



**Trinity College Dublin**

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

School of Computer Science and Statistics

# Using SUMO and Unity 3D to analyse the effect of roadside obstacles on the safety and performance of bicycles in Dublin

Eoin Bergin



April 30, 2020

A Final Year Project submitted in partial fulfilment  
of the requirements for the degree of  
MAI (Computer Engineering)

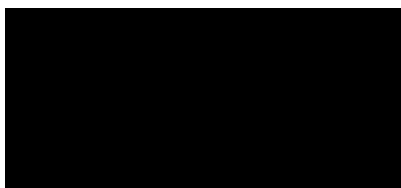
# Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at <http://www.tcd.ie/calendar>.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at <http://tcd-ie.libguides.com/plagiarism/ready-steady-write>.

Signed:



Date:

30/04/20

# Abstract

The aim of this project is to analyse the impact of obstacles such as roadside bus stops and parked cars on the safety and efficiency of bicycle operations in the Dublin Docklands area. Cycling is becoming more popular as an alternative method of transport in Dublin, however, it is becoming increasingly dangerous. According to an Irish Times article, as recently as 2013 Dublin ranked in a list of the top 20 bike-friendly cities yet it hasn't been seen on the list since this time (1). Several campaigns have and continue to protest roadside obstacles such as those mentioned above (2). This project will investigate what impact, if any, these obstacles have on the safety of cyclists.

Whilst there are several examples of work being done analysing the impact of items on road-flow etc this project addresses the specific area of cycling that's been neglected and also the problems unique to the Dublin city centre. Creating a network using the SUMO software suite and connecting it to the Smart Dublin Docklands model has never been done before and is a unique approach to planning for the future and taking into account the needs of cyclists.

# Acknowledgements

Firstly I would like to thank my supervisor Prof. Siobhán Clarke. Your constant support and patience was invaluable throughout the project.

Thank you to my parents and sister for your constant support throughout the years.

Thanks you to my classmates who are always supportive especially David, Rocky and Tejas who has offered me sound advice throughout our years together in college.

This wouldn't have been possible without all of you.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Research Questions</b>	<b>4</b>
<b>3</b>	<b>Report structure</b>	<b>5</b>
<b>4</b>	<b>Literature Review</b>	<b>6</b>
4.1	Implementation of the Project . . . . .	6
4.1.1	Sumo . . . . .	6
4.1.2	Sumo to Unity3D connection . . . . .	7
4.1.3	Bicycle movement model . . . . .	8
4.2	Background theory . . . . .	9
4.2.1	Traffic patterns in Dublin City . . . . .	9
4.2.2	Road collisions . . . . .	11
<b>5</b>	<b>Traffic network generation</b>	<b>13</b>
5.1	Create a .net.xml file . . . . .	13
5.2	Generate Traffic . . . . .	14
5.2.1	Add elements in Netedit . . . . .	15
5.2.2	Using RandomTrips.py . . . . .	15
5.2.3	Using a text editor of choice to manually edit the config files . . . . .	16
<b>6</b>	<b>Creating an interface between SUMO and Unity3D</b>	<b>18</b>
6.1	TraCl . . . . .	18
6.2	TCP/IP . . . . .	19
6.2.1	Server . . . . .	19
6.2.2	Client . . . . .	20
6.2.3	Server and Client . . . . .	21
<b>7</b>	<b>Displaying vehicles in Unity3D</b>	<b>22</b>
7.1	The basics . . . . .	22
7.1.1	Manually create each component and add to the GameObject . . . . .	22

7.1.2	Prefabs . . . . .	23
7.2	Vehicle Class . . . . .	23
7.3	Instantiating a vehicle . . . . .	24
7.4	Mapping coordinates . . . . .	24
7.5	Determining an objects orientation . . . . .	25
<b>8</b>	<b>Implementation of collision detection</b>	<b>28</b>
8.1	Attaching a collision detector . . . . .	28
8.2	Force generated . . . . .	28
<b>9</b>	<b>Project workflow</b>	<b>30</b>
<b>10</b>	<b>Results</b>	<b>32</b>
10.1	Small tests . . . . .	32
10.1.1	Scenario 1A . . . . .	33
10.1.2	Scenario 1B . . . . .	34
10.1.3	Scenario 1C . . . . .	35
10.2	Complex tests . . . . .	37
10.2.1	Scenario 2A . . . . .	37
10.2.2	Scenario 2B . . . . .	39
10.2.3	Scenario 2C . . . . .	41
10.3	Density tests . . . . .	43
10.3.1	Scenario 3A . . . . .	43
10.3.2	Scenario 3B . . . . .	45
<b>11</b>	<b>Evaluation</b>	<b>47</b>
11.1	Bus stop v no bus stop . . . . .	47
11.2	Density . . . . .	49
<b>12</b>	<b>Conclusion</b>	<b>50</b>
12.1	Does roadside bus stops have an impact on the safety of cyclists and if so what is this impact? . . . . .	50
12.2	Is using a SUMO and Unity3D combination an effective tool? . . . . .	50
12.3	Limitations . . . . .	51
<b>13</b>	<b>Future Work</b>	<b>52</b>
13.1	Enlarging the area Dublin-wide . . . . .	52
13.2	Developing a more sophisticated cyclist model . . . . .	52
13.3	City planning tool . . . . .	53
<b>A1</b>	<b>Appendix</b>	<b>57</b>

A1.1 Code . . . . .	57
---------------------	----

# List of Figures

1.1	Roadside bus stop with no indent(3)	2
1.2	Smart Dublin Docklands model(4)	3
4.1	Cost function	8
4.2	(5)	9
5.1	Final road layout	13
5.2	OSM map export	14
5.3	Final road layout	14
5.4	Demands section	15
5.5	Network section	15
5.6		15
5.7		17
6.1	(6)	19
6.2	Connection	21
7.1		23
7.2		23
7.3		23
7.4	Vehicle information received	24
7.5		24
7.6	Sumo coordinate system	25
7.7	Unity coordinate system	25
7.8	Conversion formula	25
7.9	Time step 1	26
7.10	Time step 2	26
7.11		27
8.1	Example of collision results from a simulation	28
8.2		29



9.1	Workflow of entire project when run . . . . .	30
9.2	Vehicle information sent over TCP connection Left to right: vehicle ID, x-coordinate, y-coordinate, vehicle type . . . . .	30
9.3	Vehicle information stored in Unity3D Top to bottom: vehicle ID, x-coordinate, y-coordinate, z-coordinate, orienta- tion angle, game object displayed . . . . .	31
9.4	End result with vehicle being displayed . . . . .	31
9.5	Collision information written to csv file . . . . .	31
10.1	. . . . .	33
10.2	. . . . .	33
10.3	. . . . .	34
10.4	. . . . .	34
10.5	. . . . .	35
10.6	. . . . .	36
10.7	. . . . .	37
10.8	. . . . .	38
10.9	. . . . .	39
10.10	. . . . .	40
10.11	. . . . .	41
10.12	. . . . .	42
10.13	. . . . .	43
10.14	. . . . .	44
10.15	. . . . .	45
10.16	. . . . .	46
11.1	. . . . .	48
11.2	. . . . .	49
13.1	Indented bus stop . . . . .	52

# Nomenclature

Force	$F$
Mass	$m$
Acceleration	$a$
Impulse (change in momentum)	$\Delta p$
Time	$t$
Velocity	$V$

# 1 Introduction

With the advent of Smart City planning using technology as a tool to plan and evaluate future city development decisions is becoming increasingly useful. Senior editor, BCC Research Michael Sullivan stated:

The smart cities market opportunity is driven by the convergence of information and communications technology, in particular the development of advanced connectivity and analytic software and hardware. A branch of the Internet of Things (IoT), smart cities leverage the array of connected sensors and analytics platforms to drive stronger coordination within departments and across city agencies and community groups.

By applying this combination of statistics, analytics and technology there is a multitude of different possible uses from city planning to enhancing public-government engagement. Using smart city tools, local authorities can more efficiently examine current situations and use this data to plan a more efficient future for their area. Smart Dublin is one current initiative by the four local Dublin councils aimed at implementing these new techniques in an Irish setting. They provide a Docklands area model that is used in the development of this project. They state:

We aim to position Dublin as a world leader in the development of new urban solutions, using open data, and with the city region as a test bed.

Smart Dublin is delivering a programme that encourages the creation of solutions to address city needs. It has an emphasis on using the opportunities offered by emerging technology and public data. Smart Dublin has identified mobility, environment, energy, waste and emergency management as priority challenges.(7)

This project will look to analyse the one of these situations. The impact of obstacles such as roadside bus stops and parked cars on the safety and efficiency of bicycle operations in the Dublin Docklands area will be analysed and discussed. Cycling is becoming more popular as an alternative method of transport in Dublin, however, it is becoming increasingly dangerous. According to an Irish Times article, as recently as 2013 Dublin ranked in a list of the top 20

bike-friendly cities yet it hasn't been seen on the list since this time. Several campaigns have and continue to protest roadside obstacles such as those mentioned above. This project will investigate what impact, if any, these obstacles have on the safety of cyclists.



Figure 1.1: Roadside bus stop with no indent(3)

The traffic network is created in SUMO with information regarding vehicle positioning and information being obtained from this simulation via Traci. Vehicles are displayed in the Unity simulation using a C# assembly script. Unfortunately there is no significant C# Traci module so a python script (which supports Traci) is used in order to pull this information. this python script pulls this information and then dumps it as a string via TCP to the C# assembly script. This parses the string and creates and updates game objects based on this info. Collision detectors are attached to each game object to monitor the states of the object.

Whilst there are several examples of work being done analysing the impact of items on road-flow etc. My project addresses the specific area of cycling that's been neglected and also the problems unique to the Dublin city centre. Creating a network using the SUMO software suite and connecting it to the DCC model has never been done before and is a unique approach to planning for the future and taking into account cyclist needs.

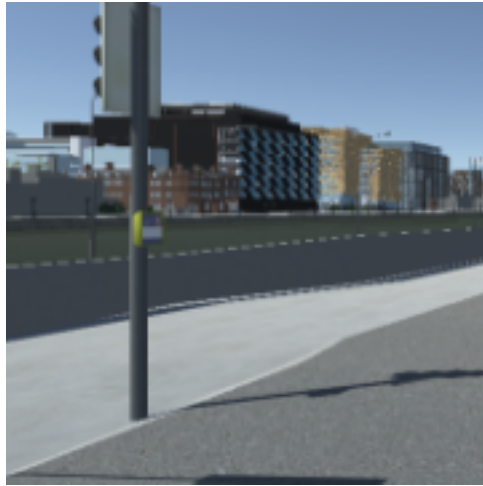


Figure 1.2: Smart Dublin Docklands model(4)

To evaluate the research questions several simulations are run and the statistics gained from each scenario are compared. By comparing collision rates we can see if there is any impact by changing the environment state.

## 2 Research Questions

Following on from the previous chapter where the concerns on cyclists were outlined and the aim of the project was discussed the key questions to be answered in the thesis are as follows:

Primary Question:

- Does roadside bus stops have an impact on the safety of cyclists and if so what is this impact?

Secondary Question:

- Is using a SUMO and Unity3D combination an effective tool?

### 3 Report structure

Chapter 2 will outline the key research performed in anticipation of the project. It will examine areas such as traffic patterns, cyclist behaviour and the physics of road collisions. It will also examine the use cases of tools used in the project and the state of the art in examining questions such as those posed in Section 1.2.

Chapters 5,6,7 and 8 will provide a deeper look at the tools used to examine this particular traffic situation and convey the positives and negatives of certain choices in this area. Chapter 8 then looks at how each of these tools are combined together to produce a linked program whereby we can gain results.

In Chapter 10, the scenarios designed and run in the program are shown and the results produced outlined. These results are then amalgamated and analysed as a sum in Chapter 11 with Chapter 12 examining how these evaluations answer the original research questions.

The potential of the program as a whole is discussed in Chapter 13 along with some recommended future upgrades.

## 4 Literature Review

Previous and related work reviewed can be divided into three sections, work relevant to the implementation of the project, current developments using similar set-ups and work relevant to the overall theme of the project. e.g. an article on connecting the Sumo traffic simulator to Unity's 3D game engine, an article on identifying types of collisions involving road users. These combine to give an understanding of the complexities of the question and of how to go about implementing a way to examine the question. This section will contain a discussion of the papers of major importance the project at large.

### 4.1 Implementation of the Project

#### 4.1.1 Sumo

In creating a realistic traffic network it was important to ensure the best tool was used for the job. Without a realistic traffic network with realistic road user-to-road user interaction the results obtained would be meaningless. "A Review of Traffic Simulation Software" (8) is a paper outlining the use-fulness of several different tools.

In this paper SUMO, Quadstone Paramics Modeller, Treiber's Microsimulation of Road Traffic, Aimsun, Trafficware SimTraffic and CORSIM TRAFVU are compared across a range of topics. The most important topic for this thesis is the realism of traffic interactions and therefore the "Creating traffic networks and associated vehicle patterns" is what is reviewed in this paper. In this section the author writes "SUMO software package is very different in this sense from the other applications because it is the only application where each vehicle (agent) knows its own destination and list of edges it needs to pass until this destination is reached." (8). This is crucial as although it creates extra overhead it ensures the impartiality of traffic interactions as they are guaranteed to be dynamic with no prevention measures implemented due to foreknowledge.



### 4.1.2 Sumo to Unity3D connection

Porting a generated network created in SUMO into the Unity simulation is another major step for the project. The paper "Connection of the SUMO Microscopic Traffic Simulator and the Unity 3D Game Engine to Evaluate V2X Communication-Based Systems" (6) talks about this process and explains a method of doing this using TraCI. It is quite an extensive paper evaluating Vehicle-to-Everything (V2X) communication technologies, however, the main focus of this review will be on its description of connecting SUMO to Unity. That being said the rest of the paper is quite interesting with regards to the use of technologies in assisting the driver of a vehicle on the road and is a recommended read.

The authors created a simulated scenario of a neighbourhood in Vienna using geographic information and procedural modelling tool CityEngine as well as creating a road network scenario in SUMO. The process described in the report for displaying this scenario in Unity is quoted below:

1. Generation of the road set, which built the network scenario through the process of retrieving all of the vertices, allowing for the definition of lane shape, and created a GameObject per segment. Every segment was noted in meters as is required by the SUMO X-Y coordinate system and then mapped to the X-Z coordinate system in Unity 3D (6)
2. In accordance with its position and angle, each simulated vehicle generated by SUMO was placed in Unity as a GameObject (6)

This connection between the two platforms is created via the TraCI protocol.

Figure 2 in Section 3 of the paper shows the communication to initialise the scenario. Section 4 of the paper describes the algorithms created in order to maintain the scenario in Unity.

These algorithms and protocols are an ideal basis to start from in connecting a scenario in SUMO with Unity and therefore is a good starting point for connecting a bicycle centered network in SUMO with the Docklands model in Unity although as mentioned in the report the performance of this implementation could be improved and would need to be tested. In the research done for this project this approach is the closest to the approach carried out in the experiment. It is lacking however in several key areas:

- It does not have any practical applications.
- The area investigated is not relevant and therefore any results obtain are not relevant to the key area investigated in this project.
- The Docklands model provided by Smart Dublin is extremely realistic and provides

much greater accuracy than generating a model through CityEngine.

### 4.1.3 Bicycle movement model

An improvement on the default models would be to create an improved bicycle movement model for Sumo. The research paper "Integration of an External Bicycle Model in SUMO" (9) describes an approach of overcoming bicycle model problems using an external control script.

As the author describes in the Motivation section of the paper "In this paper a method for simulating the path-finding behaviour of bicyclists as they cross an intersection is presented. Issues with indirect left turns, bidirectional bicycle lanes and shared space are addressed". The main issue looked at was the behaviour of cyclists at intersections with how cyclists interact with other road users addressed as well. In order to do this the authors collected video data from a road intersection in Germany and used open source software to analyse the trajectories of the road users. These studied trajectories are used as a guideline for the simulated cyclists by selecting the lowest cost action for the cyclist using the cost formula shown in Fig.1 below.

$$Cost_{\alpha} = \beta_{dist}dist_{\alpha} + \beta_{\Delta v}\Delta v_{\alpha} + \beta_{iSG}iSG_{\alpha}$$

Where:

$Cost_{\alpha}$  = cost of carrying out a given action  $\alpha$   
 $dist_{\alpha}$  = distance between the position after action  $\alpha$  and the guideline ( $m$ )  
 $v_{obs}$  = velocity at a given point on the guideline ( $m/s$ )  
 $v_{\alpha}$  = velocity after carrying out a given action  $\alpha$  ( $m/s$ )  
 $\Delta v_{\alpha} = |v_{obs} - v_{\alpha}|$   
 $iSG_{\alpha}$  = inverse space gap ( $m^{-1}$ ) ( $iSG_{\alpha} = 0$  if  $SG \geq 5m$ ,  $iSG_{\alpha} = \infty$  if  $SG \leq 0m$ )  
 $\beta_{dist}, \beta_{\Delta v}, \beta_{iSG}$  = weighting parameters ( $m^{-1}$ ,  $s/m$ ,  $m$ , respectively)

Figure 4.1: Cost function

Through the authors approach, the implementation of an external method "delivered promising results" (9). It found that the simulated approach followed the modelled approach during normal phases with minimal placement error but during transition phases (intersections etc.) the placement error can grow to 6-7m (9). It provides useful realism for modelling cyclist behaviour in SUMO, however, as the authors mention this can and should be improved upon using machine learning. For the purpose of this project, this paper gives useful information and theory to create an external control model.

## 4.2 Background theory

### 4.2.1 Traffic patterns in Dublin City

In order to create simulations with realistic traffic density and distribution real word Dublin traffic patterns must be researched. The National Transport Authority published a report "Canal Cordon Report 2017"(5) which described the trends in mode share of vehicles and pedestrians in an area comprising the Dublin City Centre shown in Figure 4.2.

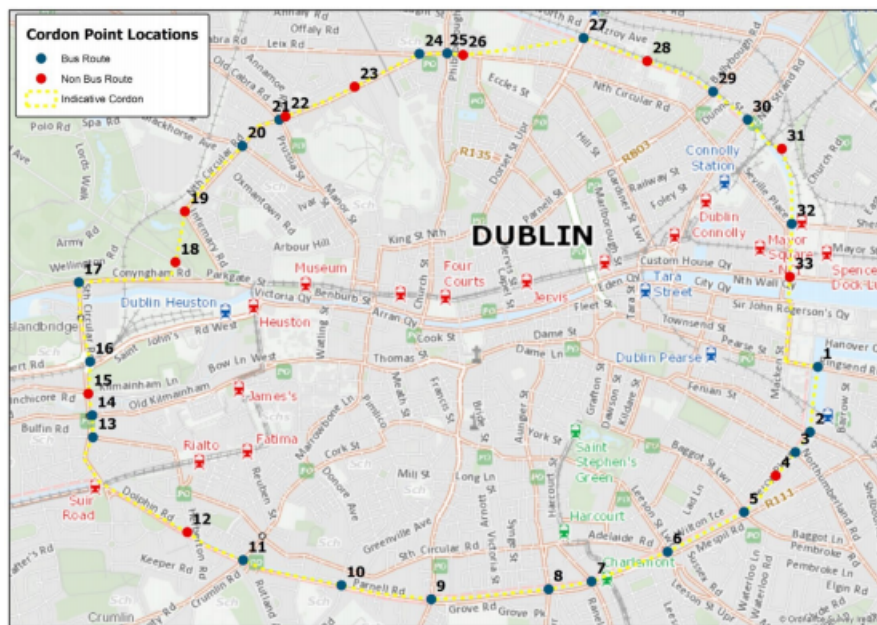


Figure 4.2: (5)

This data is collected via several sources (5):

- Dublin City Council has undertaken surveys at the Canal Cordon in November annually since 1980. Surveys are undertaken over two days at each location and an average across the two days is reported. The survey counts pedestrians, cyclists, cars, taxis, buses, goods vehicles and motorbikes crossing the cordon points in the inbound direction in the three hour, AM peak period 0700-1000
- To complement the Dublin City Council Canal Cordon annual surveys, Dublin Bus have undertaken their own surveys annually on a single day at each location in November. This is not necessarily the same day as the DCC cordon counts. Since 1997 this survey has counted the number of passengers on all buses crossing inbound over the canal cordon points. This survey is undertaken at the 22 cordon points that are on bus routes into the City
- Since 2012, Iarnród Éireann has undertaken a census of passengers boarding and

alighting on all services passing through all stations in the national rail network on a single day. In 2017 the national rail census was carried out on 16th November. Prior to 2012 and since 1997, Iarnród Éireann had undertaken a similar passenger census for services operating within the Greater Dublin Area. Analysis of this data enables a calculation of the numbers of rail passengers crossing the three Canal Cordon points (inbound) between 0700-1000 on the census day.

- Transport Infrastructure Ireland undertakes an annual census of passengers boarding and alighting at all LUAS tram stops. This census is undertaken on a single day in November. It has been undertaken every year since both LUAS lines became operational in 2004. This data enables calculation of the number of LUAS passengers crossing the two Canal Cordon points (inbound) between 0700-1000 on census day

From the data collected, the ratio of cars, bicycles and buses in 2017 was:

Cars	50158
Buses	1637
cyclists	12447

Table 4.1: Ratio of road users

with the number of cyclists and buses increasing in the previous five years indicating a trend. This ratio combined with the information regarding bus timetables (10) in the Grand Canal dock area is what will be used to create the Scenarios described in Chapter 10.

## 4.2.2 Road collisions

Understanding and evaluating results is a crucial element of any report. One of the results that will be obtained from simulations run is collision force whilst know the incident vehicle's positions. Evaluating this information is crucial to giving weight to the observations made in end of project analysis.

According to Mobility and Transport - European Commission (11) There are four types of serious collisions:

1. Single vehicle (run-off road - no road border causes vehicle to leave the road and crash into external object)
2. Head-on collisions (collisions involving vehicles travelling in opposing directions)
3. Side Impact collisions at junctions (Vehicle impacted on its side at an intersection)
4. Collisions involving pedestrians and cyclists. Pedestrians do not survive if struck by a vehicle at above 40km/h.

The scenarios outlined in this paper monitor types two, three and four with incidents involving cyclists taking priority. Any combination of type four and type two/ three is predicted (11) to be cause extremely large impact force on the cyclist/ pedestrian and be fatal.

Minor collisions such as side-to-side impacts produces the most numerous types of injuries (9,199) according to Garda PC16 report forms (12). One interesting outcome of the project and an indicator of accuracy will be the level of minor incidents occurring. These can be identified using the impact force involved and position information of incident vehicles.

Understanding the physics behind impart forces involved in collisions is of vital importance in background work for this project. The paper "On the Mutual Coefficient of Restitution in Two Car Collinear Collisions" (13) provides an interesting discussion on two method's of

calculating collinear collisions. These methods are (13):

1. One based on the masses of vehicles involved
2. One based on the stiffness of vehicles involved

The author extends a linear force model described in "A Note on Linear Force Model in Car Accident Reconstruction" (14) to more collisions involving than one vehicle. In it calculations determine the vehicle's post impact velocity. The approach used by the author can serve as an starting point for calculations done in this project.

## 5 Traffic network generation

In order to create an accurate, realistic scenario displayed in Unity3D, we first have to generate a traffic network in SUMO. In doing this we can get realistic step-by-step information and realistic modelling for each vehicle in the simulation.

### 5.1 Create a .net.xml file

A network file can be simply thought of as the road network that provides the base for a traffic network. The documents for SUMO produced by the DLR outlines network files as follows "A SUMO network file describes the traffic-related part of a map, the roads and intersections the simulated vehicles run along or across" (15).



Figure 5.1: Final road layout

The first step in creating a SUMO simulation is creating this background for the traffic network. This needs to be the same area as outlined in the DCC 3D simulation seen in Fig 3.1. In order to do this we can use OSM's built in map export tool (16) to export a map area like that shown in Figure 3.1 which can be converted into a .net.xml file (readable and editable in SUMO software) using Netconvert (17). By using these tools and by pruning excess features, a network compatible with the Unity3D model is produced as seen in Fig 3.3.

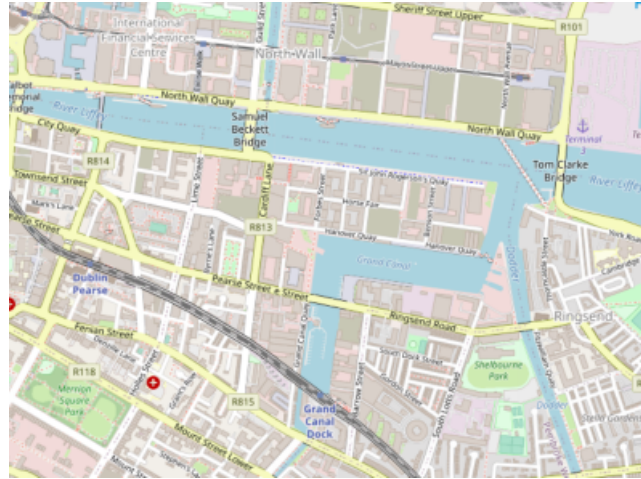


Figure 5.2: OSM map export



Figure 5.3: Final road layout

## 5.2 Generate Traffic

Now that there is this file with interconnected roads and intersections we need to opulate it with routes and vehicles. There are several different methods for generating the "Demand Elements" part of this SUMO configuration. These methods are outlined as follows:

1. Add elements in Netedit.
2. Using RandomTrips.py.
3. Using a text editor of choice to manually edit the config files.

All three were used during different stages in the project and a combination of the methods is typically most effective. For the majority of simulations methods 1 and 3 were used.

When the demands and additionals are created the .net.xml, .add.xml and .rou.xml files are combined into one .sumocfg file.



### 5.2.1 Add elements in Nedit

The netedit interface has a "Demand" tab allows a user to add and edit demand elements ranging from vehicles to routes to bus stops. This method is the most user friendly of the three discussed. Below outlines the main tools used in this method

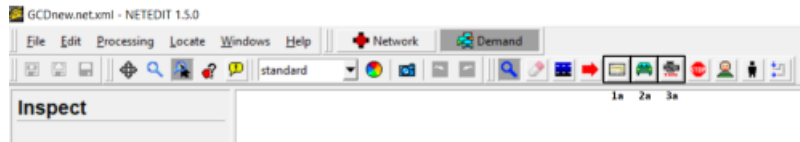


Figure 5.4: Demands section



Figure 5.5: Network section

From Fig 3.3 and 3.4:

- 1a Route mode - create a new route with colour and name through different points on roads
- 2a Vehicle mode - add vehicles with start position, speed etc. to these generated routes
- 3a Vehicle type editor - customise the vehicles to your specifications
- 4a Additional - create and position additional elements such as bus stops etc. in the network

### 5.2.2 Using RandomTrips.py

RandomTrips.py (18) is a useful script which comes with the SUMO package. It automatically generates the files that are created by the user in section 3.2.1.

It is particularly useful if you want to generate random networks every-time you run the program. By writing a command level line of code you could call this script every-time you run a program (Figure 3.5 - inputs: -n network file used, -e end time of simulation). If it is being run by a user then the user can have greater control over the simulation.

```
randomTrips.py -n input net.net.xml -e 50
```

Figure 5.6

There are multiple extra optional inputs that can be added in order to generate a more complex network. Some useful examples of these include:

- **-vehicle-class**

This allows the user to specify a non-default vehicle class to be added to the output files.

- **-trip-attributes**

This allows the user to use addition parameters in the creation of vehicles. e.g. the speed at which it departs starting position at etc.

- **-period** - **-binomial**

The options allow for the control of the rate at which vehicles arrive in the network (i.e. added to the simulation). - -period is used to set a constant rate e.g. 2 vehicles per time step whilst - -binomial uses a binomial distribution to randomise arrivals(18)

The limitations of using this script include its lack of easily added complexity. Unless you manually edit a lot of the generated files and add several option in the command, it produces methodical, unrealistic entries whereby vehicles are entered into the simulation once per time step with little realistic distribution and all of default values.

### **5.2.3 Using a text editor of choice to manually edit the config files**

This technique can be used in combination with the two previously discussed options or as a standalone method. It involves using the same structure as before, routes, vehicles, additional but instead of using a script or UI to create them all the work is done manually in a text editor. An example of this work is shown in Figure 3.6. This shows an example of a .rou.xml route file for one of the scenarios discussed in Chapter 7. It shows:

1. A new custom vehicle type being declared
2. Several routes being declared with information such as edge nodes contained and stops contained in route.
3. Numerous vehicles of different types being instantiated with values

This technique gives a lot more control, flexibility and options when creating the files necessary for traffic generation. It is the only option where you can easily add bus stops to particular routes and it allows greater control over default values when instantiating vehicles. The limitations include the time taken to create large networks, route edge nodes have to be

```

<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dlr.de/aut/routes_file.xsd">
  <vType id="vType_3" <Class="bus"/>

  <route edges="-102694945 -482525983 -482525982 -482525981 -482525980 482525980 482525981 482525982 482525983 102694945 -102694945 -482525983"
    <route edges="-102694945 -482525983 -482525982 -482525981 -482525980 482525980 482525981 482525982 482525983 102694945 -102694945 -482525983"
    <stop busStop="busStop1" duration="30"/>
  </route>
  <route edges="-102694945 -482525983 -482525982 -482525981 -482525980 482525980 482525981 482525982 482525983 102694945 -102694945 -482525983"
    <route edges="-102694945 -482525983 -482525982 -482525981 -482525980 482525980 482525981 482525982 482525983 102694945 -102694945 -482525983"
    <stop busStop="busStop1" duration="30"/>
  </route>
  <route edges="-102694945 -482525983 -482525982 -482525981 -482525980 482525980 482525981 482525982 482525983 102694945 -102694945 -482525983"
    <route edges="-102694945 -482525983 -482525982 -482525981 -482525980 482525980 482525981 482525982 482525983 102694945 -102694945 -482525983"
    <stop busStop="busStop1" duration="30"/>
  </route>

  <vehicle id="bus_0" type="vType_3" depart="0.00" color="red" route="route_bus_0"/>
  <vehicle id="bike_1" type="DEFAULT_BIKETYPE" depart="1.00" color="yellow" route="route_1" departPosLat="right" departSpeed="random"/>
  <vehicle id="bike_2" type="DEFAULT_BIKETYPE" depart="2.00" color="yellow" route="route_1" departPosLat="right" departSpeed="random"/>
  <vehicle id="vehicle_3" type="DEFAULT_VEHTYPE" depart="3.00" color="red" route="route_3"/>
  <vehicle id="vehicle_4" type="DEFAULT_VEHTYPE" depart="4.00" color="red" route="route_3"/>
  <vehicle id="vehicle_5" type="DEFAULT_VEHTYPE" depart="5.00" color="red" route="route_3"/>

```

Figure 5.7

manually found and added, and a slow learning curve if unfamiliar with coding. However this technique is extremely valuable and should be used in concert with the first method mentioned (as done in this project), for example:

1. Easily create the routes needed for the simulation in Netedit and save file as example.rou.xml
2. Open example.rou.xml in your favorite text editor
3. Create any custom vehicle types needed
4. Edit any routes created with any additional you require it to have e.g. bus stop
5. Add or edit any new vehicles you need in the simulation
6. This process can be repeated for any additional .add.xml needed

## 6 Creating an interface between SUMO and Unity3D

### 6.1 TraCI

Now that a traffic network is generated and running the information surrounding it needs to be accessible and transferred to the Unity3D simulation. This is done using a tool in the SUMO suite called TraCI.

TraCI is the short term for "Traffic Control Interface". Giving access to a running road traffic simulation, it allows to retrieve values of simulated objects and to manipulate their behaviour "on-line".

...TraCI uses a TCP based client/server architecture to provide access to SUMO. Thereby, SUMO acts as server that is started with additional command-line options.(19)

The architecture behind TraCI is shown in Figure 6.1. It uses a TCP connection to transfer requested data and accept control commands. The main commands used to request information and control the simulation are as follows:

```
traci.start("sumo", "-c", "scenario1.sumocfg")
```

This initiates the start of the Scenario 1 simulation with the executable SUMO and with a .sumocfg file input.

```
traci.simulationStep()
```

This initiates the start of each step in the simulation. Without receiving this command the server will not begin a time step.

`traci.simulation.getArrivedIDList()`

This returns a list of the id's of the vehicles which have entered the simulation on a time step.

`traci.vehicle.getPosition("bus_0")`

This returns a the x and y coordinates for vehicle with id "bus\_0".

`traci.close()`

Sends a `TCP_close` message to the server to end the connection.

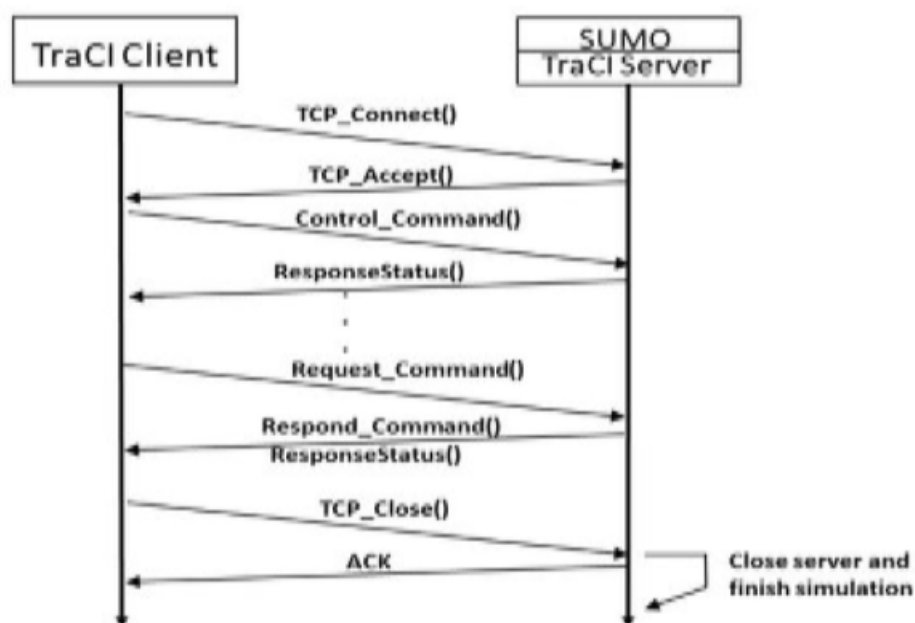


Figure 6.1: (6)

## 6.2 TCP/IP

### 6.2.1 Server

In the `sumo_server.py` python script that contain the Tracl code used to pull information, it also creates a TCP server in order to make this information available to other sources. For this section a module called `Socket(20)` is used to accomplish this. Python's own documentation about socket states:

This module provides access to the BSD socket interface. It is available on all modern Unix systems, Windows, MacOS, and probably additional platforms.

The Python interface is a straightforward transliteration of the Unix system call and library interface for sockets to Python's object-oriented style: the `socket()` function returns a socket object whose methods implement the various socket system calls. Parameter types are somewhat higher-level than in the C interface: as with `read()` and `write()` operations on Python files, buffer allocation on receive operations is automatic, and buffer length is implicit on send operations.(20)

These built in high-level functions make creating a server a straightforward matter however this server needs to be tailored to the specific demands of this project.

## 6.2.2 Client

In the c# assembly script `Sumo_to_unity_connector.cs` that is used to manage the Unity3D game scene, a TCP client is created in order to connect to the server and request positional information. For this implementation the default `System.Net.Sockets` module is used. Microsoft's documentation on this module states:

The `Socket` class provides a rich set of methods and properties for network communications. The `Socket` class allows you to perform both synchronous and asynchronous data transfer using any of the communication protocols listed in the `ProtocolType` enumeration.

The `Socket` class follows the .NET Framework naming pattern for asynchronous methods. For example, the synchronous `Receive` method corresponds to the asynchronous `BeginReceive` and `EndReceive` methods.

If your application only requires one thread during execution, use the following methods, which are designed for synchronous operation mode.

If you are using a connection-oriented protocol such as TCP, your server can listen for connections using the `Listen` method. The `Accept` method processes any incoming connection requests and returns a `Socket` that you can use to communicate data with the remote host. Use this returned `Socket` to call the `Send` or `Receive` method. Call the `Bind` method prior to calling the `Listen` method if you want to specify the local IP address and port number. Use a port number of zero if you want the underlying service provider to assign a free port for you. If you want to connect to a listening host, call the `Connect` method. To communicate data, call the `Send` or `Receive` method.(21)

These built in function make creating a client a reasonable task however the lack of several high-end functions like the python module contains adds to the difficulty. The client also needs to be tailored to the specific demands of this project and be able to parse the recieved data into usable information.

### 6.2.3 Server and Client

The below Figure 6.2 shows the architecture of the TCP connection between `sumo_server.py` and `Sumo_to_unity_connector.cs`. Having initiated the beginning of the sumo simulation the python script creates a socket and then listens for any requests for this information. On the binding to the socket of a client and initiating message it begins to run each time step whilst sending positional information about vehicles to the client.

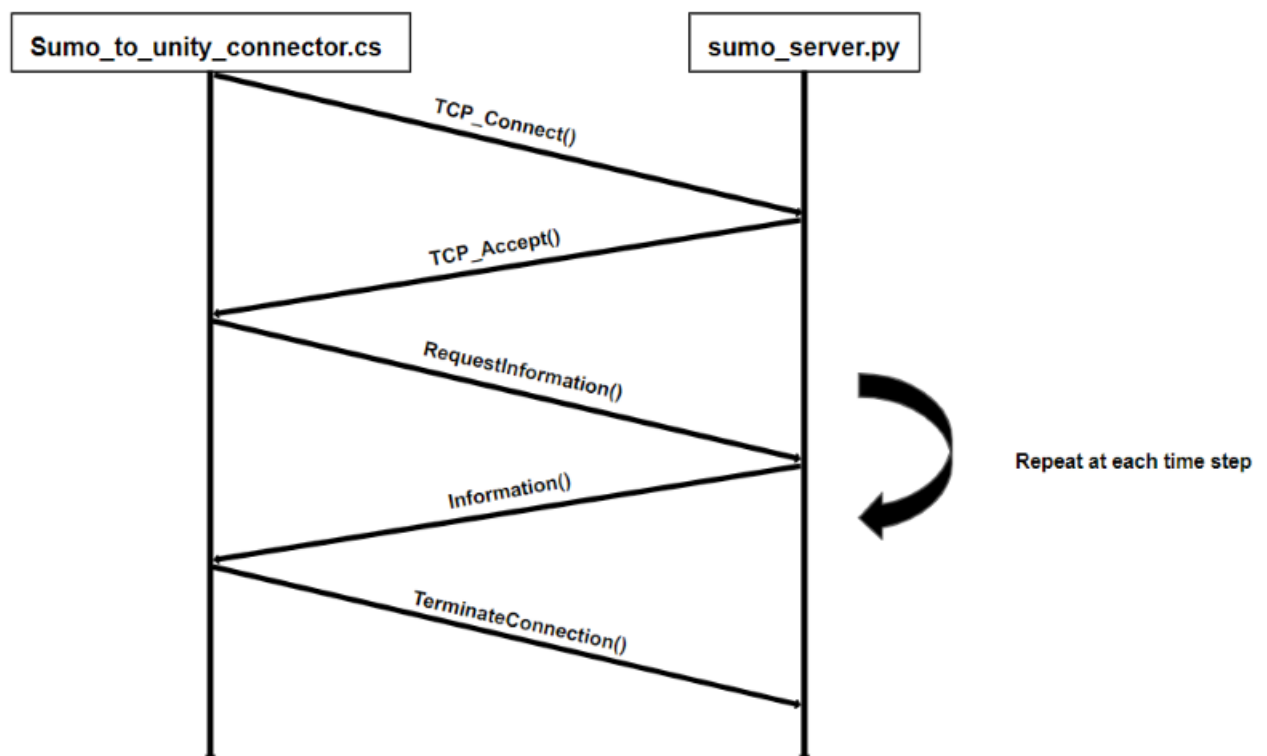


Figure 6.2: Connection

## 7 Displaying vehicles in Unity3D

### 7.1 The basics

When starting development of a game in Unity3D the first step is typically to create the environment it will be played in. For the purpose of this project the Smart Dublin Docklands model is being used (as shown in Section 3.1). This model is a highly accurate physical representation of the area with buildings and features represented. If you wish to display and interact with any sort of entity in this Unity3D scene you do this by creating a "GameObject" (22). This object by default has no physical characteristics, positional information or behavioural traits. All these features must be added as components. There are two main ways to do this:

1. Manually create each component and add to the GameObject
2. Load premade prefabs that you or someone else has previously created

#### 7.1.1 Manually create each component and add to the GameObject

Unity by default has some simple primitive objects that can be used to build up the parts to the object you want to display e.g. rectangle, sphere, cylinder etc. By creating a hierarchy of these objects the custom GameObject being created can begin to take shape. Doing this in combination with using another graphics program like Autodesk 3ds Max (23) can lead you to creating the objects you need.

The amount of components that you create to make up your object depends on the complexity required. A simple combination of cylinders, spheres and rectangles can be used to create an animal like a cat or dog but an extremely realistic representation might have millions of hairs attached etc.

Adding positional information is crucial as well. An object is default created with a null attribute "transform". By editing this value and applying other transform vectors you can



position and move the object.

Components also include characteristics of the object such as the physics that govern the actions of an object. E.g. in Figure 5.1 a "rigidbody" component is added, this means that the object will now obey the Unity3D physics engine, it will get pulled down by gravity and will react to other objects in the scene. A BoxCollider can also be attached which is a bounding box around the object monitoring for any contact. This is discussed further in Section 6.1.

```
g.AddComponent<Rigidbody>();
```

Figure 7.1

## 7.1.2 Prefabs

Prefabs are widely used in creating Unity3D scenes whether it be your own custom fabs or ones downloaded from Unity3D's store. This is done by having your prefabs saved in the resources folder and loaded in on the instantiation of the game object as shown in Figure 5.2.

```
g = Instantiate(Resources.Load(type_veh)) as GameObject;
```

Figure 7.2

## 7.2 Vehicle Class

For this project several prefabs were used to represent cars, buses and bicycles however a custom class was created to house these objects differently than discussed above due to the nature of SUMO controlling movement. Attribute information needed to be stored separately from the game object. In this situation the game object is stored in the class as the visual appearance whilst the other information (position, orientation angle, id) is stored as attributes. In Figure 5.3 we can see this implemented.

```
class Vehicle
{
    public string id;
    public float current_x, current_y, current_z, prev_x, prev_z, angle_c;
    public GameObject g;
```

Figure 7.3

## 7.3 Instantiating a vehicle

Chapter 4 outlined how information regarding vehicles is passed into the c# script but on receiving this information its just one long string (Figure 5.4). This information has to be used to create and update vehicles. The string is parsed for individual components. The components are then used in the construction on a vehicle instance. The constructor is shown in Figure 5.5

vehicle\_4 1104.94983571 1081.82249782 Car

Figure 7.4: Vehicle information received

```
public Vehicle(string id, float curx, float curz, string type_veh)
{
    //constructor
    this.id = id;
    current_x = - (curx - 680);
    current_y = 4;
    current_z = -(curz - 1131);
    g = Instantiate(Resources.Load(type_veh)) as GameObject;
    g.name = "VehicleID: " + id + type_veh;
    g.AddComponent(Type.GetType("collision_detection_m"));
}
```

Figure 7.5

## 7.4 Mapping coordinates

Sumo and Unity3D operate off of different coordinate systems. Sumo has a basic 2-d coordinate system (x,y axis) whilst Unity3D operates on a 3-d coordinate system (x,y,z axis) (shown in Figure 5.3, The vertical axis in this case is y-axis). Neither systems common axis are equivalent either,  $x_{SUMO} \neq x_{Unity3D}$  and  $y_{SUMO} \neq y_{Unity3D}$ . A conversion system must be created and implemented.



Figure 7.6: Sumo coordinate system

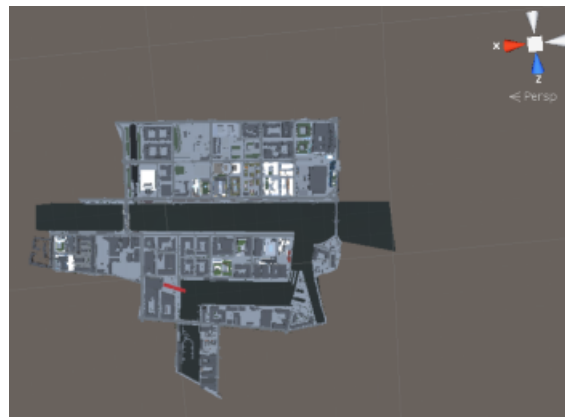


Figure 7.7: Unity coordinate system

In order to do this an equivalency needs to be created. This is done by lining up the two systems as shown in Figure 5.2 and 5.3, picking points at the same real world location and comparing x,y and x,z values. In doing this we can now map coordinates from one system to another. In this project's case the coordinate conversion is shown below in Figure 5.4.

```
Unity_X = -(SUMO_X - 680)
Unity_Z = -(SUMO_Z - 1131)
```

Figure 7.8: Conversion formula

## 7.5 Determining an objects orientation

As this project uses SUMO to control the traffic network the position of objects in Unity is determined based on SUMO. This has been established in previous sections and will be expanded on in Chapter 6 however this leads to one major issue which will be discussed in this section.

As mentioned in Section 5.1 the default method for moving a game object in Unity3D is through using the Transform() method by applying a three dimensional vector (x,y,z). This is done through user control i.e. using 'w', 'a', 's', 'd' keys to indicate a movement 'up', 'left', 'down', 'right'. The appropriate transforms are determined from this key press and applied and the orientation of the object can be determined based on inferring from user control. This is not applicable in the SUMO method used in this project as Unity never knows the starting orientation. A method for arriving at this orientation at each time step therefore is needed to be implemented.

On the first time step all that is known is current position, from this nothing can be determined regarding the orientation however at the second and all future time steps the current and previous positions are known i.e. from figure 5.5 at time step 1, the vehicle was at position (a), from 5.6 at time step 2, the vehicle was at position (b). Therefore we can infer that the vehicle is travelling in a direction towards marker (m1) and therefore facing that direction.

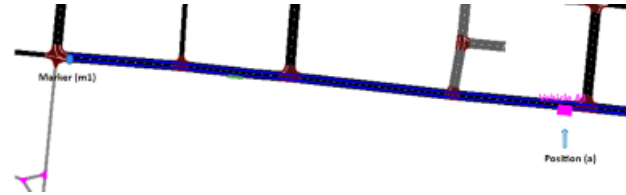


Figure 7.9: Time step 1

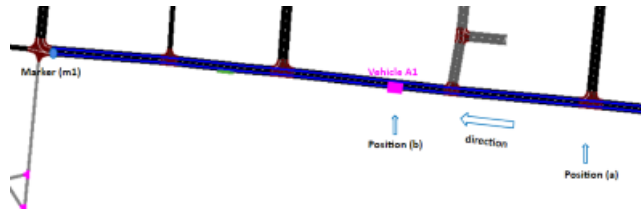


Figure 7.10: Time step 2

The resultant vector calculated from these positions is as follows:

$$ab = (x_a - x_b, z_a - z_b)$$

From this vector an angle with respect to the origin, can be determined which is used to calculate orientation:

$$angle = Atan2(ab) * \frac{180}{\pi}$$

Using the aforementioned transform method which is attached to the object, orientation can now be set. The objects rotation is aligned in the y-axis to the angle determined above. This is shown in Figure 5.7

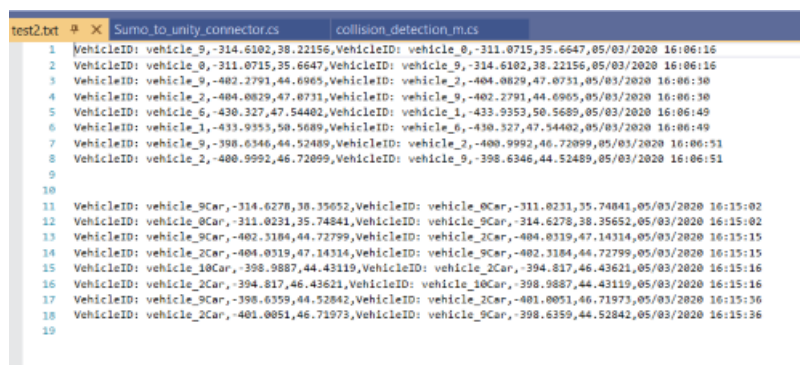
```
item.g.transform.rotation = Quaternion.Euler(0, (float)item.angle_c, 0);
```

Figure 7.11

## 8 Implementation of collision detection

### 8.1 Attaching a collision detector

In order to analyse the safety and status of this generated traffic network we need to monitor the status of each of these newly generated cars, bicycles, buses etc. Unity3D has a built in "OnCollisionEnter" function which passes in information regarding a collision eg. objects involved, object positions etc. For the purposes of monitoring each object's collision status a "collisiondetection\_m" class is added as a component to each generated game-object. This monitors for collision and on a collision event enters the OnCollisionEnter function. This function gets passed all information Unity3D default monitors about a collision. It then uses this information about the collision and writes it to a results text file. An example of this text is seen below in Fig 5.1.



```
test2.txt # Sumo to unity connector.cs collision_detection_m.cs
1 VehicleID: vehicle_9,-314.6102,38.22156,VehicleID: vehicle_0,-311.0715,35.6647,05/03/2020 16:06:16
2 VehicleID: vehicle_0,-311.0715,35.6647,VehicleID: vehicle_9,-314.6102,38.22156,05/03/2020 16:06:16
3 VehicleID: vehicle_9,-402.2791,44.6965,VehicleID: vehicle_2,-404.0829,47.0731,05/03/2020 16:06:30
4 VehicleID: vehicle_2,-404.0829,47.0731,VehicleID: vehicle_9,-402.2791,44.6965,05/03/2020 16:06:30
5 VehicleID: vehicle_6,-430.327,47.54402,VehicleID: vehicle_1,-433.9353,50.5689,05/03/2020 16:06:49
6 VehicleID: vehicle_1,-433.9353,50.5689,VehicleID: vehicle_6,-430.327,47.54402,05/03/2020 16:06:49
7 VehicleID: vehicle_9,-398.6346,44.52489,VehicleID: vehicle_2,-400.9992,46.72099,05/03/2020 16:06:51
8 VehicleID: vehicle_2,-400.9992,46.72099,VehicleID: vehicle_9,-398.6346,44.52489,05/03/2020 16:06:51
9
10
11 VehicleID: vehicle_9Car,-314.6278,38.35652,VehicleID: vehicle_0Car,-311.0231,35.74841,05/03/2020 16:15:02
12 VehicleID: vehicle_0Car,-311.0231,35.74841,VehicleID: vehicle_9Car,-314.6278,38.35652,05/03/2020 16:15:02
13 VehicleID: vehicle_9Car,-402.3104,44.72799,VehicleID: vehicle_2Car,-404.0319,47.14314,05/03/2020 16:15:15
14 VehicleID: vehicle_2Car,-404.0319,47.14314,VehicleID: vehicle_9Car,-402.3104,44.72799,05/03/2020 16:15:15
15 VehicleID: vehicle_10Car,-398.9887,44.43119,VehicleID: vehicle_2Car,-394.817,46.43621,05/03/2020 16:15:16
16 VehicleID: vehicle_2Car,-394.817,46.43621,VehicleID: vehicle_10Car,-398.9887,44.43119,05/03/2020 16:15:16
17 VehicleID: vehicle_9Car,-398.6359,44.52842,VehicleID: vehicle_2Car,-401.0051,46.71973,05/03/2020 16:15:36
18 VehicleID: vehicle_2Car,-401.0051,46.71973,VehicleID: vehicle_9Car,-398.6359,44.52842,05/03/2020 16:15:36
19
```

Figure 8.1: Example of collision results from a simulation

### 8.2 Force generated

In order to analyse the results, a fuller picture is needed. In order to do this, not only is information about vehicles involved and positions needed but information about the force

involved in the collision is needed.

From Newton's Second law of Motion we have:

$$F = m.a$$

However we know that acceleration is rate of velocity change given a time period or:

$$a = m.\frac{\Delta V}{\Delta t}$$

and impulse or the change in momentum is:

$$\Delta p = m.\Delta V$$

therefore an equation for Force that is compatible in Unity3D can be obtained:

$$F = \frac{\Delta p}{\Delta t}$$

Now that this equation for Force is generated it needs to be implemented. This is done by using the information Unity3D passes into the collision detector. At each step  $\Delta t$  is recorded and sent to this detector. Impulse can be manually calculated by determining change in velocity:

$$magnitude(currentpositionalvalues - previouspositionalvalues)$$

and the mass stored for each game object. It can also be obtained by using built in `collision.impulse.magnitude` stored value. These  $\Delta p$  and  $\Delta t$  values are then plugged into the formula and stored in the results csv file.

```
float colForce = collision.impulse.magnitude / Time.fixedDeltaTime;
```

Figure 8.2

## 9 Project workflow

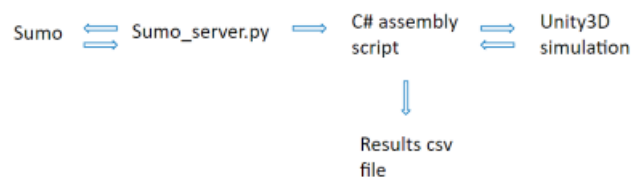


Figure 9.1: Workflow of entire project when run

Having gone over each component of the project, this section will show how they all connect and interact. The main connections are shown in Fig. 6.1 and are outlined below.

The workflow starts with the `Sumo_server.py` script. As mentioned in previous sections this code uses TracI to start a SUMO simulation. It launches with whatever `.sumocfg` file is referenced in the script and controls when each time-step occurs. On running the script it also creates a TCP server and waits for a connection. On the initiation of each time step, using TracI, the script pulls information about each vehicle's state and makes this information (shown in Figure 6.2) available to a connecting client.

```

Command Prompt
0
Bus_0 1134.43031420 1078.77053877 Car
vehicle_3 1185.89718178 1073.44501822 Car
bike_1 1167.19285612 1074.00890157 Motor
bike_2 1177.77106924 1073.80409582 Motor
10
Bus_0 1126.34195001 1079.6085599 Car
vehicle_3 1181.74170138 1073.58788289 Car
bike_1 1161.85812402 1074.65186971 Motor
bike_2 1172.89749961 1073.50857084 Motor
11
Bus_0 1118.45365815 1080.4204552 Car
vehicle_3 1179.46873634 1074.11038281 Car
bike_1 1156.49155279 1075.20647316 Motor
bike_2 1167.54634086 1074.06217836 Motor
12
Bus_0 1110.27138968 1081.27175344 Car
vehicle_3 1185.89718178 1073.44501822 Car
vehicle_4 1173.65088537 1077.93442281 Car
bike_1 1151.0078488 1075.77399807 Motor
bike_2 1162.49544256 1074.50511169 Motor
13
Bus_0 1102.54850498 1082.07184705 Car
vehicle_4 1183.72913299 1073.60938814 Car
vehicle_3 1165.92542115 1070.73436892 Car
bike_1 1145.92095225 1076.20981062 Motor
bike_2 1157.4745776 1075.18473683 Motor
14
Bus_0 1094.25791913 1082.92081704 Car
vehicle_4 1179.65481218 1074.09112723 Car
  
```

Figure 9.2: Vehicle information sent over TCP connection  
Left to right: vehicle ID, x-coordinate, y-coordinate, vehicle type

The C# assembly script is attached in game to the Smart Dublin model docklands model.



On command it creates a client and connects to the server created above. On initial connection it receives vehicle information and creates a game object for each vehicle. It attaches a collision detector and displays in-game. Each time-step the script receives the new vehicle information, calculates any other necessary information (vehicle orientation) and displays this updated game objects. Any objects entering the simulation after the first time step will also be created, monitored and displayed the same as the initial vehicles.

```
[11:26:04] bus_0
UnityEngine.Debug:Log(Object)
[11:26:04] -498.1698
UnityEngine.Debug:Log(Object)
[11:26:04] 4
UnityEngine.Debug:Log(Object)
[11:26:04] 56.75525
UnityEngine.Debug:Log(Object)
[11:26:04] 95.90707
UnityEngine.Debug:Log(Object)
[11:26:04] VehicleID: bus_0Car (UnityEngine.GameObject)
UnityEngine.Debug:Log(Object)
```

Figure 9.3: Vehicle information stored in Unity3D

Top to bottom: vehicle ID, x-coordinate, y-coordinate, z-coordinate, orientation angle, game object displayed

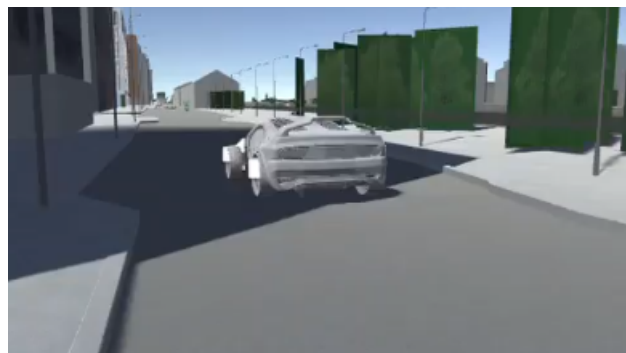


Figure 9.4: End result with vehicle being displayed

On a collision being detected, Unity passes all information about the collision into the attached detector. This then leads to the script pulling selected information about the collision and writing to a results csv file to be stored and analysed later (shown in Figure 6.5).

Vehicle ID (V_a)	V_a x coordinate	V_a y coordinate	Vehicle ID (V_b)	V_b x coordinate	V_b y coordinate	Force in collision
VehicleID: bus_12Car	-209.3747	-409.155	VehicleID: bike_1	-210.9015	-406.3455	15/04/2020 16:35 281466.4
VehicleID: bike_13Motor	-210.9015	-406.3455	VehicleID: bus_12	-209.3747	-406.155	15/04/2020 16:35 281466.4
VehicleID: bus_12Car	-209.3787	-409.1484	VehicleID: bike_1	-211.3198	-411.715	15/04/2020 16:35 723336.3
VehicleID: bike_13Motor	-211.3198	-411.715	VehicleID: bus_12	-209.3787	-409.1484	15/04/2020 16:35 723336.3
VehicleID: bus_12Car	-209.5603	-409.0754	VehicleID: bike_1	-211.1645	-408.8876	15/04/2020 16:35 6083614
VehicleID: bike_14Motor	-211.1645	-409.3876	VehicleID: bus_12	-209.5603	-409.0754	15/04/2020 16:35 6083614
VehicleID: vehicle_5Car	-218.1704	-510.3914	VehicleID: vehicle	-218.7418	-510.4237	15/04/2020 16:35 175324.2
VehicleID: vehicle_3Car	-216.7418	-510.4237	VehicleID: vehicle	-218.1704	-510.3914	15/04/2020 16:35 175324.2
VehicleID: vehicle_4Car	-219.7153	-519.2981	VehicleID: bus_0C	-219.7584	-518.8038	15/04/2020 16:35 188890.2
VehicleID: bus_0Car	-219.7584	-518.8038	VehicleID: vehicle	-219.7153	-519.2981	15/04/2020 16:35 188890.2

Figure 9.5: Collision information written to csv file

# 10 Results

In this chapter the scenarios run using the program described in Chapter 9 are outlined and the results obtained are published. Three scenarios are described in Section 10.1, 10.2 and two scenarios in Section 10.3.

Each scenario will be described with the same format. This format is outlined here:

- Description of the traffic situation in the scenario.
- Diagram showing location of routes implemented with routes highlighted in blue or yellow, bus-stops circled in green.
- Traffic density and rules followed in the scenario.
- Results obtained in scenario.

The results obtained are shown in a bar chart with collisions detected in scenarios with bus stops active denoted by a blue bar and collisions where bus stops are inactive denoted by a orange bar.

## 10.1 Small tests

These are limited tests across small areas designed in order to examine specific small situations e.g. a section of straight narrow road. Each scenario will be run twice, once with bus stops active and once with bus stops inactive and results will compare the difference.

### 10.1.1 Scenario 1A

This scenario examines a limited peak rush-hour situation on a straight road. It is designed to represent the peak morning rush hour traffic i.e 8-9.00 am

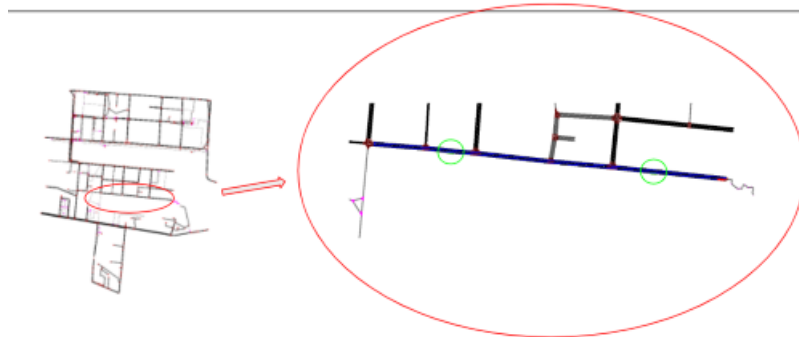


Figure 10.1

Density:

- 67 Cars
- 10 Buses
- 35 Bicycles

Rules:

- Approximately 1m between motor vehicles
- Bicycles can occupy the same lane space as other vehicles
- Overtaking allowed and road users will overtake if the option is available

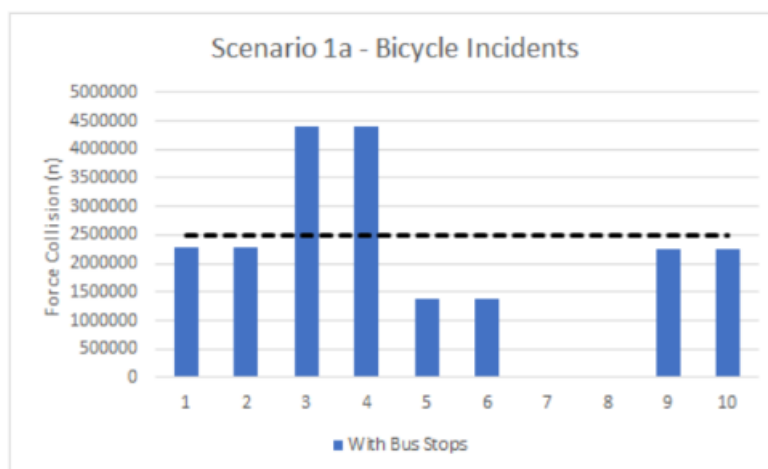


Figure 10.2

### 10.1.2 Scenario 1B

This scenario examines a limited off-peak rush-hour situation on a straight road. The routes and area is the same however there is more space between motor vehicles and more time between bus stop arrivals compared to Scenario 1A. This scenario is designed to represent a 11.00am situation whereby there is still reoccurring traffic but a lot less density than peak rush-hour morning traffic. Density:

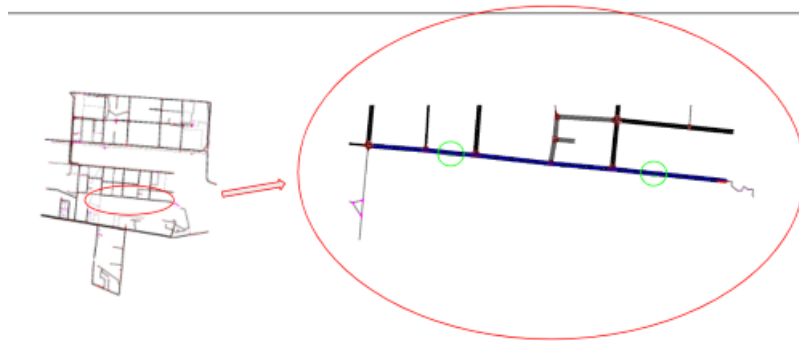


Figure 10.3

- 24 Cars
- 6 Buses
- 12 Bicycles

Rules:

- Approximately 3m between motor vehicles
- Bicycles can occupy the same lane space as other vehicles
- Overtaking allowed and road users will overtake if the option is available

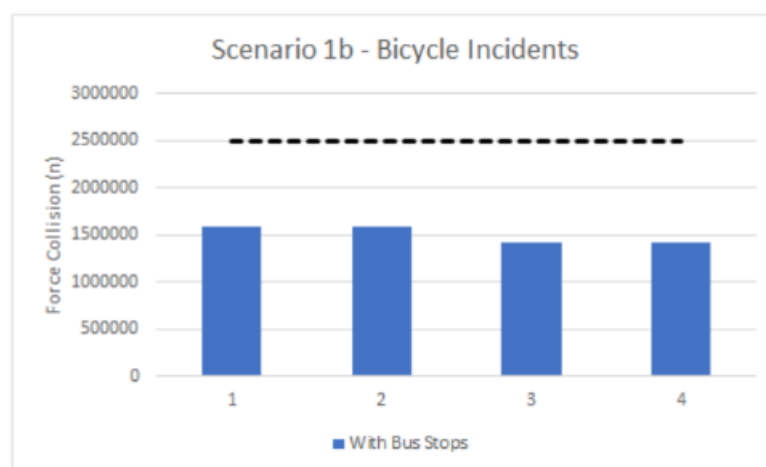


Figure 10.4

### 10.1.3 Scenario 1C

This scenario examines a limited peak rush-hour situation at an intersection. It is designed to represent the peak morning rush hour traffic i.e 8-9.00 am but ensuring flow.

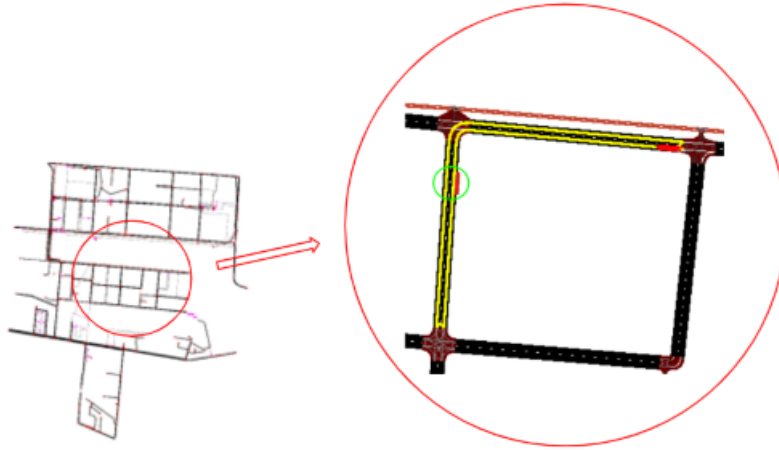


Figure 10.5

Density:

- 67 Cars
- 9 Buses
- 35 Bicycles

Rules:

- Approximately 1m between motor vehicles
- Bicycles can occupy the same lane space as other vehicles
- Overtaking allowed and road users will overtake if the option is available

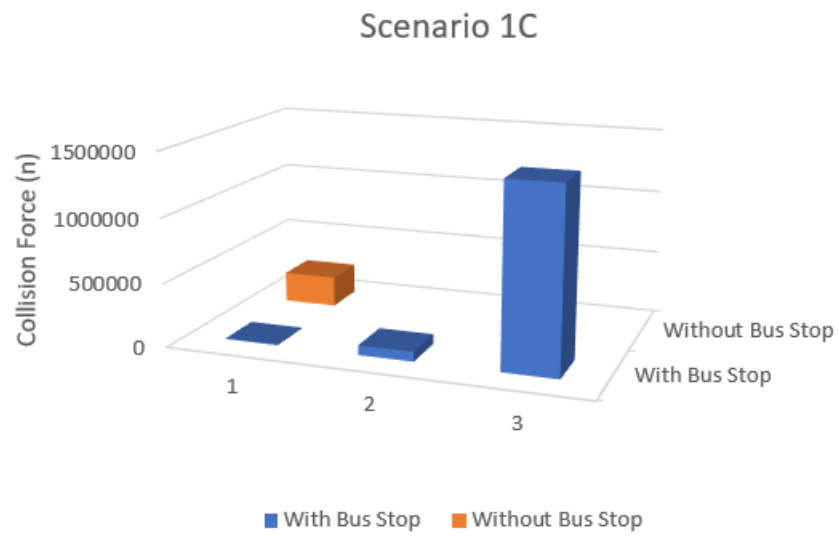


Figure 10.6

## 10.2 Complex tests

These are expanded tests across larger areas often combining multiple tests from the previous section into one designed to examine specific large situations across a block. Each scenario will be run twice, once with bus stops active and once with bus stops inactive. Results will compare the difference.

### 10.2.1 Scenario 2A

These are expanded tests across larger areas often combining multiple tests from the previous section into one designed to examine specific large situations across a block. Each scenario will be run twice, once with bus stops active and once with bus stops inactive and results will compare the difference. Density:

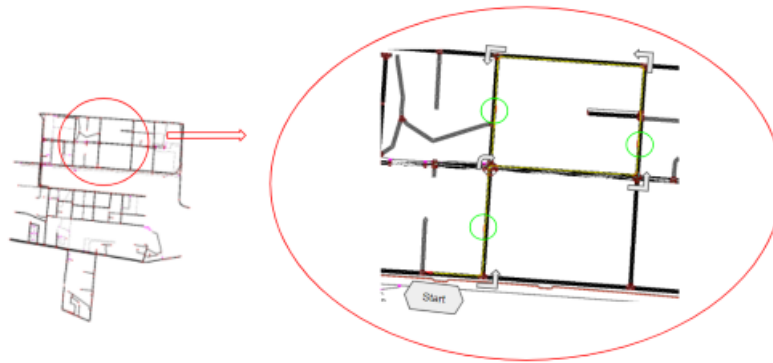


Figure 10.7

- 67 Cars
- 10 Buses
- 35 Bicycles

Rules:

- Approximately 1m between motor vehicles
- Bicycles can occupy the same lane space as other vehicles
- Overtaking allowed and road users will overtake if the option is available
- Traffic lights at intersections

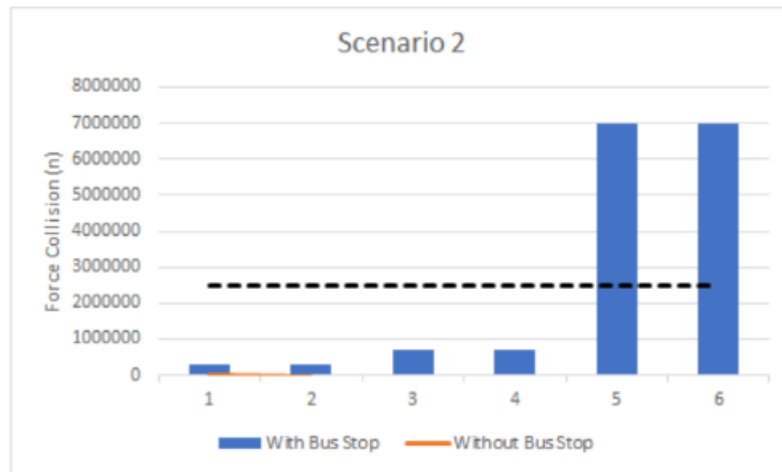


Figure 10.8



### 10.2.2 Scenario 2B

This scenario examines a block-wide peak rush-hour situation with intersections. It is designed to represent the peak morning rush hour traffic i.e 8-9.00 am but ensuring flow.

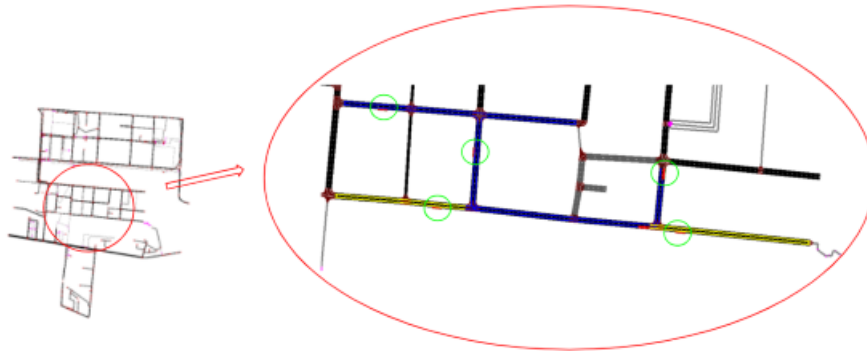


Figure 10.9

Density:

- 82 Cars
- 14 Buses
- 42 Bicycles

Rules:

- Approximately 1m between motor vehicles
- Bicycles can occupy the same lane space as other vehicles
- Overtaking allowed and road users will overtake if the option is available
- Multiple routes interacting (4)

### Scenario 2B

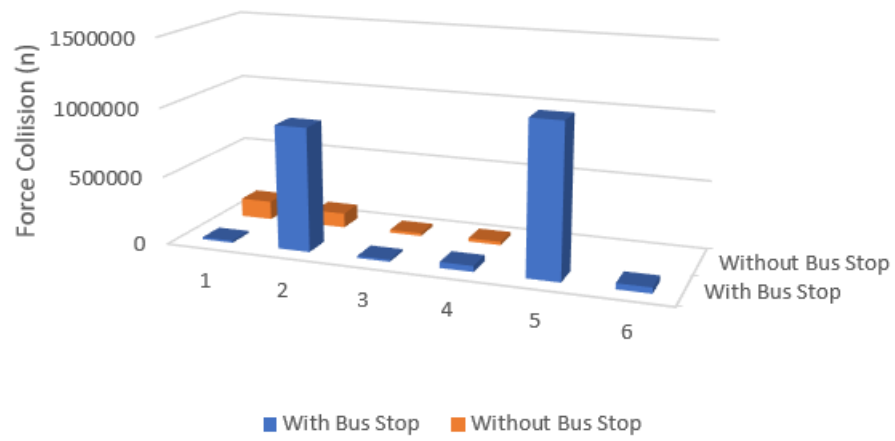


Figure 10.10

### 10.2.3 Scenario 2C

This scenario examines several block-wide peak rush-hour situations with intersections and bridges involved. It is designed to represent the peak morning rush hour traffic i.e 8-9.00 am but ensuring flow.

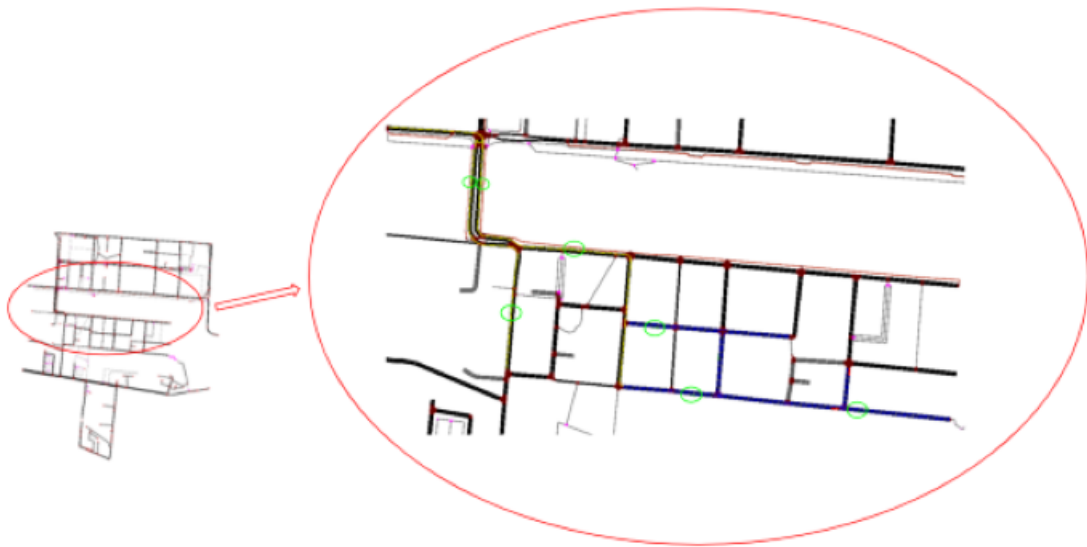


Figure 10.11

Density:

- 101 Cars
- 19 Buses
- 63 Bicycles

Rules:

- Approximately 1m between motor vehicles
- Bicycles can occupy the same lane space as other vehicles
- Overtaking allowed and road users will overtake if the option is available
- Multiple routes interacting (8)

Scenario 2C

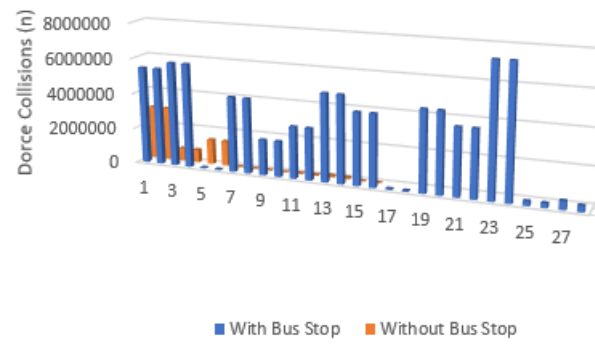


Figure 10.12

## 10.3 Density tests

These scenarios are similar in layout to those shown previously however instead of comparing the results with/without bus stops these will look at comparing each scenario with different bus stop densities. Each scenario will be run twice, once with bus stops active at a "high" density and once with bus stops active at a "low" density and results will compare the difference. Bus stops in low density simulations are shown in green, bus stops in high density simulations shown in blue with green stops also included.

### 10.3.1 Scenario 3A

This scenario examines an off rush-hour situation on a straight road. It is designed to represent the quiet afternoon traffic i.e 2-3.00 pm

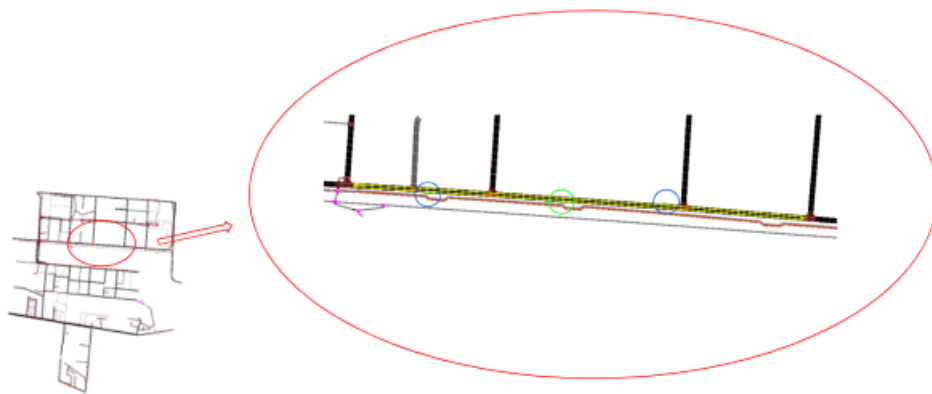


Figure 10.13

Density:

- 18 Cars
- 4 Buses
- 13 Bicycles
- Low Bus Stop count - 1
- High Bus Stop count - 3

Rules:

- Approximately 1m between motor vehicles
- Bicycles can occupy the same lane space as other vehicles
- Overtaking allowed and road users will overtake if the option is available
- Each bus stops at each available stop along the route

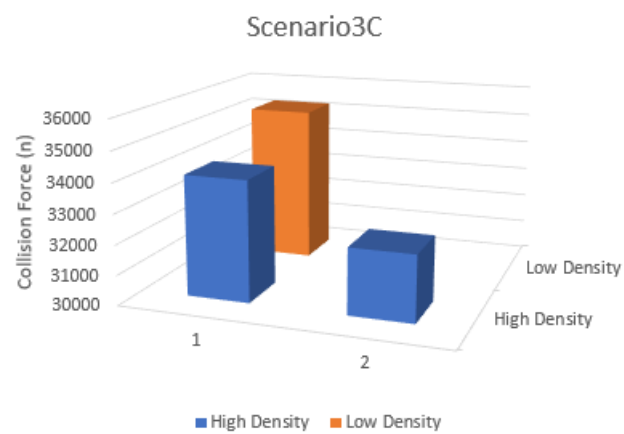


Figure 10.14

### 10.3.2 Scenario 3B

This scenario examines an mid-peak rush-hour situation in a city block. It is designed to represent the mid level afternoon traffic i.e 12-2.00 pm

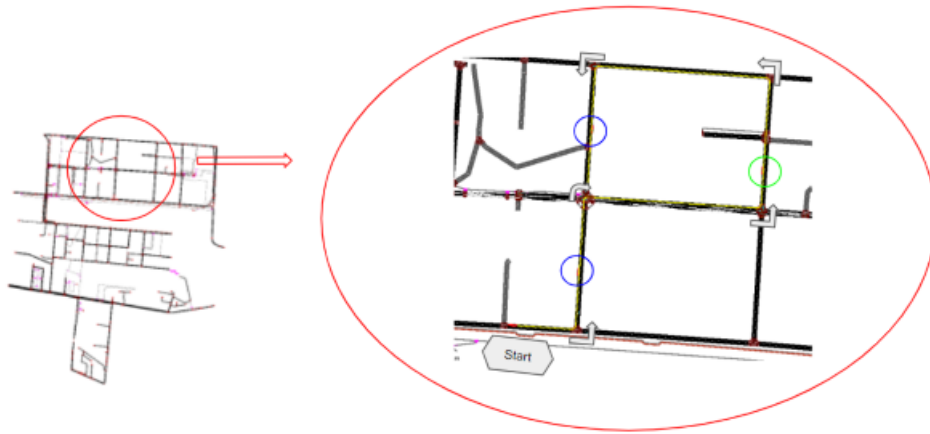


Figure 10.15

Density:

- 18 Cars
- 4 Buses
- 13 Bicycles
- Low Bus Stop count - 1
- High Bus Stop count - 3

Rules:

- Approximately 1m between motor vehicles
- Bicycles can occupy the same lane space as other vehicles
- Overtaking allowed and road users will overtake if the option is available
- Each bus stops at each available stop along the route

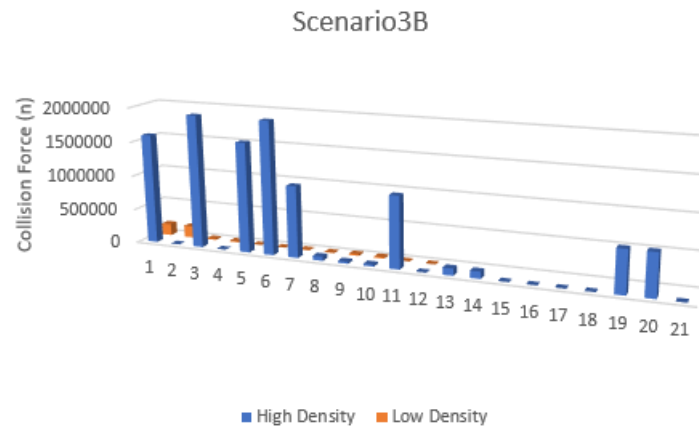


Figure 10.16



# 11 Evaluation

## 11.1 Bus stop v no bus stop

Scenario 1A shows a situation typical in most Dublin Bus routes, a stretch of straight road with one - two bus stops. The time period reflects peak morning rush hour traffic. This scenario shows an increase of collisions from zero to ten with eight of them being serious collisions and two minor grazing. The majority of collisions being serious indicates that head on collisions are more prevalent in this environment of a straight road where there is a significant stretch of area for overtaking.

Scenario 1B shows a situation similar to Scenario 1A, a stretch of straight road with one - two bus stops. The time period reflects off peak morning rush hour traffic. This scenario shows an increase of collisions from zero to four with all four of them being serious collisions. The rate of collisions being serious indicates that again head on collisions are more prevalent in this environment of a straight road where there is a significant stretch of area for overtaking. The reduction in collisions from Scenario 1A to Scenario 2B can be explained through the density of traffic being reduced and therefore the space between vehicles being reduced.

Scenario 1C shows a situation found in Dublin Bus route through the city centre and especially the docklands area, an intersection with one bus stop after the intersection. The time period reflects peak morning rush hour traffic. This scenario shows an increase of collisions from one to three with one of them being serious collisions and two minor grazing. The lack of serious collisions and the lower number of collisions indicates the lack of opportunity in this environment for overtaking.

Scenario 2A shows a further expanded situation relative to the first set. It contains a combination of straight road and intersections with three bus stops. The time period reflects peak morning rush hour traffic. This scenario shows an increase of collisions from two to six with two of them being serious collisions and four minor grazing. The lack of serious collisions and the lower number of collisions indicates the lack of opportunity in this environment for overtaking.

Scenario 2B shows a further expanded situation relative to the first set. It contains a combination of straight road and intersections across a small city block with five bus stops. The time period reflects peak morning rush hour traffic for an area this size. This scenario shows an increase of collisions from two to six with two of them being serious collisions and four minor grazing. The lack of serious collisions and the lower number of collisions indicates the lack of opportunity in this environment for overtaking where there is not a consistent straight stretch of road.

Scenario 2C shows a further expanded situation relative to the first set. It contains a combination of straight road and intersections across a large city block with five bus stops and a river crossing. The time period reflects peak morning rush hour traffic for an area this size. This scenario shows an increase of collisions from fifteen to twenty-seven with eighteen of them being serious collisions and nine minor grazing. The complex nature of this simulation with a large number of interconnecting routes shows the potential for a large number of serious collisions if cyclists take each opportunity to overtake if there is an obstacle like a parked bus at a bus stop.

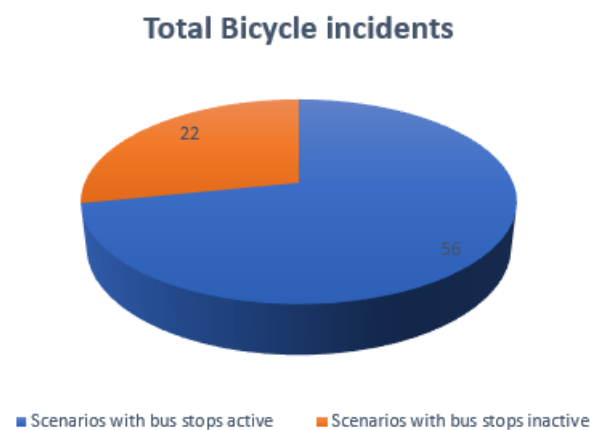


Figure 11.1

## 11.2 Density

Scenario 3A examines a scenario where the density of bus stops is examined instead of the presence of bus stops. It contains a stretch of straight road with one - three bus stops. The time period reflects off-peak early morning traffic for an area this size (4-5.00 am). This scenario shows an increase of collisions from one to two with eighteen of one being serious collisions. Although there is an increase based on density there isn't enough collisions to show any correlation.

Scenario 3B examines a scenario where the density of bus stops is examined instead of the presence of bus stops. It contains a stretch of straight road and several intersections with one - three bus stops. The time period reflects off-peak early morning traffic for an area this size (4-5.00 am). This scenario shows an increase of collisions from twelve to twenty-one with eight of one being serious collisions. This follows the pattern in Scenario3A whereby the increase of bus stop density decreases the safety of cyclists on the road though not to the level of bus stop v no bus stop.

**High v Low density of stops**

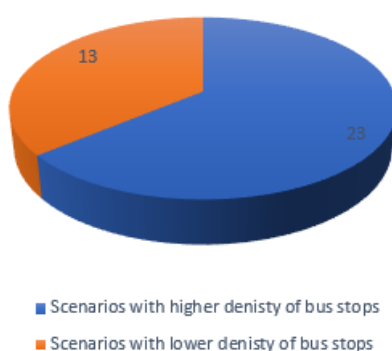


Figure 11.2

## 12 Conclusion

This report describes a method for creating a tool to investigate traffic network interactions using the SUMO software suit, Unity3D, Smart Dublin's Docklands model and a TCP scripts to connect everything together.

### 12.1 Does roadside bus stops have an impact on the safety of cyclists and if so what is this impact?

From the graphs shown in Chapter 10 and 11 it is quite clear that adding active bus stops on the roadside significantly and negatively affects the safety of cyclists as road users. Not only is there a significant increase in collisions but the rate of serious collisions rises. In simulations where bus stops are inactive there is three serious collisions whilst in simulations where bus stops are active there are thirty serious collisions which is a ninefold increase.

### 12.2 Is using a SUMO and Unity3D combination an effective tool?

Using a combination of these two development platforms proves to be quite effective. By combining them SUMO's complex traffic network platform and Unity3D's collision detection are both used which produces a superior tool. In Unity3D the GameObject's could be coded to dynamically follow a route and interact with other vehicles although this would be rudimentary and produce significant overhead costs. SUMO does have the ability to detect a collision however it is extremely limited with it only detecting major collisions and not providing critical information regarding the collision.

## 12.3 Limitations

The traffic interactions are dependant on the models used by SUMO. The bicycle model used is not as developed as other vehicle models as it does not contain the full range of actions available to real life bicycle road users. According to SUMO documentation (24) these limitations include:

1. Turning left by crossing twice does not work. Extra edges need to be added to accommodate these trajectories.
2. No bi-directional movements on bicycle lanes
3. No shared space for bicycles and pedestrians
4. No overtaking by vehicles on a single-lane road.
5. The intersection model has no special adaptations for bicycles. This results in unrealistic (large) safety gaps when bicycles are approaching a large priority intersection from a prioritized road
6. The road speed limit is not meaningful for bicycles. To model a speed distribution for bicycles with a single vehicle type, a speed limit corresponding to the median speed of the bicycles should be set for the cycling lanes.

This project addresses issue 3 by using the sublane model and issue 6 with the editing described in Chapter 5. These solutions minimise the road level problems with the bicycle model.

# 13 Future Work

## 13.1 Enlarging the area Dublin-wide

Expanding the area used for the project city wide instead of limited to the Docklands area would give a multitude of benefits. First it would allow for testing in a wider and more diverse range of situations and elements.

One such situation would be examining the effect of fully or partially indented bus stops (shown in Figure 10.1). This project has looked at roadside bus stops with no indent in the road but Dublin city (particularly the outer regions) contains bus stops that are either partially indented or fully indented.



Figure 13.1: Indented bus stop

## 13.2 Developing a more sophisticated cyclist model

One of the limitations of the SUMO suite of tools is the not fully developed bicycle model. As a road user, cyclists have the most diverse range of options when it comes to movement.

Especially in situations such as overtaking and navigating intersections. As mentioned in previous chapters TraCI allows for interfacing and controlling elements in a simulation. Writing a script that uses modelling such as one shown in the paper - "Integration of an external bicycle model in SUMO" (9) that could control bicycle movement and would produce even more realistic results. One caveat with this would be the potential overhead cost of implementing this in conjunction with the entire project especially if there are large numbers of vehicles being controlled in this manner. Any future work on this topic will have to work with this issue,

### 13.3 City planning tool

After incorporating the above two ideas the project would be extremely robust and well rounded. Expanding it with a complex User interface system would allow for use as a city planning tool by local authorities.

# Bibliography

- [1] Laura Laker. Cycling in dublin: 'we've lost our way with private cars'.  
<https://www.irishtimes.com/life-and-style/people/cycling-in-dublin-we-ve-lost-our-way-with-private-cars-1.3937975>.
- [2] Dominic McGrath.  
<https://www.thejournal.ie/cycle-protest-dublin-killed-cars-4881126-nov2019/>.  
[www.thejournal.ie](http://www.thejournal.ie).
- [3] Dublin bus stop. <http://www.dublinbusstuff.com/PhotoWeek/GTClass.html>.
- [4] Smart dublin docklands model. [https://data.smartdublin.ie/dataset/855edf52-85c5-488b-aedf-4f410ee061e2/resource/c93f26fa-a741-45e3-8a3e-9d717a70c209/download/3d\\_hack\\_u\\_nity3d\\_g\\_uide.pdf](https://data.smartdublin.ie/dataset/855edf52-85c5-488b-aedf-4f410ee061e2/resource/c93f26fa-a741-45e3-8a3e-9d717a70c209/download/3d_hack_u_nity3d_g_uide.pdf).
- [5] National Transport Authority. Canal corden report 2017.  
[https://www.nationaltransport.ie/wp-content/uploads/2018/05/Canal\\_Cordon\\_Report\\_2017.pdf](https://www.nationaltransport.ie/wp-content/uploads/2018/05/Canal_Cordon_Report_2017.pdf).
- [6] Alberto Díaz-Álvarez Carlos Biurrun-Quel Luis Serrano-Arriezu Markus Kuba nCristina Olaverri-Monreal, Javier Errea-Moreno. Connection of the sumo microscopic traffic simulator and the unity 3d game engine to evaluate v2x communication-based systems. *1 Chair for Sustainable Transport Logistics 4.0, Johannes Kepler University, 4040 Linz, Austria 2 Eurecom, Campus SophiaTech, 06410 Biot, France 3 Instituto Universitario de Investigación del Automóvil (INSIA), Universidad Politécnica de Madrid, 28031 Madrid, Spain 4 Institute of Smart Cities; Electrical, Electronic and Communication Engineering, Universidad Pública de Navarra, 31006 Pamplona, Spain 5 Applied Mathematics and Physics, University of Applied Sciences Technikum Wien, 1200 Wien, Austria.*
- [7] Smart dublin initiative. <https://smartdublin.ie/about/>.
- [8] G. Kotusevski and K.A. Hawick. A review of traffic simulation software.  
[https://www.researchgate.net/publication/228966705\\_Areview\\_of\\_traffic\\_simulation\\_software](https://www.researchgate.net/publication/228966705_Areview_of_traffic_simulation_software).



- [9] Georgios Grigoropoulosk Heather Kath. Integration of an external bicycle model in sumo. *Conference: SUMO Conference 2016 At: Berlin, Germany.*
- [10] Dublin bus timetable information.  
<https://www.dublinbus.ie/Your-Journey1/Timetables/>.
- [11] Monitoring  
road safety in the eu: towards a comprehensive set of safety performance indicators 2017.  
[https://ec.europa.eu/transport/road\\_safety/sites/roadsafety/files/pdf/ersosynthesis2017-detail-performanceindicators15\\_en.pdf](https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/pdf/ersosynthesis2017-detail-performanceindicators15_en.pdf).
- [12] Risk Solutions. Contributory factors analysis for road traffic collisions, a report for the national roads authority of ireland, november 2012. <https://www.tii.ie/tii-library/road-safety/Road%20Safety%20Research/Collision-Contributory-Factors.pdf>.
- [13] Milan Batista. On the mutual coefficient of restitution in two car collinear collisions.  
<https://arxiv.org/ftp/physics/papers/0601/0601168.pdf>, .
- [14] Milan Batista. A note on linear force model in car accident reconstruction.  
<http://xxx.arxiv.org/ftp/physics/papers/0511/0511127.pdf>, .
- [15] Sumo network file documentation.  
<https://sumo.dlr.de/docs/Networks/SUMORoadNetworks.html>, .
- [16] Osm map export tool. <https://www.openstreetmap.org/exportmap=5/51.500/-0.100>.
- [17] Netconvert documentation. <https://sumo.dlr.de/docs/NETCONVERT.html>.
- [18] Randomtrips.py documentation. <https://sumo.dlr.de/docs/Tools/Trip.html>.
- [19] Sumo documentation on traci. <https://sumo.dlr.de/docs/TraCI.html>.
- [20] Python socket documentation. <https://docs.python.org/3/library/socket.html>.
- [21] C# socket documentation. <https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.socket?view=netcore-3.1>.
- [22] Unity documentation on gameobjects.  
<https://docs.unity3d.com/ScriptReference/GameObject.html>.
- [23] Autodesk 3ds max. [https://www.autodesk.eu/products/3ds-max/overview?mktvar002=3360535/SEM/2053343431/77881279889/kwd-300333301645gclsrc=aw.dsefid=CjwKCAjwqJ1BRBZEiwAv73uwK0B88i2YxoSzx5VpLzi54dJzloP23bPGN5cuMEWjMQugeuBKXLBoCG : ss\\_k wcid =](https://www.autodesk.eu/products/3ds-max/overview?mktvar002=3360535/SEM/2053343431/77881279889/kwd-300333301645gclsrc=aw.dsefid=CjwKCAjwqJ1BRBZEiwAv73uwK0B88i2YxoSzx5VpLzi54dJzloP23bPGN5cuMEWjMQugeuBKXLBoCG : ss_k wcid =)

$AL!11172!3!359759228062!e!!g!!3ds\%20max!2053343431!77881279889mkwid =$   
 $s3Lya3b8h|pcrid|359759228062|pkw|3ds\%20max|pmt|e|pdv|c|slid||pgrid|77881279889|ptaid|kwd-$   
 $300333301645|pid|utm\_medium = cpcutm\_source = googleutm\_campaign =$   
 $GGL_3ds + Max_I E_B R_S EM\%28eStore\%29_N Vutm\_term = 3ds\%20maxutm\_content =$   
 $s3Lya3b8h|pcrid|359759228062|pkw|3ds\%20max|pmt|e|pdv|c|slid||pgrid|77881279889|ptaid|kwd-$   
 $300333301645|gclid =$   
 $CjwKCAjwqJ_1BRBZEiwAv73uwK0B88i2YxoSzx5VpLzi_54dJzloP23bPGN5cuMEWjMQugeuBKXLBoC$

[24] Sumo documentation on the bicycle model.

<https://sumo.dlr.de/docs/Simulation/Bicycles.html>, .

# A1 Appendix

## A1.1 Code

The code for this project can be accessed at <https://github.com/eb2k12/Thesis>