Ramtin Sumo Notes

November 2017

1 Building Sumo with /ucbtrans/sumo-project

1.1 Warnings

I have only built SUMO with the additions on Ubuntu 16.04, so be warned this may not work on all other systems. Also be warned, it may be possible to get around using these dependencies by disabling certain features of SUMO, or it may be possible to use different packages to suffice for the dependencies, but this worked for me.

1.2 Setting up the directories

First, you need to download the source files of the repository, the link is here, and the source files of SUMO 0.30.0 which can be found here.

After extracting the files from the Github repository, you can find a directory sumo-project-master/sumo-0.30.0, in which you should copy the source files for SUMO 0.30.0. You can simply drag the SUMO 0.30.0 source folder into the sumo-project-master directory, and a reasonable GUI will ask you if you'd like to merge the folders. You should click yes, then make sure NOT to replace the sumo-project-master files, as these add the IIDM functionality we desire.

1.3 Dependencies

Now we are almost ready to compile the project. There are only two dependencies (that I did not already have on a relatively clean system) required to do this. The first is Xerces-C++ (some kind of XML parser). You can install it with the package manager.

```
sudo apt-get install libxerces-c-dev
```

The second dependency is FOX (some kind of GUI toolkit), which can also be installed via the package manager.

```
sudo apt-get install libfox-1.6-dev
```

1.4 Compiling the project

Warning: I do not have a great understanding of this process, but bugs aren't too hard to track since they usually point to missing dependencies/files.

To compile the project, you need to first need to cd into the sumo-project-master/sumo-0.30.0 directory.

```
cd ~/<specific to you>/sumo-project-master/sumo-0.30.0
```

Then, you need to run the ./configure command (again not really sure what this does...)

```
./configure
```

Finally, we are ready to make the project. There are a few optional arguments to the make command, although I am not quite sure what we need yet, so I have just used the default settings.

```
make
```

If this works successfully, pat yourself on the back. Then, before you get too excited, remember to add the environment variable to your /.bashrc file, so that you can communicate to SUMO with Python. To do this, simply open your bash profile.

```
gedit ~/.bashrc
```

and paste this line into the text file

```
export SUMO_HOME=$HOME/<specific to you>/sumo-project-master/sumo-0.30.0
```

Also recommended: paste this line, which will allow you to run SUMO from the command line

```
export PATH=$PATH:$HOME/sumo/sumo-0.30.0/bin
```

2 Running Examples

There are Python scripts that run each example, each with a name that starts with "runner" and ends with ".py". These files can be executed directly if you change the file access permissions to add executability.

```
chmod +x runner.py
```

Then you can simply run the example.

```
./runner.py
```

Some of these examples will work right off the bat - like /example/redLight_simulations. Some will not.

2.0.1 Platoon Bug Fix

One of the cases I have been able to debug is those examples that use platooning functions. If you run the script without modifying the runner file, you may run into an error complaining that the traci module doesn't exist.

The bug fix is simple - in the runner.py file, near the beginning, you will find a line.

```
from platoon_functions import *
```

In the platoon_functions file, traci is imported, although Python does not yet know where the traci module can be found. In the original runner file, after the line shown above, you will see a big try/except statement resembling

```
# import python modules from $SUMO_HOME/tools directory
try:
    blah
except ImportError:
    blah
```

This block tells Python where to find your SUMO files. Move the line

```
from platoon_functions import *
```

to be below that try/except statement. This should resolve the error and allow traci to be imported in the platoon_functions module.

3 Modifications to Platoon Functions

In order to accomodate the notation of ACC and CACC enabled vehicles (REALLY PUT REFERENCE HERE), I have changed the platooning vehicle type to be CarCACC rather than CarA, which is a little misleading since this can refer to simply adaptive cruise control. This makes defining vehicle types in the route xml file more intuitive - as we can now have "CarACC," "CarCACC," and "CarM" without the old ambiguous "CarA." This can all be changed easily.

4 Questions/To Do

4.1 Questions

- Papers in Github repo?
- how to start distributing cars to measure throughput

4.2 To Do

- Python script to make cars leave/run trials of different distributions
- Python script to look at throughput measurements