Incorporating Human Drivers into SUMO

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Aarti Modani
Bachelor of Technology
GITAM University, 2015

Director: Dr. Duminda Wijesekera, Professor
Department of Computer Science

Spring Semester 2019
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my family and friends.

Acknowledgments

At the outset, I thank my Advisor, Dr.Duminda Wijesekera, for his guidance at each stage of the research and thesis. He has been very generous to spare time from his pressing schedule to guide me. I am thankful to my colleagues, Raghavendra Somepalli and Chaitanya Yuvvari for assisting me in my work. Their support has been invaluable. I wish to express my gratitude to my family and friends who have been my pillars of strength and a great source of inspiration. Their constant motivation has been a guiding force in my research. My friend, Jai Vora for supporting throughout. Finally, I wish to thank my committee for their encouragement and co-operation.

Table of Contents

List of Tables

List of Figures

Abstract

INCORPORATING HUMAN DRIVERS INTO SUMO

Aarti Modani, M.S.

George Mason University, 2019

Thesis Director: Dr. Duminda Wijesekera

Multiple large-scale traffic simulators have been developed to represent and analyze the consequences of real-life traffic scenarios. These tools can be used to understand the complexity of urban traffic situations and should be used to evaluate human behavior in driver-centered simulations. Most driving simulators fail to incorporate both aspects in one model or tool. The objective of my research is to enable realistic 3-dimensional (3D) driver centric within large scale realistic traffic simulators. In order to do so, I integrated the *Simulation of Urban Mobility (SUMO)* simulator with the *Unity* 3D game engine to provide a human driver with simulated realistic traffic on chosen roads. I used the *TraCI* protocol to communicate between the SUMO simulator and the *Unity* gaming engine. The implementation of the simulation and future lines of work is presented in the thesis.

Chapter 1: Introduction

With increasing numbers of vehicles on the road, traffic congestion has become an ongoing urbanization hurdle. There is a need for a planning system to increase the efficiency of road networks and improve driver safety, satisfaction and performance. Many traffic simulators were developed and implemented as a solution to evaluate traffic management strategies and their impact in real-life scenarios. These traffic simulators can be categorized based on the level of detail of analysis. The categories are:

1. **Macroscopic models:** Simulation of average vehicle dynamics like traffic volume, speed and density. [1]

2. **Microscopic models:** Simulation of individual vehicle dynamics in a network.

3. **Submicroscopic model:** Simulation of internal vehicular GUIs like dashboards to assess driver behavior.

4. **Mesoscopic models:** Mixture of both macroscopic and microscopic traffic simulation models.

In this thesis, I use a microscopic traffic simulator. The capability of microscopic traffic simulators are to study each vehicle individually and emulate the flow through road network makes them most powerful and versatile. Over the years, many microscopic simulation tools like TSS-AIMSUN [2], MOTUS [3], PTV-VISSIM [4] and SUMO [5] have been developed to provide analysis of higher levels of detail such as vehicle movements, flow, speed, etc. in a network. While some of them are available as open-source tools, others require a paid license to use.

Due to its open-source availability, multi-modal capabilities (vehicles, pedestrians and public transport), cross-platform support, and ability to interact with other applications using its TraCI communication protocol, SUMO traffic simulator has been selected for this research. SUMO is a microscopic traffic simulator where each vehicle is defined explicitly by name (an identifier), vehicle's route, departure and arrival time, allowed lanes, velocity. SUMO also allows a programmer to define physical properties and appearance of vehicle within the simulation's graphical user interface. The TraCI (Traffic Control Interface) allows retrieval of data about vehicle movement during a simulation run [6].

I integrated SUMO with the 3D graphic engine *Unity* to create a driver centric simulation using TraCI. The Unity 3D game engine for chosen due to its powerful runtime engine, easy of access to the community for support, vast documentation and large number of projects made with it [7].

Chapter 2: Related Work

This area of vehicular simulations have have become popular in recent times due to arising traffic problems and to re-enigeer traffic patters in otder to cope with increasing traffic. Some research has been done in the past with different traffic simulators to address the issue. Some of them are presented in this section.

A driver-centric simulation was developed using *DIVERT* to simulate traffic interfacing with a realistic driving simulator in a vehicle to study Virtual Traffic Lights and See-Through System. The driver may affect other simulated cars as the actions are updated in the VANET simulation [8].

As an integrated traffic, driving and networking simulator, *ITDNS* was proposed which used Paramics [9] to simulate traffic along with NS-2 and a driving simulator. This group of researchers were able to create a mutual data transfer between Paramics and NS-2 which allows a vehicle in Paramics to follow the behaviour of a human driver in the driving simulator. Leveraging this, they were able to study the human perception of autonomous driving, human-centric data fusion for safety application and echosignal applications [10]. Echosignal applications involves transmitting a potential speed to the vehicles approaching an intersection which can let the vehicle arrive without stopping and at green.

A HUD (Heads-Up Display) was included in [11] but it made use of offline communication between *SUMO* and *Unity 3D.* SUMO was used to create the vehicular movements that were used to develop the traffic simulation in *Unity3D.* These generated vehicular movements did not create a direct link of communication between SUMO and Unity3D and hence the human driver could not interact in response to other vehicle's movements in the simulation and the respective movements could not be reflected in SUMO either.

Chapter 3: SUMO

SUMO the chosen acronym for *Simulation of Urban Mobility* is an open-source,inter-and multi-modal, space-continuous and time-discrete traffic flow simulation platform. SUMO supports (vehicular) network import and (vehicular) demand modelling. SUMO can also be used to develop traffic light control algorithms. Given the Origin-Destination matrix, SUMO can simulate and analyze traffic in that network [12]. The sumo package includes many tools needed to generate a road network. These tools help generate road, vehicles, routes, trips, traffic lights etc [13]. All the developed SUMO applications can be seen in Table 3.1.

Table 3.1: SUMO Applications

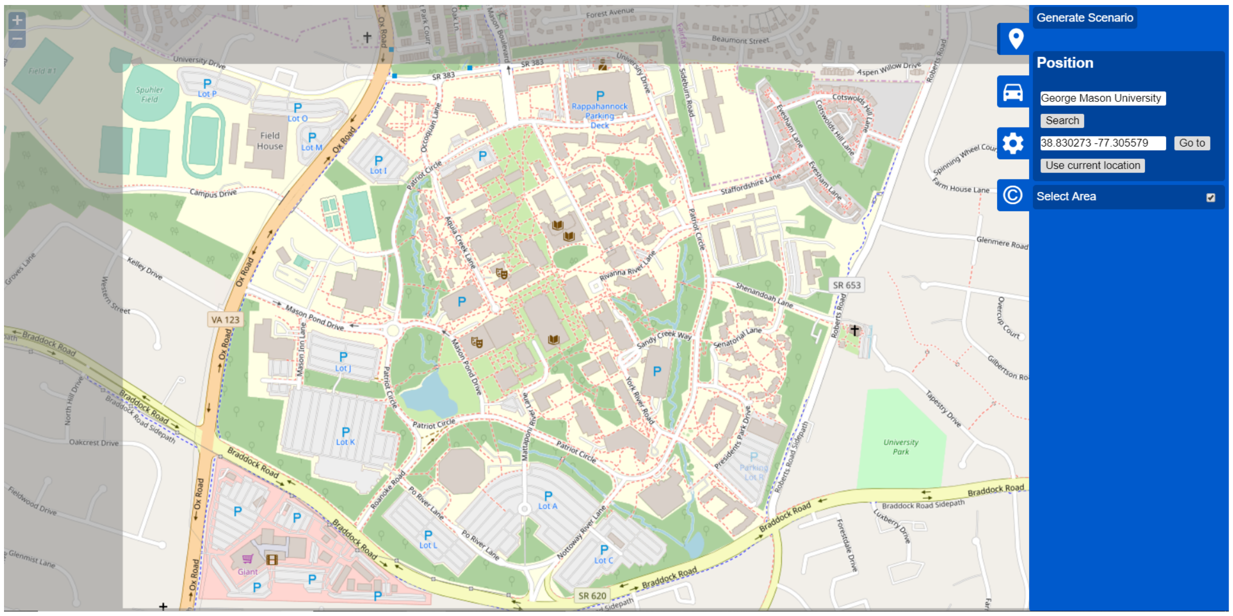
Application Name	**Description**
SUMO	Command Line Interface for simulation, execution and visualization
SUMO-GUI	Graphic User Interface for simulation
NETCONVERT	Network importer and generator; reads road networks from different formats and converts them into the SUMO-format
NETEDIT	A graphical network editor
NETGENERATE	Generates abstract networks for the SUMO simulation
DUAROUTER	Computes routes through the network, importing different types of demand description
POLYCONVERT	Imports points of interest and polygons from different formats and translates them into a description that may be visualized by SUMO-GUI

3.1 Generating a Network Scenario in SUMO

Sumo generates a road network using a network file that represents the traffic network and has the file extension `.net.xml`. The traffic network can be created using *OSMWeb-Wizard* [14], manually or from Open Street Map (OSM) data. The OSM is open-source database of all the geographical data of the world [15].

3.1.1 Generating a Network using OSMWebWizard

The OSM Web Wizard is the easiest method to create a traffic network in SUMO. With a few clicks, one can generate randomized traffic demand on the selected area from open street map. This can be then visualized in SUMO-GUI tool. By running the python script `osmWebWizard.py` , the screen shown in the Figure 3.1 opens up. One can select the area and click on generate scenario to create `.sumocfg` file which can be run in SUMO.



George Mason University

38.830273 -77.305579

Figure 3.1: Simulation through OSMWebWizard.

3.1.2 Manually Creating a Road Network

In SUMO, a road network consists of nodes connected by edges. The roads or lanes are represented by edges and junctions by nodes. Each node has an (x,y) co-ordinates. In order to build our network, the following steps have to be followed.

1. The first step is to create a node file and the extension of the node file is `.nod.xml`.

2. The second step is to create an edge file and the extension of the edge file is `.edg.xml`.

3. The third step is to create an edge type file with the extension `.type.xml`. The type file includes the properties of lanes such as speed limit, number of lanes, type of vehicles allowed, etc.

4. The fourth step is to generate a network from the node, edge and type file. This network file has an extension `.net.xml` can be generated using the following command

   ```
   netconvert --node-files node.nod.xml --edge-files edge.edg.xml -t type.type.xml -o network.net.xml
   ```

5. After generating the network, one can define the routes, we can add the `.rou.xml` file.

6. Lastly, after getting the network and the route file, one can build the sumo configuration file with inputs from both files.

This method is feasible for smaller networks. As the roads and vehicles increase in the network, creating all the components can be tedious and this method is not advisable.

3.1.3 Generating a Network using OSM

To simulate larger traffic scenarios, the network file can be created by importing it from external applications. For my research, I created the traffic network from OSM to get a more realistic scenarios. The George Mason University, Fairfax, VA area was chosen for this simulation. The process is described below.

1. First, the map of George Mason University is extracted from the OSM webpage [16]. Select the area manually and click export as shown in Figure 3.2 and 3.3. The map is stored as an .osm file.

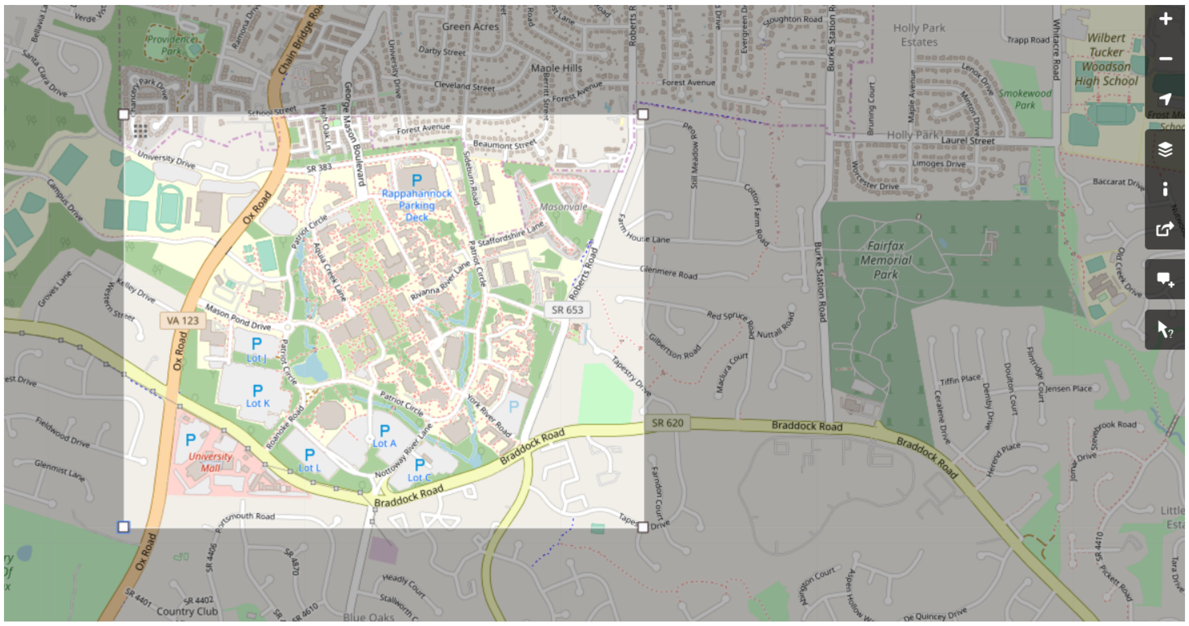
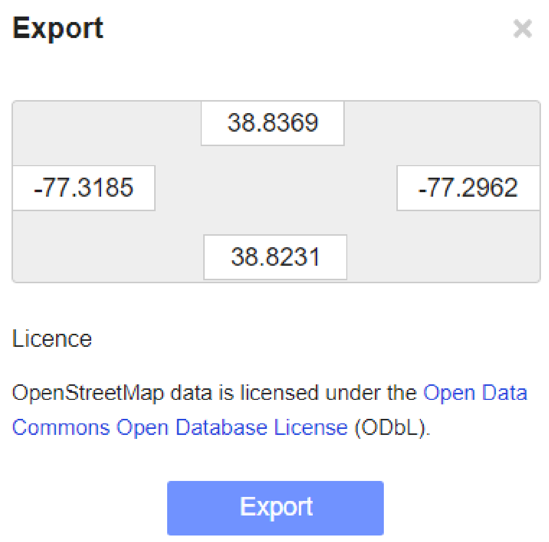


Figure 3.2: Selection of desired area from OSM.



Export

38.8369

-77.3185

-77.2962

38.8231

Licence

OpenStreetMap data is licensed under the Open Data Commons Open Database License (ODbL).

Export

Figure 3.3: Latitude and Longitude boundary values to be exported.

2. The NETCONVERT tool from the SUMO package is used to generate the corresponding SUMO file. The sumo network file has a `.net.xml` extension. The NETCONVERT tool is not perfect and while converting the `.osm` file, there might be some warnings which can be edited using NETEDIT tool.

   ```
   netconvert --osm-files test.osm -o test.net.xml
   ```

3. Once the network file is generated, for the initial phase, traffic through vehicles can be randomly generated using the `randomTrips.py` script and the *DUAROUTER* tool provided by the SUMO package. The route files have an `.rou.xml` extension.

   ```
   py randomTrips.py -n gmu.net.xml -r gmu.rou.xml -e 50 -l
   ```

4. The routes file, network file, trips file are all combined to form a sumo configuration `.sumocfg` file which can be simulated in the SUMO-GUI shown in Figures 3.4 and 3.5.

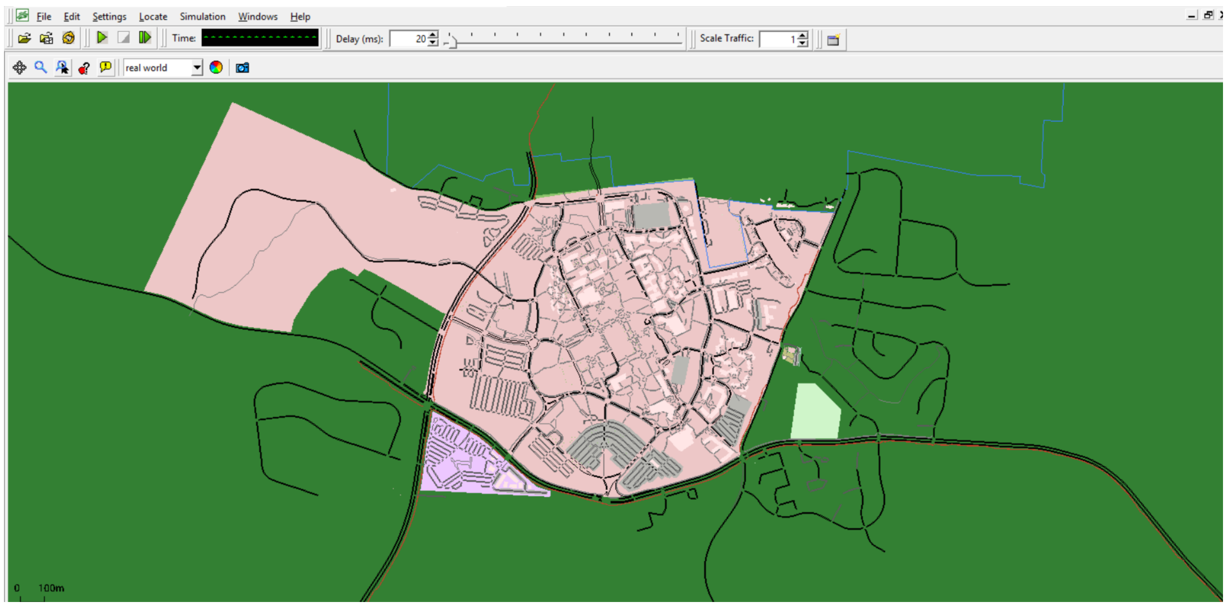
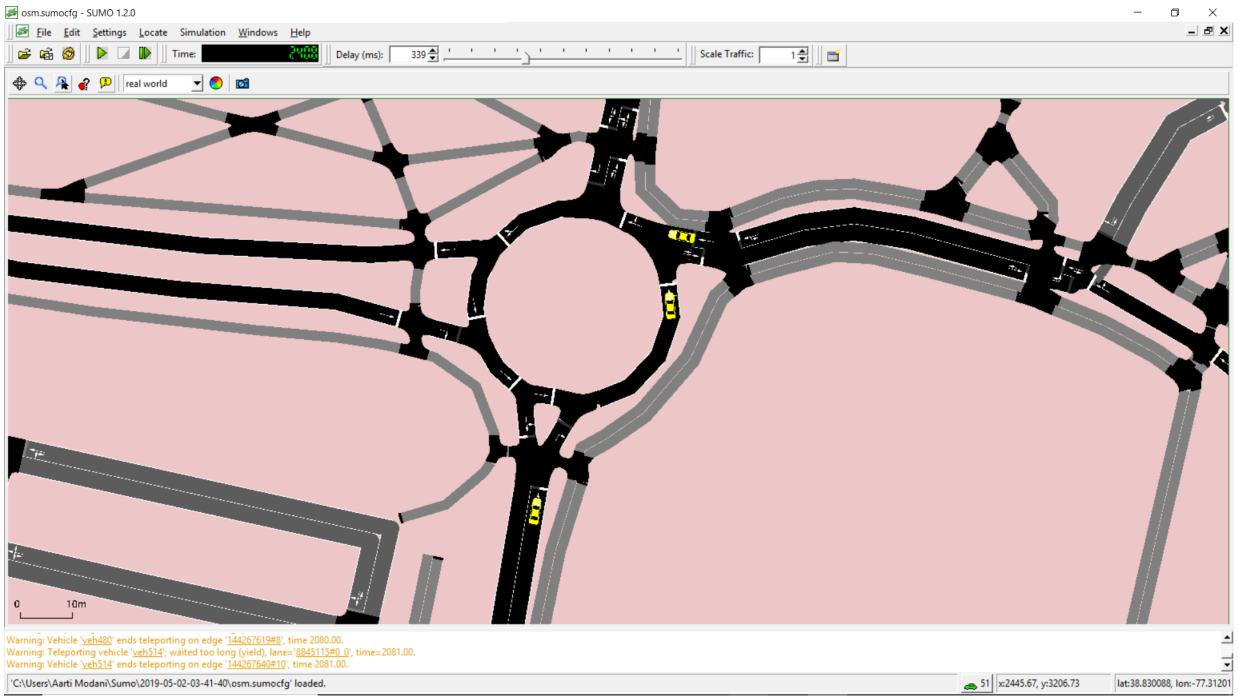


Figure 3.4: SUMO configuration file.



```
Warning: Vehicle 'veh480' ends teleporting on edge '144267619#8', time 2080.00.
Warning: Teleporting vehicle 'veh514'; waited too long (yield), lane='8845115#0_0', time=2081.00.
Warning: Vehicle 'veh514' ends teleporting on edge '144267640#10', time 2081.00.
'C:\Users\Aarti Modani\Sumo\2019-05-02-03-41-40\osm.sumocfg' loaded.   51   x:2445.67, y:3206.73   lat:38.830088, lon:-77.31201
```

Figure 3.5: SUMO simulation.

Chapter 4: Communication Between SUMO and Unity using TraCI

The SUMO simulator can interface with multiple clients using the *TraCI* protocol. TraCI enables retrieval of data from SUMO and pass it to the client interfacing with SUMO. TraCI allows the retrieval of vehicular data such as position of vehicles, routes, traffic lights used in the simulation. The protocol follows a Client-Server architecture. A TCP/IP connection is established between the SUMO and TraCI client and it follows a request-response pattern. The TraCI client requests a connection to SUMO which acts as a TraCI server, listening on a specific port 4.1. Once, it receives the request from the client application, SUMO accepts the connection and the control of simulation is transferred to the client [17]. The client can control the simulation and can make modifications to vehicles or retrieve data for the elements in simulation.

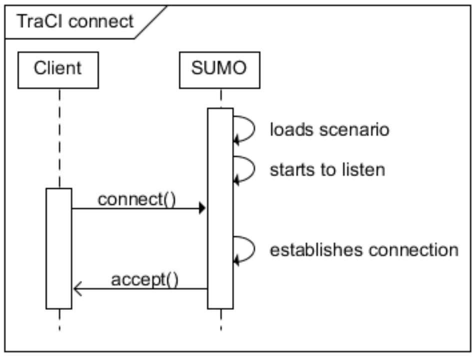


Figure 4.1: TraCI connection establishment between SUMO and client.

The client ends the simulation by closing the TCP connection and no longer have control over the simulation as shown in Figure 4.2.

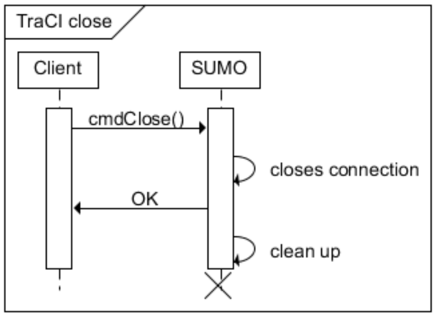


Figure 4.2: TraCI connection close between SUMO and client.

The TraCI protocol needs a library to be implemented in *Unity 3D* in order to communicate with *SUMO*. Since there is no existing C# library, TraCI as a Service (TraaS) library is chosen. It is an open source java library which can be used as a web service to work with multiple clients from any programming language. To import it to a C# library, `IKVM.NET` [18] tool is used.

4.1 IKVM.NET

The IKVM.NET tools is a Java implementation for Microsoft .Net Framework. It generates .NET libraries (`.dll`) from `.class` or `.jar` files. The ikvmc tool is used to achieve this which can be used with Unity3D without having to develop C# library from scratch. The following steps are adopted to convert the TraaS library to C# that can be used in Unity3D

1. Firstly, we need to download the TraaS library (`TraaS.jar`) from its webpage [19].

2. Second, some classes were not useful and were thus removed to make the `.jar` file compatible with IKVM.

3. Third, a new `.jar` file (`myDLL.jar`) is built after the removal of classes which can be converted using the IKVM tool.

4. The `myDLL.jar` is converted into `myDLL.dll` using the IKVMC of the IKVM tool.

5. Lastly, all the `.dll` files provided by IKVM along with created `myDLL.dll` are imported into Assets folder in Unity3D.

Chapter 5: Implementation

A microscopic traffic simulator was used with a real-time game engine to carry out the simulation. A simulation may offer a variety of features with respect to its implementation. We would be able to make use of different models and specify different parameters for the various elements that are part of the simulation and analyze the results in each of those cases. As a part of this, we can test out our assumption in a simulation before we perform a real life experiment to avert any avoidable dangers which may be discovered during a simulation. And in this process, we are also reducing expenses since we can easily change our initial prototype and also because we are using a virtual reproduction for our experiment. But using a simulation might sometime require a huge amount of computation power and if we create the simulation on a huge scale, we might encounter delays.

There are three critical aspects to generate a 3D scenario in Unity from the data retrieved from SUMO through `SumoBehavior.cs` script.

5.1 Initialization

The SUMO and Unity 3D communicate through TraCI protocol in a Server-Client structure. To establish the connection, we use the SumoTraciConnection object which enables SUMO to listen on a specific port and connects to it as TCP client. The runServer() is responsible for performing the above task by creating a runtime process. Once this connection is established, data can be retrieved and exchanged between the two applications. In this process, we also specify paths to the SUMO-GUI executable and the configuration file `.sumocfg` for which the simulation is performed. The GameObjects for vehicles and lanes were initialized during the simulation. The next step is to retrieve information about vehicles and lanes from SUMO and print in Unity. This is described in the following sections.

5.2 Read and Print Vehicles

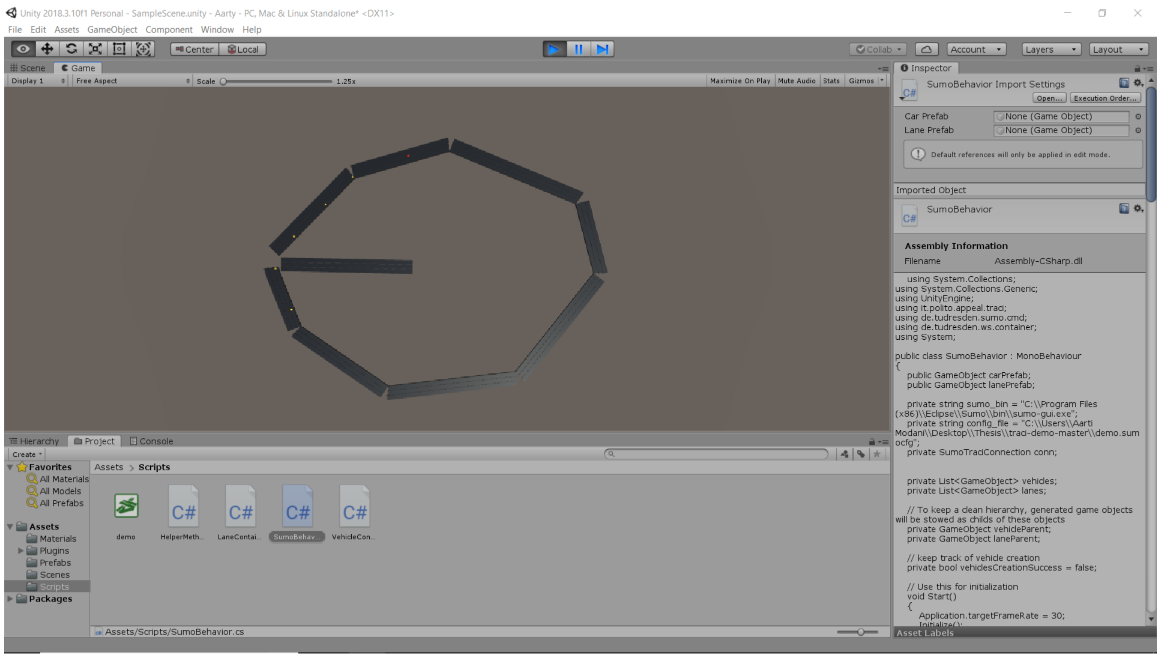
The information about each vehicle such as its position and speed in the traffic network in SUMO can be retrieved by using their respective IDs. We can get the vehicles from Vehicle.getIDList() and store it as a SumoStringList. The list is parsed to get data of each vehicle is retrieved using the do_job_get() method from the SumoTraciConnection object using the parameters Vehicle.getPosition(id) and Vehicle.getSpeed(id) for vehicle's position and speed respectively [20].

The position of vehicles in SUMO is represented in the (x-y) co-ordinate system and needs to be mapped to (x-z) co-ordinate system used by Unity 3D. This can be achieved by using the Vector3 structure [21]. We need to instantiate the vehicles through its GameObject first and then assign position and speed retrieved from SUMO through TraCI to it. The position of the vehicles need to be updated at each simulation step. This can be done using the UpdateVehicles() method where the position is requested from SUMO and updated in Unity [22].

5.3 Read and Print Lanes

We need to get information about each lane in the road network along with lane vertices,width and allowed vehicles information.This can be achieved with the readLanes() method. The list of lanes is created by getting data about each lane through Lane.getIDList() method. We can get the vertices that determine the shape of the lane from the Lane.getShape(id) parameter of the do_job_get() method. We need to determine what kind of data does the getShape() function returns. We see that this function returns a list of 2D vertices describing the lane geometry.

We then parse this data from SumoStringList to list of Vector2's or Vector3's with y component as 0. This is done using a Helper method. The data is in the form of "x,y x,y"


Unity 2018.3.10f1 Personal - SampleScene.unity - Aarty - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Collab Account Layers Layout
Scene Game
Display 1 Free Aspect Scale 1.25x Maximize On Play Mute Audio Stats Gizmos
Hierarchy Project Console
Favorites All Materials All Models All Prefabs
Assets Materials Plugins Prefabs Scenes Scripts Packages
Assets > Scripts
demo HelperMeth... LaneContai... SumoBehav... VehicleCon...
Assets/Scripts/SumoBehavior.cs
Inspector
SumoBehavior Import Settings
Open... Execution Order...
Imported Object
SumoBehavior
Asset Labels

Car Prefab: None (Game Object)
Lane Prefab: None (Game Object)

Default references will only be applied in edit mode.

Assembly Information
Filename: Assembly-CSharp.dll

```
    using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using it.polito.appeal.traci;
using de.tudresden.sumo.cmd;
using de.tudresden.ws.container;
using System;

public class SumoBehavior : MonoBehaviour
{
    public GameObject carPrefab;
    public GameObject lanePrefab;

    private string sumo_bin = "C:\\Program Files
(x86)\\Eclipse\\Sumo\\bin\\sumo-gui.exe";
    private string config_file = "C:\\Users\\Aarti
Modani\\Desktop\\Thesis\\traci-demo-master\\demo.sum
ocfg";
    private SumoTraciConnection conn;


    private List<GameObject> vehicles;
    private List<GameObject> lanes;

    // To keep a clean hierarchy, generated game objects
will be stowed as childs of these objects
    private GameObject vehicleParent;
    private GameObject laneParent;

    // keep track of vehicle creation
    private bool vehiclesCreationSuccess = false;

    // Use this for initialization
    void Start()
    {
        Application.targetFrameRate = 30;
```

Figure 5.1: Simulation in Unity.

format and it can be divided by a space and stored as Vector3 by adding a y component. Once we have the list of vertices, we can instantiate GameObjects at those locations to see the layout of lanes and modify them to get the shape of lanes.

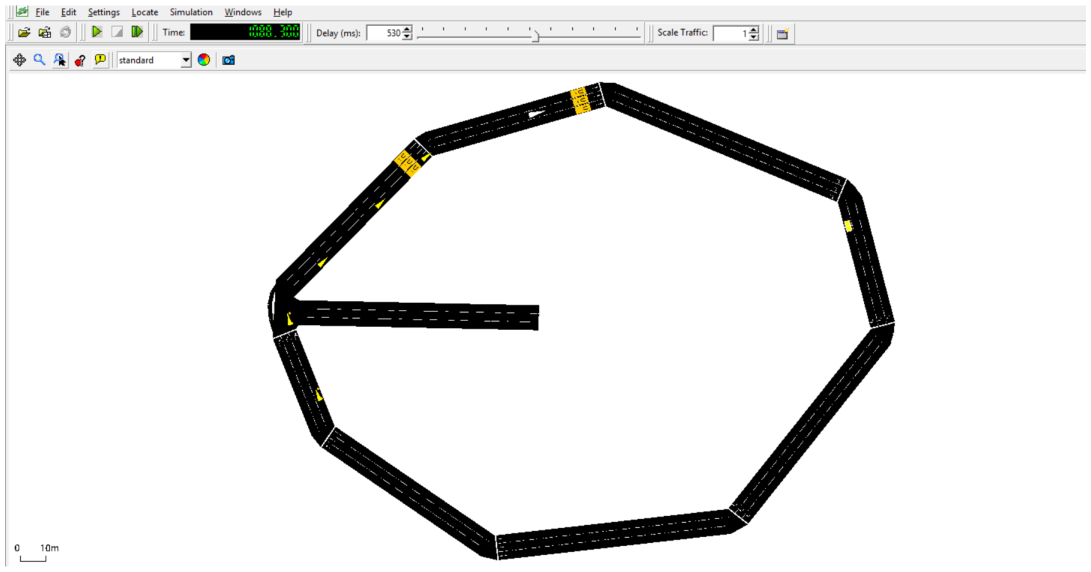


Figure 5.2: Simulation in SUMO started by Unity [?]

.

5.4 Driver Centric Simulation

The traffic simulation in SUMO can be started by Unity 3D using the above implementation. The Prefabs for vehicles, lanes are included and another Prefab for creating human controlled car is loaded into unity. When the play button is clicked, the Unity 3D starts SUMO and we can control the simulation through the human driven car. This gives us a better insight of the traffic.

During the simulation, some observations were made. The maps loaded through OSM are not entirely correct. There are slight errors in the junctions, traffic lights and lanes. This can be edited using NETEDIT and using parameters like `--junction.join` in NETCONVERT command. The maps from OSM also take a little longer to load due to large number of lanes and elements. We also observed that when the information is retrieved from TraCI during simulation, a delay is experienced when many vehicles are running through the road network.

Chapter 6: Conclusion and Future Work

In this research, a driver centric simulation was developed which represented realistic traffic on a real-world road network. This was achieved by integrating a microscopic simulator, SUMO with a powerful game engine Unity3D through TraCI protocol. The TraCI protocol, available as a Java library was converted into a .NET C# using IKVM tool in order to work with Unity3D. There were microscopic simulations which managed to generate a 3D scenario in Unity through offline traffic data from SUMO simulator. This research extends the previous works by simulating live traffic movements of vehicles. The current works not only depicts data generated from SUMO into Unity but also relays back the information about the human driven vehicle generated in Unity such as location and speed back to SUMO.

The current work was developed to generate a realistic 3D driver centric simulation for a single human driven vehicle in Unity. The goal in future could be to extend this simulation to multiple human driven vehicles. The multiple human driven vehicles generated in Unity can be controlled by individual people during the simulation. This will give a more realistic scenario of real-life traffic with multiple human driven vehicles involved.

In addition to this, the pedestrian traffic can be generated and simulated in a 3D scenario using SUMO and Unity. The incorporation of multiple vehicles along with pedestrians will generate a simulation experience close to the real experience.

References

[1] W. Burghout, "Mesoscopic simulation models for short-term prediction," *PREDIKT project report CTR2005*, vol. 3, 2005.

[2] "Top features for aimsun 8 aimsun." [Online]. Available: https://www.aimsun.com/aimsun-next/top-features-for-aimsun-8/.

[3] W. Schakel, B. van Arem, H. van Lint, and G. Tamminga, "A modular approach for exchangeable driving task models in a microscopic simulation framework," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, 2013, pp. 565–571.

[4] J. Barceló *et al.*, *Fundamentals of traffic simulation.* Springer, 2010, vol. 145.

[5] B. Pattberg, "Sumo simulation of urban mobility." [Online]. Available: https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/

[6] "Traci." [Online]. Available: https://sumo.dlr.de/wiki/TraCI

[7] C. Guindon, "Simulation of urban mobility - sumo." [Online]. Available: https://www.eclipse.org/community/eclipse_newsletter/2017/august/article2.php

[8] P. Gomes, C. Olaverri-Monreal, M. Ferreira, and L. Damas, "Driver-centric vanet simulation," in *International Workshop on Communication Technologies for Vehicles.* Springer, 2011, pp. 143–154.

[9] "Paramics microsimulation." [Online]. Available: https://www.paramics.co.uk/en/

[10] Y. Hou, Y. Zhao, A. Wagh, L. Zhang, C. Qiao, K. F. Hulme, C. Wu, A. W. Sadek, and X. Liu, "Simulation-based testing and evaluation tools for transportation cyber–physical systems," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 3, pp. 1098–1108, 2016.

[11] V. Charissis, S. Papanastasiou, W. Chan, and E. Peytchev, "Evolution of a full-windshield hud designed for current vanet communication standards," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013).* IEEE, 2013, pp. 1637–1643.

[12] D. Krajzewicz, L. Bieker, and J. Erdmann, "Sumo-simulation of urban mobility-an overview." [Online]. Available: https://www.academia.edu/17294749/Sumo-simulation_of_urban_mobility-an_overview

[13] C. Biurrun Quel, “Development of a microscopic driver-centric simulator with unity3d and sumo,” 2016.

[14] “Tutorials/osmwebwizard.” [Online]. Available: http://sumo.sourceforge.net/userdoc/Tutorials/OSMWebWizard.html

[15] “Openstreetmap,” Apr 2019. [Online]. Available: https://en.wikipedia.org/wiki/OpenStreetMap

[16] “Openstreetmap.” [Online]. Available: https://www.openstreetmap.org/

[17] “Traci/traas.” [Online]. Available: https://sumo.dlr.de/wiki/TraCI/TraaS

[18] “Ikvm.net home page.” [Online]. Available: http://www.ikvm.net/

[19] “Traas.” [Online]. Available: https://sourceforge.net/projects/traas/

[20] “Unity user manual (2019.1).” [Online]. Available: https://docs.unity3d.com/Manual/index.html

[21] U. Technologies, “Vector3.” [Online]. Available: https://docs.unity3d.com/ScriptReference/Vector3.html

[22] “Welcome to the unity scripting reference!” [Online]. Available: https://docs.unity3d.com/2017.4/Documentation/ScriptReference/index.html

[23] Avcourt, “avcourt/traci-demo,” Feb 2019. [Online]. Available: https://github.com/avcourt/traci-demo

Curriculum Vitae

Aarti Modani received her Bachelor of Technology degree from GITAM University, India in 2015. She worked for 2 years as a Systems Engineer for Cisco at the Tata Consultancy Services.