Rogue Car

TraCI script:

The script imports TraCI functions from the tools folder of the user's SUMO base directory, set as SUMO_HOME. The script then begins SUMO as a subprocess, then connects to sumo and runs depending on the command line configuration (with or without the gui). If the user would like to run the simulation normally with the gui, they execute ./my.py. If the user would like to run the simulation without the gui, they execute ./my.py --nogui. The script then runs the TraCI control loop, interacting with the simulation, using functions from within the tools/traci function library. By default, the simulation runs for 5000 steps. If the user writes a number on the command line, for example, ./my.py 3500, then the script will run for said number of steps.

TraCI python functions (_vehicle.py):

Note: All togglers assume user is using default values. If the user is using special values, they will be overwritten by the toggler.

- rogueToggleFollowDistance
    - Toggles car follow distance between default (2.5m) and rogue setting (0.5m). Pass vehicle ID.
- rogueToggleLightRun
    - Makes car run stoplights and not respect right of way at intersections. Toggles on and off depending on current status. Pass vehicle ID.
- rogueToggleFollowSpeed
    - Toggles car speed factor between default (follow the speed limit) and rogue (double the speed limit). Pass vehicle ID.
- rogueNodeException
    - Makes previously declared rogue car respect an intersection and stop for a stoplight. This includes all connected edges that pass into an intersection. Pass vehicle ID, and x, y coordinates of the designated node.
- rogueEdgeException
    - Makes previously declared rogue car respect the speed limit along a particular edge. Pass vehicle ID, and x, y coordinates of the connecting points of the desired edge.

Rogue Vehicle Class

Characterized by doubling the speed limit, not respecting minimum follow distance, merging instantly when the rogue vehicle must decelerate for another vehicle, and running red lights if it does not pose a collision risk. The vehicle will not intentionally collide with other cars, and will follow traffic laws if it is surrounded by regular traffic. It will not run red lights when behind another car without a means of merging around it. Within the gui, the vehicle appears as a regular passenger car unless other characteristics are specified.

Testbed

After finding out a target "map", export/download it from Openstreetmap. Convert the OSM file into a SUMO network file and then add trips/routes using randomTrips.py

Netconvert --osm-files map.osm -o map.net.xml --geometry.remove --ramps.guess --junctions.join -- tls.guess-signals --tls.discard-simple --tls.join --remove-edges.isolated --tls.half-offset --remove-edges.by-vclass bicycle

The following options in order, removes complexities from the network topology without changing it, adds ramp lanes, merges junctions automatically, transcribes traffic light information, discard traffic light info, joins traffic light info, discard any edges without a connection, shift traffic light timing by half phase, and removes bicycle lanes.

Create a polygon file to import additional polygons

polyconvert --net-file map.net.xml --osm-files map.osm typemap.xml -o map.poly.xml

Use typemap.xml file from /home/veins/src/sumo-0.32/data/typemap/osmPolyconvert.typ.xml Typemap.xml

```
<polygonTypes>
   <polygonType id="waterway"          name="water"      color=".71,.82,.82" layer="-4"/>
   <polygonType id="natural"           name="natural"    color=".55,.77,.42" layer="-4"/>
   <polygonType id="natural.water"     name="water"      color=".71,.82,.82" layer="-4"/>
   <polygonType id="natural.wetland"   name="water"      color=".71,.82,.82" layer="-4"/>
   <polygonType id="natural.wood"      name="forest"     color=".55,.77,.42" layer="-4"/>
   <polygonType id="natural.land"      name="land"       color=".98,.87,.46" layer="-4"/>

   <polygonType id="landuse"           name="landuse"    color=".76,.76,.51" layer="-3"/>
```

```
  <polygonType id="landuse.forest"          name="forest"      color=".55,.77,.42" layer="-3"/>
  <polygonType id="landuse.park"            name="park"        color=".81,.96,.79" layer="-3"/>
  <polygonType id="landuse.residential"     name="residential" color=".92,.92,.89" layer="-3"/>
  <polygonType id="landuse.commercial"      name="commercial"  color=".82,.82,.80"
layer="-3"/>
  <polygonType id="landuse.industrial"      name="industrial"  color=".82,.82,.80" layer="-3"/>
  <polygonType id="landuse.military"        name="military"    color=".60,.60,.36" layer="-3"/>
  <polygonType id="landuse.farm"            name="farm"        color=".95,.95,.80" layer="-3"/>
  <polygonType id="landuse.greenfield"      name="farm"        color=".95,.95,.80" layer="-3"/>
  <polygonType id="landuse.village_green"   name="farm"        color=".95,.95,.80"
layer="-3"/>

  <polygonType id="tourism"                 name="tourism"     color=".81,.96,.79" layer="-2"/>
  <polygonType id="military"                name="military"    color=".60,.60,.36" layer="-2"/>
  <polygonType id="sport"                   name="sport"       color=".31,.90,.49" layer="-2"/>
  <polygonType id="leisure"                 name="leisure"     color=".81,.96,.79" layer="-2"/>
  <polygonType id="leisure.park"            name="tourism"     color=".81,.96,.79" layer="-2"/>
  <polygonType id="aeroway"                 name="aeroway"     color=".50,.50,.50" layer="-2"/>
  <polygonType id="aerialway"               name="aerialway"   color=".20,.20,.20" layer="-2"/>

  <polygonType id="shop"                    name="shop"        color=".93,.78,1.0" layer="-1"/>
  <polygonType id="historic"                name="historic"    color=".50,1.0,.50" layer="-1"/>
  <polygonType id="man_made"                name="building"    color="1.0,.90,.90" layer="-1"/>
  <polygonType id="building"                name="building"    color="1.0,.90,.90" layer="-1"/>
  <polygonType id="amenity"                 name="amenity"     color=".93,.78,.78" layer="-1"/>
  <polygonType id="amenity.parking"         name="parking"     color=".72,.72,.70" layer="-1"/>
  <polygonType id="power"                   name="power"       color=".10,.10,.30" layer="-1"
discard="true"/>
  <polygonType id="highway"                 name="highway"     color=".10,.10,.10" layer="-1"
discard="true"/>

  <polygonType id="boundary" name="boundary"    color="1.0,.33,.33" layer="0" fill="false"
discard="true"/>
  <polygonType id="admin_level" name="admin_level"    color="1.0,.33,.33" layer="0"
fill="false" discard="true"/>
</polygonTypes>
```

To create random 100 validated trips over routes use this command

/home/veins/src/sumo-0.32/tools/randomTrips.py -n map.net.xml map.rou.xml -e 100 -l --validate

Using the DSRC device module on SUMO v1

This document will provide steps to installing and running the DSRC Device module on any vehicle(s) on the SUMO traffic simulator. <insert some information on DSRC and its uses>.

When using the SUMO simulator, you can find the directory of all pre-installed device modules under ~/src/sumo/src/microsim/devices. This folder contains different device modules that can be implemented to any vehicle via the command line or the .rou.xml file used by the SUMO configuration file. These devices collect data from the vehicle and can either export it to a standard output or save it to an xml file. A copy of the microsim directory is available on our Veins github account with the addition of the DSRC device and modifications to the makefiles responsible for compiling the device modules.The DSRC device module is named MSDevice_DSRC.cpp and had a header file named MSDevice_DSRC.h, respectively under the devices directory. There are two ways you can use the DSRC modules on your local machine:

Method 1: The easiest way of usage

1. Go to the Veins team (UCSC) main github repo, and navigate to **cmps116-IoT/dev/aflore37/**. There you will find a copy of the microsim folder. The difference between the microsim folder found here and the one under the ~/src/sumo/src/ is the addition of the DSRC modules and the modified makefiles needed to compile it.
2. Download the folder and save it locally. After doing so, replace the microsim folder provided by the UCSC Veins team with the microsim folder found on the local veins VM.
3. Navigate to ~/src/sumo/src/microsim/devices. Now that you have replaced the microsim folder, you will have to make all the required executables so that SUMO is able to use the added DSRC device. Once you run the first make, go back one folder (cd ..) and run the next make until you reach ~/src/. There are 4 makes that you have to run before being able to use the DSRC device.
4. Now we are able to use the DSRC device but have to instantiate it on a vehicle in order to collect BSM data from that specified vehicle.
5. Assuming there is a simulation already created, go to the simulation folder where the sumocfg file is located. There is a .rou.xml file in which you must instantiate the DSRC device for any vehicle. In order to add the DSRC device module on a vehicle, you must go to any vehicle declaration of your choosing and add a new parameter key “has.dsrc.device” and set its value to “true” in order to activate the device during runtime. An example of it is shown highlighted and in bold.

   <vehicle depart="20" id="veh2" route="route2" type="Car">

<param key="has.dsrc.device" value="true"/>
</vehicle>

You can repeat this step for as many vehicles needed.

6. Now the DSRC is activated and will be able to produce BSM information for DSRC applications. You can run a sumo-gui simulation and you will be able to see captured BSM data on the terminal output as shown below