Tutorial 6: Basic Platooning Implementation

Basic Platooning Implementation

Prof. Sangyoung Park
Module ”Vehicle-2-X: Communication and Control“

Implementation of Simple Platooning Algorithm

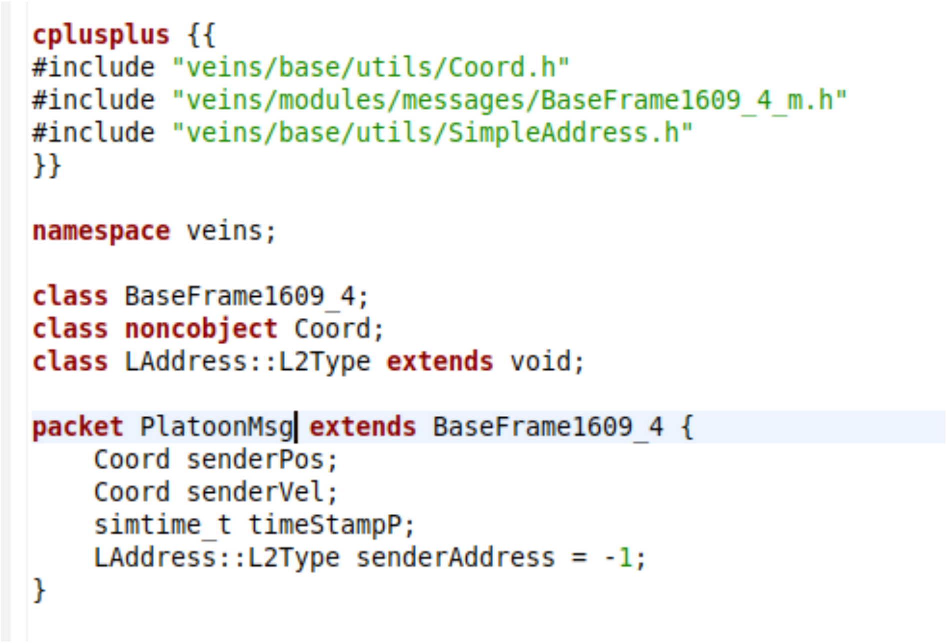
- Let‘s start with something simple
- Let‘s read the distance to the preceding vehicle only and try to adjust the acceleration of the current vehicle
- Would you be able to implement this?
- $a = p \cdot (d - d_{desired}) + d \cdot (v - v_{pred})$

Make a New Folder

- simple_platoon folder within veins/src folder
- Let‘s make SimplePlatoon.cc and SimplePlatoon.h, which will be vehicle controllers

Let‘s Define a Message

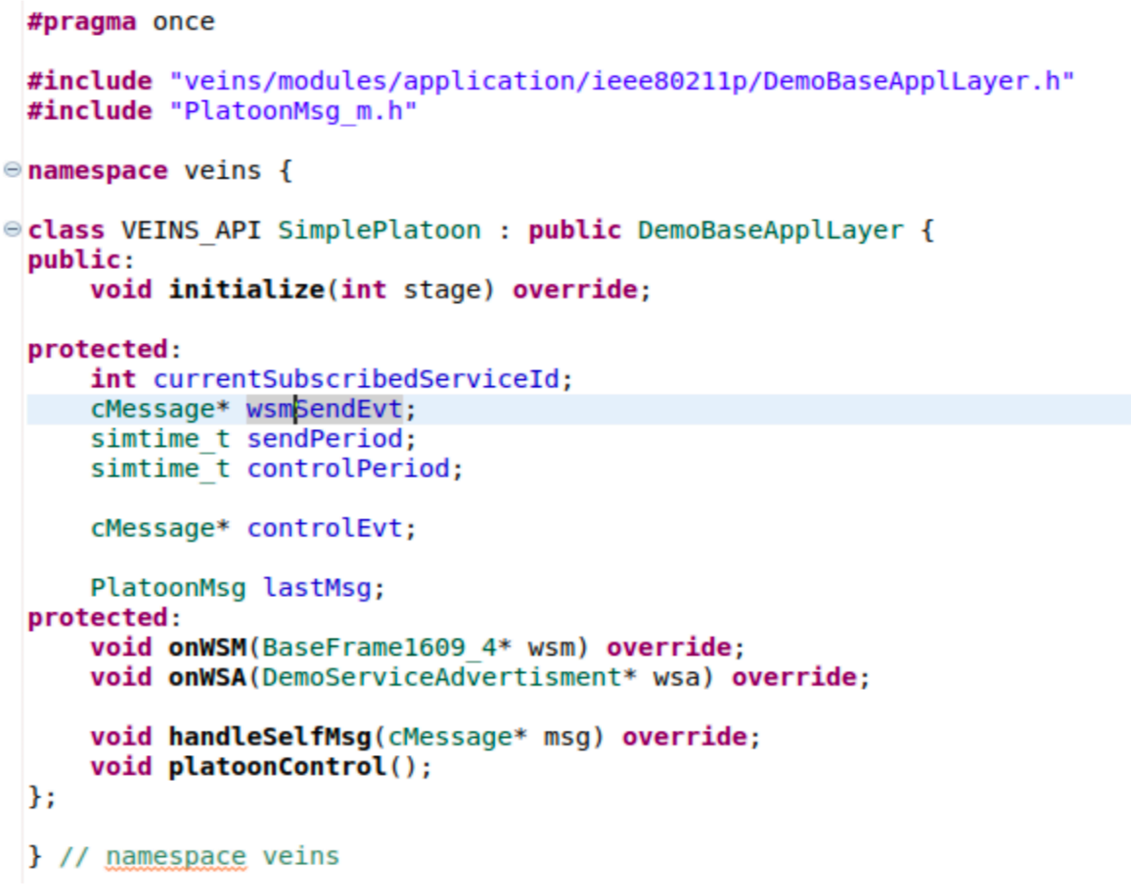
- We need a message which contains velocity information
- Let‘s name it PlatoomMsg.msg



```
cplusplus {{
#include "veins/base/utils/Coord.h"
#include "veins/modules/messages/BaseFrame1609_4_m.h"
#include "veins/base/utils/SimpleAddress.h"
}}

namespace veins;

class BaseFrame1609_4;
class noncobject Coord;
class LAddress::L2Type extends void;

packet PlatoonMsg extends BaseFrame1609_4 {
    Coord senderPos;
    Coord senderVel;
    simtime_t timeStampP;
    LAddress::L2Type senderAddress = -1;
}
```

Definining a Periodic Task (Platoon Control)

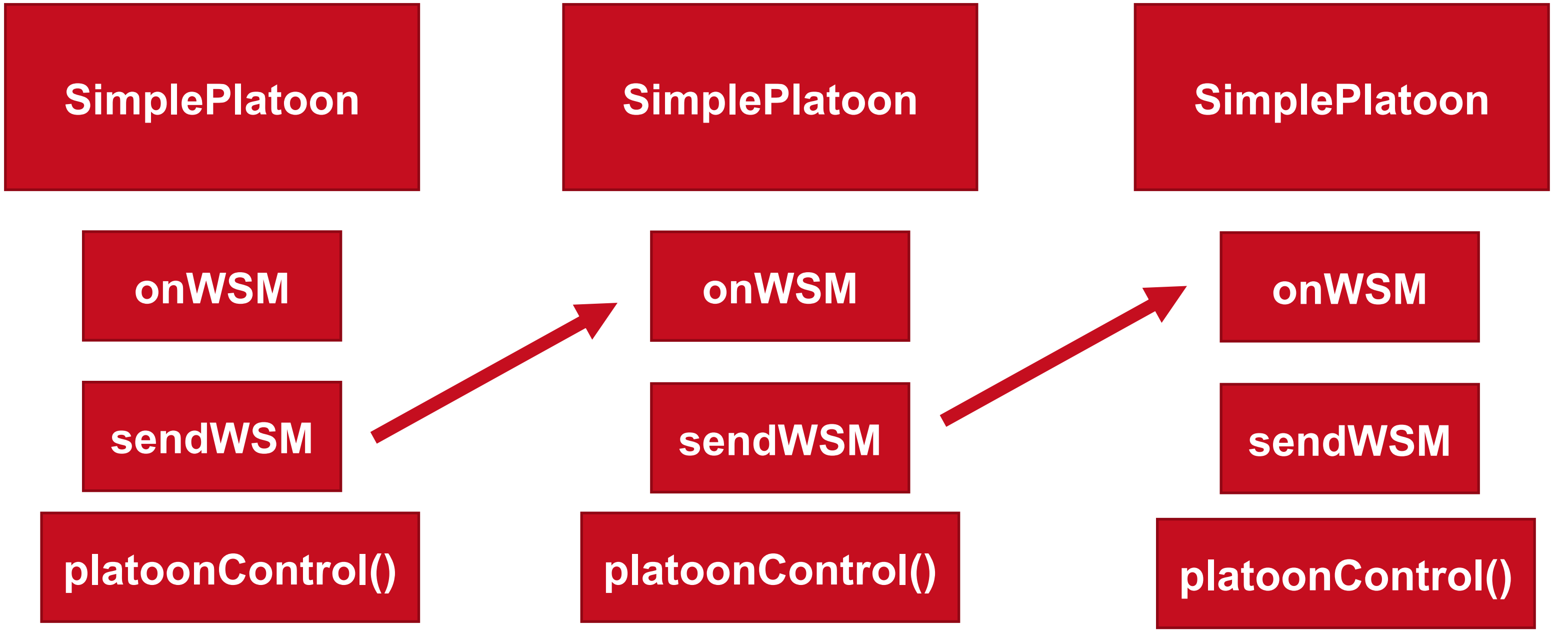
- SimplePlatoon.h
- Take a note at controlEvt;
- What we intend to do is
  - When WSM is received
  - We save it in lastMsg
- There‘s a separate loop
  - Executed every controlPeriod



```cpp
#pragma once

#include "veins/modules/application/ieee80211p/DemoBaseApplLayer.h"
#include "PlatoonMsg_m.h"

namespace veins {

class VEINS_API SimplePlatoon : public DemoBaseApplLayer {
public:
    void initialize(int stage) override;

protected:
    int currentSubscribedServiceId;
    cMessage* wsmSendEvt;
    simtime_t sendPeriod;
    simtime_t controlPeriod;

    cMessage* controlEvt;

    PlatoonMsg lastMsg;
protected:
    void onWSM(BaseFrame1609_4* wsm) override;
    void onWSA(DemoServiceAdvertisment* wsa) override;

    void handleSelfMsg(cMessage* msg) override;
    void platoonControl();
};

} // namespace veins
```

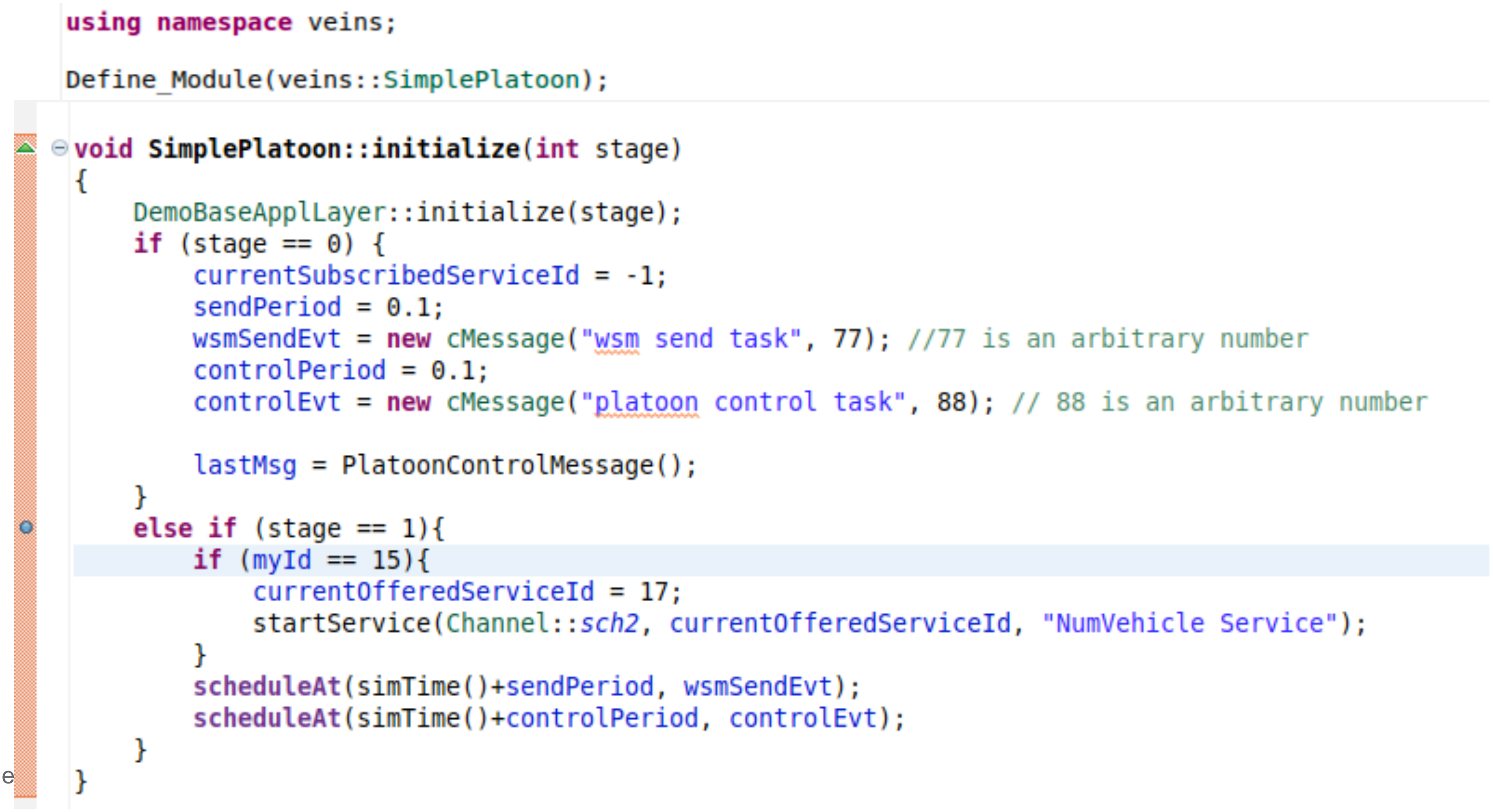
Don‘t Get Confused

- We are writing a single C++ file
- But there‘ll be multiple instances of SimplePlatoon each running on respective vehicles



SimplePlatoon.cc

- We use scheduleAt() two times now
- The lead vehicle (myId: 15, it‘s 9 if there‘s no RSU) announces this service, and start broadcasting it‘s own speed/position information



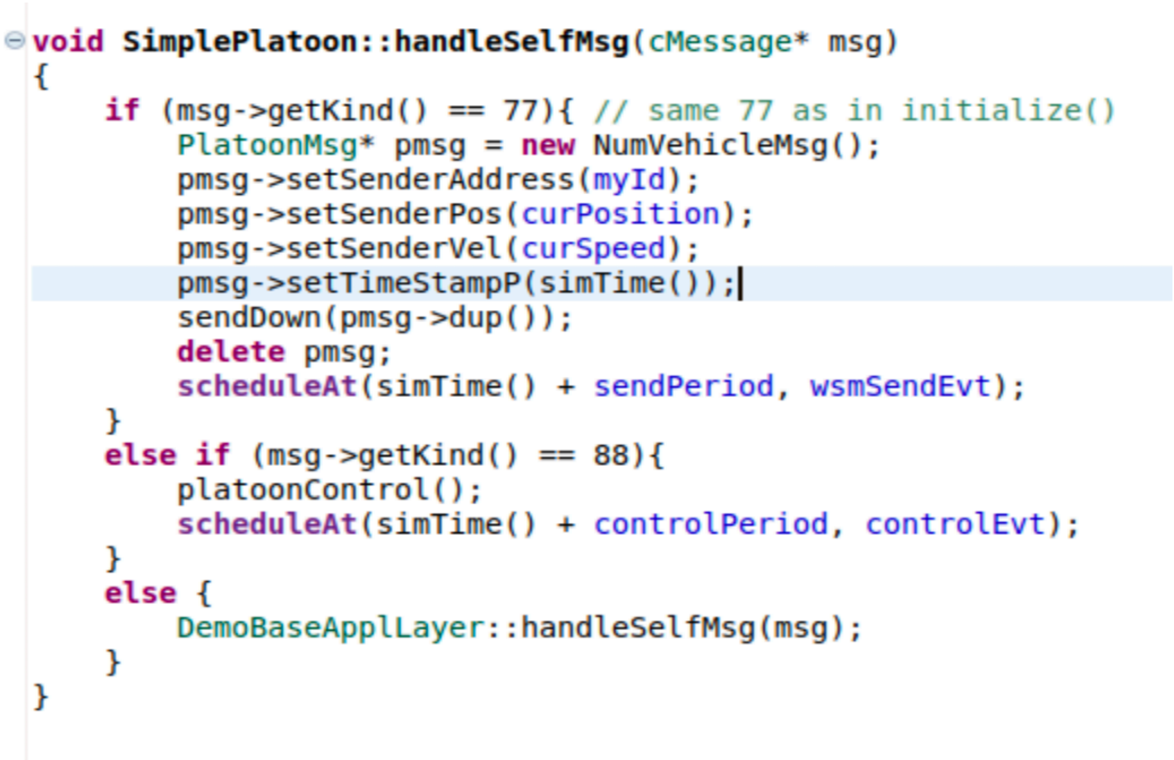
```
using namespace veins;

Define_Module(veins::SimplePlatoon);

void SimplePlatoon::initialize(int stage)
{
    DemoBaseApplLayer::initialize(stage);
    if (stage == 0) {
        currentSubscribedServiceId = -1;
        sendPeriod = 0.1;
        wsmSendEvt = new cMessage("wsm send task", 77); //77 is an arbitrary number
        controlPeriod = 0.1;
        controlEvt = new cMessage("platoon control task", 88); // 88 is an arbitrary number

        lastMsg = PlatoonControlMessage();
    }
    else if (stage == 1){
        if (myId == 15){
            currentOfferedServiceId = 17;
            startService(Channel::sch2, currentOfferedServiceId, "NumVehicle Service");
        }
        scheduleAt(simTime()+sendPeriod, wsmSendEvt);
        scheduleAt(simTime()+controlPeriod, controlEvt);
    }
}
```

Identifying the Sender

- The ID of the VehicleControlApp can be obtained by directly accessing myId of the class
- Try to make use of the debugger to find out the IDs appearing in the simulation
  - std::cout << myId << std::endl;
  - In initialize() and see what is printed in the console and use debugger toset breakpoints and read the myIds
- So, the IDs will be 15, 21, 27, 33, 39, ... (starts with 9 without RSU, but you can check yourself)

SimplePlatoon.cc

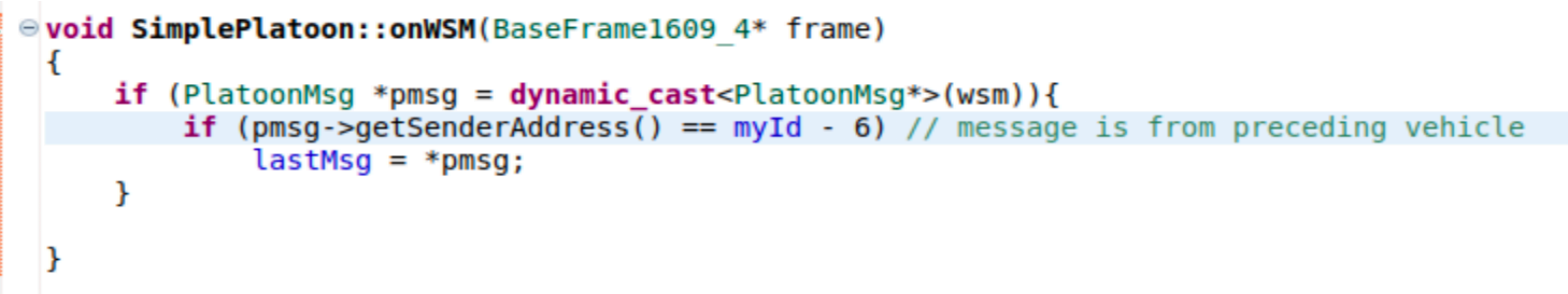
- Periodically executing the tasks can be handled here (see second if clause)
- First if clause is similar to the previous tutorial except that there‘s „curSpeed“



```
void SimplePlatoon::handleSelfMsg(cMessage* msg)
{
    if (msg->getKind() == 77){ // same 77 as in initialize()
        PlatoonMsg* pmsg = new NumVehicleMsg();
        pmsg->setSenderAddress(myId);
        pmsg->setSenderPos(curPosition);
        pmsg->setSenderVel(curSpeed);
        pmsg->setTimeStampP(simTime());
        sendDown(pmsg->dup());
        delete pmsg;
        scheduleAt(simTime() + sendPeriod, wsmSendEvt);
    }
    else if (msg->getKind() == 88){
        platoonControl();
        scheduleAt(simTime() + controlPeriod, controlEvt);
    }
    else {
        DemoBaseApplLayer::handleSelfMsg(msg);
    }
}
```

onWSM

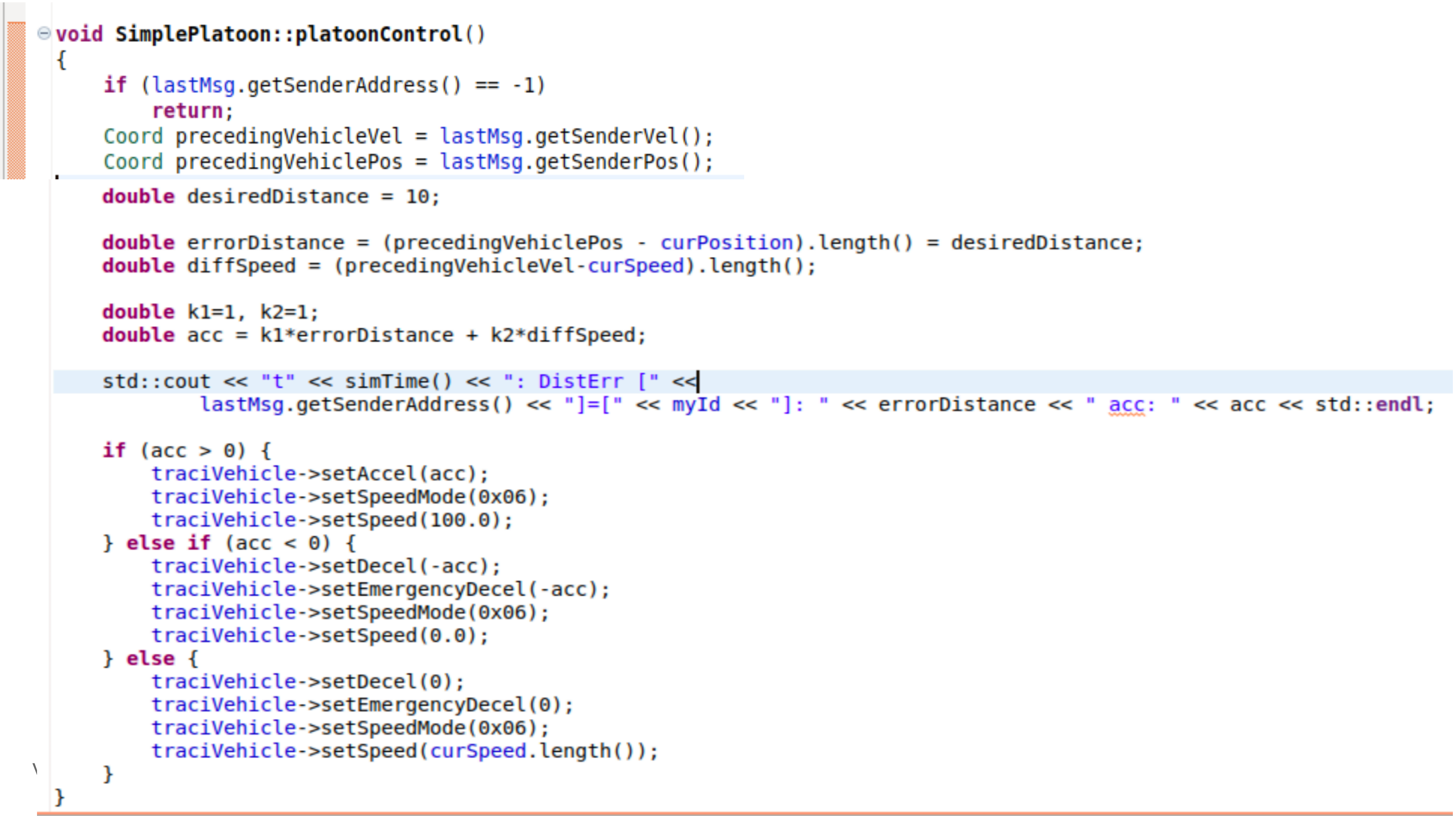
- We save WSM upon receiving WSM and use it later in platoonControl()
- But we are only interested in WSM from the preceding vehicle only
- I found out that the myId of preceding vehicle is smaller by 6
    - How did I find out? (Debugger!)



```
void SimplePlatoon::onWSM(BaseFrame1609_4* frame)
{
    if (PlatoonMsg *pmsg = dynamic_cast<PlatoonMsg*>(wsm)){
        if (pmsg->getSenderAddress() == myId - 6) // message is from preceding vehicle
            lastMsg = *pmsg;
    }

}
```

platoonControl()

- We make use of the data
- Notice the code is going to print logs on console, you can use it to debug



```
void SimplePlatoon::platoonControl()
{
    if (lastMsg.getSenderAddress() == -1)
        return;
    Coord precedingVehicleVel = lastMsg.getSenderVel();
    Coord precedingVehiclePos = lastMsg.getSenderPos();
    double desiredDistance = 10;

    double errorDistance = (precedingVehiclePos - curPosition).length() = desiredDistance;
    double diffSpeed = (precedingVehicleVel-curSpeed).length();

    double k1=1, k2=1;
    double acc = k1*errorDistance + k2*diffSpeed;

    std::cout << "t" << simTime() << ": DistErr [" <<
            lastMsg.getSenderAddress() << "]=[" << myId << "]: " << errorDistance << " acc: " << acc << std::endl;

    if (acc > 0) {
        traciVehicle->setAccel(acc);
        traciVehicle->setSpeedMode(0x06);
        traciVehicle->setSpeed(100.0);
    } else if (acc < 0) {
        traciVehicle->setDecel(-acc);
        traciVehicle->setEmergencyDecel(-acc);
        traciVehicle->setSpeedMode(0x06);
        traciVehicle->setSpeed(0.0);
    } else {
        traciVehicle->setDecel(0);
        traciVehicle->setEmergencyDecel(0);
        traciVehicle->setSpeedMode(0x06);
        traciVehicle->setSpeed(curSpeed.length());
    }
}
```

Setting Acceleration Values

- Veins does not provide an interface to directly control the acceleration of vehicles
- We could do the following work around (maybe there’s a better way)
  - Set maximum acceleration or deceleration value
  - Set a very high speed or low speed to ensure that the vehicle is taking that maximum acceleration or deceleration value
- But Veins doesn’t provide an interface to control the *max acceleration* and *max deceleration* either
- Let’s try implement the functionalities

New Functions to the TraCI Interface

- TraCI interface is no magic, all the commands and API (functions we could use) are defined in the following three files in the folder veins/src/veins/modules/mobility/traci/
  - TraCICommandInterface.cc and TraCICommandInterface.h
  - TraCIConstants.h
- For example, if you look at the function we already used, “setSpeed()”
  - You can see that variableId = VAR_SPEED
  - VAR_SPEED is defined in TraCIConstants.h as 0x40
  - You can also see 0x40 from https://sumo.dlr.de/wiki/TraCI/Change_Vehicle_State

```
void TraCICommandInterface::Vehicle::setSpeed(double speed) {
uint8_t variableId = VAR_SPEED;
uint8_t variableType = TYPE_DOUBLE;
TraCIBuffer buf = traci->connection.query(CMD_SET_VEHICLE_VARIABLE, TraCIBuffer() << variableId << nodeId <<
variableType << speed);
ASSERT(buf.eof());
}
```

New Functions to the TraCI Interface

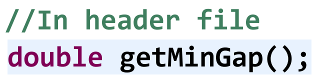
- So, we could implement the functions setAccel() and setDecel() in a similar way
- Define the function format in the header file (.h)
- Define the function in the cc file (.cc)

```
// in TraCICommandInterface.h
void setAccel(double accel);
void setDecel(double decel);
```

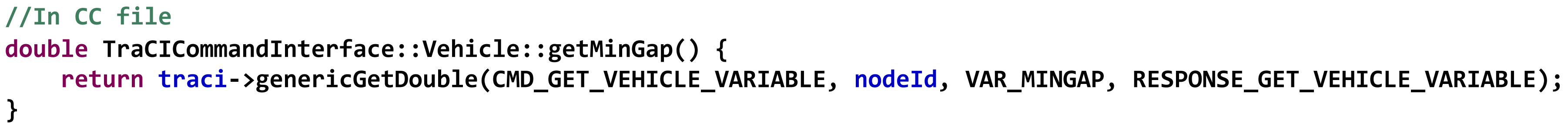
```
// added by spark
void TraCICommandInterface::Vehicle::setAccel(double
accel) {
    uint8_t variableId = VAR_ACCEL;
    uint8_t variableType = TYPE_DOUBLE;
    TraCIBuffer buf = traci-
>connection.query(CMD_SET_VEHICLE_VARIABLE, TraCIBuffer()
<< variableId << nodeId << variableType << accel);
    ASSERT(buf.eof());
}

// added by spark
void TraCICommandInterface::Vehicle::setDecel(double
decel) {
    uint8_t variableId = VAR_DECEL;
    uint8_t variableType = TYPE_DOUBLE;
    TraCIBuffer buf = traci-
>connection.query(CMD_SET_VEHICLE_VARIABLE, TraCIBuffer()
<< variableId << nodeId << variableType << decel);
    ASSERT(buf.eof());
}
```

What About Reading Variables using TraCI?

- For example, you can read the minGap parameter in the car following model (recall car following model lecture)
- https://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes



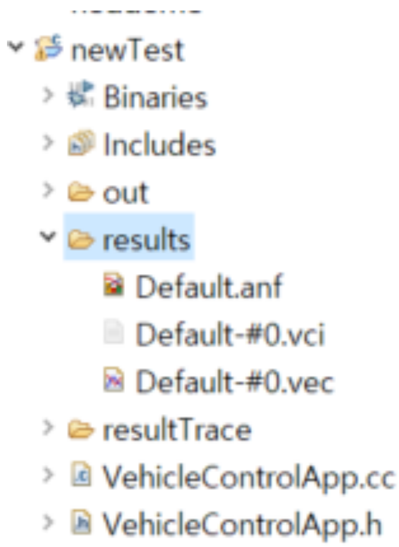
```
//In header file
double getMinGap();
```



```
//In CC file
double TraCICommandInterface::Vehicle::getMinGap() {
    return traci->genericGetDouble(CMD_GET_VEHICLE_VARIABLE, nodeId, VAR_MINGAP, RESPONSE_GET_VEHICLE_VARIABLE);
}
```

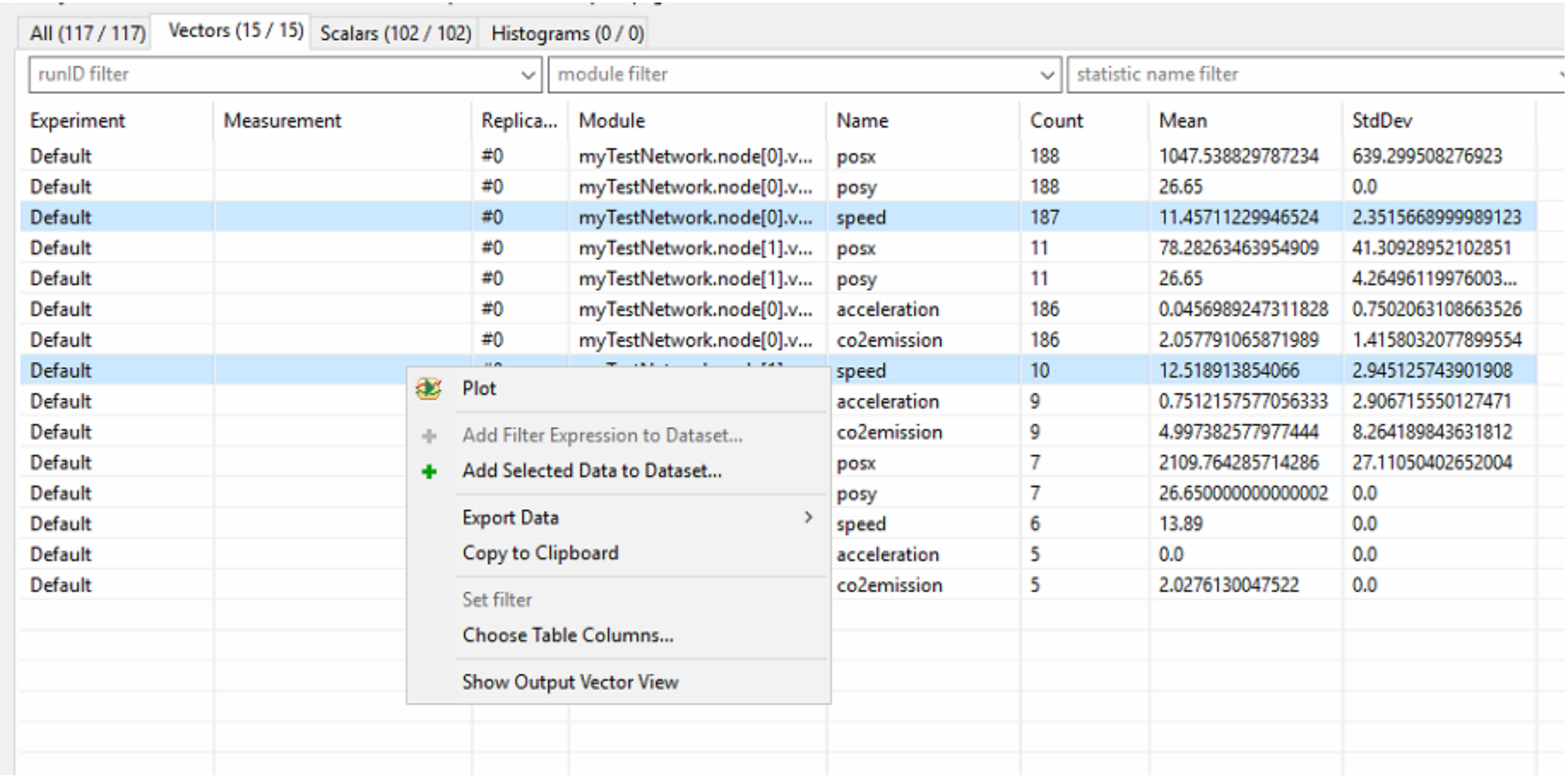
Plotting the Results

- Fortunately, Veins provides its own statistics mechanism, so we can just make use of it
- After you simulate anything, data will be generated in the results folder
- If you double click .vec file you will be able to generate .anf file
- In the tab “browse data” at the bottom,
  and then “vectors” tab at the top,
  you will be able to generate graphs about
  the position, velocity, and acceleration of vehicles



```
v newTest
  > Binaries
  > Includes
  > out
  v results
      Default.anf
      Default-#0.vci
      Default-#0.vec
  > resultTrace
  > VehicleControlApp.cc
  > VehicleControlApp.h
```

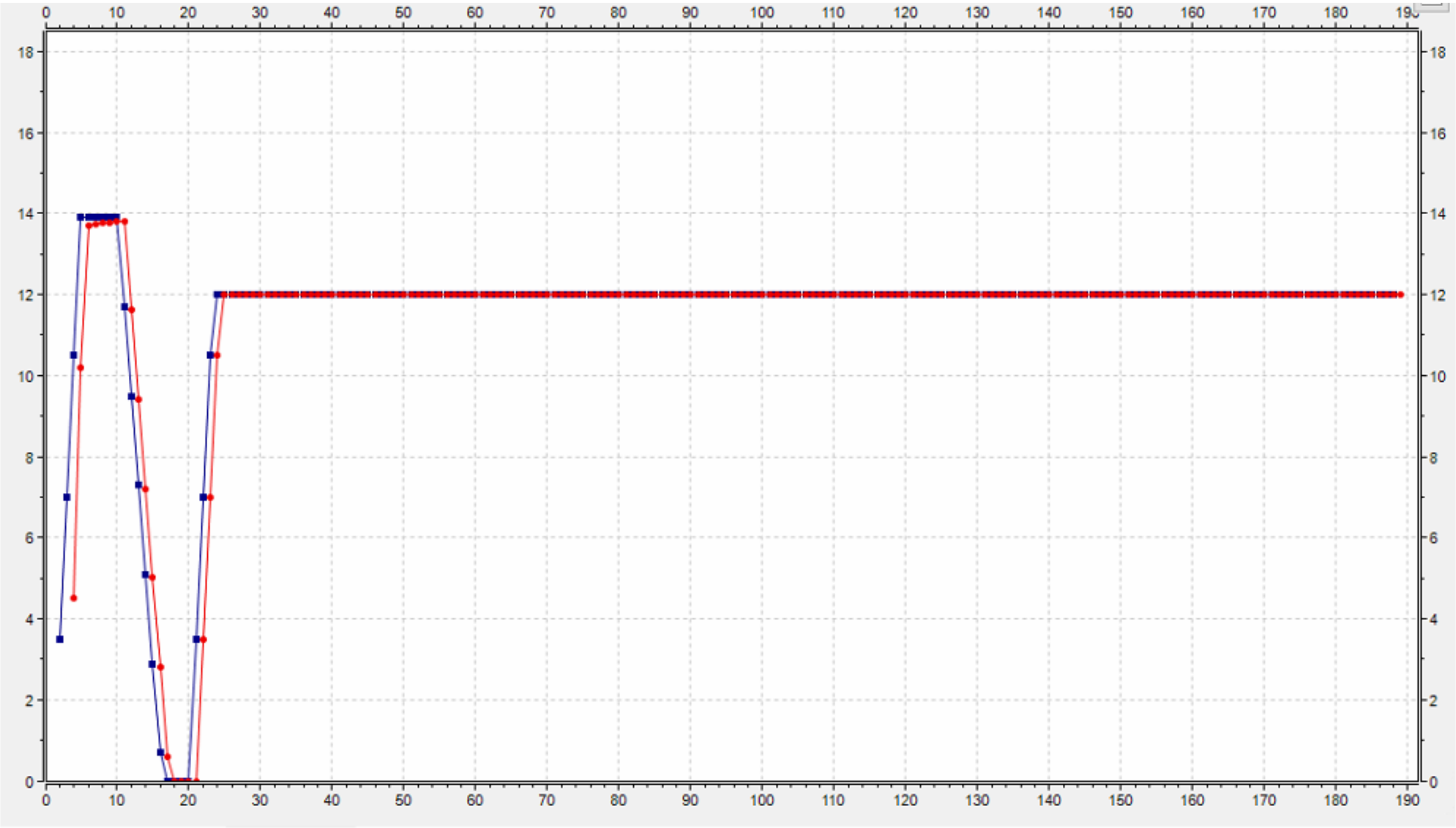
Plotting the Results



Experiment	Measurement	Replica...	Module	Name	Count	Mean	StdDev
Default		#0	myTestNetwork.node[0].v...	posx	188	1047.538829787234	639.299508276923
Default		#0	myTestNetwork.node[0].v...	posy	188	26.65	0.0
Default		#0	myTestNetwork.node[0].v...	speed	187	11.45711229946524	2.3515668999989123
Default		#0	myTestNetwork.node[1].v...	posx	11	78.28263463954909	41.30928952102851
Default		#0	myTestNetwork.node[1].v...	posy	11	26.65	4.26496119976003...
Default		#0	myTestNetwork.node[0].v...	acceleration	186	0.0456989247311828	0.7502063108663526
Default		#0	myTestNetwork.node[0].v...	co2emission	186	2.057791065871989	1.415803207789554
Default				speed	10	12.518913854066	2.945125743901908
Default				acceleration	9	0.7512157577056333	2.906715550127471
Default				co2emission	9	4.997382577977444	8.264189843631812
Default				posx	7	2109.764285714286	27.11050402652004
Default				posy	7	26.650000000000002	0.0
Default				speed	6	13.89	0.0
Default				acceleration	5	0.0	0.0
Default				co2emission	5	2.0276130047522	0.0

Speed vs Time Graph

- Wait why is the the velocity the same and the gap is 19.05 m? The control doesn't work!

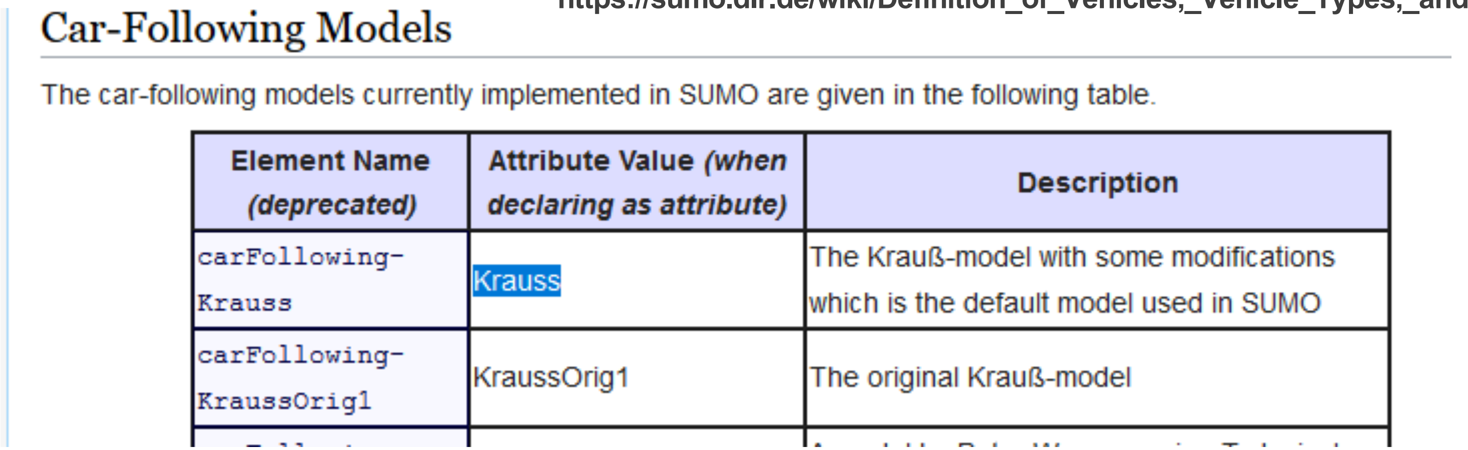


Overriding the SUMO Driver Models

- One thing to note is that SUMO does not allow direct control of vehicle acceleration and deceleration, but rather lets you configure parameters in “driver models”

- SUMO default model is “carFollowing-Krauss”

https://sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes



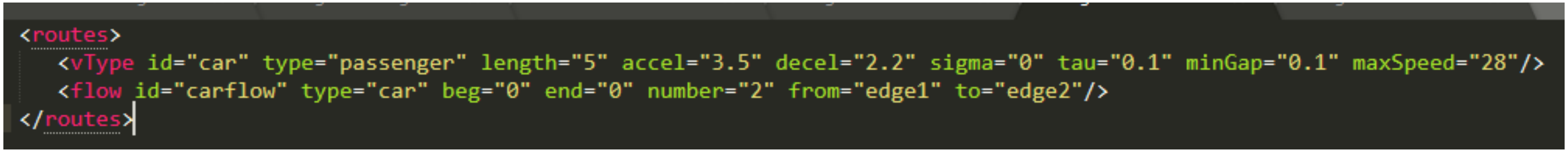
Car-Following Models

The car-following models currently implemented in SUMO are given in the following table.

Element Name *(deprecated)*	Attribute Value *(when declaring as attribute)*	Description
`carFollowing-Krauss`	Krauss	The Krauß-model with some modifications which is the default model used in SUMO
`carFollowing-KraussOrig1`	KraussOrig1	The original Krauß-model

Overriding the SUMO Driver Models

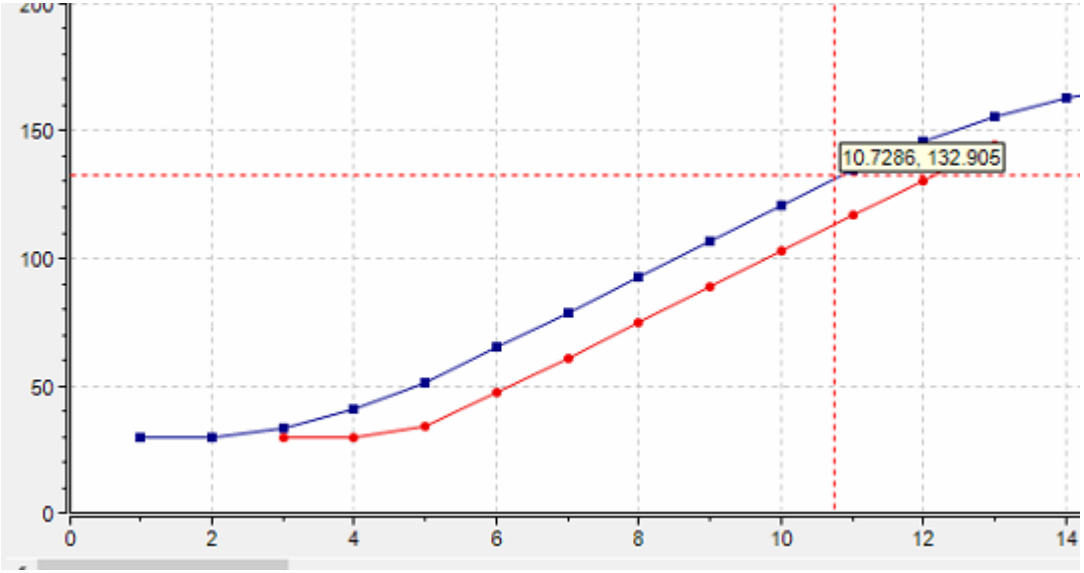
- So, the vehicles are trying to maintain the minimum time and space gap to the preceding vehicle
- The distance we'd like to achieve 6 m is going to be overridden by the driver model from SUMO
- So far, I haven't found a way to directly control acceleration, but we can try to do it by setting the values minGap (space headway) and tau (time headway) to a small value
- We can do that in the .rou.xml file
- Let's say we set the values tau and minGap to be both 0.1 (default values are 1.0 and 2.5)



```
<routes>
   <vType id="car" type="passenger" length="5" accel="3.5" decel="2.2" sigma="0" tau="0.1" minGap="0.1" maxSpeed="28"/>
   <flow id="carflow" type="car" beg="0" end="0" number="2" from="edge1" to="edge2"/>
</routes>
```

Debugging the Code

- Something has happened
- Vehicles collide and disappear in the simulation because our algorithm can’t handle the situation
- X pos vs time graph
  - Red line disappears after 13 seconds



Debugging the Code

- Something’s not right about the results, the positions are not being updated frequently as we want
- If you look into the console window (in Omnetpp IDE), something is wrong
- The vehicle distance is not as often updated (1 sec interval) as the BSM send interval
- This means we can’t rely on current handleUpdatePosition() to update the position of velocity values of the vehicles

```
t3.029858499977: Distance [13]-[19]: 10.5 acc: 4.5
t3.129870016741: Distance [13]-[19]: 10.5 acc: 4.5
t3.229870016741: Distance [13]-[19]: 10.5 acc: 4.5
t3.329870016741: Distance [13]-[19]: 10.5 acc: 4.5
t3.429870016741: Distance [13]-[19]: 10.5 acc: 4.5
t3.529870016741: Distance [13]-[19]: 10.5 acc: 4.5
t3.629870016741: Distance [13]-[19]: 10.5 acc: 4.5
t3.729870016741: Distance [13]-[19]: 10.5 acc: 4.5
t3.829870016741: Distance [13]-[19]: 10.5 acc: 4.5
t3.929870016741: Distance [13]-[19]: 10.5 acc: 4.5
t4.029870056769: Distance [13]-[19]: 16.5 acc: 10.5
t4.129870056769: Distance [13]-[19]: 16.5 acc: 10.5
```

Debugging the Code

- Try using debuggers!

Traffic Light Control

- TraCI interface to traffic light control is given in TraCICommandInterface.cc as well

```
class Trafficlight {
public:
Trafficlight(TraCICommandInterface* traci, std::string trafficLightId) : traci(traci), trafficLightId(trafficLightId)
{
connection = &traci->connection;
}

std::string getCurrentState() const;
int32_t getDefaultCurrentPhaseDuration() const;
std::list<std::string> getControlledLanes() const;
std::list<std::list<TraCITrafficLightLink> > getControlledLinks() const;
int32_t getCurrentPhaseIndex() const;
std::string getCurrentProgramID() const;
TraCITrafficLightProgram getProgramDefinition() const;
int32_t getAssumedNextSwitchTime() const;

void setProgram(std::string program);/**< set/switch to different program */
void setPhaseIndex(int32_t index); /**< set/switch to different phase within the program  */
void setState(std::string state);
void setPhaseDuration(int32_t duration); /**< set remaining duration of current phase in milliseconds */
void setProgramDefinition(TraCITrafficLightProgram::Logic program, int32_t programNr);

protected:
TraCICommandInterface* traci;
TraCIConnection* connection;
std::string trafficLightId;
};
```

Importing Realistic Maps

- If you want to work on realistic maps, you can import maps from openstreetmap
- https://sumo.dlr.de/wiki/Tutorials/Import_from_OpenStreetMap