String'));
if sum(size(VehinStart)) > 2
    set(hObject, 'String', 1);
    errordlg('Input must be a one dimensional number','Error');
end
if (VehinStart <= 0) || (VehinStart > 240) || (rem(VehinStart, 1)>0)
    set(hObject, 'String', 1);
    errordlg('Input must be a Natural number between 1 to 240','Error');
end
function VehinStart_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function scenario1_Callback(hObject, eventdata, handles)
function scenario2_Callback(hObject, eventdata, handles)
function scenario3_Callback(hObject, eventdata, handles)
if get(handles.scenario3,'Value') == 0
    set(handles.ThresHold,'Visible','off');
else
    set(handles.ThresHold,'Visible','on');
```

```
end
function ThresHold_Callback(hObject, eventdata, handles)
ThresHold = str2double(get(hObject, 'String'));
if isnan(ThresHold)
    set(hObject, 'String', 2.2);
    errordlg('Input must be a number','Error');
end
ThresHold=str2num(get(hObject, 'String'));
if sum(size(ThresHold)) > 2
    set(hObject, 'String', 2.2);
    errordlg('Input must be a one dimensional number','Error');
end
if ThresHold < 0
    set(hObject, 'String', 2.2);
    errordlg('Input must be a possitive number','Error');
end
function ThresHold_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function Run2nd_Callback(hObject, eventdata, handles)
if get(handles.Run2nd,'Value') == 0
    set(handles.Increase,'Visible','off');
else
    set(handles.Increase,'Visible','on');
end
function Increase_Callback(hObject, eventdata, handles)
Increase = str2double(get(hObject, 'String'));
if isnan(Increase)
    set(hObject, 'String', 350);
    errordlg('Input must be a number','Error');
end
Increase=str2num(get(hObject, 'String'));
if sum(size(Increase)) > 2
    set(hObject, 'String', 350);
    errordlg('Input must be a one dimensional number','Error');
end
if Increase < 0
    set(hObject, 'String', 350);
    errordlg('Input must be a possitive number','Error');
end
function Increase_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function Quickmode_Callback(hObject, eventdata, handles)
```

Main program

In the next Table 11, source code for the main program appears (*vis_decin.m*).

Table 11: Source code of the main program

```
%decin
clear all;
close all;
disp('Program vis_decin.m running');
disp('Initializing... (It might take a while)');
Vissim = actxserver('Vissim.Vissim.700'); % Start Vissim
Vissim.LoadNet
sim=Vissim.simulation;
vnet=Vissim.Net;
%Declaration:
scenario=zeros(3,2);
step_time = 10;
max_cores = 4;
window = 5;
maxgap = getappdata(0,'ThresHold');
increase = (getappdata(0,'Increase')/100)+1;
period_time = getappdata(0,'PeriodTime');
random_seed = getappdata(0,'RandomSeed');
```

```
startcol = ceil(getappdata(0,'VehinStart')/5);
seed_change = getappdata(0,'RandomChange');
qm = getappdata(0,'Quickmode');
if getappdata(0,'scenario1')==1
    scenario(1,1) = 1;
    if getappdata(0,'Run2nd')==1
        scenario(1,2) = 1;
    end
end
if getappdata(0,'scenario2')==1
    scenario(2,1) = 1;
    if getappdata(0,'Run2nd')==1
        scenario(2,2) = 1;
    end
end
if getappdata(0,'scenario3')==1
    scenario(3,1) = 1;
    if getappdata(0,'Run2nd')==1
        scenario(3,2) = 1;
    end
end
rmappdata(0,'ThresHold')
rmappdata(0,'Increase')
rmappdata(0,'PeriodTime')
rmappdata(0,'RandomSeed')
rmappdata(0,'VehinStart')
rmappdata(0,'RandomChange')
rmappdata(0,'scenario1')
rmappdata(0,'scenario2')
rmappdata(0,'scenario3')
rmappdata(0,'Quickmode')
%Sim parameters assignment
sim.set('AttValue', 'SimPeriod', period_time);
sim.set('AttValue', 'SimRes', step_time);
sim.set('AttValue', 'NumCores', max_cores);
set(Vissim.Simulation, 'AttValue', 'RandSeed', random_seed);
set(Vissim.Graphics.CurrentNetworkWindow, 'AttValue', 'QuickMode', qm);
%Defining signal controllers & detectors
SCs = vnet.SignalControllers.GetAll;
SG(1,1)=SCs{1}.SGs.ItemByKey(1);
SG(1,2)=SCs{1}.SGs.ItemByKey(2);
SG(2,1)=SCs{2}.SGs.ItemByKey(1);
SG(2,2)=SCs{2}.SGs.ItemByKey(2);
dets = SCs{1}.Detectors.GetAll;
det1 = dets{1};
%Defining composition
Compos= vnet.VehicleCompositions.GetAll;
Rel_Flows=Compos{2}.VehCompRelFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        100);
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   50);
Rel_Flows=Compos{3}.VehCompRelFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        200);
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   50);
Rel_Flows=Compos{4}.VehCompRelFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        300);
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   50);
%Defining routing VehClasses
routingsource=vnet.VehicleRoutingDecisionsStatic.GetAll;
routingsource{1}.set('AttValue', 'AllVehTypes', 'false');
routingsource{1}.set('AttValue', 'VehClasses', 10);
routingsource{2}.set('AttValue', 'AllVehTypes', 'false');
routingsource{2}.set('AttValue', 'VehClasses', 20);
routingsource{3}.set('AttValue', 'AllVehTypes', 'false');
routingsource{3}.set('AttValue', 'VehClasses', 30);
routingsource{4}.set('AttValue', 'AllVehTypes', 'false');
routingsource{4}.set('AttValue', 'VehClasses', 10);
routingsource{5}.set('AttValue', 'AllVehTypes', 'false');
routingsource{5}.set('AttValue', 'VehClasses', 20);
routingsource{6}.set('AttValue', 'AllVehTypes', 'false');
routingsource{6}.set('AttValue', 'VehClasses', 30);
routingsource{7}.set('AttValue', 'AllVehTypes', 'false');
routingsource{7}.set('AttValue', 'VehClasses', 10);
routingsource{8}.set('AttValue', 'AllVehTypes', 'false');
routingsource{8}.set('AttValue', 'VehClasses', 20);
routingsource{9}.set('AttValue', 'AllVehTypes', 'false');
routingsource{9}.set('AttValue', 'VehClasses', 30);
routingsource{10}.set('AttValue', 'AllVehTypes', 'false');
```

```
routingsource{10}.set('AttValue', 'VehClasses', 10);
routingsource{11}.set('AttValue', 'AllVehTypes', 'false');
routingsource{11}.set('AttValue', 'VehClasses', 30);
routingsource{12}.set('AttValue', 'AllVehTypes', 'false');
routingsource{12}.set('AttValue', 'VehClasses', 10);
routingsource{13}.set('AttValue', 'AllVehTypes', 'false');
routingsource{13}.set('AttValue', 'VehClasses', 10);
%Defining VehCompositions on Inputs
vehins=vnet.VehicleInputs.GetAll;
vehins{1}.set('AttValue', 'VehComp(1)', 2);
vehins{2}.set('AttValue', 'VehComp(1)', 3);
vehins{3}.set('AttValue', 'VehComp(1)', 4);
vehins{4}.set('AttValue', 'VehComp(1)', 2);
vehins{5}.set('AttValue', 'VehComp(1)', 3);
vehins{6}.set('AttValue', 'VehComp(1)', 4);
vehins{7}.set('AttValue', 'VehComp(1)', 2);
vehins{8}.set('AttValue', 'VehComp(1)', 3);
vehins{9}.set('AttValue', 'VehComp(1)', 4);
vehins{10}.set('AttValue', 'VehComp(1)', 2);
vehins{11}.set('AttValue', 'VehComp(1)', 4);
vehins{12}.set('AttValue', 'VehComp(1)', 2);
vehins{13}.set('AttValue', 'VehComp(1)', 2);
%other simulation parameters
verify = [1 5*60];
offset = 22;
EOS=sum(scenario(:,1))+sum(scenario(:,2));
period_meas = 30;
%zero matrix for evaluation
x=period_meas:period_meas:period_time;
if rem(period_time, period_meas) ~= 0
   x(length(x)+1)=period_time;
end
DelayA=zeros(3,length(x));
DelayB=zeros(3,length(x));
TTA=zeros(3,length(x));
TTB=zeros(3,length(x));
QA=zeros(3,length(x));
QB=zeros(3,length(x));
DelayA2=zeros(3,length(x));
DelayB2=zeros(3,length(x));
TTA2=zeros(3,length(x));
TTB2=zeros(3,length(x));
QA2=zeros(3,length(x));
QB2=zeros(3,length(x));
maxQA=zeros(3,1);
maxQB=zeros(3,1);
avgQA=zeros(3,1);
avgQB=zeros(3,1);
maxDelayA=zeros(3,1);
maxDelayB=zeros(3,1);
avgDelayA=zeros(3,1);
avgDelayB=zeros(3,1);
maxTTA=zeros(3,1);
maxTTB=zeros(3,1);
avgTTA=zeros(3,1);
avgTTB=zeros(3,1);
avgDelaypA=zeros(3,1);
avgDelaypB=zeros(3,1);
avgDelaytA=zeros(3,1);
avgDelaytB=zeros(3,1);
avgDelaybA=zeros(3,1);
avgDelaybB=zeros(3,1);
avgTTpA=zeros(3,1);
avgTTpB=zeros(3,1);
avgTTtA=zeros(3,1);
avgTTtB=zeros(3,1);
avgTTbA=zeros(3,1);
avgTTbB=zeros(3,1);
maxQA2=zeros(3,1);
maxQB2=zeros(3,1);
avgQA2=zeros(3,1);
avgQB2=zeros(3,1);
maxDelayA2=zeros(3,1);
maxDelayB2=zeros(3,1);
avgDelayA2=zeros(3,1);
avgDelayB2=zeros(3,1);
maxTTA2=zeros(3,1);
```

```
maxTTB2=zeros(3,1);
avgTTA2=zeros(3,1);
avgTTB2=zeros(3,1);
avgDelaypA2=zeros(3,1);
avgDelaypB2=zeros(3,1);
avgDelaytA2=zeros(3,1);
avgDelaytB2=zeros(3,1);
avgDelaybA2=zeros(3,1);
avgDelaybB2=zeros(3,1);
avgTTpA2=zeros(3,1);
avgTTpB2=zeros(3,1);
avgTTtA2=zeros(3,1);
avgTTtB2=zeros(3,1);
avgTTbA2=zeros(3,1);
avgTTbB2=zeros(3,1);
%simulation
for j=1:EOS
if scenario(1,1)==1
    if (seed_change == 1) || (seed_change == 2)
        format longG
        random_seed = round(2147483646*rand)+1;
        set(Vissim.Simulation, 'AttValue', 'RandSeed', random_seed);
        format short
    end
    cycle = 60;
    green(1) = 35;
    green(2) = green(1);
    maxgreen=0;
    load vehinsmatrix.txt;
    scenario(1,1)=2;
    disp('Scenario 1,1 running');
elseif scenario(1,2)==1
    if (seed_change == 1)
        format longG
        random_seed = round(2147483646*rand)+1;
        set(Vissim.Simulation, 'AttValue', 'RandSeed', random_seed);
        format short
    end
    cycle = 60;
    green(1) = 35;
    green(2) = green(1);
    maxgreen=0;
    load vehinsmatrix.txt;
    vehinsmatrix=vehinsmatrix*increase;
    scenario(1,2)=2;
    disp('Scenario 1,2 running');
elseif scenario(2,1)==1
    if (seed_change == 1) || (seed_change == 2)
        format longG
        random_seed = round(2147483646*rand)+1;
        set(Vissim.Simulation, 'AttValue', 'RandSeed', random_seed);
        format short
    end
    cycle = 70;
    green(1) = 45;
    green(2) = green(1);
    maxgreen=0;
    load vehinsmatrix.txt;
    scenario(2,1)=2;
    disp('Scenario 2,1 running');
elseif scenario(2,2)==1
    if (seed_change == 1)
        format longG
        random_seed = round(2147483646*rand)+1;
        set(Vissim.Simulation, 'AttValue', 'RandSeed', random_seed);
        format short
    end
    cycle = 70;
    green(1) = 45;
    green(2) = green(1);
    maxgreen=0;
    load vehinsmatrix.txt;
    vehinsmatrix=vehinsmatrix*increase;
    scenario(2,2)=2;
    disp('Scenario 2,2 running');
elseif scenario(3,1)==1
    if (seed_change == 1) || (seed_change == 2)
```

```
        format longG
        random_seed = round(2147483646*rand)+1;
        set(Vissim.Simulation, 'AttValue', 'RandSeed', random_seed);
        format short
    end
    cycle = 60;
    green(1) = 35;
    green(2) = green(1);
    maxgreen=45;
    load vehinsmatrix.txt;
    scenario(3,1)=2;
    disp('Scenario 3,1 running');
elseif scenario(3,2)==1
    if (seed_change == 1)
        format longG
        random_seed = round(2147483646*rand)+1;
        set(Vissim.Simulation, 'AttValue', 'RandSeed', random_seed);
        format short
    end
    cycle = 60;
    green(1) = 35;
    green(2) = green(1);
    maxgreen=45;
    load vehinsmatrix.txt;
    vehinsmatrix=vehinsmatrix*increase;
    scenario(3,2)=2;
    disp('Scenario 3,2 running');
end
stage = [0 0];
change = [0 0;0 0];
terminate = [0 0];
start_time = [0 0];
sim_sec = 0;
claim1 = 1;
startshift = 0;
gap1 = 0;
c=1;
vehTTs1 = vnet.VehicleTravelTimeMeasurements.ItemByKey(1);
vehTTs2 = vnet.VehicleTravelTimeMeasurements.ItemByKey(2);
queue1 = vnet.QueueCounters.ItemByKey(1);
queue2 = vnet.QueueCounters.ItemByKey(2);
del1 = vnet.DelayMeasurement.ItemByKey(1);
del2 = vnet.DelayMeasurement.ItemByKey(2);
TT = [0;0];
Q = [0;0];
TTactual = [0;0];
TTsum = [0;0];
Qlenactual = [0;0];
Qsum = [0;0];
maxQ = [0;0];
delactual = [0;0];
DLactual = [0;0];
DLsum = [0;0];
DL = [0;0];
%Loading VehInputs
row2 = 1;
column = startcol;
for var1 = 1:length(vehins)
    vehins{var1}.set('AttValue', 'Volume(1)', vehinsmatrix(row2,column));
    if row2 < size(vehinsmatrix,1)
    row2=row2+1;
    end
end
%Loading routing data
row = 1;
load routmatrix.txt
routing=vnet.VehicleRoutingDecisionsStatic.GetAll;
for var1 = 1:length(routing)
routingx=routing{var1}.VehRoutSta.GetAll;
    for var2 = 1:length(routingx)
    routingx{var2}.set('AttValue', 'RelFlow(1)', routmatrix(row,column));
        if row < size(routmatrix,1)
        row=row+1;
        end
    end
end
for i=0:(period_time*step_time)
```

```
    sim.RunSingleStep;
    if (maxgreen ~= 0) && (stage(1) == 1) && (start_time(1) >= (green(1)-3)) && (start_time(1) < maxgreen)
        if claim1 == 1
            gap1=det1.get('AttValue', 'GapTm');
            if gap1 > maxgap
                claim1 = 0;
            end
        end
    end
    if rem(i/step_time, verify(1))==0
        if stage(1) == 0
           if (start_time(1) ~= 0) && (start_time(1) == terminate(1))
               stage(1) = 21;
           end
           if start_time(1) == 0
               SG(1,1).set('AttValue', 'State', 1);
               SG(1,2).set('AttValue', 'State', 3);
               SG(2,1).set('AttValue', 'State', 1);
               SG(2,2).set('AttValue', 'State', 3);
               terminate(1) = 3;
           end
           start_time(1) = start_time(1) + 1;
           if stage(1) == 21
               start_time(1) = 0;
               terminate(1) = 0;
           end
        end
        if stage(1) == 21
           if (start_time(1) ~= 0) && (start_time(1) == change(1,1))
               SG(1,1).set('AttValue', 'State', 2);
           end
           if (start_time(1) ~= 0) && (start_time(1) == change(1,2))
               SG(1,2).set('AttValue', 'State', 1);
           end
           if (start_time(1) ~= 0) && (start_time(1) == terminate(1))
               stage(1) = 1;
           end
           if start_time(1) == 0
               SG(1,2).set('AttValue', 'State', 4);
               change(1,1) = 2;
               change(1,2) = 3;
           end
           start_time(1) = start_time(1) + 1;
           if stage(1) == 1
               start_time(1) = 0;
               terminate(1) = 0;
               change(1,:) = 0;
           end
        end
        if stage(1) == 12
           if (start_time(1) ~= 0) && (start_time(1) == change(1,1))
               SG(1,1).set('AttValue', 'State', 1);
           end
           if (start_time(1) ~= 0) && (start_time(1) == change(1,2))
               SG(1,2).set('AttValue', 'State', 2);
           end
           if (start_time(1) ~= 0) && (start_time(1) == terminate(1))
               stage(1) = 2;
           end
           if start_time(1) == 0
               SG(1,1).set('AttValue', 'State', 4);
               terminate(1) = 6;
               change(1,1) = 3;
               change(1,2) = 4;
           end
           start_time(1) = start_time(1) + 1;
           if stage(1) == 2
               start_time(1) = 0;
               terminate(1) = 0;
               change(1,:) = 0;
           end
        end
        if stage(1) == 1
           if (maxgreen ~= 0) && (start_time(1) >= (terminate(1)-1)) && (start_time(1) < (maxgreen-1))
               if claim1 == 1
```

```
                        terminate(1)=terminate(1)+1;
                    end
            end
            if (start_time(1) ~= 0) && (start_time(1) == (terminate(1)-1))
                stage(1) = 12;
            end
            if start_time(1) == 0
                SG(1,1).set('AttValue', 'State', 3);
                terminate(1) = green(1);
            end
            start_time(1) = start_time(1) + 1;
            if stage(1) == 12
                start_time(1) = 0;
                terminate(1) = 0;
                startshift = 1;
            end
        end

        if stage(1) == 2
            if (start_time(1) ~= 0) && (start_time(1) == (terminate(1)-1))
                stage(1) = 21;
            end
            if start_time(1) == 0
                SG(1,2).set('AttValue', 'State', 3);
                terminate(1) = cycle - (4 + 6 + green(1));
            end
            start_time(1) = start_time(1) + 1;
            if stage(1) == 21
                start_time(1) = 0;
                terminate(1) = 0;
            end
        end
%stages(2)
        if stage(2) == 0
            if (start_time(2) ~= 0) && (start_time(2) == terminate(2))
                stage(2) = 21;
            end
            if start_time(2) == 0
                terminate(2) = (offset - 4);
            end
            start_time(2) = start_time(2) + 1;
            if stage(2) == 21
                start_time(2) = 0;
                terminate(2) = 0;
            end
        end
        if stage(2) == 21
            if (start_time(2) ~= 0) && (start_time(2) == change(2,1))
                SG(2,1).set('AttValue', 'State', 2);
            end
            if (start_time(2) ~= 0) && (start_time(2) == change(2,2))
                SG(2,2).set('AttValue', 'State', 1);
            end
            if (start_time(2) ~= 0) && (start_time(2) == terminate(2))
                stage(2) = 1;
            end
            if start_time(2) == 0
                SG(2,2).set('AttValue', 'State', 4);
                terminate(2) = 8;
                change(2,1) = 6;
                change(2,2) = 3;
            end
            start_time(2) = start_time(2) + 1;
            if stage(2) == 1
                start_time(2) = 0;
                terminate(2) = 0;
                change(2,:) = 0;
            end
        end
        if stage(2) == 12
            if (start_time(2) ~= 0) && (start_time(2) == change(2,1))
                SG(2,1).set('AttValue', 'State', 1);
            end
            if (start_time(2) ~= 0) && (start_time(2) == change(2,2))
                SG(2,2).set('AttValue', 'State', 2);
            end
            if (start_time(2) ~= 0) && (start_time(2) == terminate(2))
```

```
                stage(2) = 2;
            end
            if start_time(2) == 0
                SG(2,1).set('AttValue', 'State', 4);
                terminate(2) = 8;
                change(2,1) = 3;
                change(2,2) = 6;
            end
            start_time(2) = start_time(2) + 1;
            if stage(2) == 2
                start_time(2) = 0;
                terminate(2) = 0;
                change(2,:) = 0;
            end
        end
        if stage(2) == 1
            if startshift == 1
                terminate(2) = start_time(2) + offset;
                startshift = 0;
            end
            if (start_time(2) ~= 0) && (start_time(2) == (terminate(2)-1))
                stage(2) = 12;
            end
            if start_time(2) == 0
                SG(2,1).set('AttValue', 'State', 3);
            end
            start_time(2) = start_time(2) + 1;
            if stage(2) == 12
                start_time(2) = 0;
                terminate(2) = 0;
                if (maxgreen ~= 0)
                    claim1 = 1;
                end
            end
        end
        if stage(2) == 2
            if (start_time(2) ~= 0) && (start_time(2) == (terminate(2)-1))
                stage(2) = 21;
            end
            if start_time(2) == 0
                SG(2,2).set('AttValue', 'State', 3);
                terminate(2) = cycle - green (2) - 8 - 8;
            end
            start_time(2) = start_time(2) + 1;
            if stage(2) == 21
                start_time(2) = 0;
                terminate(2) = 0;
            end
        end
        sim_sec = sim_sec + 1;
    end
    if (rem(i/step_time, verify(2))==0) && i~=0
        column = column + 1;
        row = 1;
        row2 = 1;
        if column <= length(vehinsmatrix)
        for var1 = 1:length(vehins)
            vehins{var1}.set('AttValue', 'Volume(1)',
vehinsmatrix(row2,column));
            row2=row2+1;
        end
        end
        if column <= length(routmatrix)
        for var2 = 1:length(routingx)
            routingx{var2}.set('AttValue', 'RelFlow(1)',
routmatrix(row,column));
            row=row+1;
        end
        end
    end
    %data collecting
    if (i~=0) && (rem((i)/step_time, period_meas)==0) &&
(i<((period_time*step_time)-1))
    TTactual(1,1) = get(vehTTs1,'AttValue', 'TravTm(Current,Total,All)');
    TTactual(2,1) = get(vehTTs2,'AttValue', 'TravTm(Current,Total,All)');
    Qlenactual(1,1) = get(queue1,'AttValue', 'QLen(Current,Total)');
    Qlenactual(2,1) = get(queue2,'AttValue', 'QLen(Current,Total)');
```

```
    DLactual(1,1) = get(del1, 'AttValue', 'VehDelay(Current,Total,All)');
    DLactual(2,1) = get(del2, 'AttValue', 'VehDelay(Current,Total,All)');
     if c ~= 1
        if isnan(TTactual(1,1))
            TT(1,c)=0;
        else
            TTsum(1,1) = TTsum(1,1) + TT(1,c-1);
            TT(1,c) = (TTactual(1,1) - TTsum(1,1));
        end
        if isnan(DLactual(1,1))
            DL(1,c)=0;
        else
            DLsum(1,1) = DLsum(1,1) + DL(1,c-1);
            DL(1,c) = (DLactual(1,1) - DLsum(1,1));
        end
        if isnan(Qlenactual(1,1))
            Q(1,c)=0;
        else
            Qsum(1,1) = Qsum(1,1) + Q(1,c-1);
            Q(1,c) = (Qlenactual(1,1) - Qsum(1,1));
        end
        %second direction
        if isnan(TTactual(2,1))
            TT(2,c)=0;
        else
            TTsum(2,1) = TTsum(2,1) + TT(2,c-1);
            TT(2,c) = (TTactual(2,1) - TTsum(2,1));
        end
        if isnan(DLactual(2,1))
            DL(2,c)=0;
        else
            DLsum(2,1) = DLsum(2,1) + DL(2,c-1);
            DL(2,c) = (DLactual(2,1) - DLsum(2,1));
        end
        if isnan(Qlenactual(2,1))
            Q(2,c)=0;
        else
            Qsum(2,1) = Qsum(2,1) + Q(2,c-1);
            Q(2,c) = (Qlenactual(2,1) - Qsum(2,1));
        end
        c=c+1;
     else
        if isnan(TTactual(1,1))
            TT(1,c)=0;
        else
            TT(1,c) = TTactual(1,1);
        end
        if isnan(DLactual(1,1))
            DL(1,c)=0;
        else
            DL(1,c) = DLactual(1,1);
        end
        if isnan(Qlenactual(1,1))
            Q(1,c)=0;
        else
            Q(1,c) = Qlenactual(1,1);
        end
        %second direction
        if isnan(TTactual(2,1))
            TT(2,c)=0;
        else
            TT(2,c) = TTactual(2,1);
        end
        if isnan(DLactual(2,1))
            DL(2,c)=0;
        else
            DL(2,c) = DLactual(2,1);
        end
        if isnan(Qlenactual(2,1))
            Q(2,c)=0;
        else
            Q(2,c) = Qlenactual(2,1);
        end
        c=c+1;
     end
    end
if i==((period_time*step_time)-1)
```

```
    TTactual(1,1) = get(vehTTs1,'AttValue', 'TravTm(Current,Total,All)');
    TTactual(2,1) = get(vehTTs2,'AttValue', 'TravTm(Current,Total,All)');
    Qlenactual(1,1) = get(queue1,'AttValue', 'QLen(Current,Total)');
    Qlenactual(2,1) = get(queue2,'AttValue', 'QLen(Current,Total)');
    DLactual(1,1) = get(del1,'AttValue', 'VehDelay(Current,Total,All)');
    DLactual(2,1) = get(del2,'AttValue', 'VehDelay(Current,Total,All)');
        if isnan(TTactual(1,1))
            TT(1,c)=0;
        else
            TTsum(1,1) = TTsum(1,1) + TT(1,c-1);
            TT(1,c) = (TTactual(1,1) - TTsum(1,1));
        end
        if isnan(DLactual(1,1))
            DL(1,c)=0;
        else
            DLsum(1,1) = DLsum(1,1) + DL(1,c-1);
            DL(1,c) = (DLactual(1,1) - DLsum(1,1));
        end
        if isnan(Qlenactual(1,1))
            Q(1,c)=0;
        else
            Qsum(1,1) = Qsum(1,1) + Q(1,c-1);
            Q(1,c) = (Qlenactual(1,1) - Qsum(1,1));
        end
        %second direction
        if isnan(TTactual(2,1))
            TT(2,c)=0;
        else
            TTsum(2,1) = TTsum(2,1) + TT(2,c-1);
            TT(2,c) = (TTactual(2,1) - TTsum(2,1));
        end
        if isnan(DLactual(2,1))
            DL(2,c)=0;
        else
            DLsum(2,1) = DLsum(2,1) + DL(2,c-1);
            DL(2,c) = (DLactual(2,1) - DLsum(2,1));
        end
        if isnan(Qlenactual(2,1))
            Q(2,c)=0;
        else
            Qsum(2,1) = Qsum(2,1) + Q(2,c-1);
            Q(2,c) = (Qlenactual(2,1) - Qsum(2,1));
        end
        maxQ(1) = get(queue1,'AttValue', 'QLen(Current,Max)');
        maxQ(2) = get(queue2,'AttValue', 'QLen(Current,Max)');
        avgQ(1) = get(queue1,'AttValue', 'QLen(Current,Avg)');
        avgQ(2) = get(queue2,'AttValue', 'QLen(Current,Avg)');
        maxDelay(1) = get(del1,'AttValue', 'VehDelay(Current,Max,All)');
        maxDelay(2) = get(del2,'AttValue', 'VehDelay(Current,Max,All)');
        avgDelay(1) = get(del1,'AttValue', 'VehDelay(Current,Avg,All)');
        avgDelay(2) = get(del2,'AttValue', 'VehDelay(Current,Avg,All)');
        maxTT(1) = get(vehTTs1,'AttValue', 'TravTm(Current,Max,All)');
        maxTT(2) = get(vehTTs2,'AttValue', 'TravTm(Current,Max,All)');
        avgTT(1) = get(vehTTs1,'AttValue', 'TravTm(Current,Avg,All)');
        avgTT(2) = get(vehTTs2,'AttValue', 'TravTm(Current,Avg,All)');
        avgDelayp(1) = get(del1,'AttValue', 'VehDelay(Current,Avg,10)');
        avgDealyp(2) = get(del2,'AttValue', 'VehDelay(Current,Avg,10)');
        avgDelayt(1) = get(del1,'AttValue', 'VehDelay(Current,Avg,20)');
        avgDelayt(2) = get(del2,'AttValue', 'VehDelay(Current,Avg,20)');
        avgDelayb(1) = get(del1,'AttValue', 'VehDelay(Current,Avg,30)');
        avgDelayb(2) = get(del2,'AttValue', 'VehDelay(Current,Avg,30)');
        avgTTp(1) = get(vehTTs1,'AttValue', 'TravTm(Current,Avg,10)');
        avgTTp(2) = get(vehTTs2,'AttValue', 'TravTm(Current,Avg,10)');
        avgTTt(1) = get(vehTTs1,'AttValue', 'TravTm(Current,Avg,20)');
        avgTTt(2) = get(vehTTs2,'AttValue', 'TravTm(Current,Avg,20)');
        avgTTb(1) = get(vehTTs1,'AttValue', 'TravTm(Current,Avg,30)');
        avgTTb(2) = get(vehTTs2,'AttValue', 'TravTm(Current,Avg,30)');
end
end
    sim.Stop;
    if scenario(1,1)==2
        DelayA(1,:)=movmean(DL(1,:),window);
        DelayB(1,:)=movmean(DL(2,:),window);
        prov = TT(1,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
```

```
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTA(1,:)=movmean(prov,window);
        prov = TT(2,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTB(1,:)=movmean(prov,window);
        QA(1,:)=movmean(Q(1,:),window);
        QB(1,:)=movmean(Q(2,:),window);
        maxQA(1)=maxQ(1);
        maxQB(1)=maxQ(2);
        avgQA(1)=avgQ(1);
        avgQB(1)=avgQ(2);
        maxDelayA(1)=maxDelay(1);
        maxDelayB(1)=maxDelay(2);
        avgDelayA(1)=avgDelay(1);
        avgDelayB(1)=avgDelay(2);
        maxTTA(1)=maxTT(1);
        maxTTB(1)=maxTT(2);
        avgTTA(1)=avgTT(1);
        avgTTB(1)=avgTT(2);
        avgDelaypA(1)=avgDelayp(1);
        avgDelaypB(1)=avgDealyp(2);
        avgDelaytA(1)=avgDelayt(1);
        avgDelaytB(1)=avgDelayt(2);
        avgDelaybA(1)=avgDelayb(1);
        avgDelaybB(1)=avgDelayb(2);
        avgTTpA(1)=avgTTp(1);
        avgTTpB(1)=avgTTp(2);
        avgTTtA(1)=avgTTt(1);
        avgTTtB(1)=avgTTt(2);
        avgTTbA(1)=avgTTb(1);
        avgTTbB(1)=avgTTb(2);
    scenario(1,1)=3;
    elseif scenario(1,2)==2
        DelayA2(1,:)=movmean(DL(1,:),window);
        DelayB2(1,:)=movmean(DL(2,:),window);
        prov = TT(1,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTA2(1,:)=movmean(prov,window);
        prov = TT(2,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTB2(1,:)=movmean(prov,window);
        QA2(1,:)=movmean(Q(1,:),window);
        QB2(1,:)=movmean(Q(2,:),window);
        maxQA2(1)=maxQ(1);
        maxQB2(1)=maxQ(2);
        avgQA2(1)=avgQ(1);
        avgQB2(1)=avgQ(2);
        maxDelayA2(1)=maxDelay(1);
        maxDelayB2(1)=maxDelay(2);
        avgDelayA2(1)=avgDelay(1);
        avgDelayB2(1)=avgDelay(2);
        maxTTA2(1)=maxTT(1);
        maxTTB2(1)=maxTT(2);
        avgTTA2(1)=avgTT(1);
        avgTTB2(1)=avgTT(2);
        avgDelaypA2(1)=avgDelayp(1);
        avgDelaypB2(1)=avgDealyp(2);
        avgDelaytA2(1)=avgDelayt(1);
        avgDelaytB2(1)=avgDelayt(2);
        avgDelaybA2(1)=avgDelayb(1);
        avgDelaybB2(1)=avgDelayb(2);
        avgTTpA2(1)=avgTTp(1);
        avgTTpB2(1)=avgTTp(2);
```

```
        avgTTtA2(1)=avgTTt(1);
        avgTTtB2(1)=avgTTt(2);
        avgTTbA2(1)=avgTTb(1);
        avgTTbB2(1)=avgTTb(2);
    scenario(1,2)=3;
    elseif scenario(2,1)==2
        DelayA(2,:)=movmean(DL(1,:),window);
        DelayB(2,:)=movmean(DL(2,:),window);
        prov = TT(1,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTA(2,:)=movmean(prov,window);
        prov = TT(2,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTB(2,:)=movmean(prov,window);
        QA(2,:)=movmean(Q(1,:),window);
        QB(2,:)=movmean(Q(2,:),window);
        maxQA(2)=maxQ(1);
        maxQB(2)=maxQ(2);
        avgQA(2)=avgQ(1);
        avgQB(2)=avgQ(2);
        maxDelayA(2)=maxDelay(1);
        maxDelayB(2)=maxDelay(2);
        avgDelayA(2)=avgDelay(1);
        avgDelayB(2)=avgDelay(2);
        maxTTA(2)=maxTT(1);
        maxTTB(2)=maxTT(2);
        avgTTA(2)=avgTT(1);
        avgTTB(2)=avgTT(2);
        avgDelaypA(2)=avgDelayp(1);
        avgDelaypB(2)=avgDealyp(2);
        avgDelaytA(2)=avgDelayt(1);
        avgDelaytB(2)=avgDelayt(2);
        avgDelaybA(2)=avgDelayb(1);
        avgDelaybB(2)=avgDelayb(2);
        avgTTpA(2)=avgTTp(1);
        avgTTpB(2)=avgTTp(2);
        avgTTtA(2)=avgTTt(1);
        avgTTtB(2)=avgTTt(2);
        avgTTbA(2)=avgTTb(1);
        avgTTbB(2)=avgTTb(2);
    scenario(2,1)=3;
    elseif scenario(2,2)==2
        DelayA2(2,:)=movmean(DL(1,:),window);
        DelayB2(2,:)=movmean(DL(2,:),window);
        prov = TT(1,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTA2(2,:)=movmean(prov,window);
        prov = TT(2,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTB2(2,:)=movmean(prov,window);
        QA2(2,:)=movmean(Q(1,:),window);
        QB2(2,:)=movmean(Q(2,:),window);
        maxQA2(2)=maxQ(1);
        maxQB2(2)=maxQ(2);
        avgQA2(2)=avgQ(1);
        avgQB2(2)=avgQ(2);
        maxDelayA2(2)=maxDelay(1);
        maxDelayB2(2)=maxDelay(2);
        avgDelayA2(2)=avgDelay(1);
        avgDelayB2(2)=avgDelay(2);
        maxTTA2(2)=maxTT(1);
```

```
        maxTTB2(2)=maxTT(2);
        avgTTA2(2)=avgTT(1);
        avgTTB2(2)=avgTT(2);
        avgDelaypA2(2)=avgDelayp(1);
        avgDelaypB2(2)=avgDealyp(2);
        avgDelaytA2(2)=avgDelayt(1);
        avgDelaytB2(2)=avgDelayt(2);
        avgDelaybA2(2)=avgDelayb(1);
        avgDelaybB2(2)=avgDelayb(2);
        avgTTpA2(2)=avgTTp(1);
        avgTTpB2(2)=avgTTp(2);
        avgTTtA2(2)=avgTTt(1);
        avgTTtB2(2)=avgTTt(2);
        avgTTbA2(2)=avgTTb(1);
        avgTTbB2(2)=avgTTb(2);
    scenario(2,2)=3;
    elseif scenario(3,1)==2
        DelayA(3,:)=movmean(DL(1,:),window);
        DelayB(3,:)=movmean(DL(2,:),window);
        prov = TT(1,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTA(3,:)=movmean(prov,window);
        prov = TT(2,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTB(3,:)=movmean(prov,window);
        QA(3,:)=movmean(Q(1,:),window);
        QB(3,:)=movmean(Q(2,:),window);
        maxQA(3)=maxQ(1);
        maxQB(3)=maxQ(2);
        avgQA(3)=avgQ(1);
        avgQB(3)=avgQ(2);
        maxDelayA(3)=maxDelay(1);
        maxDelayB(3)=maxDelay(2);
        avgDelayA(3)=avgDelay(1);
        avgDelayB(3)=avgDelay(2);
        maxTTA(3)=maxTT(1);
        maxTTB(3)=maxTT(2);
        avgTTA(3)=avgTT(1);
        avgTTB(3)=avgTT(2);
        avgDelaypA(3)=avgDelayp(1);
        avgDelaypB(3)=avgDealyp(2);
        avgDelaytA(3)=avgDelayt(1);
        avgDelaytB(3)=avgDelayt(2);
        avgDelaybA(3)=avgDelayb(1);
        avgDelaybB(3)=avgDelayb(2);
        avgTTpA(3)=avgTTp(1);
        avgTTpB(3)=avgTTp(2);
        avgTTtA(3)=avgTTt(1);
        avgTTtB(3)=avgTTt(2);
        avgTTbA(3)=avgTTb(1);
        avgTTbB(3)=avgTTb(2);
    scenario(3,1)=3;
    elseif scenario(3,2)==2
        DelayA2(3,:)=movmean(DL(1,:),window);
        DelayB2(3,:)=movmean(DL(2,:),window);
        prov = TT(1,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTA2(3,:)=movmean(prov,window);
        prov = TT(2,:);
        pr = size(find(prov~=0));
        if pr(2)>=2
            prov = interp1(find(prov~=0),prov(prov~=0),1:length(prov),'nearest','extrap');
        end
        TTB2(3,:)=movmean(prov,window);
```

```
        QA2(3,:)=movmean(Q(1,:),window);
        QB2(3,:)=movmean(Q(2,:),window);
        maxQA2(3)=maxQ(1);
        maxQB2(3)=maxQ(2);
        avgQA2(3)=avgQ(1);
        avgQB2(3)=avgQ(2);
        maxDelayA2(3)=maxDelay(1);
        maxDelayB2(3)=maxDelay(2);
        avgDelayA2(3)=avgDelay(1);
        avgDelayB2(3)=avgDelay(2);
        maxTTA2(3)=maxTT(1);
        maxTTB2(3)=maxTT(2);
        avgTTA2(3)=avgTT(1);
        avgTTB2(3)=avgTT(2);
        avgDelaypA2(3)=avgDelayp(1);
        avgDelaypB2(3)=avgDealyp(2);
        avgDelaytA2(3)=avgDelayt(1);
        avgDelaytB2(3)=avgDelayt(2);
        avgDelaybA2(3)=avgDelayb(1);
        avgDelaybB2(3)=avgDelayb(2);
        avgTTpA2(3)=avgTTp(1);
        avgTTpB2(3)=avgTTp(2);
        avgTTtA2(3)=avgTTt(1);
        avgTTtB2(3)=avgTTt(2);
        avgTTbA2(3)=avgTTb(1);
        avgTTbB2(3)=avgTTb(2);
    scenario(3,2)=3;
    end
end
%Delays
figure('Name','Delays');
subplot(2,2,1); plot(x,DelayA(1,:),'r')
hold on
subplot(2,2,1); plot(x,DelayA(2,:),'g')
hold on
subplot(2,2,1); plot(x,DelayA(3,:),'b')
title('Delay Ustecka-Podmokelska'), xlabel('sample period'), ylabel('seconds'), xlim([period_meas period_time])
if sum(sum(DelayA))>0
    ylim([0 max(max(DelayA))])
end
subplot(2,2,2); plot(x,DelayA2(1,:),'r')
hold on
subplot(2,2,2); plot(x,DelayA2(2,:),'g')
hold on
subplot(2,2,2); plot(x,DelayA2(3,:),'b')
title('Delay Ustecka-Podmokelska with traffic increased'), xlabel('sample period'), ylabel('seconds'), xlim([period_meas period_time])
if sum(sum(DelayA2))>0
    ylim([0 max(max(DelayA2))])
end
subplot(2,2,3); plot(x,DelayB(1,:),'r')
hold on
subplot(2,2,3); plot(x,DelayB(2,:),'g')
hold on
subplot(2,2,3); plot(x,DelayB(3,:),'b')
title('Delay Podmokelska-Ustecka'), xlabel('sample period'), ylabel('seconds'), xlim([period_meas period_time])
if sum(sum(DelayB))>0
    ylim([0 max(max(DelayB))])
end
subplot(2,2,4); plot(x,DelayB2(1,:),'r')
hold on
subplot(2,2,4); plot(x,DelayB2(2,:),'g')
hold on
subplot(2,2,4); plot(x,DelayB2(3,:),'b')
title('Delay Podmokelska-Ustecka with traffic increased'), xlabel('sample period'), ylabel('seconds'), xlim([period_meas period_time])
if sum(sum(DelayB2))>0
    ylim([0 max(max(DelayB2))])
end
legend('Cycle = 60s', 'Cycle = 70s', '60-70 with dynamic', 'Location','Best')
%Queues
figure('Name','Queues')
subplot(2,2,1); plot(x,QA(1,:),'r')
hold on
subplot(2,2,1); plot(x,QA(2,:),'g')
```

```
hold on
subplot(2,2,1); plot(x,QA(3,:),'b')
title('Queue Ustecka-Podmokelska'), xlabel('sample period'), ylabel('meters'), xlim([period_meas period_time])
if sum(sum(QA))>0
    ylim([0 max(max(QA))])
end
subplot(2,2,2); plot(x,QA2(1,:),'r')
hold on
subplot(2,2,2); plot(x,QA2(2,:),'g')
hold on
subplot(2,2,2); plot(x,QA2(3,:),'b')
title('Queue Ustecka-Podmokelska with traffic increased'), xlabel('sample period'), ylabel('meters'), xlim([period_meas period_time])
if sum(sum(QA2))>0
    ylim([0 max(max(QA2))])
end
subplot(2,2,3); plot(x,QB(1,:),'r')
hold on
subplot(2,2,3); plot(x,QB(2,:),'g')
hold on
subplot(2,2,3); plot(x,QB(3,:),'b')
title('Queue Podmokelska-Ustecka'), xlabel('sample period'), ylabel('meters'), xlim([period_meas period_time])
if sum(sum(QB))>0
    ylim([0 max(max(QB))])
end
subplot(2,2,4); plot(x,QB2(1,:),'r')
hold on
subplot(2,2,4); plot(x,QB2(2,:),'g')
hold on
subplot(2,2,4); plot(x,QB2(3,:),'b')
title('Queue Podmokelska-Ustecka with traffic increased'), xlabel('sample period'), ylabel('meters'), xlim([period_meas period_time])
if sum(sum(QB2))>0
    ylim([0 max(max(QB2))])
end
legend('Cycle = 60s', 'Cycle = 70s', '60-70 with dynamic', 'Location','Best')
figure('Name','Travel times');
subplot(2,2,1); plot(x,TTA(1,:),'r')
hold on
subplot(2,2,1); plot(x,TTA(2,:),'g')
hold on
subplot(2,2,1); plot(x,TTA(3,:),'b')
title('Travel time Ustecka-Podmokelska'), xlabel('sample period'), ylabel('seconds'), xlim([period_meas period_time])
if sum(sum(TTA))>0
    ylim([0 max(max(TTA))])
end
subplot(2,2,2); plot(x,TTA2(1,:),'r')
hold on
subplot(2,2,2); plot(x,TTA2(2,:),'g')
hold on
subplot(2,2,2); plot(x,TTA2(3,:),'b')
title('Travel time Ustecka-Podmokelska with traffic increased'), xlabel('sample period'), ylabel('seconds'), xlim([period_meas period_time])
if sum(sum(TTA2))>0
    ylim([0 max(max(TTA2))])
end
subplot(2,2,3); plot(x,TTB(1,:),'r')
hold on
subplot(2,2,3); plot(x,TTB(2,:),'g')
hold on
subplot(2,2,3); plot(x,TTB(3,:),'b')
title('Travel time Podmokelska-Ustecka'), xlabel('sample period'), ylabel('seconds'), xlim([period_meas period_time])
if sum(sum(TTB))>0
    ylim([0 max(max(TTB))])
end
subplot(2,2,4); plot(x,TTB2(1,:),'r')
hold on
subplot(2,2,4); plot(x,TTB2(2,:),'g')
hold on
subplot(2,2,4); plot(x,TTB2(3,:),'b')
title('Travel time Podmokelska-Ustecka with traffic increased'), xlabel('sample period'), ylabel('seconds'), xlim([period_meas period_time])
if sum(sum(TTB2))>0
```

```
    ylim([0 max(max(TTB2))])
end
legend('Cycle = 60s', 'Cycle = 70s', '60-70 with dynamic', 'Location','Best')
disp('All done! See results in figures and tables above:');
%Tables
TTust_podm = [avgTTpA(1),avgTTtA(1),avgTTbA(1),avgTTA(1),maxTTA(1);
              avgTTpA2(1),avgTTtA2(1),avgTTbA2(1),avgTTA2(1),maxTTA2(1);
              avgTTpA(2),avgTTtA(2),avgTTbA(2),avgTTA(2),maxTTA(2);
              avgTTpA2(2),avgTTtA2(2),avgTTbA2(2),avgTTA2(2),maxTTA2(2);
              avgTTpA(3),avgTTtA(3),avgTTbA(3),avgTTA(3),maxTTA(3);
              avgTTpA2(3),avgTTtA2(3),avgTTbA2(3),avgTTA2(3),maxTTA2(3);];
TTpodm_ust = [avgTTpB(1),avgTTtB(1),avgTTbB(1),avgTTB(1),maxTTB(1);
              avgTTpB2(1),avgTTtB2(1),avgTTbB2(1),avgTTB2(1),maxTTB2(1);
              avgTTpB(2),avgTTtB(2),avgTTbB(2),avgTTB(2),maxTTB(2);
              avgTTpB2(2),avgTTtB2(2),avgTTbB2(2),avgTTB2(2),maxTTB2(2);
              avgTTpB(3),avgTTtB(3),avgTTbB(3),avgTTB(3),maxTTB(3);
              avgTTpB2(3),avgTTtB2(3),avgTTbB2(3),avgTTB2(3),maxTTB2(3);];
Delust_podm =
[avgDelaypA(1),avgDelaytA(1),avgDelaybA(1),avgDelayA(1),maxDelayA(1);
avgDelaypA2(1),avgDelaytA2(1),avgDelaybA2(1),avgDelayA2(1),maxDelayA2(1);
avgDelaypA(2),avgDelaytA(2),avgDelaybA(2),avgDelayA(2),maxDelayA(2);
avgDelaypA2(2),avgDelaytA2(2),avgDelaybA2(2),avgDelayA2(2),maxDelayA2(2);
avgDelaypA(3),avgDelaytA(3),avgDelaybA(3),avgDelayA(3),maxDelayA(3);
avgDelaypA2(3),avgDelaytA2(3),avgDelaybA2(3),avgDelayA2(3),maxDelayA2(3);];
Delpodm_ust =
[avgDelaypB(1),avgDelaytB(1),avgDelaybB(1),avgDelayB(1),maxDelayB(1);
avgDelayB2(1),avgDelaytB2(1),avgDelaybB2(1),avgDelayB2(1),maxDelayB2(1);
avgDelaypB(2),avgDelaytB(2),avgDelaybB(2),avgDelayB(2),maxDelayB(2);
avgDelaypB2(2),avgDelaytB2(2),avgDelaybB2(2),avgDelayB2(2),maxDelayB2(2);
avgDelaypB(3),avgDelaytB(3),avgDelaybB(3),avgDelayB(3),maxDelayB(3);
avgDelaypB2(3),avgDelaytB2(3),avgDelaybB2(3),avgDelayB2(3),maxDelayB2(3);];
Qust_podm = [avgQA(1), maxQA(1);
             avgQA2(1), maxQA2(1);
             avgQA(2), maxQA(2);
             avgQA2(2), maxQA2(2);
             avgQA(3), maxQA(3);
             avgQA2(3), maxQA2(3);];
Qpodm_ust = [avgQB(1), maxQB(1);
             avgQB2(1), maxQB2(1);
             avgQB(2), maxQB(2);
             avgQB2(2), maxQB2(2);
             avgQB(3), maxQB(3);
             avgQB2(3), maxQB2(3);];
Scen = {'60s', 'increase', '70s', 'increase', '60-70s', 'increase'};
disp('Travel time Ustecka-Podmokelska [s]');
T1=table;
T1.Scenario = Scen';
T1.personal_veh = TTust_podm(:,1);
T1.trucks = TTust_podm(:,2);
T1.buses = TTust_podm(:,3);
T1.all = TTust_podm(:,4);
T1.max_all = TTust_podm(:,5)
disp('Travel time Podmokelska-Ustecka [s]:')
T2=table;
T2.Scenario = Scen';
T2.personal_veh = TTpodm_ust(:,1);
T2.trucks = TTpodm_ust(:,2);
T2.buses = TTpodm_ust(:,3);
T2.all = TTpodm_ust(:,4);
T2.max_all = TTpodm_ust(:,5)
disp('Delay Ustecka-Podmokelska [s]:')
T3=table;
T3.Scenario = Scen';
T3.personal_veh = Delust_podm(:,1);
T3.trucks = Delust_podm(:,2);
T3.buses = Delust_podm(:,3);
T3.all = Delust_podm(:,4);
```

```
T3.max_all = Delust_podm(:,5)
disp('Delay Podmokelska-Ustecka [s]:')
T4=table;
T4.Scenario = Scen';
T4.personal_veh = Delpodm_ust(:,1);
T4.trucks = Delpodm_ust(:,2);
T4.buses = Delpodm_ust(:,3);
T4.all = Delpodm_ust(:,4);
T4.max_all = Delpodm_ust(:,5)
disp('Queue Ustecka-Podmokelska [m]:')
T5=table;
T5.Scenario = Scen';
T5.all = Qust_podm(:,1);
T5.max_all = Qust_podm(:,2)
disp('Queue Podmokelska-Ustecka [m]:')
T6=table;
T6.Scenario = Scen';
T6.all = Qpodm_ust(:,1);
T6.max_all = Qpodm_ust(:,2)
```

Autonomous vehicles

In the last table (Table 12), there is a source code for the autonomous vehicle chapter (*autonomous.m*).

Table 12: Source code for Autonomous Vehicles

```
%autonomous vehicles
clear all;
close all;
Vissim = actxserver('Vissim.Vissim.700'); % Start Vissim
Vissim.LoadNet
sim=Vissim.simulation;
vnet=Vissim.Net;
SC_number = 1;
SignalController = vnet.SignalControllers.ItemByKey(SC_number);
SG(1)=SignalController.SGs.ItemByKey(1);
SG(2)=SignalController.SGs.ItemByKey(2);
SH_1=vnet.SignalHeads.ItemByKey(1);
SH_1pos=get(SH_1, 'AttValue', 'Pos');
dets=SignalController.Detectors;
det_all=dets.GetAll;
det_1=det_all{1};
det_2=det_all{2};
det_1pos=get(det_1, 'AttValue', 'Pos');
v_veh = 58;
s = SH_1pos - det_1pos - 10;
v = v_veh/3.6;
t = floor(s/v)-1;
period_time=3600;
sim.set('AttValue', 'SimPeriod', period_time);
sim.get('AttValue', 'SimPeriod')
step_time=10;
sim.set('AttValue', 'SimRes', step_time);
max_cores=4;
sim.set('AttValue', 'NumCores', max_cores);
vehins=vnet.VehicleInputs;
load vehs.txt
veh_id = 1;
vehin(1)=vehins.ItemByKey(1);
Veh_composition_number = 2;
Rel_Flows =
vnet.VehicleCompositions.ItemByKey(Veh_composition_number).VehCompRelFlows.GetAll;
set(Rel_Flows{1}, 'AttValue', 'VehType',        100);
set(Rel_Flows{1}, 'AttValue', 'DesSpeedDistr',   1047);
set(Rel_Flows{1}, 'AttValue', 'RelFlow',        1);
vehin(1).set('AttValue', 'VehComp(1)', 2);
vehin(1).set('AttValue', 'Volume(1)', vehs(1,veh_id));
pedins=vnet.PedestrianInputs;
pedin(1)=pedins.ItemByKey(1);
pedin(1).set('AttValue', 'Volume(1)', 1000);
pedin(2)=pedins.ItemByKey(2);
```

```
pedin(2).set('AttValue', 'Volume(1)', 1100);
verify = 1;
time = 0;
tsg1 = 17;
tsg1max = 45;
tsg2min = 2;
sg1_time = 0;
sg2_time = 0;
relative_time = 0;
t12 = 0;
t21 = 0;
stage = 2;
demand = 0;
transition = 3;
shift = t - (transition+1);
wait = 0;
for i=0:(period_time*step_time)
sim.RunSingleStep;
if rem(i/step_time, verify)==0
   time=time+1;
   if stage == 1
    detect=det_1.get('AttValue', 'Detection');
    if detect == 1
        relative_time = 0;
    end
    if (relative_time >= (t-1))
        stage = 12;
        t12 = 0;
        sg1_time = 0;
        relative_time = 0;
        demand = 0;
    elseif (sg1_time > tsg1max)
        stage = 12;
        t12 = 0;
        sg1_time = 0;
        relative_time = 0;
        demand = 0;
    else
    SG(1).set('AttValue', 'State', 3);
    SG(2).set('AttValue', 'State', 1);
    sg1_time = sg1_time+1;
    relative_time = relative_time + 1;
    end
   end
   if stage == 2
      detect=det_1.get('AttValue', 'Detection');
      detect2=det_2.get('AttValue', 'Detection');
      if (detect == 1) || (demand == 1) || (detect2 == 1)
          if t >= tsg2min
              if (wait >= shift) && (sg2_time > tsg2min)
                  stage = 21;
                  sg2_time = 0;
                  t21 = 0;
                  wait = 0;
              else
                  SG(1).set('AttValue', 'State', 1);
                  SG(2).set('AttValue', 'State', 3);
                  sg2_time = sg2_time+1;
                  wait = wait + 1;
              end
          elseif (sg2_time >= tsg2min)
              stage = 21;
              sg2_time = 0;
              t21 = 0;
          else
              SG(1).set('AttValue', 'State', 1);
              SG(2).set('AttValue', 'State', 3);
              sg2_time = sg2_time+1;
          end
          demand = 1;
      else
          SG(1).set('AttValue', 'State', 1);
          SG(2).set('AttValue', 'State', 3);
          sg2_time = sg2_time+1;
      end
   end
   if stage == 12
```

```
    if (t12 == 0) || (t12 == 1)
     SG(1).set('AttValue', 'State', 4);
     SG(2).set('AttValue', 'State', 1);
    end
    if t12 == 2
     SG(1).set('AttValue', 'State', 1);
     SG(2).set('AttValue', 'State', 1);
     stage = 2;
     sg2_time = 0;
    end
    t12 = t12 + 1;
   end
   if stage == 21
    if t21 < transition
     SG(1).set('AttValue', 'State', 1);
     SG(2).set('AttValue', 'State', 1);
    end
    if t21 == transition
     SG(1).set('AttValue', 'State', 2);
     SG(2).set('AttValue', 'State', 1);
     stage = 1;
     sg1_time = 0;
     demand = 0;
    end
    t21 = t21 + 1;
   end
vehin(1).set('AttValue', 'Volume(1)', vehs(1,veh_id));
    if veh_id < length(vehs)
                veh_id = veh_id + 1;
    end
end
end
```