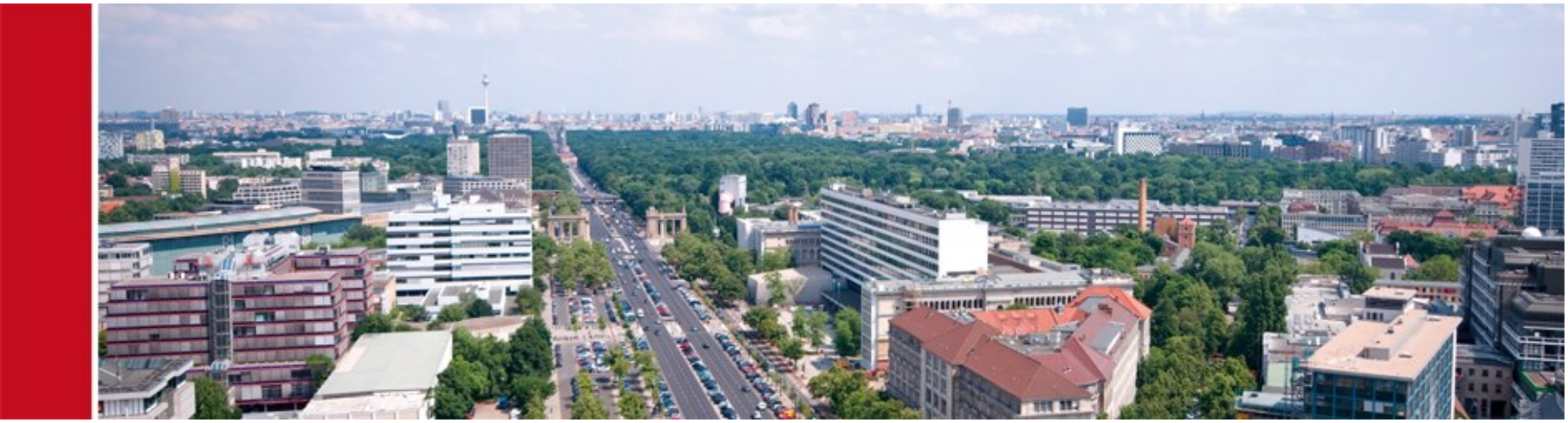
EINSTEIN CENTER Digital Future

Technische Universität Berlin



Tutorial 3: Custom Veins Example

Custom Veins Example

Prof. Sangyoung Park

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

For those of you not familiar with C++

- C++ is an objective-oriented programming language
  - Deals with classes
  - Classes have member functions and member variables
  - Public: any member can access the functions and variables
  - Private: only the object itself can access the functions and variables
- Inheritance
  - Classes can inherit other classes
- The best way to deal with it is to do it yourself
- If you are not familiar with C++, I can provide a very simple example code
  - Test.cc is about basics of classes
  - Car.cc is about basics of class inheritance

For those of you not familiar with C++

- Quickest way to test your code is to
- On Msss terminal, type
  - >> g++ test.cc
  - >> ./a.exe
- You will be able to find out how the code runs
- Besides this, the programming is basically Googling
  - For example, if you are wondering what „printf“ is,
  - Please google it for the function description
  - C++ standard library (STL) documentation is available on the web

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 simpler road network and traffic https://sumo.dlr.de/wiki/Tutorials/Driving_in_Circles
- Step 2: Check whether the code works with omnetpp.ini from the 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 (demo will be shown)

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 error you must add the route information to circles.rou.xml

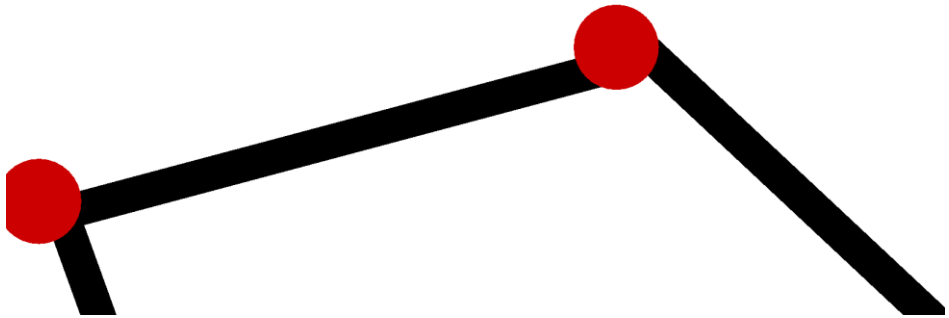
    <flow id="carflow" type="car" beg="0" end="0" number="5" from="edge1" to="edge2"/>

  - 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 to circles.sumocfg. Otherwise, SUMO simulation will not recognize the additional file
    <additional-files value="circles.add.xml"/>

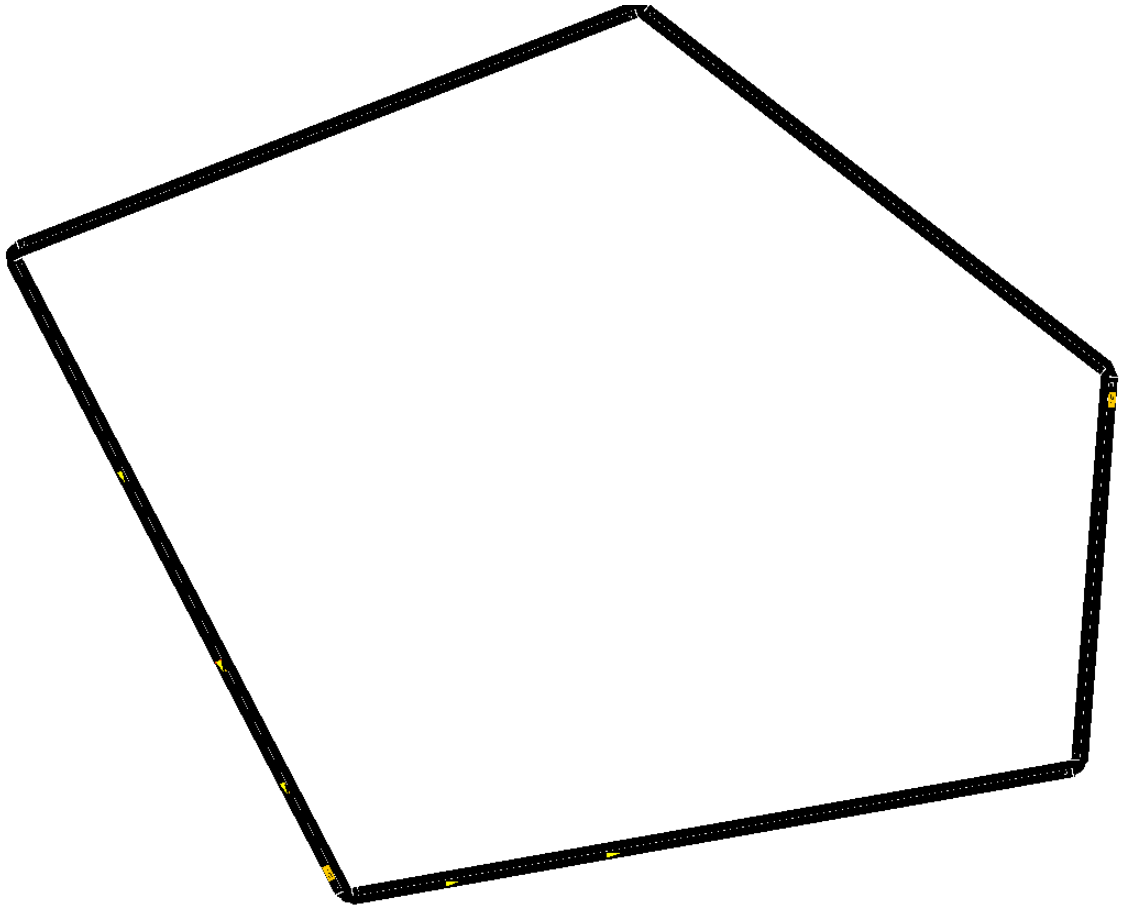
Step 1: Driving in Circles

- 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 adding two-way for all streets



Step 1: Driving in Circles

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



Step 2: Running the Simulation from Veins

- Make a new Omnet++ project from Omnet++
  - File -> New -> Omnet++ Project
  - Use whatever project name (but should not overlap with other existing project names) and location you prefer
  - Choose an empty project
  - Finish
- Copy SUMO simulation files into your project folder
  - circles.*.xml
  - Yet you need another file „circles.launchd.xml“

```
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 the 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 your choice of name
- Copy contents of RSUExampleScenario.ned to our 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)

```
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 the Simulation from Veins

- Reference to Veins libraries
  - There will be lots of errors because the Omnet++ simulator is a network simulator. By default, it is not aware of RSUs, Cars, etc., which is implemented in Veins
  - We need to reference the libraries that Veins developers have made
  - Right-click your project in the project explorer (in my case newTest below)
  - Properties -> Project References -> Click Veins -> “Apply and Close”

Step 2: Running the Simulation from Veins

- Copying and modifying the omnetpp.ini file
  - There are a lot of things (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 have to modify it
    - We should change the name of the network we are simulating (myTestNetwork)
    - We should comment out the obstacle model because there's no obstacle such as buildings in our simulation

```
[General]
cmdenv-express-mode = true
cmdenv-autoflush = true
cmdenv-status-frequency = 1s
**.cmdenv-log-level = info

ned-path = .
image-path = ../../images

network = myTestNetwork

##########################################################
#           Simulation parameters                        #
##########################################################
```

```
##########################################################
# Obstacle parameters                                    #
##########################################################
*.obstacles.debug = false
#*.obstacles.obstacles = xmldoc("config.xml", "//AnalogueModel[@type='SimpleObstacleShadowing']
```

Step 2: Running the Simulation from Veins

- Copying and modifying the omnetpp.ini file
  - And we have to let the ini file know that we are running circles traffic simulation

```
##########################################################
#            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

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

Step 3: Custom Application

- Please recall OSI model layers
  - 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 upon 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 the run simulation!
  - Right click the ini file and run as omnetpp simulation
- You‘ll see cars running in circles as you have seen from the SUMO simulation

Step 4: Let‘s make the vehicles change behavior

- Enable beacon messages from the RSU
  - SendBeacons every 10 seconds
- Define cars‘ behavior upon receiving the beacon message
  - https://sumo.dlr.de/wiki/TraCI/Change_Vehicle_State#speed_mode_.280xb3.29
- Now the cars repeatedly stop-and-go upon receiving beacon msg

```
##########################################################
#                        RSU SETTINGS                    #
#                                                        #
#                                                        #
##########################################################
*.rsu[0].mobility.x = 250
*.rsu[0].mobility.y = 250
*.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
```

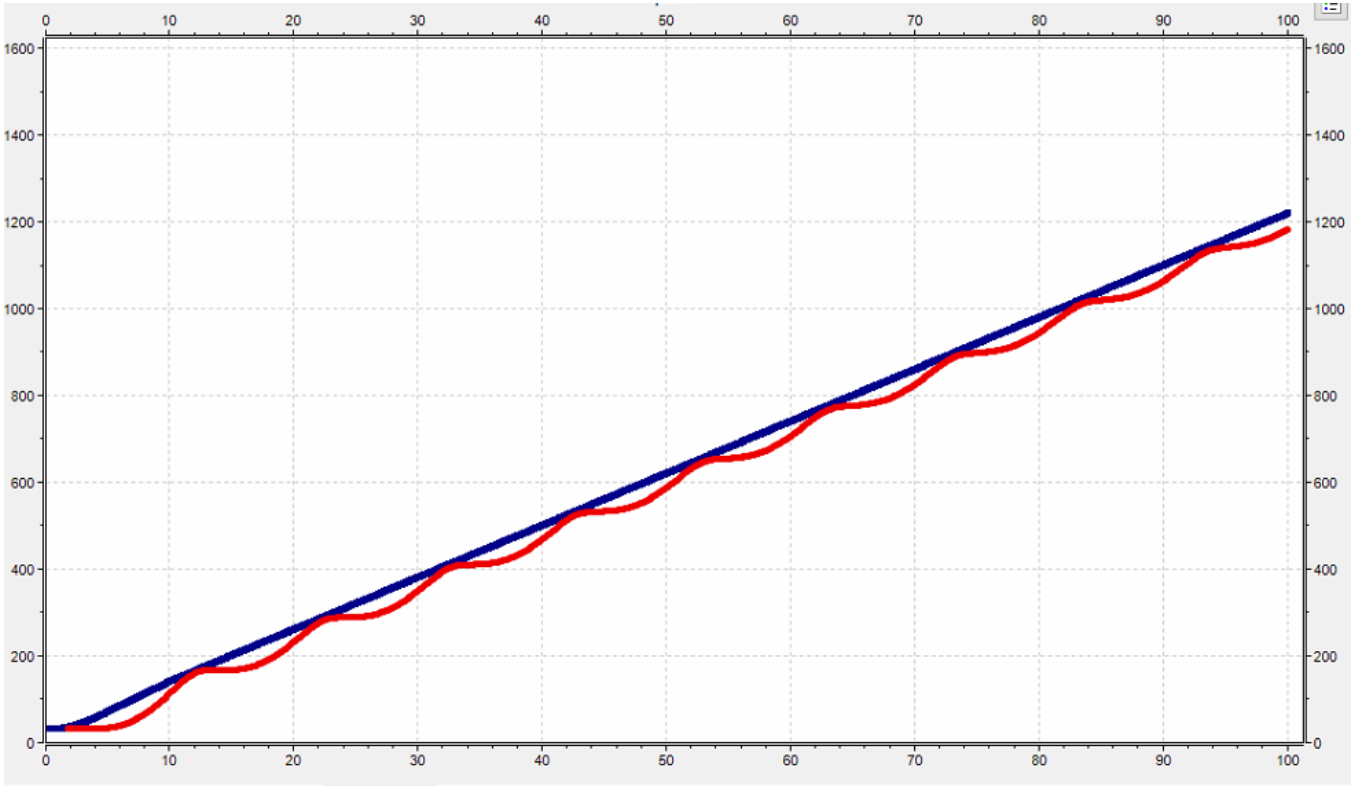
```
void MyVeinsApp::onBSM(BasicSafetyMessage* bsm) {
    //Your application has received a beacon message from another car or RSU
    //code for handling the message goes here

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

}
```

Step 5: Let‘s initiate a WAVE Service

- But the simulation speed is too slow!
- Run with express speed
- You can see the results afterwards
- In the results folder in your project, there are multiple files generated
- Double click *.vec file and an output file (with extension .anf) will be generated
- Go to “vectors” tab, and select the data you want to display, right click and plot
- You will have graphs like the one on the next page (not exactly the same)


0
10
20
30
40
50
60
70
80
90
100
1600
1400
1200
1000
800
600
400
200
0