EINSTEIN CENTER Digital Future

Technische Universität Berlin

Tutorial 3: Custom Veins Example

Custom Veins Example

Prof. Dr. Sangyoung Park

Module ”Vehicle-2-X: Communication and Control“

Veins Simulator

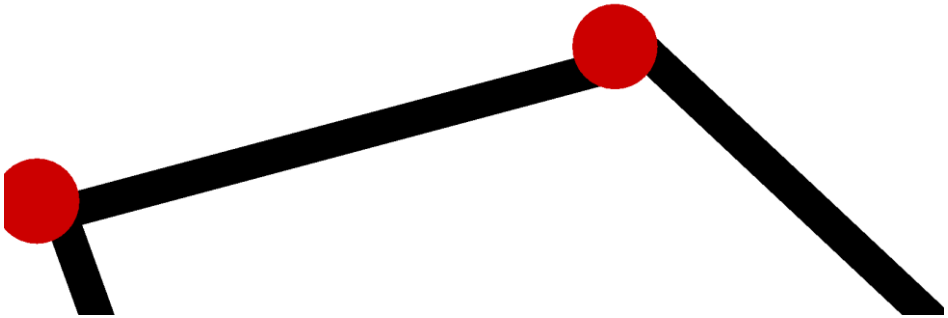
- We've ran a tutorial example before, but we don't know what it actually does
- Let's make a working example from scratch
- Step 1: Let's make a simple road network and traffic https://sumo.dlr.de/wiki/Tutorials/Driving_in_Circles
- Step 2: Check whether the code works with omnetpp.ini from veins tutorial
- Step 3: Let's make an application (or service which does nothing)
- Step 4: Let's play around with it a little bit

Step 1: Driving in Circles

- Faithfully follow the instructions from https://sumo.dlr.de/wiki/Tutorials/Driving_in_Circles
- Common mistakes
  - First try must end in an error. You must add the route information to circle.rou.xml
    <flow id="carflow" type="car" beg="0" end="0" number="5" from="edge1" to="edge2"/>
  - Don't forget to change the "id"s of the "edges" (not vertices) to edge1 and edge2
  - When adding circles.add.xml, you must add the following line to circles.sumocfg. Otherwise, SUMO simulation will not recognize the additional file
    <additional-files value="circles.add.xml"/>

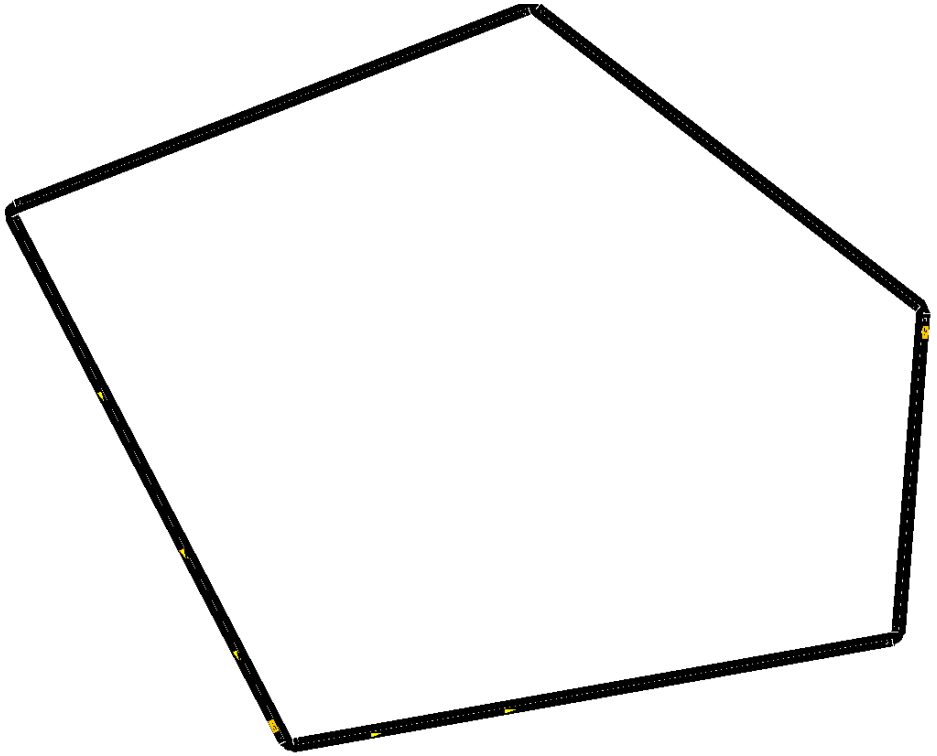
Step 1: Driving in Circles

- Common mistakes
  - You might encounter an error where the vehicles cannot find the path. This could be due to the fact that only one-way streets are used (see figure below, no path due to wrong alignment)
  - You could solve this by aligning the one-way streets, or using two-way for all streets



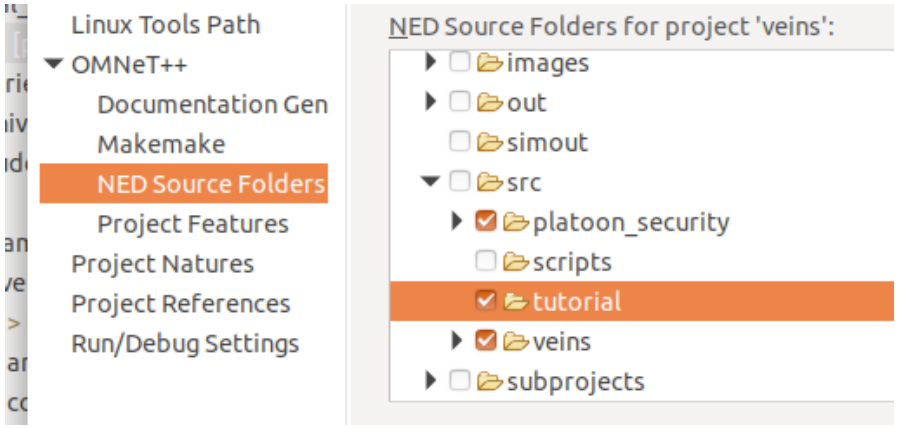
Step 1: Driving in Circles

- If you follow the steps correctly, you will see cars circulating forever due to rerouters



Step 2: Running Simulation from Veins

- Make a new folder for your sources codes in Veins project
  - Right click the [veins_folder]/src -> New -> Folder
  - Use whatever folder name
  - In this folder, you will put all your SUMO and Veins (OMNeT++) files
  - But before that let's do one thing
  - OMNeT++ simulator keeps track of all the NED files
    - We should let OMNeT++ know that NED files will exist in this folder
  - Right click veins project in the project explorer
  - In the pop-up window, expand OMNeT++ -> NED Source Folders, and check the folder you just created (in my case it was "tutorial")
  - Click apply and close



Step 2: Running Simulation from Veins

- Copy SUMO simulation files into your project folder
  - circles.*.xml
  - Yet you need another file “circles.launchd.xml”
    - This file will let Veins know all the SUMO files that will be used in Veins simulation
    - Change circles.sumo.cfg to circles.sumocfg if necessary

```
1  <?xml version="1.0"?>
2  <!-- debug config -->
3  <launch>
4      <copy file="circles.net.xml" />
5      <copy file="circles.rou.xml" />
6      <copy file="circles.add.xml" />
7      <copy file="circles.sumo.cfg" type="config" />
8  </launch>
9
10
```

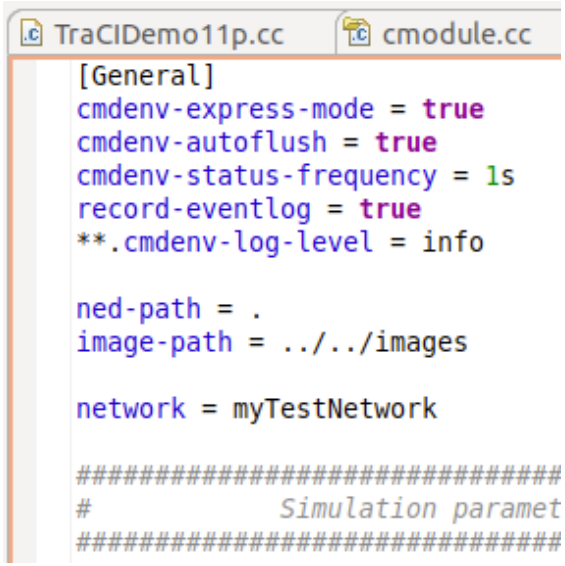
Step 2: Running Simulation from Veins

- Copy files from Veins example folder to your project folder
  - Antenna.xml
  - Config.xml
- Let's make a network description file
  - File -> New -> Network Description File (NED)
  - Make an empty file with a name of your choice
- Copy contents of RSUExampleScenario.ned to your NED file
  - It's in [veins_folder]/examples/veins/RSUExampleScenario.ned
  - But let's change the network name, because it will overlap with the original network name (I changed it to myTestNetwork in the figure below)

```
import org.car2x.veins.nodes.RSU;
import org.car2x.veins.nodes.Scenario;

network myTestNetwork extends Scenario
{
    submodules:
        rsu[1]: RSU {
            @display("p=50,50;i=veins/sign/yellowdiamond;is=vs");
        }
}
```

Step 2: Running Simulation from Veins

- Copying and modifying the omnetpp.ini file
  - There are a lot of things, e.g. simulation parameters, which can be configured from the file
  - As we are already using lots of codes from Veins such as RSU, cars, etc., it's more convenient to start with the existing omnetpp.ini file, which is in veins/examples/veins/omnetpp.ini
  - But, of course, we would have to modify it to use it for our simulation
    - We should change the name of the network we are simulating (myTestNetwork)



Step 2: Running Simulation from Veins

- Copying and modifying the omnetpp.ini file
  - And we have to let the ini file know that we are running our own SUMO traffic simulation (circles.launchd.xml)

```
##########################################################
#            TraCIScenarioManager parameters             #
##########################################################
*.manager.updateInterval = 1s
*.manager.host = "localhost"
*.manager.port = 9999
*.manager.autoShutdown = true
*.manager.launchConfig = xmldoc("circles.launchd.xml")
```

  - Finally, we have to define the behavior of RSUs and cars
    - Let's use MyVeinsApp
    - The source code is in veins/src/modules/application/traci
    - You can see the MyVeinsApp.cc, MyVeinsApp.h and MyVeinsApp.ned files in the folder

```
##########################################################
#                      WaveAppLayer                      #
##########################################################
*.node[*].applType = "MyVeinsApp"
*.node[*].appl.headerLength = 80 bit
*.node[*].appl.sendBeacons = false
*.node[*].appl.dataOnSch = false
*.node[*].appl.beaconInterval = 1s
```

Step 2: Running Simulation from Veins

```
##########################################################
#                       WaveAppLayer                     #
##########################################################
*.node[*].applType = "MyVeinsApp"
*.node[*].appl.headerLength = 80 bit
*.node[*].appl.sendBeacons = false
*.node[*].appl.dataOnSch = false
*.node[*].appl.beaconInterval = 1s
```

- The first line indicates that all node's (cars in this case), which appear in the Veins simulation will be assigned MyVeinsApp to it's property "applType"
- All nodes will be assigned "MyVeinsApp.ned" in the OMNeT++ simulation
- If you open MyVeinsApp.ned, you will see that it's designating the C++ class in MyVeinsApp.cc for behavior description

```
simple MyVeinsApp extends DemoBaseApplLayer
{
    @class(veins::MyVeinsApp);
    string appName = default("My first Veins App
}
```

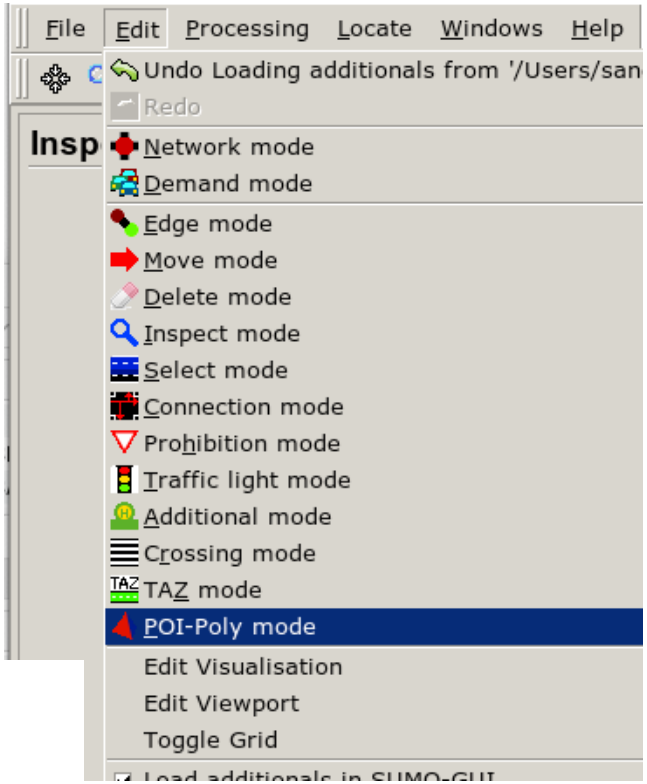
- In short, 1) omnetpp.ini let's you know which NED file cars will be assigned. 2) NED files point to the C++ class that cars will use

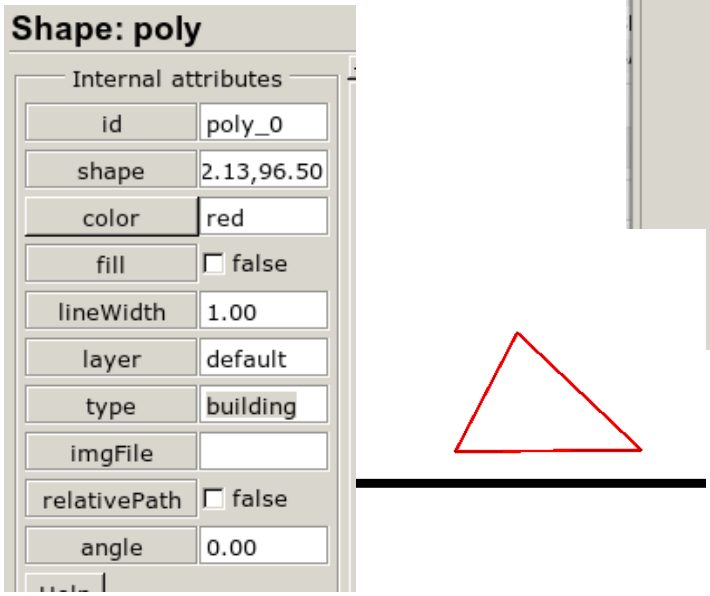
Step 3: Custom Application

- Please recall OSI model layers in the lecture part (communications)
  - PHY/MAC layers are also defined in the ini file
  - The ini file lets you configure various parameters
  - But now, we are interested in “application” layer
  - We’ve designated WAVE application as MyVeinsApp
- If you open MyVeinsApp.cc, there is nothing in the functions
  - This means that the application will do nothing upont receiving a WAVE packet
- Current ini file, by default, generates an accident
  - For now, let’s remove it from the ini file
  - *.node[*0].veinsmobility.accidentCount = 0
- Let’s run the simulation!
  - Right click the ini file and run as omnetpp simulation
- However, this will result in an error: OMNeT++ requires an obstacle

Adding Obstacles

- If you don‘t have any obstacles on the map, the simulation will show an error
- We can add obstacles on NETEDIT
- POI-Poly mode
- Draw any shape
- But *.poly.xml doesn’t yet know what the shape is
- Click the shape and designate the type is “building”





Adding Obstacles

- But the shapes are not saved in the *.net.xml
- Save the shape from
  - File->Additionals and shapes->Save additionals
  - *.poly.xml
- You should also let SUMO and Veins know that .poly.xml file exists
- Modify the sumocfg file and launchd.xml file

Add in sumocfg file…

```
<input>
    <net-file value="circles.net.xml"/>
    <route-files value="circles.rou.xml"/>
    <additional-files value="circles.add.xml, circles.poly.xml"/>
</input>
```

Also in launchd.xml file…

```
<?xml version="1.0"?>
<!-- debug config -->
<launch>
<copy file="circles.net.xml"/>
<copy file=" circles.rou.xml"/>
<copy file=" circles.add.xml"/>
<copy file=" circles.poly.xml"/>
<copy file=" circles.sumocfg" type="config"/>
</launch>
```

Step 4: Let's Make Vehicles Change Behavior

- Now, you can run Veins simulation without errors
- Enable beacon message from the RSU
  - Send beacons every 10 seconds (bottom figure)
  - If you run the simulation, you'll see beacon messages repeatedly sent from the RSU

```
##########################################################
#                       RSU SETTINGS                     #
#                                                        #
#                                                        #
##########################################################
*.rsu[0].mobility.x = 2000
*.rsu[0].mobility.y = 2000
*.rsu[0].mobility.z = 3

*.rsu[*].applType = "TraCIDemoRSU11p"
*.rsu[*].appl.headerLength = 80 bit
*.rsu[*].appl.sendBeacons = true
*.rsu[*].appl.dataOnSch = false
*.rsu[*].appl.beaconInterval = 10s
*.rsu[*].appl.beaconUserPriority = 7
*.rsu[*].appl.dataUserPriority = 5
*.rsu[*].nic.phy80211p.antennaOffsetZ = 0 m
```

Step 4: Let's Make Vehicles Change Behavior

- You'd have to define "bool hasStopped" somewhere, where should you define it? (recall C++ basics lecture)
- You'll have to initialize it somewhere, where would you initialize it?
- Now the cars repeatedly stop-and-go upon receiving beacon messages when you run the simulation

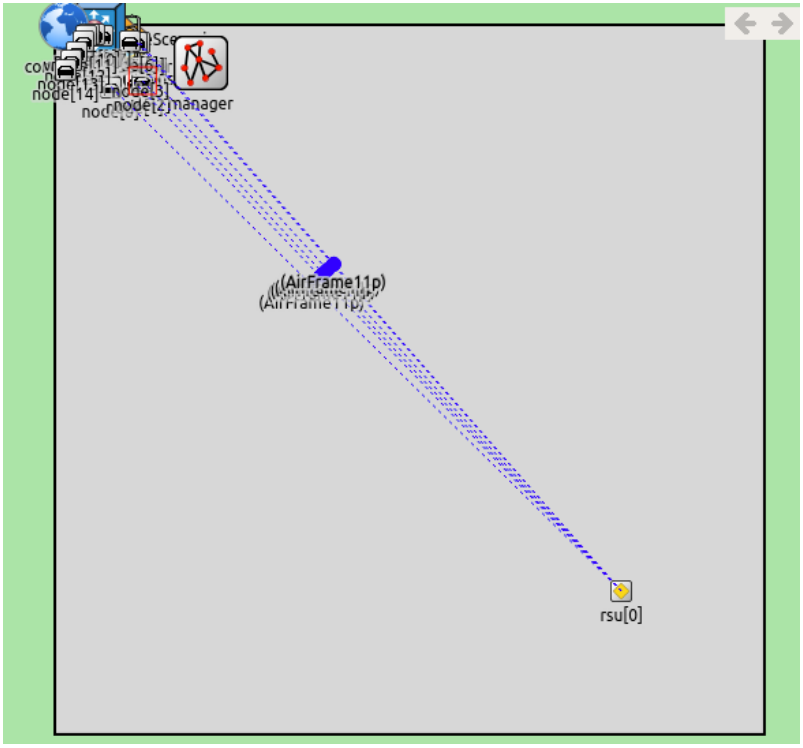
```
void MyVeinsApp::onBSM(DemoSafetyMessage* bsm)
{
    // Your application has received a beacon message from another car or RSU
    // code for handling the message goes here

    if (hasStopped == false)
    {
        traciVehicle->setSpeedMode(0x1f);
        traciVehicle->setSpeed(0);
        hasStopped = true;
    }
    else
    {
        traciVehicle->setSpeedMode(0x1f);
        traciVehicle->setSpeed(20);
        hasStopped = false;
    }
}
```

Step 4: Let's Make Vehicles Change Behavior

- What are the meanings of traciVehicle functions?
- traciVehicle is a pointer defined in a parent class of MyVeinsApp
  - Try right-clicking traciVehicle and "Open Declaration"
  - You will see where it's defined
  - It allows you to change the motion state of the vehicle in SUMO through TraCI interface
- Define cars' behavior upon receiving the beacon message
  - https://sumo.dlr.de/docs/TraCI/Change_Vehicle_State.html
  - "0x" if C++ language denotes that the integer is in hexadecimals
  - 0x1f = 11111 in binary numbers
- If you access the link above and go to "speed mode (0xb3)" you will see the meaning of each bits
  - 11111 means, regard everything
  - 00110 means, regard only maximum acceleration and deceleration

Step 4: Let's Make Vehicles Change Behavior

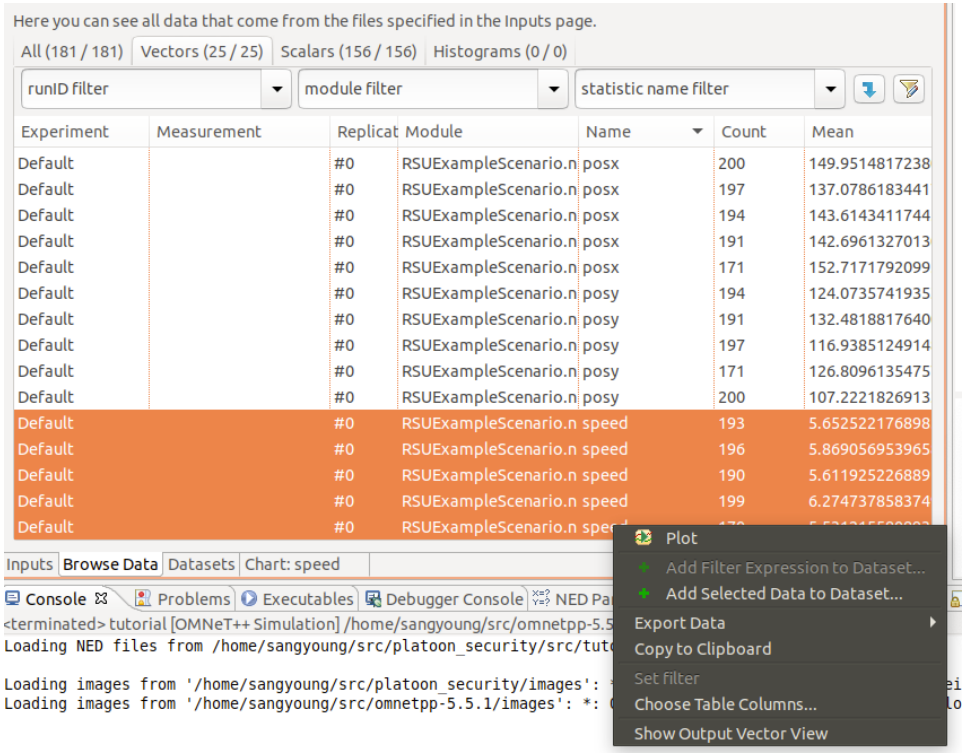
- Depending on the road network you have drawn, the location of RSU might be too far from the cars to communicate
- In such a case, adjust the location of the RSU from omnetpp.ini




```
##########################################################
#                       RSU SETTINGS                     #
#                                                        #
#                                                        #
##########################################################
*.rsu[0].mobility.x = 2000
*.rsu[0].mobility.y = 2000
*.rsu[0].mobility.z = 3
```

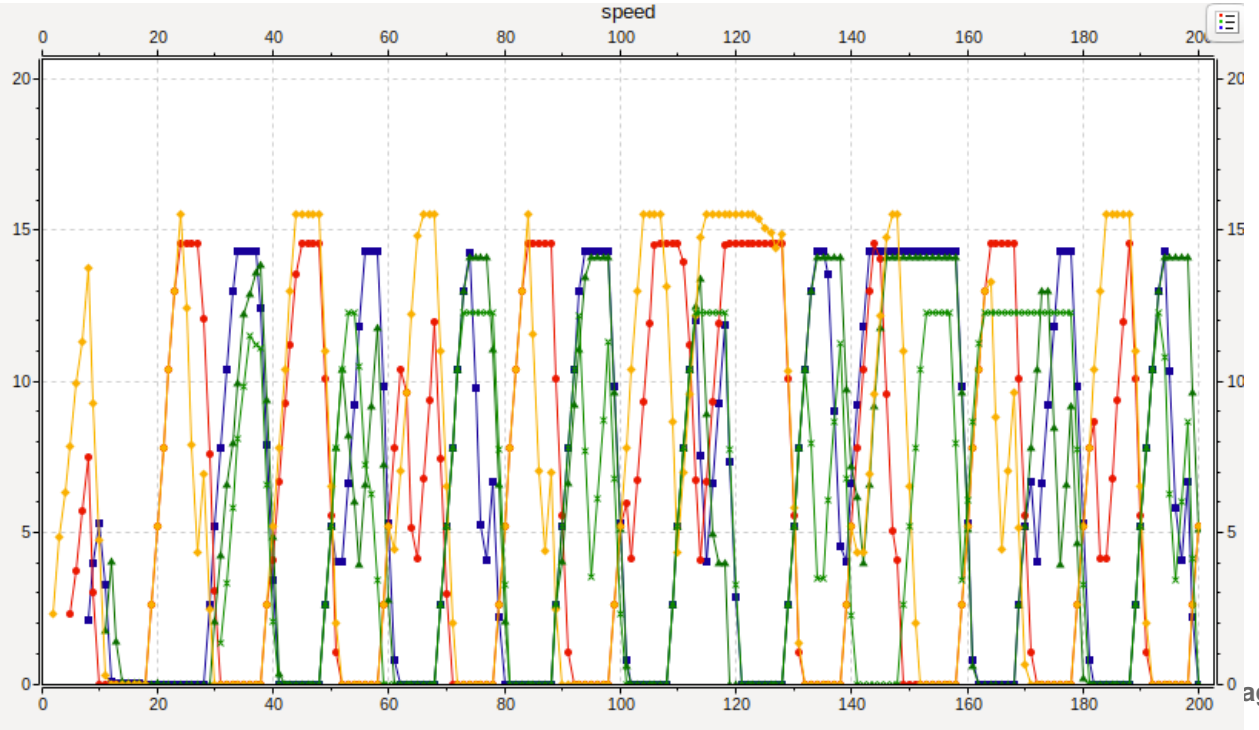
Step 5: Look at the Graphs

- From Veins simulation, it's hard to get a grasp of how the cars are actually moving
- There are two ways to analyze the movements
- First, draw the graph
  - After the simulation ends, "results" folder is generated
  - Double click Default-#0.vec and generate Default.anf
  - Double click Default.anf -> |Browse Data (from bottom tab) -> Vectors (from upper tab) -> select data you want to display -> right-click -> Plot



Step 5: Look at the Graphs

- Graphs are generated from the simulation data
- You can see that the vehicle are brought to repeated stops every 20 seconds
- It's not very clean because not all the cars appear in the simulation at the same time, so upon receiving a message, some cars stop and some cars go (alternatively)



Step 5: Look at the Graphs

- Second way to check how the vehicle move would be to use SUMOGUI instead of SUMO
- When you launch the python script, you can also type
  - [yourveinspath]/sumo-launchd.py –vv –c [yoursumopath]/sumo-gui
- Then, when you run OMNeT++ simulation, SUMO-GUI will be launched
- This time try using “Express” instead of “Run” in SUMO-GUI
- SUMO-GUI will launch, input some delay (e.g. 100 ms) and run the SUMO simulation
- You’ll be able to see how the cars are moving in real-time