
Curso - Simulation of Urban MObility (SUMO)

Requisitos previos:

- Computador personal
- Sistema Operativo Linux (Ubuntu 22 -<https://ubuntu.com/download/desktop>). Puede ser nativo o máquina virtual.
- Acceso a Internet
- Conocimiento básicos de programación.
- Conocimiento de comandos básicos de Linux.

Resumen:

Este curso incluye una serie de prácticas destinadas a generar escenarios de simulación de tráfico vehicular. Cada práctica incluye pasos y recomendaciones necesarios para lograr los objetivos establecidos. Las implementaciones se basan en paquetes de código abierto (SUMO) e incluyen: (i) Generación de escenarios de micro movilidad vehicular, (ii) Generación de datos y estadísticas vehiculares, (iii) Interfaces de control de tráfico en línea, (iii) Sistemas inteligentes de transportación.

Autores

Erick Patricio Perez Peralta
erickpatriciopp9@gmail.com
<https://sites.google.com/view/erick-perez-p/>

Pablo Andrés Barbecho Bautista
pablo.barbecho@ucuenca.edu.ec
<https://www.pbarbecho.com/>

Tabla de Contenidos

1	Introducción al Simulador de movilidad Urbana (SUMO)	1
1.1	Objetivos	1
1.2	Introducción	1
1.3	Materiales	2
1.4	Instrucciones	2
1.4.1	Parte I: Instalación de SUMO	2
1.4.2	Parte II: Generación de archivos de simulación de SUMO	3
1.4.3	Parte III: Generación de una simulación usando las consolas CLI y GUI de SUMO.	9
2	Simulación de tráfico vehicular con SUMO WebWizard	11
2.1	Objetivos	11
2.2	Introducción	11
2.3	Materiales	11
2.4	Instrucciones	12
3	Estadísticas de movilidad vehicular en SUMO	15
3.1	Objetivos	15
3.2	Introducción	15
3.3	Materiales	16

3.4	Instrucciones	16
3.4.1	Parte I: Generar archivo de salida (outputs) en formato .csv	16
3.4.2	Parte II: Generar gráficas generales de visualización.	17
3.4.3	Parte III: Generar gráficas usando herramientas de visualización de SUMO.	18
3.4.4	Trayectorias:	18
3.4.5	Velocidad:	19
3.4.6	Densidad de vehículos:	20
3.4.7	Emisiones de gases:	21
4	Introducción a TraCI (Traffic Control Interface)	23
4.1	Objetivos	23
4.2	Introducción	23
4.3	Materiales	25
4.4	Instrucciones	25
4.4.1	Parte I: Requisitos iniciales de TraCI.	25
4.4.2	Parte II: Ejecutar una simulación usando la interfaz TraCI.	26
4.4.3	Parte III: Ejemplos de interacción TraCI - SUMO.	28
4.4.4	Ejemplo 1: Conteo de vehículos en línea.	29
4.4.5	Ejemplo 2: Cálculo de velocidad promedio de vehículos en línea.	29
4.4.6	Ejemplo 3: Cálculo de emisiones CO ₂ de vehículos en línea.	30
4.4.7	Ejemplo 4: Cambio de trayectoria de vehículos en línea.	32
5	Clasificación de movilidad con aprendizaje automático	34
5.1	Objetivos	34
5.2	Introducción	34
5.3	Materiales	35
5.4	Instrucciones	36
5.4.1	Aprendizaje Supervisado	36
5.4.2	Aprendizaje No Supervisado	38
6	Aprendizaje automático aplicado a ITS	40
6.1	Objetivos	40
6.2	Introducción	40
6.3	Materiales	41
6.4	Instrucciones	41
7	Aprendizaje por refuerzo (RL) aplicado a ITS.	44
7.1	Objetivos	44

7.2	Introducción	44
7.3	Materiales	46
7.4	Instrucciones	46
References		59



1. Introducción al Simulador de movilidad Urbana (SUMO)

*En esta práctica el estudiante instalará la plataforma de simulación de movilidad urbana (SUMO) y ejecutará una simulación sencilla. **Modalidad:** Trabajo Individual. **Recursos:** Computador personal con sistema operativo Linux o máquina virtual (VM). Recomendado Ubuntu 22 <https://ubuntu.com/download/desktop>.*

1.1 Objetivos

1. Utilizar los archivos binarios para instalar el simulador sobre el sistema operativo Linux (Guía realizada sobre - Ubuntu 22.04.3 LTS).
2. Conocer los archivos necesarios para correr una simulación sobre SUMO.
3. Lanzar una simulación desde la línea de comandos (CLI) y desde la interfaz gráfica (GUI).

1.2 Introducción

SUMO (Simulación de Movilidad Urbana), es un paquete de simulación de tráfico multimodal (i.e., vehículos livianos, transporte público, peatones, bicicletas, etc) continuo, microscópico (i.e., configuración por nodo). SUMO es una suite de simulación de tráfico gratuita y de código abierto que incluye diferentes herramientas para la generación de tráfico, estadísticas de movilidad y visualización. SUMO incluye modelos de movilidad realista, cambio de línea, emisiones de contaminantes, entre otros. Además, incluye la interface TraCI para control en línea de la simulación.

SUMO es ampliamente utilizado en diferentes temáticas de investigación:

- Evaluación de los ciclos semáforicos.
- Optimización de rutas.
- Reducción de emisiones contaminantes.
- Conducción autónoma.
- Predicción y mejora de tráfico vehicular.
- Evaluación de redes vehiculares (VANETs).
- Manejo cooperativo (Platooning).
- Seguridad vial y análisis de riesgos.
- Planificación de rutas de transporte público.

1.3 Materiales

- **Ordenador** con sistema operativo Linux (Ubuntu 22.04.3 LTS).

1.4 Instrucciones

Esta práctica se compone de 3 partes:

1. **Parte I:** Instalación de SUMO,
2. **Parte II:** Generación de archivos de simulación de SUMO,
3. **Parte III:** Generación de una simulación usando las consolas CLI y GUI de SUMO.

1.4.1 Parte I: Instalación de SUMO

En esta primera parte vamos a instalar la suite de SUMO. Existen varias formas para instalar SUMO, incluso sobre diferentes sistemas operativos <https://sumo.dlr.de/docs/Downloads.php>; sin embargo, en esta sección presentamos un resumen de la instalación manual de la suite de SUMO sobre Ubuntu a partir de los binarios. Una guía completa se puede encontrar en https://sumo.dlr.de/docs/Installing/Linux_Build.html.

1. Actualizar los paquetes del sistema operativo.

```
#sudo apt update
#sudo apt upgrade
```

2. Instalar prerequisitos de SUMO:

```
#sudo apt install git cmake python3 g++ libxerces-c-dev libfox-1.6-dev
libgdal-dev libproj-dev libgl2ps-dev python3-dev swig default-jdk
maven libeigen3-dev

#sudo apt install python3-pandas python3-rtree python3-pyproj
```

Descripción general de prerequisitos:

- git: permite clonar repositorios de Github
- cmake: g++ generadores y compiladores,
- libxerces: manejo de archivos xml
- libfox-1.6-dev: manejo de GUI de SUMO
- libgdal-dev: para manejo de formatos geoespaciales,
- libproj-dev: soporte para la geoconversión y referencias
- libgl2ps-dev: usado por OpenGL,
- ccache: acelerar las construcciones,
- ffmpeg-devel: para manejo de vídeo,
- libOpenSceneGraph-devel: para la GUI en 3D,
- gtest: para pruebas de unidad, no utilice 1.13 o posterior,
- gettext: manejo de idiomas (interno),
- texttest, xvfb y tkdiff: para los ensayos de aceptación,
- copos, un estilo y autopep para la comprobación de estilo,
- swig: conector de programas de bajo a alto nivel,
- python3-dev: entorno para python3,

- jdk: entorno de java,
 - maven: gestor de proyectos de software,
 - libeigen3-dev: para manejo de matrices en C.
3. Una vez instalados los prerrequisitos, es momento de obtener los binarios de SUMO. Para esto, nos dirigimos al repositorio de Github del proyecto <https://github.com/eclipse-sumo/sumo>. Se recomienda descargar SUMO dentro del directorio /opt/ de Linux, propio para instalar aplicaciones de terceros y opcionales.

```
# git clone --recursive https://github.com/eclipse-sumo/sumo
```

Note que este procedimiento descargará la versión más reciente (1.19.0).

4. Luego ha de instalar algunas librerías comunes de python (e.g., pandas). SUMO incluye un archivo de estos requerimientos y es necesario estar ubicados dentro de la carpeta **sumo** descargado en el paso anterior, con el fin de llamar al archivo **requirements.txt**:

```
pip install -r tools/requirements.txt
```

5. Antes de compilar el proyecto ha de definir la variable de entorno permanente **SUMO_HOME**. Este paso es muy importante ya que durante el curso se requiere que el sistema operativo conozca donde se encuentran los binarios del programa. Puede usar el comando de Linux **pwd** para ver el path completo donde se encuentra SUMO.

Es necesario modificar los archivos **/etc/bash.bashrc** y **/home/<user>/.bashrc**. Ha de agregar el path al SUMO al final de los archivos mencionados:

```
export SUMO_HOME="/home/user/sumo-version/"
```

Puede verificar que la variable de entorno se encuentre correctamente configurada con:

```
echo $SUMO_HOME
```

6. Finalmente, ha de crear una carpeta **build** dentro la carpeta raíz SUMO. Se ubica dentro de la carpeta **build** y ejecuta uno a uno los siguientes comando que construyen los binarios de SUMO:

```
cmake -B build .  
cmake --build build -j $(nproc)
```

El comando **nproc** es el número de procesadores de su PC. Este proceso puede tomar varios minutos.

7. Puede verificar la instalación abriendo una terminal de Linux y ejecutando **sumo** que mostrará la versión instalada.

1.4.2 Parte II: Generación de archivos de simulación de SUMO

Una vez instalado SUMO, el primer paso es generar el escenario de simulación. Para construir el escenario, usamos las herramientas pre-instaladas en SUMO. Podemos ver una clasificación éstas herramientas en azul en la Fig. 1.1. Además, la Fig. 1.1 muestra los archivos de entrada y salida de cada herramienta, en verde.

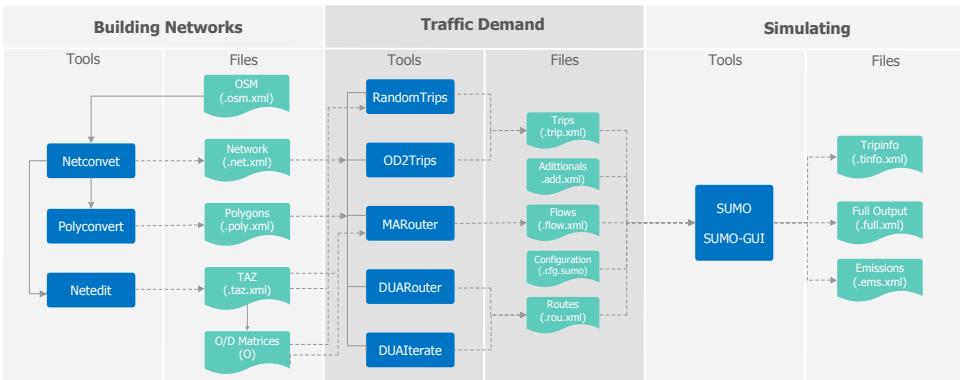


Figure 1.1: Flujo de trabajo de simulación de movilidad de tráfico. Herramientas integradas SUMO y archivos relacionados para realizar simulaciones. Tomado de [1].

Tenemos 3 grupos principales de herramientas, según sus funciones (ver Building Networks, Traffic Demand y, Simulating en la Fig. 1.1):

1. **El primer grupo de herramientas incluye:** red de carreteras, elementos de movilidad (ej., semáforos) y polígonos como edificios, parques etc. Los polígonos son de especial interés cuando evaluamos redes inalámbricas (modelos de obstrucción). En la Fig. 1.1, a la izquierda tenemos las diferentes herramientas de SUMO para generar la red de carreteras (**Netconvert**), los polígonos (**Polyconvert**). Si deseamos modificar gráficamente el mapa, lógica de semáforos, o crear zonas de interés, podemos usar un editor gráfico llamado **Netedit**.
2. **El segundo grupo de herramientas incluye:** generación de la demanda de tráfico. Aquí se definen las rutas de los vehículos, cantidad de vehículos, tipo de vehículos (transporte público, pesados, livianos, bicicletas, peatones). En la Fig. 1.1, en el centro tenemos las herramientas para generación de tráfico.
3. **El tercer grupo de herramientas incluye:** interfaces de simulación. Tenemos dos interfaces: línea de comandos (CLI), gráfica (GUI).

A continuación introducimos algunas de las herramientas que utilizaremos durante la práctica, según la clasificación de la Fig. 1.1:

- **Building Networks:**

- **Netconvert:** Esta herramienta convierte un archivo de mapa de entrada OpenStreetMaps (.osm) en un archivo de red de carreteras legible por SUMO. Este archivo se guarda en ASCII con un formato XML simple conocido como archivo de red (.net.xml). La herramienta netconvert proporciona un conjunto de opciones de procesamiento (por ejemplo, vías cortadas, pasos a desnivel, rotundas, etc.). Este archivo es obligatorio.
- **Polyconvert:** Esta herramienta genera todos los polígonos (por ejemplo, edificios, terrenos, etc.) a partir de la fuente del mapa ingresada (.osm). El archivo de salida (.poly.xml) contiene todas las formas geométricas del mapa. Este archivo

es opcional.

- *Netedit*: Esta herramienta permite a los usuarios editar/crear mapas personalizados. Viene con una interfaz gráfica de usuario donde los usuarios pueden editar las propiedades de los elementos de un mapa, como carreteras, semáforos, etc.

- **Herramientas de generación de movilidad de tráfico:**

- *RandomTrips*: Está destinado a implementaciones rápidas. Aquí, los puntos de origen/destino se seleccionan aleatoriamente en el mapa y los vehículos se distribuyen uniformemente dentro de un período de tiempo.
- *MARouter*: Generar una demanda de tráfico macroscópica. Durante el proceso de generación de tráfico se considera la distribución de la ruta, es decir, cada ruta incluye la probabilidad de ser seleccionada.
- *DUARouter*: Genera una lista de rutas que incluye la ruta completa entre los puntos de origen y destino. Utiliza el algoritmo de Dijkstra para calcular la ruta más corta. Además, el método de asignación de tráfico considera una red vacía cada vez que se genera un vehículo, lo cuál puede decantar en congestión durante la simulación (deadlock).
- *DUAIterate*: Esta herramienta utiliza un método de asignación llamado iterativo, que intenta calcular el equilibrio. Esto significa que las rutas de los vehículos se generan con el mínimo coste (por ejemplo, tiempo de viaje). Esto se hace llamando iterativamente a la herramienta *DUARouter*. Las situaciones de congestión son menos probables.
- *OD2Trips*: Genera una lista de definiciones de viaje según la cantidad de vehículos a insertar en la simulación dentro del intervalo de tiempo codificado en un archivo adicional llamado O/D matrices. Al igual que la herramienta *RandomTrips*, utiliza un método de asignación incremental.

En la Fig. 1.2, se presenta un resumen de los archivos de entrada y salida esperados en cada herramienta de generación de tráfico. En este punto, se recomienda revisar el artículo [2] que amplía los modelos y diferencias entre las diferentes herramientas de generación de tráfico como Duarouter, Randomtrips, MARouter, OD2Trips, etc.

Tool	Inputs	Outputs	Traffic assignment method
OD2Trips (OD2)	Network, O/D Matrices, Additionals	Trips	Incremental
DUARouter (DUAR)	Network, Trips, Additionals	Routes	Empty-network
MARouter (MAR)	Network, O/D Matrices, Additionals	Flows	Incremental
RandomTrips (RT)	Network, Additionals	Trips	Incremental
DUAIterate (DUA)	Network, Trips, Additionals	Routes	Iterative

Figure 1.2: Herramientas de generación de tráfico de SUMO. Tomada de [2]

Finalmente, un escenario de simulación en SUMO se compone por los menos de tres archivos (ver Fig. 1.3). La extensión de los archivo es .xml. Note que el archivo *.poly* es opcional.

Veamos una descripción de los archivos presentados en la Fig. 1.3:

- **Archivo de red (.net)**: Este archivo es generado con la herramienta *Netconvert* y

contiene toda la información acerca del mapa, carreteras y calles.

- **Archivo de polígonos (.poly):** Este archivo es generado con la herramienta *Polyconvert* contiene toda la información acerca de edificios y otros obstáculos como parques que existen en el mapa.
- **Archivo de viajes (.rou):** Este archivo contiene toda la información referente a las rutas de vehículos dentro de la simulación. La manera más sencilla para generar este archivo es usando la herramienta *RandomTrips*.
- **Archivo de Configuración (.cfg):** Los archivos antes mencionados están organizados en un archivo de configuración único donde se indica el tiempo de simulación, los directorios de ubicación de los archivos .net y .rou entre otros.

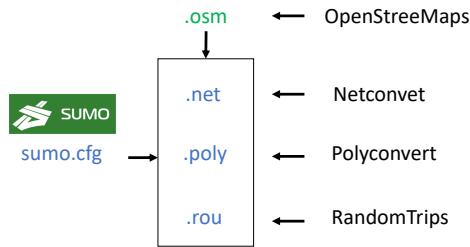


Figure 1.3: Archivos de simulación de SUMO.

A continuación se describen los pasos para generar los archivos necesarios (ver Fig. 1.3) para correr una simulación:

1. Antes de comenzar a generar los archivos, se sugiere crear una carpeta por proyecto, donde se almacenarán todos los archivos de simulación.
2. El primer archivo a generar es el mapa (ver .osm en Fig.1.3). SUMO soporta diferentes formatos, vamos a exportar un mapa descargado de OpenStreetMaps (OSM), <https://www.openstreetmap.org/>. Seleccione una sección pequeña de su ciudad y haga clic en "Exportar" y luego seleccione la opción "Seleccionar manualmente un área diferente", ver Fig. 1.4.



Figure 1.4: Seleccionar área y exportar mapa en formato .osm.

Nota: A través de este método (OSM), solo se permite exportar un número máximo de

50000 nodos, incluyendo carreteras, señales de tráfico, rutas de transporte público, etc.

3. Vamos a generar el siguiente archivo que corresponde a la red de carreteras (.net) (ver Fig. 1.3). Para esto, usamos la herramienta *netconvert*:

```
netconvert --ramps.guess --remove-edges.isolated --edges.join --  
geometry.remove --osm-files map.osm -o bcn.net.xml
```

Donde:

- `--ramps.guess` permite predecir rampas que no se lograron especificar en el mapa.
- `--remove-edges.isolated` elimina los bordes asilados del mapa.
- `--edges.join` fusiona nodos que están cercanos entre sí.
- `--geometry.remove` reemplaza los nodos que solo definen la geometría del borde por puntos de geometría.
- `--osm-files` indica la ruta y el nombre al archivo `.osm` (descargado desde OSM).
- `-o` indica la ruta y el nombre del archivo de salida.

La herramienta tiene varias opciones que se pueden agregar, por ejemplo, excluir rutas de tranvía. Para conocer las opciones nos referimos a la documentación [3]. La entrada de *netconvert* es el archivo `.osm` previamente descargado y la salida es el archivo de la red (.net), por ejemplo `bcn.net.xml`, donde `bcn` es un nombre cualquiera que le damos al mapa y se mantiene las extensiones `.net.xml`.

4. Ahora generamos el archivo de los polígonos (ver Fig. 1.3). Invocamos la herramienta *polyconvert*:

```
polyconvert --net-file bcn.net.xml --osm-files map.osm -o bcn.poly.xml
```

Donde:

- `--net-file` indica la ruta y el nombre del archivo de la red (net).
- `--osm-files` indica la ruta y el nombre al archivo `.osm` (descargado desde OSM).
- `-o` indica la ruta del archivo de polígonos (edificios, parques, etc.) de salida con extensión `.poly.xml`.

Otras opciones de *polyconvert* se encuentran en [4].

5. Ahora generamos las rutas (.rou.xml). Usamos la herramienta *RandomTrips* que hace referencia a un script en Python. ***RandomTrips*** está ubicado en el directorio: `/opt/sumo/tools/`. Corremos el script de Python `randomTrips.py`.

```
python3 randomTrips.py -n bcn.net.xml -r bcn.rou.xml
```

Donde:

- `-r` genera un archivo de rutas con DUAROUTER. Archivo de rutas de salida con extensión `.rou.xml`.
- `-n` indica la ruta y el nombre del archivo de la red (.net).

Una opción de utilidad es `-p`. Esta opción permite generar tráfico en el que consten n vehículos distribuidos uniformemente en un período de tiempo. `-p` se obtiene de:

$$p = \frac{t_1 - t_0}{n} \quad (1.1)$$

Donde, t_1 y t_0 son el tiempo final e inicial de la simulación, respectivamente. Por ejemplo, 3600 segundos, comenzando en el segundo 0. Reemplazando en la ecuación 1.1 se obtiene el valor de $p=36$.

$$p = \frac{3600}{100} = 36 \quad (1.2)$$

Finalmente, el comando sería el siguiente:

```
python3 randomTrips.py -b 0 -e 3600 -p 36 -n bcn.net.xml -r bcn.rou.xml
```

Otras opciones de la herramienta *RandomTrips* están disponibles en [5].

6. Finalmente, necesitamos el archivo de configuración de sumo (ver Fig. 1.3). Este archivo contiene a todos los archivos necesarios para la simulación (.net, .poly y .rou).

El archivo `sumo.cfg` (ver Código 1.1) tiene un formato .xml e incluye directivas del escenario de simulación. La configuración de este archivo se divide en bloques. En el Código 1.1 identificamos el bloque `<input>`, `<processing>`, y `<time>`.

La sección de entrada `<input>` establece la ubicación de los archivos de la red, rutas de vehículos y polígonos. La sección de procesamiento `<processing>` incluye la directiva `<ignore-route-errors>`. Esto permite ignorar los errores que pueden generarse por rutas que no se pueden completar (no hay conexión entre el punto origen y destino). Finalmente, la sección de tiempo establece el tiempo de inicio, finalización y la duración de los pasos de tiempo de la simulación.

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
    <net-file value="bcn.net.xml"/>
    <route-files value="bcn.rou.xml"/>
    <additional-files value="bcn.poly.xml"/>
</input>

<processing>
    <ignore-route-errors value="true"/>
</processing>

<time>
    <begin value="0"/>
    <end value="10000"/>
    <step-length value="1"/>
</time>
</configuration >
```

Listing 1.1: Archivo `sumo.cfg`.

1.4.3 Parte III: Generación de una simulación usando las consolas CLI y GUI de SUMO.

A continuación, se describen los pasos necesarios para correr una simulación usando la CLI y GUI de SUMO.

1. **Simulación usando la CLI:** necesitamos indicar el archivo de configuración con extensión `sumo.cfg`.

```
#sumo -c bcn.sumo.cfg
```

Donde, `-c` indica la ubicación y el nombre del archivo de configuración `xx.sumo.cfg`.

```
sumo@sumo-VirtualBox:~/sumo/bin$ ./sumo -c cuenca.sumo.cfg
Step #0.00 (12ms ~= 83.33*RT, -83.33UPS, vehicles TOT 1 ACT 1 BUF 0)
Warning: No connection between edge '118366336#2' and edge '45420539#0' found.
Warning: No route for vehicle '8' found.
Warning: No connection between edge '48202935' and edge '334948974#6' found.
Warning: No route for vehicle '23' found.
Warning: No connection between edge '389769675#0' and edge '256907298#0' found.
Warning: No route for vehicle '12' found.
Warning: No connection between edge '408449634#1' and edge '334948974#7' found.
Warning: No route for vehicle '95' found.
Step #100.00 (2ms ~= 500.00*RT, -49500.00UPS, vehicles TOT 100 ACT 99 BUF 1)
Warning: No connection between edge '495093213#0' and edge '48859314#3' found.
Warning: No route for vehicle '132' found.
Warning: No connection between edge '408463706#1' and edge '334948974#2' found.
Warning: No route for vehicle '16' found.
Warning: No connection between edge '4950933657#1' and edge '311212627#3' found.
Warning: No route for vehicle '178' found.
Warning: No connection between edge '45420545#2' and edge '42201281#1' found.
Warning: No route for vehicle '181' found.
Warning: No connection between edge '48537834' and edge '408713354' found.
Warning: No route for vehicle '199' found.
Step #200.00 (3ms ~= 333.33*RT, -62333.33UPS, vehicles TOT 200 ACT 187 BUF 1)
Warning: No connection between edge '49577750' and edge '-97264414#4' found.
```

Figure 1.5: Simulacion en marcha desde la línea de comandos de Linux.

Como vemos en la Fig. 1.5, se presentan ciertas advertencia refiriéndose a rutas sin conexión. La **CLI** suele utilizarse para ejecutar simulaciones largas y pesadas.

2. **Simulación usando la GUI:** Para ejecutar la simulación con interfaz gráfica (GUI), abrimos una terminal de Linux y ejecutamos `sumo-gui`.
3. En el menú superior de la GUI de SUMO (ver Fig. 1.6), seleccionamos *File → Open Simulation* y buscamos el archivo de configuración del escenario `xx.sumo.cfg`.

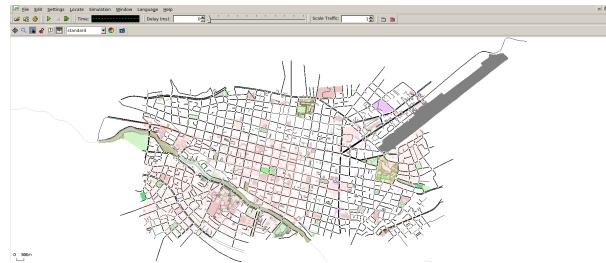


Figure 1.6: Archivo de configuración cargado.

En la parte inferior de la GUI se muestran posibles errores al intentar cargar los archivos `.net` y `.poly` definidos en el archivo de configuración `.cfg`.

4. Finalmente, para ejecutar la simulación consideraremos la velocidad de simulación en la parte superior (recuadro rojo de la Fig. 1.7). Luego, es necesario dar clic en la opción *Run* (recuadro azul de la Fig. 1.7) y la simulación estará en marcha (ver Fig. 1.8).

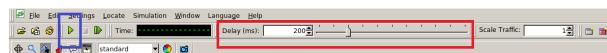


Figure 1.7: Controles de la velocidad de simulación.

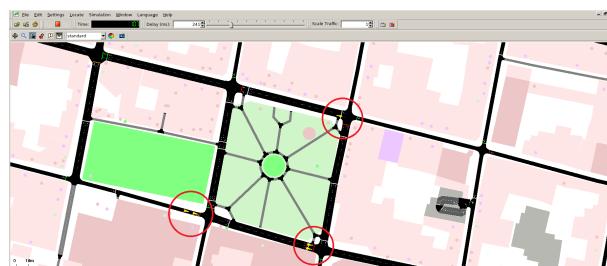


Figure 1.8: Simulación en marcha desde la interfaz gráfica de SUMO.



2. Simulación de tráfico vehicular con SUMO WebWizard

En esta práctica el estudiante generará los archivos de simulación de SUMO utilizando la herramienta OSMWebWizard. Esta herramienta, incluida en la suite de SUMO permite generar simulaciones de manera muy sencilla, automatizando la generación de archivos de simulación. **Modalidad:** Trabajo Individual. **Recursos:** Computador personal con sistema operativo Linux y SUMO.

2.1 Objetivos

- Generar una simulación de SUMO usando la herramienta WebWizard.
- Comprender las ventajas y desventajas de usar OSMWebWizard.

2.2 Introducción

Dentro de la suite de herramientas de SUMO, una herramienta particularmente útil es OSMWebWizard. Este componente permite a los usuarios aprovechar los datos geoespaciales disponibles en OpenStreetMap (OSM) para generar escenarios de simulación realistas y detallados y sin limitación del número de nodos.

OSMWebWizard simplifica significativamente el proceso de generación de archivos de simulación de SUMO. Se basa en una interfaz gráfica Web (se recomienda usar Firefox) que guía a los usuarios a través de los pasos necesarios para importar mapas, definir atributos de la red vial, y configurar parámetros de simulación.

Un inconveniente de la herramienta, es que no permite modificaciones directas en la generación de archivos. Por ejemplo, *netconvert* permite agregar opciones para mejorar la calidad de la red de carreteras (.net) que no se pueden configurar con OSMWebWizard.

En esta práctica, exploraremos cómo utilizar el OSMWebWizard para crear escenarios de simulación en SUMO.

2.3 Materiales

- Ordenador con sistema operativo Linux (Ubuntu 22.04.3 LTS).
- SUMO instalado.

2.4 Instrucciones

A continuación se describen los pasos para ejecutar la herramienta OSMWebWizard de SUMO.

1. La herramienta se localiza dentro del directorio **tools** de SUMO en la ruta: `/opt/sumo/tools`. OSMWebWizard es un script en Python `osmWebWizard.py` y para ejecutarlo escribimos en la terminal:

```
# ./ osmWebWizard . py
```

Se abrirá un navegador web con el mapa de OSM (ver Fig. 2.1). Se puede modificar la ubicación (e.j., Quito). Podemos seleccionar la opción *Seleccionar Área* o simplemente el mapa en pantalla será el mapa que se exporte. De igual manera, podemos marcar la opción *Add Polygons* si necesitamos generar el archivo de polígonos (opcional). En el icono del carro, nos aparece un segundo menú que nos permite configurar el número de vehículos en la simulación, así como otro tipo de vehículos (ver Fig. 2.2).

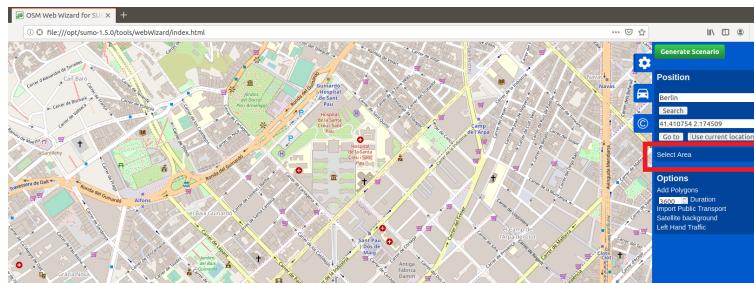


Figure 2.1: osmWebWizard tool.

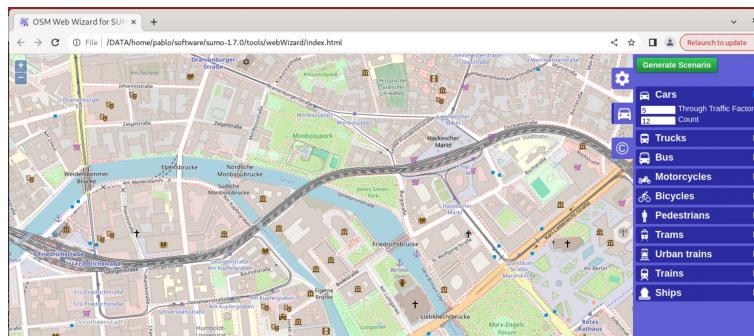


Figure 2.2: osmWebWizard tool.

2. Una vez presionemos en *Generate Scenario* (botón verde en Fig. 2.1), podemos mirar en consola los comandos que se ejecutan automáticamente. Finalizado el proceso, se nos presenta un resumen en consola y se agrega una carpeta dentro del directorio `/opt/sumo/tools/` cuyo nombre corresponde a la fecha en la que se generaron los

archivos. Además, se ejecutará automáticamente la simulación en la GUI de SUMO.

En este punto es importante identificar los archivos generados dentro de la carpeta. En la Fig. 2.3 se listan los archivos generados de manera automática.

```
-rw-rw-r-- 1 pablo pablo    354 Mar  5 03:35 build.bat
-rw-rw-r-- 1 pablo pablo 7354804 Mar  5 03:34 osm.net.xml
-rw-rw-r-- 1 pablo pablo   1212 Mar  5 03:34 osm.netccfg
-rw-rw-r-- 1 pablo pablo  59092 Mar  5 03:35 osm.passenger.trips.xml
-rw-rw-r-- 1 pablo pablo 1416755 Mar  5 03:34 osm.poly.xml
-rw-rw-r-- 1 pablo pablo    691 Mar  5 03:34 osm.polycfg
-rw-rw-r-- 1 pablo pablo   851 Mar  5 03:35 osm.sumocfg
-rw-rw-r-- 1 pablo pablo     88 Mar  5 03:35 osm.view.xml
-rw-rw-r-- 1 pablo pablo 13234119 Mar  5 03:34 osm_bbox.osm.xml
-rwxr-xr-x 1 pablo pablo     23 Mar  5 03:35 run.bat
```

Figure 2.3: Archivos SUMO generados.

A continuación se presenta una descripción de cada archivo:

- `build.bat` es un bash script que contiene el comando para correr la herramienta `randomTrips.py` indicando cada una de las opciones agregadas.
- `osm.netccfg` es el archivo de configuración usado para el netconvert. Construye el archivo final `.net`.
- `osm.net.xml` es el archivo de carreteras final `.net`.
- `osm.poly.xml` es el archivo de configuración final de polígonos `.poly`.
- `osm_bbox.osm.xml` es el archivo del mapa en formato `.osm`.
- `osm.passenger.trips.xml` contiene las rutas de los vehículos (pasajeros).
- `osm.polycfg` contiene la información del archivo usado para generar los polígonos.
- `osm.sumocfg` es el archivo de configuración de la simulación de sumo.
- `osm.view.xml` contiene la configuración de la ejecución en el GUI de SUMO.
- `run.bat` es un script que contiene el comando para correr la simulación de sumo desde la interfaz gráfica.

Los archivos marcados en azul son los que nos interesa para reproducir la simulación. Podemos copiarlos a otro directorio y modificarlos o usarlos para ejecutar una nueva simulación.

Recordando de la práctica anterior, en el archivo `sumo.cfg`, teníamos como entrada `<input>` los archivos: `.net`, `.rou` y `.poly`. En este caso tenemos los archivos `.net`, `trips` y `.poly`. Se observa que ahora se utiliza el archivo `.trips` es cual reemplaza al archivo `.rou`, más adelante explicaremos la diferencia entre `.trips` y `.rou`.

3. Genere nuevamente la simulación usando los archivos generados con `osmWebWizard`.
-

NOTA: Si al correr el script `osmWebWizard.py` salta el siguiente error:

```
osmWebWizard.py: error: typemap file "\${SUMO\_HOME}/data/typemap/
osmPolyconvert.typ.xml" not found
```

Es necesario verificar que la variable de entorno se encuentre configurada correctamente `echo $SUMO_HOME`, y verificar la existencia del archivo `osmPolyconvert.typ.xml` en el path especificado en el error.



3. Estadísticas de movilidad vehicular en SUMO

En esta práctica el estudiante explorará como extraer estadísticas de las simulaciones en SUMO

Modalidad: Trabajo Individual. **Recursos:** Computador personal con sistema operativo Linux y la SUMO pre-instalado.

3.1 Objetivos

- Generar archivos de salida (outputs) de la simulación en SUMO.
- Entender el contenido de los archivos de salida de las simulaciones.
- Analizar los datos de las simulaciones con scripts en Python.

3.2 Introducción

SUMO permite generar una gran cantidad de estadísticas, basada en los datos de vehículos, simulación, carreteras, intersecciones, semáforos, entre otros. Los archivos de salida escritos por SUMO están en formato XML. SUMO incluye la herramienta `xml2csv.py` que permite convertir los outputs a un formato de archivo plano (csv) que se puede procesar con la mayoría de programas para análisis de datos (R, Python, Excel, etc).

En esta práctica nos enfocamos en 3 ejemplos (outputs) que capturan datos de la red vehicular:

1. ***tripinfo***: Esta salida contiene la información sobre el tiempo de viaje del vehículo, distancia recorrida, numero de cambios de rutas entre otros datos. Nota: La información se genera para cada vehículo tan pronto como el vehículo llega a su destino y se retira de la red. En el caso que el vehículo no llegue a su destino durante el tiempo de simulación, este vehículo no se considera en la estadística <https://sumo.dlr.de/docs/Simulation/Output/TripInfo.html>.
2. ***fcd***: La exportación FCD (datos de automóviles flotantes) contiene la ubicación, la velocidad, y tipo de vehículo para cada vehículo en la red. Esta información se captura para paso de la simulación (cada 1s por defecto). Esta captura se comporta como un dispositivo GPS de alta frecuencia súper preciso para cada vehículo. Los resultados se pueden procesar aún más utilizando la herramienta TraceExporter para adaptar la frecuencia, las velocidades del equipo, la precisión y el formato de los datos <https://sumo.dlr.de/docs/Simulation/Output/FCDOutput.html>.

3. **trajectories**: Los datos de trayectoria incluyen el nombre, la posición, la velocidad y la aceleración de cada vehículo <https://sumo.dlr.de/docs/Simulation/Output/AmitranOutput.html>.

3.3 Materiales

- **Ordenador** con sistema operativo Linux (Ubuntu 22.04.3 LTS).
- **SUMO** instalado.

3.4 Instrucciones

Dividimos la práctica en tres partes:

1. **Parte I**: Generar archivo de salida (outputs) en formato .csv
2. **Parte II**: Generar gráficas generales de visualización.
3. **Parte III**: Generar gráficas usando herramientas de visualización de SUMO.

3.4.1 Parte I: Generar archivo de salida (outputs) en formato .csv

A continuación se describen los pasos para configurar los archivos de salida en una simulación.

1. El primer paso es generar los archivos necesarios para la simulación. Esto se logra de manera manual con las herramientas de SUMO (Práctica 3), o utilizando la herramienta *osmWebWizard* (Práctica 4).
2. Luego, nos enfocamos en modificar el archivo de configuración del escenario *sumo.cfg* o *.sumocfg*. En este archivo XML, es necesario agregar un nuevo bloque con la etiqueta *<outputs>*. En este bloque agregamos las salidas que se requieren capturar, como sigue:

```
<output>
    <tripinfo-output value="tripinfo.xml"/>
    <fcd-output value="fcd.xml">
        <amitran-output value="trajectories.xml">
    </output>
```

Dentro del bloque *<outputs>* se indica tres salidas: *tripinfo*, *fcd* y *amitran* outputs. Además, se especifica el nombre del archivo de salida y la ruta donde se guardara el archivo de salida. Cabe anotar que la documentación no presenta la sintaxis de los tópicos de salida. Para conocer la sintaxis, podemos verificar las posibles opciones dentro del tópico *<outputTopicType>* localizado en el archivo */opt/sumo/data/xsd/sumoConfiguration.xsd*.

3. Una vez agregados los outputs, podemos ejecutar la simulación mediante la CLI o GUI de SUMO. Por ejemplo mediante la CLI, abrimos una terminal y ejecutamos:

```
sumo -c osm.sumocfg
```

4. Una vez finalizada la simulación, de acuerdo a las rutas especificadas para los archivos de salida, podemos encontrarlos en dicha ubicación. Por ejemplo, la Fig. 3.1 se muestra el archivo *tripinfo.xml* generado una vez finaliza la simulación. Dado que este

archivo mantiene un formato xml, en el siguiente paso se convierte a .csv para facilitar el manejo de los datos.

5. Para convertir archivos xml en archivos csv, localizamos la herramienta `xml2csv.py`. Esta se ubica en la ruta `/opt/sumo/tools/xml/`. Para ejecutar el script `xml2csv.py`, hemos de indicar el archivo de entrada con la opción `-s`, por ejemplo el archivo `tripinfo.csv`, y la opción `-o` nos permite indicar el archivo de salida (`xxxx.csv`).

```
python3 /opt/sumo/tools/xml/xm1csv.py -s tripinfo.xml, -o tripinfo.csv
```

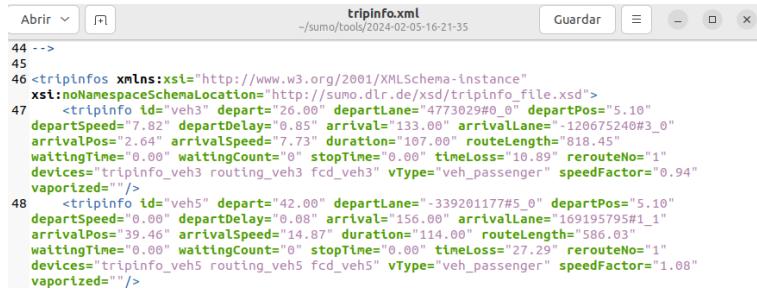


Figure 3.1: Archivos `tripinfo.xml`.

3.4.2 Parte II: Generar gráficas generales de visualización.

En este punto, con el archivo .csv se pueden hacer un análisis de los datos o generar gráficas de las estadísticas que se requiera.

Por último, presentamos un script de ejemplo en Python que permite graficar los datos presentes en el archivo tripinfo.csv. Mediante línea de comandos el usuario deba ingresar el path al archivo .csv más las columnas que desea graficar. El nombre de la columna seleccionada debe coincidir con el nombre de la columna en el archivo .csv.

El código ejemplo está disponible de manera pública en el Github <https://github.com/erickperez9/ManualSUMO.git> y fue realizado por el estudiante. Erick Perez. El código se encuentra dentro en ManualSUMO/Scripts plots /plots-v1-4.py.

Por ejemplo, para graficar la columna tripinfo_routeLength, que representan la distancia recorrida, ejecutamos:

```
python3 plots-v1-4.py -i tripinfo.csv --columns tripinfo_routeLength
```

O si deseamos incluir más columnas en la gráfica podemos agregar `emissions_CO2_abs` y `tripinfo_duration`, siendo las emisiones de CO2 y la duración de cada viaje respectivamente.

```
python3 plots-v1-4.py -i tripinfo.csv --columnas tripinfo_routeLength ,  
emissions_CO2_abs ,tripinfo_duration
```

La Fig. 3.2 presenta un ejemplo del plot generado por el script `plots-v1-4.py`. Las Fig. 3.2 ilustra los resultados del último comando.

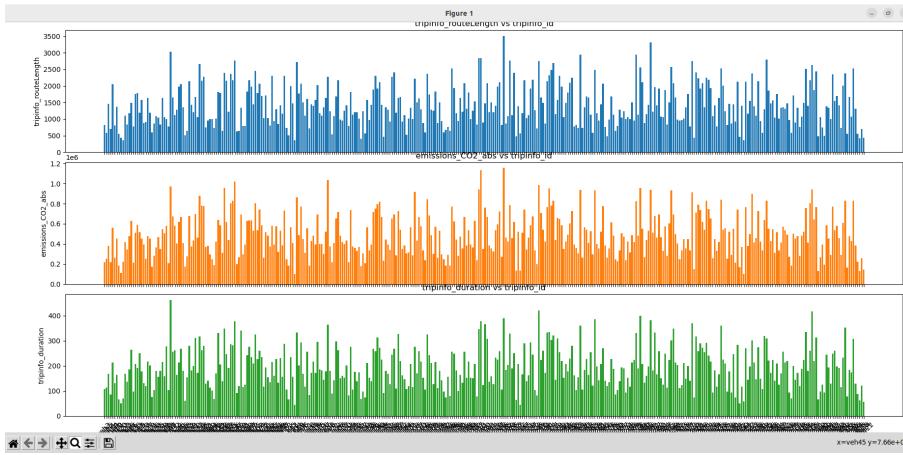


Figure 3.2: Gráficas de: tripinfo_routeLength, emissions_CO2_abs y tripinfo_duration vs tripinfo_id.

3.4.3 Parte III: Generar gráficas usando herramientas de visualización de SUMO.

1. Adicional a los ejemplos de gráficas personalizadas de la Parte II, SUMO incluye diferentes herramientas de visualización de los datos <https://sumo.dlr.de/docs/Tools/Visualization.html>. En esta parte final de la práctica usamos las herramientas de visualización para generar plots para una inspección inicial de la simulación:

- Trayectorias de vehículos.
- Velocidad de vehículos.
- Densidad de vehículos.
- Emisiones de vehículos.

3.4.4 Trayectorias:

Para graficar las trayectorias de los vehículos de la simulación en SUMO, se puede utilizar dos scripts. El primero es `plotXMLAttributes.py` que está ubicado en la ruta `tools/visualization/`. Para correr este script se requiere como entrada el archivo de salida `fcd_output` generado en la Parte I de la práctica. Ejecutamos:

```
python3 tools/visualization/plotXMLAttributes.py -x x -y y -s 1 -o
allXY_output.png fcd.xml --scatterplot
```

Donde `-x` es el atributo del eje x; `-y` es el atributo del eje y; `-s` es el tamaño; `-o` es el nombre del archivo de salida; `--scatterplot` genera un diagrama de dispersión en lugar de un diagrama de líneas.

La segunda forma para graficar trayectorias es utilizar el script `plot_trajectories.py` que está ubicado en la ruta `tools/`. Para correr este script se requiere como entrada el archivo `fcd_output`. Ejecutamos:

3.4 Instrucciones

```
python3 tools/plot_trajectories.py -t xy -o allLocations_output.png  
fcd.xml
```

Donde **-t** es el tipo de trayectoria antes mencionado; **-o** es el nombre del archivo de salida. Además, es posible seleccionar y graficar la trayectoria de uno o más vehículos. Para esto se requiere de aplicar un filtro al script `plot_trajectories.py`, con el parámetro **-filter-ids** el cual indica el ID del vehículo seleccionado. Si se requiere filtrar un vehículo adicional, basta con colocar el ID del otro vehículo seguido de una coma.

La Fig. 3.3 muestra un ejemplo de la gráfica de trayectorias de vehículos. Cada trayectoria se muestra en un color diferente. Vemos que las trayectorias describen la red de carreteras urbanas.

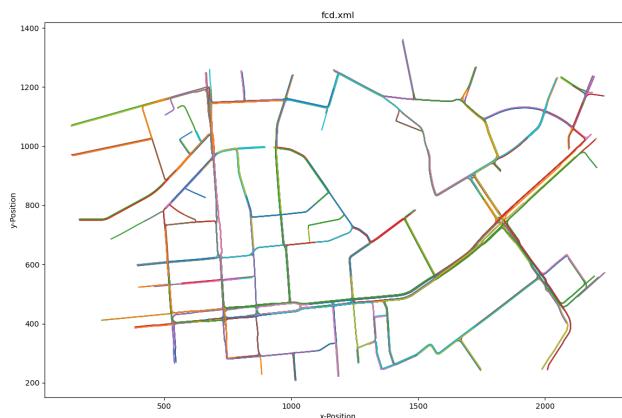


Figure 3.3: Trayectoria de vehículos en la simulación. Gráfica generada con herramientas de SUMO.

3.4.5 Velocidad:

La gráfica de velocidades de vehículos nos permite comparar la velocidad de los vehículos durante la simulación. Usamos el script `plot_trajectories.py` ubicado en `/tools`. El script requiere como entrada el archivo `fcd.xml`. A continuación se muestra un ejemplo de uso:

```
python tools/plot_trajectories.py -t ts -o timeSpeed_output.png fcd.  
xml --filter-ids veh1, veh40
```

En la Figura 3.4 se muestra las velocidades de los vehículo con IDs *1* y *40*. Se observa que el vehículo *1* comienza su viaje en el segundo 0 y termina en el segundo 220, con sus variaciones respectivas de velocidad en *m/s*. Mientras que el vehículo *40*, comienza su recorrido en el segundo 320 y termina en el segundo 500.

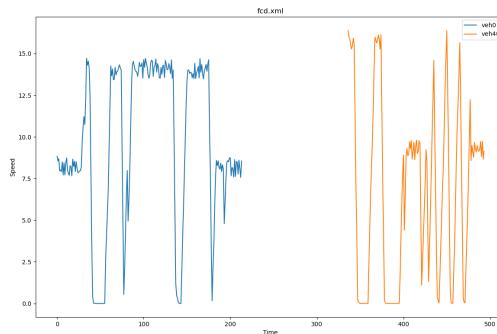


Figure 3.4: Comparación de velocidades de los vehículos *veh1* y *veh40*.

3.4.6 Densidad de vehículos:

Otra de las herramientas que incluye SUMO es `plot_net_dump.py` ubicado en `tools/visualization/`. Este script muestra una red, donde los colores y el ancho de los bordes de la red se establecen en función de los datos en las carreteras.

Las medidas de densidad se localizan en el archivo `edge.data.xml` (https://sumo.dlr.de/docs/Simulation/Output/Lane-_or_Edge-based_Traffic_Measures.html). Este archivo contiene datos de cada carretera como la ocupación o densidad (veh/km) en cada instante de la simulación.

Note que no hemos generado este output anteriormente. Bastaría con agregar el tópico respectivo (`edgedata-output`) en los outputs del archivo `sumo.cfg`

```
<output>
    <emission-output value="co2.xml"/>
    <tripinfo-output value="tripinfo.xml"/>
    <fcd-output value="fcd.xml"/>
    <edgedata-output value="edgedata.xml"/>
</output>
```

El archivo resultante `edgedata.xml` almacena entre otros datos la densidad de vehículos en cada carretera *veh/km*. A continuación se muestra un ejemplo de uso de esta herramienta para graficar la densidad de los vehículos en cada carretera:

```
python3 /home/sumo/sumo/tools/visualization/plot_net_dump.py -v -n osm
.net.xml --measures density,density --xlabel [m] --ylabel [m] --
default-width 1 -i edgedata.xml,edgedata.xml --xlim 000,2600 --
ylim 000,1500 --default-width .5 --min-color-value 0 --max-color-
value 5 --max-width-value 5 --min-width-value 0 --max-width 3 --
min-width .5 --colormap winter -o density-color-map.png
```

Donde:

- `-v` (verbose) imprime el progreso en pantalla.
- `-n` indica el archivo de entrada de red, es decir, el archivo `.net`.
- `-xlabel [m]` indica la etiqueta del eje x.
- `-ylabel [m]` indica la etiqueta del eje y.

- `-default-width` indica el valor del ancho de las líneas de las carreteras.
- `-i` indica los archivos de entrada donde se encuentra la medida a graficar.
- `-xlim` indica el límite del mapa en metros del eje x.
- `-ylim` indica el límite del mapa en metros del eje y.
- `-min-color-value` define el valor mínimo de color del borde
- `-max-color-value` define el valor máximo de color del borde
- `-min-width-value` define el valor del ancho mínimo del borde.
- `-max-width` define el ancho máximo del borde.
- `-min-width` define el ancho mínimo del borde.
- `-colormap` indica el color de la barra de valores.
- `-o` indica el archivo de salida.

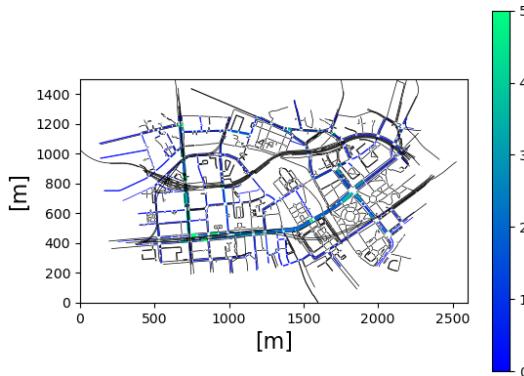


Figure 3.5: Mapa de color de densidad de flujo de vehículos.

La Fig. 3.5, muestra la densidad por carreta. Por medio de la barra de calor a la derecha, vemos sobre el mapa las carreteras con mayor (verde) y menor (azul) cantidad de vehículos durante la simulación. En este ejemplo graficamos la densidad en las carreteras, por tal motivo se indica el valor `-measures` como `density,density;` sin embargo podemos usar otras capturas como la ocupación (https://sumo.dlr.de/docs/Simulation/Output/Lane-_or_Edge-based_Traffic_Measures.html).

3.4.7 Emisiones de gases:

Para lograr esto, se requiere generar los datos de emisiones de los vehículos en la simulación. A diferencia de los ejemplos anteriores donde se agregaba el tópico en el archivo de configuración del escenario sumo.cfg, aquí se requiere (i) agregar la configuración en un archivo adicional (additional-file) y (ii) importar el archivo adicional dentro del bloque inputs del sumocfg.

(i) El nuevo archivo es el edgeData https://sumo.dlr.de/docs/Simulation/Output/Lane-_or_Edge-based_Emissions_Measures.html. Donde, se debe especificar el id, el periodo (tiempo total de simulación) de toma de datos, el tipo de datos a recibir (emisiones) y el nombre del archivo de salida. A continuación se muestra el código de este nuevo archivo.

```

<additional xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/additional_file.
    xsd">

    <edgeData id="1" period="3600" type="emissions" file="edgeData
        .xml" excludeEmpty="true"/>

</additional>

```

(ii) Para importar el nuevo archivo creado al archivo de configuración de sumo se requiere indicar la ruta de ubicación de este archivo en el apartado de archivos adicionales. Por ejemplo:

```

<input>
    <net-file value="osm.net.xml.gz"/>
    <route-files value="osm.passenger.trips.xml"/>
    <additional-files value="osm.poly.xml.gz,edgeData.xml"/>
</input>

```

La Fig. 3.6, muestra las emisiones absoultas (total) de CO₂ en cada calle del mapa. Usamos la herramienta `plot_net_dump.py` para graficar el mapa de color en base a las emisiones CO₂ medidas en las calles del mapa. Note que, adicional a los nombres del archivo de entrada y salida, el valor a graficar también cambia (`-measures CO2_abs,CO2_abs`), así como los límites respectivos.

```

python3 /home/sumo/sumo/tools/visualization/plot_net_dump.py -v -n osm
    .net.xml --measures CO2_abs,CO2_abs --xlabel [m] --ylabel [m] --
    default-width 1 -i edgeData-output-v2.xml,edgeData-output-v2.xml
    --xlim 000,2600 --ylim 000,1500 --default-width .2 --min-color-
    value 0 --max-color-value 2000000 --max-width-value 2000000 --min-
    width-value 0 --max-width 1 --min-width 1 --colormap turbo -o
    density-color-map.png

```

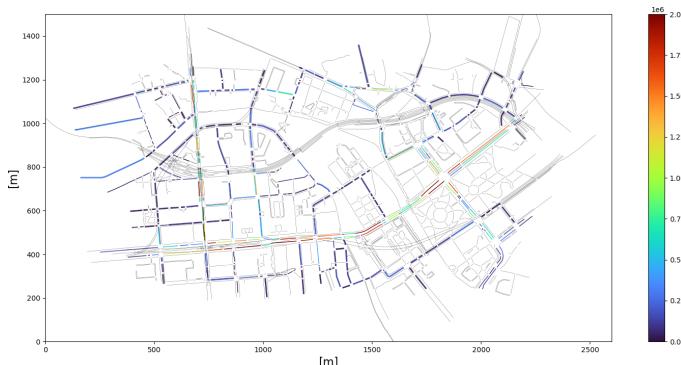


Figure 3.6: Mapa de color de CO₂ absoluto de vehículos.



4. Introducción a TraCI (Traffic Control Interface)

En esta práctica el estudiante conocerá como controlar una simulación de tráfico en tiempo real, a través de la interfaz TraCI para Python. Modalidad: Trabajo Individual. Recursos: Computador personal con SUMO, Python3.

4.1 Objetivos

- Conocer la librería TraCI para control de tráfico.
- Verificar la variable de entorno de SUMO usando script en Python.
- Integrar la interfaz TraCI con SUMO.
- Simular una red vehicular sobre SUMO utilizando TraCI en Python.

4.2 Introducción

TraCI es el término corto para “Interfaz de control de tráfico” (Traffic Control Interface). TraCI permite controlar una simulación de tráfico en marcha, permite recuperar valores de objetos simulados y manipular su comportamiento en línea. Esta interfaz es de especial interés cuando necesitamos interactuar en línea con los objetos de la simulación, por ejemplo cambiar la ruta de un vehículo, o los ciclos semafóricos en función del estado de tráfico.

TraCI utiliza una arquitectura cliente/servidor basada en TCP para proporcionar acceso a sumo. De este modo, sumo actúa como un servidor que se inicia con opciones adicionales de línea de comandos: `-remote-port <INT>` donde `<INT>` es el puerto en el que sumo escuchará las conexiones entrantes. Sumo carga los archivos de simulación y espera a que las aplicaciones externas se conecten y tomen el control. Tenga en cuenta que la opción `-end <TIME>` se ignora cuando sumo se ejecuta como servidor TraCI, sumo se ejecuta hasta que el cliente exige el fin de la simulación.

En la Fig. 4.1, después de iniciar el servidor SUMO (carga el escenario), los clientes se conectan a SUMO configurando una conexión TCP al puerto de sumo designado. La aplicación cliente envía comandos a sumo para controlar la ejecución de la simulación, influir en el comportamiento de un solo vehículo o solicitar detalles ambientales. SUMO responde con una respuesta de estado a cada comando y resultados adicionales que dependen del comando dado.

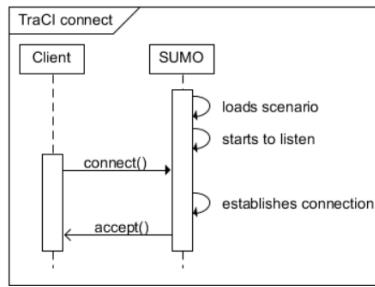


Figure 4.1: Inicio de conexión.

En la Fig. 4.2, el cliente es responsable de cerrar la conexión mediante el comando de cierre. Cuando todos los clientes emitidos cierren, la simulación finalizará, liberando todos los recursos.

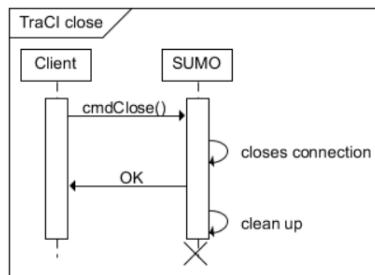


Figure 4.2: Fin de conexión.

Mediante la interfaz TraCI, podemos: recurrir valores, cambiar el estado o suscribirnos a diferentes elementos de la simulación.

- **Valores a recuperar:**
 - Objetos de tráfico
 - Detectores y Salidas (contadores)
 - Red (carreteras)
 - Infraestructura (semáforos)
- **Cambio de estado:**
 - Objetos de tráfico
 - Detectores y Salidas
 - Red
 - Infraestructura
- **Suscripciones:**
 - TraCI/Object Variable Subscription
 - TraCI/Object Context Subscription

La Fig. 4.3, muestra los principales objetos incluidos en la librería TraCI, algunos de los cuales usaremos más adelante.

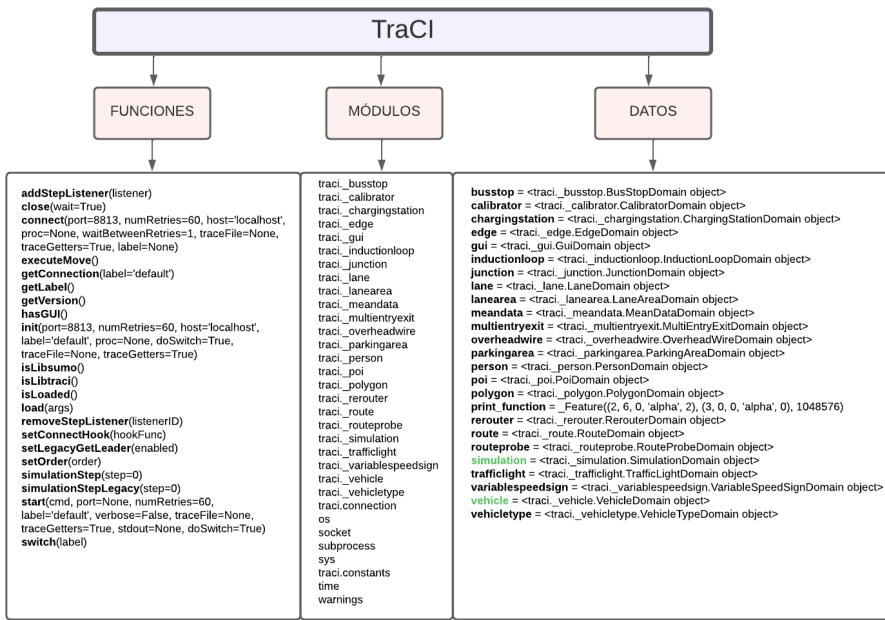


Figure 4.3: Diagrama de funcionamiento de TraCI.py.

4.3 Materiales

- **Ordenador** con sistema operativo Linux (Ubuntu 22.04.3 LTS).
- **SUMO** instalado.
- Entorno de programación Python (Jupyter Notebook, IDLE Python, IDE XXX).

4.4 Instrucciones

Dividimos esta práctica en tres partes:

1. **Parte I:** Verificar pre-requisitos de TraCI.
2. **Parte II:** Ejecutar una simulación usando la interfaz TraCI.
3. **Parte III:** Ejemplos de interacción TraCI - SUMO.

4.4.1 Parte I: Requisitos iniciales de TraCI.

1. El primer paso es verificar que Python está instalado en el sistema. Desde una consola escriba `python3 -version` y retorna la versión instalada.
2. Luego, instalamos la librería TraCI para Python. Dado que usamos Python3, usamos el gestor de paquetes de Python pip3, he instalamos la librería traci. (Documentación disponible en https://sumo.dlr.de/docs/TraCI/Interfacing_TraCI_from_Python.html). Abrimos una terminal y ejecutamos:

```
pip3 install traci
```

Note que en este punto, puede usar cualquier IDE de programación para Python de su preferencia o directamente desde el IDLE de Python. Si usa un IDE de programación y un entorno virtual de programación (venv) ha de verificar la configuración de la variable de entorno en el IDE e instalar traci en el venv correspondiente.

3. En el entorno de Python vamos a configurar las tareas generales para comenzar con TraCI. Dado que vamos a ejecutar comandos de SUMO, lo primero será verificar la variable de entorno \$SUMO_HOME esté configurada en el sistema. Para realizar la comprobación de la variable de entorno, el código lo tiene disponible en Github <https://github.com/erickperez9/ManualSUMO.git>, en la ubicación Scripts TraCI_SUMO/traci_check_sumo_home.py.

La Fig. 4.4 muestra el resultado esperado luego de ejecutar el script de validación de la variable de entorno. Vemos que nos indica además el path donde se instaló la librería traci. Si luego de ejecutar el código Python, resulta que la variable de entorno no está definida, referirse al punto 5 de la práctica 1.

```
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$ python3 traci_check_sumo_home.py
La variable de entorno SUMO_HOME está definida.
Su valor es: /home/sumo/sumo
LOADPATH: /home/sumo/sumo/tools/2024-02-05-16-21-35
/usr/lib/python310.zip
/usr/lib/python3.10
/usr/lib/python3.10/lib-dynload
/home/sumo/.local/lib/python3.10/site-packages
/usr/local/lib/python3.10/dist-packages
/usr/lib/python3/dist-packages
/home/sumo/sumo/tools
TRACIPATH: /usr/local/lib/python3.10/dist-packages/traci/_init_.py
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$
```

Figure 4.4: Verificación de variable de entorno.

En este punto, ya es posible conectar TraCI con SUMO y realizar la codificación para requerimientos específicos, por ejemplo, cambiar la ruta de un vehículo dado durante la simulación.

Un opción adicional al script presentado para verificar los binarios de sumo, es la documentada en <https://medium.com/@mohamad.razzi.my/road-traffic-simulation-using-sumo-with-traci-4a2f3a2a2a1>. El autor utiliza la librería sumolib de Python.

4.4.2 Parte II: Ejecutar una simulación usando la interfaz TraCI.

Dentro del ecosistema de herramientas de SUMO, TraCI desempeña un papel crucial al permitir la comunicación en tiempo real entre simulaciones externas y el simulador SUMO. De especial interés cuando trabajamos con programas externos como simuladores de redes (e.j., Veins) o queremos evaluar aplicaciones de IA.

TraCI ofrece a los usuarios la capacidad de controlar y monitorear de manera dinámica los elementos de la simulación, como vehículos, semáforos, y otros agentes de tráfico.

En esta parte de la práctica integramos TraCI con SUMO para llevar a cabo experimentos en el campo de la movilidad urbana. En específico, exploraremos los funciones principales de TraCI para correr una simulación de SUMO utilizando un script en Python.

En los siguientes pasos, describimos los elementos básicos de un script en Python que utiliza la interfaz TraCI para controlar una simulación de SUMO.

1. A continuación presentamos el código base en Python para ejecutar la simulación. Disponible en Github <https://github.com/erickperez9/ManualSUMO.git>, en la ubicación Scripts Traci_SUMO/traci_run_sumo.py

```
import traci

traci.start(["sumo", "-c", "/home/osm.sumocfg"])

while traci.simulation.getMinExpectedNumber() > 0:
    traci.simulationStep()

traci.close()
```

Listing 4.1: Script **traci_run_sumo.py** para correr simulación TraCI/SUMO.

2. El primer paso es importar las librerías necesarias. En este caso solo se utilizará `traci`.
3. Luego, es necesario iniciar la comunicación SUMO/TraCI a través del comando `traci.start(["cmd"])`. Donde, `cmd` hace referencia al comando que se ejecutará. En este caso se debe indicar el comando para correr una simulación SUMO, ya sea con el uso de `sumo` o `sumo-gui`. Finalmente, se debe indicar la ubicación del archivo de configuración de la simulación de SUMO (`sumocfg`), con el parámetro `-c`.
4. Luego, es necesario iniciar el bucle de la simulación. Donde, la condición debe indicar la duración que deseamos para el escenario. Esta condición lógica pueden ser el número de pasos de la simulación, velocidades, ubicaciones, etc.
Para este ejemplo, tomamos el valor del número mínimo esperado de vehículos en la simulación (`getMinExpectedNumber()`), lo que significa que la simulación se realizará hasta que todos los vehículos de la simulación hayan llegado a su destino.
5. La función `simulaciónStep(float)` dentro del bucle realiza un paso de simulación en cada iteración. Si el valor dado es 0 o está ausente, se realiza exactamente un paso.
6. Por último es importante cerrar la comunicación con el servidor, caso contrario tendremos instancias corriendo en background y puertos ocupados.

En la Fig. 4.5, presentamos el resultado esperado de ejecutar el script en python. Observamos un resumen de la simulación y algunas estadísticas.

```

Simulation ended at time: 3950.00
Reason: TraCI requested termination.
Performance:
Duration: 9.26s
TraCI-Duration: 0.70s
Real time factor: 426.796
UPS: 8284.062669
Vehicles:
Inserted: 389
Running: 0
Waiting: 0
StopDistance (avg of 389):
RouteLength: 1446.19
Speed: 7.45
Duration: 197.09
WaitingTime: 28.26
TimeLoss: 60.89
DepartDelay: 0.58
DijkstraRouter answered 389 queries and explored 354.39 edges on average.
DijkstraRouter spent 0.08s answering queries (0.19ms on average).
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$ 
    
```

Figure 4.5: Resumen de la simulación TraCI/SUMO en consola usando TraCI a través de un script de Python.

- Otro ejemplo común para ejecutar el bucle de simulación, es considerar los pasos que tiene la simulación. Estos pasos están relacionados con el tiempo de simulación, pero no es el mismo configurado en el archivo de configuración del escenario (sumocfg) en el apartado <time>. Cuando se utiliza TraCI, siempre el tiempo final es superior, por lo que se recomienda configurar el límite de pasos con un valor superior; es decir TraCI toma el control del tiempo de la simulación.

A continuación vemos el script de Python correspondiente a la lógica del bucle considerando un tiempo estimado de simulación de una hora (3600 segundos) y 3700 pasos:

```

import traci

traci.start(["sumo", "-c", "/home/osm.sumocfg"])
step = 0

while step < 3700:
    traci.simulationStep()
    step += 1

traci.close()
    
```

Listing 4.2: Variante a script **traci_run_sumo.py** para correr simulación TraCI/SUMO en base al número de pasos de la simulación.

4.4.3 Parte III: Ejemplos de interacción TraCI - SUMO.

En esta última parte de la práctica presentamos cuatro ejemplos de interacción entre TraCI y SUMO. Los código de ejemplo se puede descargar o clonar del repositorio Github <https://github.com/erickperez9/ManualSUMO.git>. Cabe mencionar que los códigos presentados son referenciales, el estudiante puede usarlos como punto de partida para sus experimentos aplicando su lógica particular. Presentamos 4 ejemplos:

- Conteo de vehículos en línea.
- Cálculo de velocidad promedio de vehículos en línea.
- Cálculo de emisiones CO2 de vehículos en línea.
- Cambio de trayectoria de vehículos en línea.

4.4.4 Ejemplo 1: Conteo de vehículos en línea.

- **Ubicación Github:** el ejemplo de conteo de vehículos se encuentra en: Scripts TraCI_SUMO/traci_cont_veh.py.
- **Importando librerías (Líneas 1 , 4):** se utilizan traci para la conexión con SUMO, y re para el manejo de expresiones regulares.
- **Iniciamos la conexión SUMO/TraCI (Línea 8):** a través del comando traci.start(['cmd']). Note que debe modificar el path donde se encuentra su propio archivo de configuración del escenario (sumo.cfg).
- **Bucle de simulación (Líneas 14-20):** usamos traci.vehicle.getIDList() (Línea 15) para obtener una lista de todos los IDs de vehículos presentes en la simulación en el paso actual. El comando devuelve los IDs en el formato *vehID*, donde *ID* indica el número de vehículo de la simulación. En la línea 20, el comando traci.simulationStep() permite avanzar un paso en la simulación.
- **Cierre de la comunicación (Línea 22):** una vez terminada la simulación, es importante cerrar el socket con SUMO.
- **Conteo de los IDs de los vehículos (Líneas 24-38):** es necesario crear una lógica con la cual se extraiga los números identificadores de cada vehículo y obtener el número mayor. En la Fig. ?? se muestra el resultado de ejecutar el script directamente desde la terminal de Linux, obteniendo el número total de vehículos que se movilizaron en el mapa.

```

Simulation ended at time: 3950.00
Reason: TraCI requested termination.
Performance:
Duration: 9.70s
TraCI-Duration: 1.24s
Real time factor: 407.427
UPS: 7908.096957
Vehicles:
Inserted: 389
Running: 0
Waiting: 0
Statistics (avg of 389):
RouteLength: 1446.19
Speed: 7.45
Duration: 197.09
WaitingTime: 28.26
TimeLoss: 60.80
DepartDelay: 0.58

DijkstraRouter answered 389 queries and explored 354.39 edges on average.
DijkstraRouter spent 0.09s answering queries (0.23ms on average).

En el mapa han circulado un total de 429 vehículos
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$
```

Figure 4.6: Conteo de vehículos con TraCI/SUMO.

4.4.5 Ejemplo 2: Cálculo de velocidad promedio de vehículos en línea.

1. **Ubicación Github:** el ejemplo de cálculo de la velocidad de vehículos se encuentra en: Scripts TraCI_SUMO/traci_vel.py.
2. **Importando librerías (Líneas 1 , 7):** En este caso se utilizará traci para la conexión con SUMO. Además de, csv para el manejo de archivos csv, pandas para análisis de datos, time para obtener los tiempos de simulación, matplotlib para generar gráficas.
3. **Iniciamos la conexión SUMO/TraCI (Línea 8):** a través del comando traci.start(['cmd']). Note que debe modificar el path donde se encuentra su propio archivo de configuración del escenario (sumo.cfg).

4. **Preparamos archivo csv para almacenar datos (Líneas 15-21):** creamos y etiquetamos las columnas del archivo csv.
5. **Bucle de simulación (Líneas 24-33):** guardamos los datos de interés: identificadores de los vehículos, velocidades, y tiempos de simulación. Los datos se guardan en un archivo de salida de tipo csv.

Para obtener el tiempo de simulación en segundos usamos (Línea 25):
`traci.simulation.getCurrentTime() / 1000.`

Para obtener los IDs usamos (Línea 27). Note que generamos un bucle adicional para iterar sobre cada vehículo:
`traci.vehicle.getIDList()`.

Para obtener las velocidad de cada vehículo, usamos (Línea 28):
`traci.vehicle.getSpeed(veh_id)`.

6. **Cierre de la comunicación (Línea 35):** una vez terminada la simulación, es importante cerrar el socket con SUMO.
7. **Análisis de datos (Líneas 37-48):** En el paso anterior creamos un archivo .csv. Ahora leemos el archivo con los datos de velocidad de cada vehículo en cada instante de tiempo y calculamos la velocidad media por vehículo y de todos los vehículos (C.I. 95%).
8. **Gráficas de los datos de velocidades (Líneas 51-69):** La Fig. 4.8, muestra las velocidades obtenidas por vehículo a la izquierda y del escenario en general a la derecha.

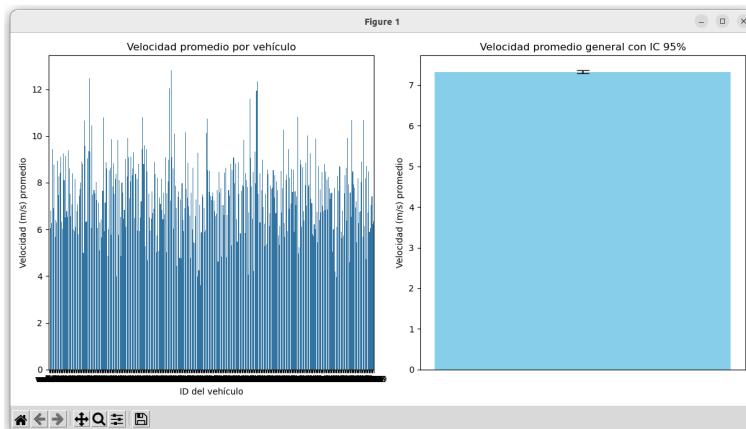


Figure 4.7: Cálculo de velocidades promedio con TraCI/SUMO.

4.4.6 Ejemplo 3: Cálculo de emisiones CO2 de vehículos en línea.

1. **Ubicación Github:** el ejemplo de cálculo de emisiones de gases se encuentra en:
`Scripts TraCI_SUMO/traci_co2.py`.

4.4 Instrucciones

2. **Importando librerías (Líneas 1 , 7):** En este caso se utilizará `traci` para la conexión con SUMO. Además de, `csv` para el manejo de archivos csv, `pandas` para análisis de datos, `time` para obtener los tiempos de simulación, `matplotlib` para generar gráficas y `seaborn`, `stats` para estadísticas y gráficas.
3. **Iniciamos la conexión SUMO/TraCI (Línea 10):** a través del comando `traci.start(['cmd'])`. Note que debe modificar el path donde se encuentra su propio archivo de configuración del escenario (`sumo.cfg`).
4. **Preparamos archivo csv para almacenar datos (Líneas 13-20):** creamos y etiquetamos las columnas del archivo csv.
5. **Bucle de simulación (Líneas 15-32):** guardamos los datos de interés: identificadores de los vehículos, emisiones de CO₂, y tiempos de simulación. Los datos se guardan en un archivo de salida de tipo csv.

Para obtener el tiempo de simulación en segundos se usa:

```
traci.simulation.getCurrentTime() / 1000
```

Para obtener los IDs se utiliza:

```
traci.vehicle.getIDList()
```

Para obtener las emisiones de CO₂ se utiliza:

```
traci.vehicle.getCO2Emission(veh_id)
```

6. **Cierre de la comunicación (Línea 35):** una vez terminada la simulación, es importante cerrar el socket con SUMO.
7. **Análisis de datos (Líneas 37-48):** En el paso anterior creamos un archivo .csv. Ahora leemos el archivo con las capturas de emisiones (mg/s) y calculamos la media de cada vehículo y de la simulación en general.
8. **Gráficas de los datos de emisiones de CO₂ (Líneas 53-68):** La Fig. 4.8, muestra las emisiones de CO₂ obtenidas por vehículo a la izquierda y en general a la derecha.

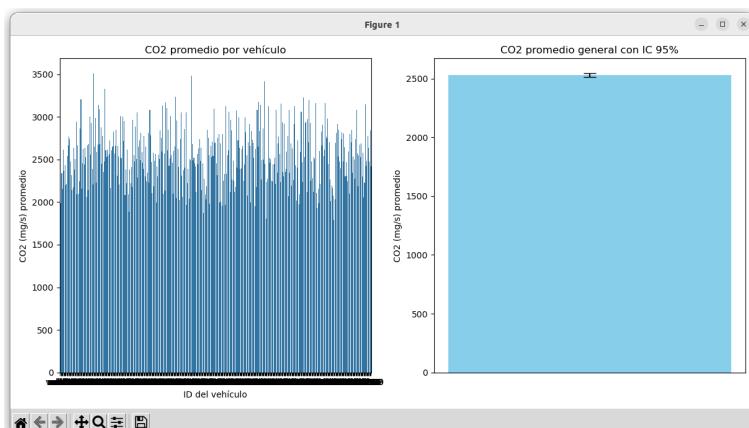


Figure 4.8: Cálculo de emisiones CO₂ promedio con TraCI/SUMO.

4.4.7 Ejemplo 4: Cambio de trayectoria de vehículos en línea.

1. **Ubicación Github:** el ejemplo de cambio de trayectoria se encuentra en:
Scripts TraCI_SUMO/traci_change_pos.py.
2. **Importando librerías (Líneas 1 - 3):** En este caso se utilizará traci para la conexión con SUMO. Además de, csv para el manejo de archivos csv.
3. **Iniciamos la conexión SUMO/TraCI (Línea 9):** a través del comando `traci.start(["cmd"])`. Note que debe modificar el path donde se encuentra su propio archivo de configuración del escenario (sumo.cfg).
4. **Preparamos archivo csv para almacenar datos (Líneas 11 - 18):** creamos y etiquetamos las columnas del archivo csv. Note que se incluye EdgeNow que hace referencia a los IDs de las carreteras en cada instante.
5. **Bucle de simulación (Líneas 21-36):** guardamos los datos de interés: identificadores de los vehículos, tiempo de simulación, IDs de las carreteras, y velocidad. Los datos se guardan en un archivo de salida de tipo csv.

Para obtener el tiempo de simulación en segundos se usa:

```
traci.simulation.getCurrentTime() / 1000
```

Para obtener los IDs se utiliza:

```
traci.vehicle.getIDList()
```

Para obtener las velocidad se utiliza:

```
traci.vehicle.getSpeed(veh_id)
```

Para obtener el ID de la calle (edge) donde se encuentra el vehículo actualmente se utiliza:

```
traci.vehicle.getRoadID(veh_id)
```

Para cambiar el destino de un vehículos se utiliza:

```
traci.vehicle.changeTarget(veh_id,EDGE_ID)
```

Para el ejemplo, creamos una lógica sencilla para generar el cambio de ruta en un vehículo. Si la velocidad es menor a 10 m/s, el destino del vehículo con ID `veh424` se modifica y se re-dirige al destino con ID `234463928#1` (ID de la carretera). SUMO utiliza el algoritmo Dijkstra para calcular la nueva ruta al destino.

Los IDs de los bordes de las carreteras se pueden verificar en los archivos de configuración de red pertinentes (.net) o utilizando la herramienta *netedit* de SUMO al dar clic sobre la carretera en cuestión. Para abrir esta herramienta basta con el comando *netedit* desde la terminal (ver Fig. 4.9).

6. **Cierre de la comunicación (Línea 38):** una vez terminada la simulación, es importante cerrar el socket con SUMO.

4.4 Instrucciones

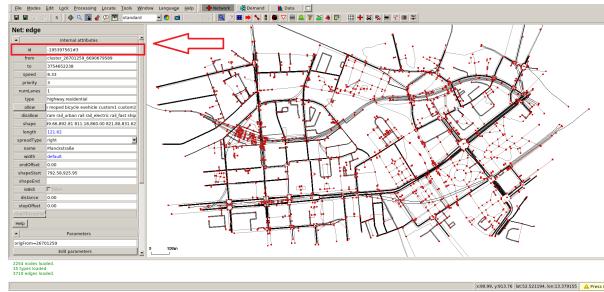
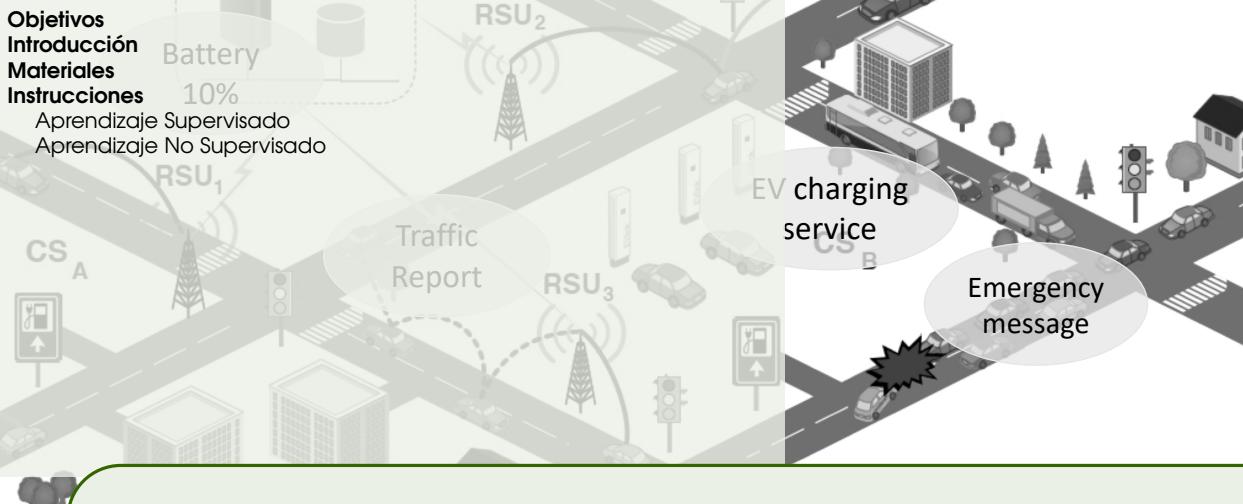


Figure 4.9: Identificador de calle desde netedit, editor gráfico del mapa en SUMO.

En la Figura 4.10, se observan los logs en la terminal de Linux. Vemos el reenrutamiento del veh424 cuyo nuevo destino es 234463928#1.

```
Tiempo: 3984.0, ID del Vehiculo: veh424, Velocidad: 15.883128615470938, EdgeNow: :2425782615_5
*** SE HA CAMBIADO LA RUTA DEL VEHICULO veh424 HACIA: 234463928#1
Tiempo: 3985.0, ID del Vehiculo: veh424, Velocidad: 15.228921826431699, EdgeNow: :1275468904_1
*** SE HA CAMBIADO LA RUTA DEL VEHICULO veh424 HACIA: 234463928#1
Tiempo: 3986.0, ID del Vehiculo: veh424, Velocidad: 15.466820430428893, EdgeNow: 234463928#1
*** SE HA CAMBIADO LA RUTA DEL VEHICULO veh424 HACIA 234463928#1
Tiempo: 3987.0, ID del Vehiculo: veh424, Velocidad: 14.763622594298035, EdgeNow: 234463928#1
Simulation ended at time: 3988.00
Reason: TraCI requested termination.
Performance:
Duration: 24.14s
TraCI-duration: 10.57s
Real time factor: 161.916
UPS: 3174.801127
Vehicles:
Inserted: 389
Running: 0
Waiting: 0
Statistics (avg of 389):
RouteLength: 1445.23
Speed: 7.45
Duration: 196.98
WaitingTime: 28.26
TimeLoss: 60.79
DepartDelay: 0.58
```

Figure 4.10: Destino cambiado para vehículo *veh424*.



5. Clasificación de movilidad con aprendizaje automático

En esta práctica el estudiante utilizará técnicas de aprendizaje automático para clasificación de diferentes medios de transporte. Modalidad: Trabajo Individual. Recursos: Computador personal con sistema operativo Linux.

5.1 Objetivos

- Generar una simulación multimodal.
- Obtener estadísticas de la simulación.
- Clasificar datos mediante algoritmos de aprendizaje automático.

5.2 Introducción

La clasificación de datos es un componente fundamental del aprendizaje automático que implica categorizar instancias en clases predefinidas o etiquetas, basándose en las características observadas de los datos. En el contexto de la simulación vehicular en SUMO (Simulador de Movilidad Urbana), la clasificación de datos puede ser una herramienta para comprender y tomar decisiones informadas en sistemas de transportación inteligente (ITS).

En esta práctica de laboratorio, exploraremos un ejemplo sencillo de clasificación para predecir el tipo de transporte (carro, bus, bicicleta) en función de sus emisiones de CO₂. Para esto, aplicamos diferentes algoritmos:

1. **Regresión Logística:** algoritmo de clasificación lineal. Modela la relación entre las variables de entrada y la probabilidad de pertenecer a una clase específica.
2. **SVC (Support Vector Classifier):** algoritmo que encuentra el hiperplano que mejor separa las clases en un espacio.
3. **Random Forest Classifier:** algoritmo de aprendizaje basado en árboles de decisión que crea múltiples árboles y combina sus resultados para mejorar la precisión y evitar el sobreajuste.
4. **Ensemble Vote Classifier:** Una técnica que combina múltiples clasificadores base para obtener un modelo más robusto y generalizable.
5. **KMeans:** Un algoritmo de agrupamiento que clasifica los datos en clusters basados en la similitud de las características. K-Means es un algoritmo de agrupación muy

simple (la agrupación pertenece al aprendizaje no supervisado). Dado un número fijo de grupos y un conjunto de datos de entrada, el algoritmo intenta dividir los datos en grupos de modo que los grupos tengan una alta similitud intraclasa y una baja similitud entre clases.

Cada uno de estos algoritmos tiene sus propias ventajas y desventajas. En esta práctica, los participantes explorarán cómo preparar los datos de emisiones de CO₂ obtenidos de una simulación vehicular en SUMO para su posterior clasificación. Aprenderán a entrenar, ajustar y evaluar los modelos de clasificación utilizando los algoritmos mencionados anteriormente, comparando su rendimiento y seleccionando el más adecuado para el problema específico.

Algoritmo:

- (a) Inicialice los centros del clúster, ya sea aleatoriamente dentro del rango de los datos de entrada o (recomendado) con algunos de los ejemplos de capacitación existentes.
- (b) Hasta la convergencia
 - i. Asigne cada punto de datos al grupo más cercano. La distancia entre un punto y el centro del grupo se mide utilizando la distancia euclídea.
 - ii. Actualice las estimaciones actuales de los centros del grupo configurándolas en la media de todas las instancias que pertenecen a ese grupo.

La función objetivo subyacente intenta encontrar centros de conglomerados de modo que, si los datos se dividen en los conglomerados correspondientes, las distancias entre los puntos de datos y sus centros de conglomerados más cercanos sean lo más pequeñas posible.

Dado un grupo de puntos de datos x_1, \dots, x_n y un número positivo k , se encuentra los clústers C_1, \dots, C_k que minimizan:

$$J = \sum_{i=1}^n \sum_{j=1}^n z_{ij} \|x_i - \mu_j\|_2 \quad (5.1)$$

Donde:

- z_{ij} define si el punto de datos x_j pertenece o no al clúster C_j
- μ_j denota el centro del grupo C_j
- $\|\cdot\|_2$ denota la distancia euclídea

5.3 Materiales

- **Ordenador** con sistema operativo Linux (Ubuntu 22.04.3 LTS).
- **SUMO** instalado.
- Entorno de programación Python (Jupyter Notebook, IDLE Python, IDE XXX).

5.4 Instrucciones

A continuación, se describen los pasos a seguir para implementar el sistema de clasificación vehicular en función de sus emisiones. Presentamos ejemplos de aplicación de algoritmos supervisados y no supervisados.

5.4.1 Aprendizaje Supervisado

- 1. Generar una simulación multimodal:** significa integrar diferentes tipos de transporte en una misma simulación (taxi, bus, motorcycle). En [6] se detallan los tipos de vehículos disponibles en SUMO.

Para el ejemplo, usaremos los siguientes tipos: taxi, motorcycle, y bus. Para definir el tipo de vehículo, se requiere modificar el archivo de rutas `.rou.xml`. Hemos de especificar el tipo de vehículo al inicio del archivo. Otra opción es definir los tipos de vehículos en un archivo adicional y agregarlo en el bloque de inputs del sumocfg. Por ejemplo:

```
<vType id="taxi" vClass="taxi"/>
<vType id="bus" vClass="bus"/>
<vType id="motorcycle" vClass="motorcycle"/>

<vehicle id="2" type="taxi" depart="2.00">
    <route edges="E2 E3"/>
</vehicle>
<vehicle id="3" type="bus" depart="3.00">
    <route edges="E5 -E1"/>
</vehicle>
<vehicle id="4" type="motorcycle" depart="4.00">
    <route edges="-E7 -E6 -E1 -E0"/>
</vehicle>
```

Note que el ejemplo implica modificar manualmente el archivo `.rou`, lo cuál resulta poco práctico si tenemos un archivo con varias entradas. Una de las opciones de la herramienta `randomTrips` es `-vehicle-class`, la cuál permite definir un tipo de vehículo predefinido e.j., bus; sin embargo, todas las rutas se generan con este tipo de vehículo `type="bus"`. En el caso de varios tipos de vehículos (multimodal), la opción factible sería generar un script que modifique el tipo de vehículo en las rutas previamente generadas con `randomtrips` con la opción `-vehicle-class` y modificar a conveniencia.

- 2. Configurar captura de emisiones de gases:** usamos el archivo de salida (output) `tripinfo` con la opción adicional de captura de emisiones. En el siguiente código mostramos el bloque adicional que se debe agregar en el archivo `sumocfg` de configuración del escenario:

```
<emissions>
    <device . emissions . probability value="1"/>
</emissions>
```

La documentación del archivo de salida `tripinfo` se puede ampliar en <https://sumo.dlr.de/docs/Simulation/Output/TripInfo.html>.

- 3. Preparación de los datos:** Simulamos y obtenemos el archivo de salida `tripinfo.xml`

que incluye las emisiones de los diferentes tipos de vehículos. Lo convertimos de XML a CSV para facilitar el análisis de los datos.

4. **Pre procesamiento de los datos:** Una vez obtenido el .csv, filtramos los datos de `tripinfo_id`, `emissions_CO2_abs`, `tripinfo_vType`, datos de interés para el ejemplo. Mapeamos los datos de la columna `tripinfo_vType`: `mapeo = {'taxi': 1, 'motorcycle': 2, 'bus': 3}`. Finalmente, almacenamos los datos en un arreglo con los valores filtrados. Definimos una función `def data_set(archivo_entrada)`: de ejemplo que se puede descargar de <https://github.com/erickperez9/ManualSUMO.git> en la ubicación *Clasificacion con machine learning/data_set.py*.
5. **Algoritmos de clasificación de los datos:** importamos librerías de python: pandas, matplotlib, intertools, sklearn para clasificar los datos con aprendizaje automático y mlxtend para observar las tendencias de la clasificación de datos. Definimos un script de ejemplo que se puede descargar de <https://github.com/erickperez9/ManualSUMO.git> en la ubicación *Clasificacion con machine learning/mlxtend test.py*. Note que este script utiliza la función `def data_set(archivo_entrada)` definida en el paso anterior para pre procesar los datos (Línea 20).

El ejemplo incluye los siguientes algoritmos de prueba de clasificación (Líneas 23-26):

- LogisticRegression,
- SCV,
- RandomForestClassifier y
- EnsembleVoteClassifier

Luego, en las líneas 36 y 37 sepáramos los datos en `X`, `Y`, donde `X` representa las medidas de CO2 absolutas, y `Y` representa la etiqueta de datos, es decir, a qué tipo de vehículo pertenece cada valor medido.

Finalmente, se obtiene el mapa de colores de clasificación del tipo de vehículo en función de sus emisiones de gases. En la Figura 5.1 se ilustra el resultado de clasificar los tipos de vehículos en función de las las emisiones CO2 medidas.

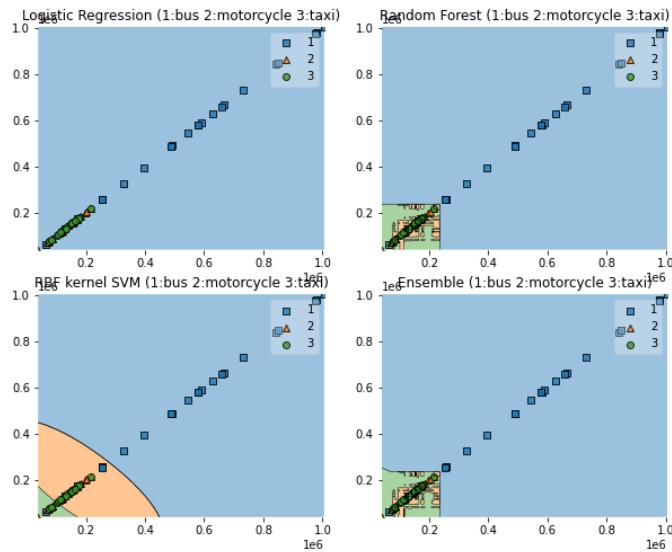


Figure 5.1: Clasificación del tipo de vehículo según sus emisiones CO2. Datos etiquetado SUMO.

5.4.2 Aprendizaje No Supervisado

Como ejemplo de aplicación de aprendizaje no supervisado, el cuál permite clasificar datos no etiquetados, usamos el algoritmo k-means, uno de los algoritmos de agrupamiento más populares. El objetivo de k-means es dividir un segmento de datos, entre un número de grupos (llamados clusters) basados en su similitud. para nuestro ejemplo los valores a dividir son las emisiones de CO2. De manera general, k-means asigna cada punto del conjunto de datos al cluster cuyo punto central (centroide) esté más cercano a él (distancia euclídea). El centroide es el promedio de todos los puntos de datos en ese cluster. Este algoritmo itera hasta que los centroides dejen de cambiar o un máximo número de iteraciones ha sido alcanzada. Se requieren 3 datos de entrada:

1. Número de clusters (k).
2. Conjunto de datos a ser agrupados.
3. Centroides iniciales para cada cluster. Pueden ser elegidos usando diferentes métodos, aquí los seleccionamos aleatoriamente.
- 4.

Definimos un script de ejemplo que se puede descargar de <https://github.com/erickperez9/ManualSUMO.git> en la ubicación *Clasificacion con machine learning/k-means.py*. Note que este script utiliza como entra el archivo tripinfo.csv de la simulación SUMO, tomando la columna `emissions_CO2_abs` para clasificar las emisiones por regiones y así posteriormente poder predecir de que tipo de vehículo de trata.

En la Fig. 5.2 se muestra el resultado de la ejecución del script `kmeans.py`. Podemos ver 3 clusters en azul, rojo y verde. Verificamos con * los centroides correspondientes a cada cluster.

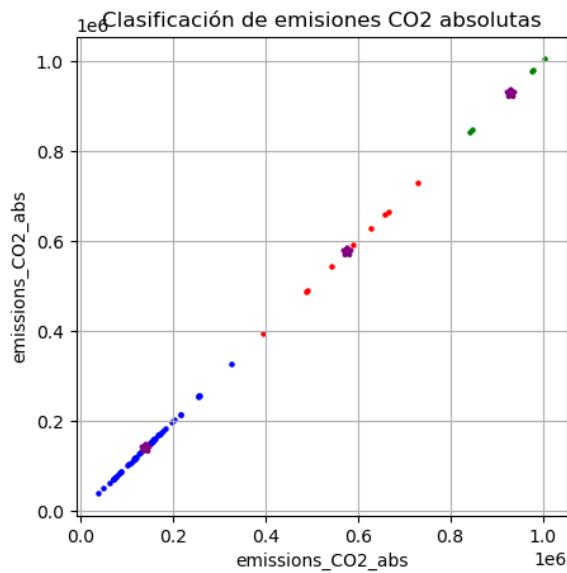


Figure 5.2: Clasificación de los tipos de vehículos en función de sus emisiones de CO2 no etiquetadas de SUMO.



6. Aprendizaje automático aplicado a ITS

En esta práctica el estudiante implementará algoritmos de aprendizaje automático supervisados, con el objetivo de clasificar el tipo de vehículo en función de sus emisiones CO₂. Modalidad: Trabajo Individual. Recursos: Computador personal con sistema operativo Linux.

6.1 Objetivos

- Simular una red vehicular sobre SUMO.
- Obtener estadísticas de la simulación en SUMO.
- Predecir datos mediante algoritmos de machine learning.

6.2 Introducción

En el contexto de los servicios inteligentes de transportación, la predicción de datos permite abordar diversas problemáticas, como la predicción de nivel de congestión vehicular o calidad de aire en ruta.

Continuando con el ejemplo anterior, en esta práctica usamos el dataset de emisiones de gases vehiculares (tripinfo.csv) junto con los algoritmos supervisados Random Forest Classifier y Logistic Regression, para predecir el tipo de vehículo (valores discretos: motorcycle, taxi, bus) en función de los datos capturados de CO₂. Note que nos mantenemos con algoritmos de clasificación cuyo objetivo es predecir/clasificar valores discretos como el tipo de vehículo, verdadero o falso, Spam o no spam, etc. Por otra parte, los algoritmos de regresión se utilizan para predecir valores continuos como precio, salario, edad, etc.

- **Algoritmo Random Forest:** se pueden utilizar para resolver problemas de regresión (variable continua) y clasificación (variable discreta/categórica). Se basa en la construcción de múltiples árboles de decisión durante el entrenamiento. Cada árbol es construido utilizando un subconjunto aleatorio de las características del conjunto de datos.
- **Algoritmo Logistic Regression :** realiza tareas de clasificación binaria al predecir la probabilidad de un resultado, evento u observación. Utiliza la función logística (sigmoid function) para modelar la probabilidad de que una instancia pertenezca a una clase específica. La regresión lógica analiza la relación entre una o más variables

independientes y clasifica los datos en clases discretas. Se utiliza ampliamente en modelos predictivos, donde el modelo estima la probabilidad matemática de si una instancia pertenece a una categoría específica o no.

6.3 Materiales

- **Ordenador** con sistema operativo Linux (Ubuntu 22.04.3 LTS).
- **SUMO** instalado.
- Entorno de programación Python (Jupyter Notebook, IDLE Python, IDE XXX).

6.4 Instrucciones

Se provee como ejemplo el código en Python, disponible en Github <https://github.com/erickperez9/ManualSUMO.git>, en la ubicación `Clasificacion con machine learning/RF_RLClassifiers_prediction model.py`. A continuación, se describen de manera general el script:

1. El primer paso realizar la simulación vehicular sobre SUMO, donde es necesario obtener como salida el archivo de información de viajes (`tripinfo`) y además de indicar que se debe agregar las emisiones emitidas por los vehículos. Referirse a la Práctica 1.4.
2. Con el objetivo de analizar las emisiones emitidas en un mapa con diferentes tipos de vehículos, es necesario configurar los vehículos que circulan sobre el mapa. En [6] se puede encontrar todos los tipos de vehículos que se puede configurar en la simulación de SUMO. Para la presente práctica usaremos: taxi, motorcycle, bus.

Para definir el tipo de vehículos, se requiere modificar el archivo de rutas `.rou.xml` y especificar el tipo de vehículo en cada ruta.

3. Luego, simular y obtener el archivo de salida `tripinfo.xml` y convertirlo en csv.
4. Con las estadísticas listas para ser analizadas, se procede con la codificación del primer ejemplo. Donde, se hace uso de la librerías: `pandas`, `matplotlib`, y `sklearn` para los algoritmos de machine learning.
5. A continuación se indica el script `RF_RLClassifiers_prediction model.py` donde se hace uso de los algoritmo `RandomForestClassifier` y `LogisticRegression` para realizar el entrenamiento y obtener un modelo de predicción de datos, usando machine learning. Además, el script en python requiere como entrada el archivo csv, obtenido de una simulación de SUMO.
6. Al correr el script se obtendrá la puntuación de precisión de cada algoritmo, así como el reporte respectivo de clasificación de cada algoritmo. Además, se mostrará la gráfica en la que compara los algoritmos con datos entrenado y predichos.

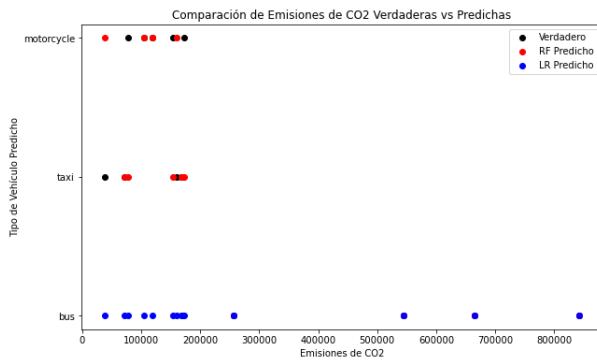


Figure 6.1: Predicción de datos de CO2 de una simulación SUMO con machine learning.

En la Figura 6.2 se indica los reportes de los algoritmos *Random Forest Classifier* y *Logistic Regression*. El reporte de clasificación ('classification_report') proporciona métricas útiles para evaluar el rendimiento de un modelo de clasificación. Estas métricas se calculan comparando las etiquetas verdaderas (ground truth) con las etiquetas predichas por el modelo. Aquí está una explicación de las métricas que se encuentran en el reporte de clasificación:

- **Precisión (Precision):** La precisión es la proporción de instancias clasificadas como positivas que son verdaderamente positivas. Se calcula como $TP / (TP + FP)$, donde TP son los verdaderos positivos y FP son los falsos positivos. Una alta precisión indica que el modelo no clasifica incorrectamente muchas instancias negativas como positivas.
- **Recall (Recuperación o Sensibilidad):** El recall es la proporción de instancias positivas que fueron correctamente identificadas por el modelo. Se calcula como $TP / (TP + FN)$, donde TP son los verdaderos positivos y FN son los falsos negativos. Un alto recall indica que el modelo identifica la mayoría de las instancias positivas.
- **F1-Score:** El F1-score es la media armónica de precisión y recall. Es útil cuando hay un desequilibrio en las frecuencias de las clases. Se calcula como $2 * (\text{precisión} * \text{recall}) / (\text{precisión} + \text{recall})$. El F1-score alcanza su mejor valor en 1 (precisión y recall perfectos) y su peor valor en 0.
- **Soporte (Support):** El soporte es el número de ocurrencias reales de la clase en los datos de prueba. Proporciona información sobre la distribución de las clases en el conjunto de datos de prueba.

Por lo general, cuando se interpreta el reporte de clasificación, se busca un equilibrio entre precisión y recall, dependiendo del problema específico que se esté abordando. En general, un F1-score alto indica un buen equilibrio entre precisión y recall.

6.4 Instrucciones

```
Random Forest Classifier:  
Accuracy Score: 0.6428571428571429  
  
Classification Report:  
precision    recall   f1-score   support  
  
      0       1.00     1.00     1.00      4  
      1       0.50     0.60     0.55      5  
      2       0.50     0.40     0.44      5  
  
accuracy          0.64  
macro avg       0.67     0.67     0.66     14  
weighted avg    0.64     0.64     0.64     14  
  
  
Logistic Regression:  
Accuracy Score: 0.2857142857142857  
  
Classification Report:  
precision    recall   f1-score   support  
  
      0       0.29     1.00     0.44      4  
      1       0.00     0.00     0.00      5  
      2       0.00     0.00     0.00      5  
  
accuracy          0.29  
macro avg       0.10     0.33     0.15     14  
weighted avg    0.08     0.29     0.13     14
```

Figure 6.2: Reportes de redicción de datos de CO₂ de una simulacián SUMO con machine learning



7. Aprendizaje por refuerzo (RL) aplicado a ITS.

En esta práctica el estudiante implementará una red CAN de 2 y 3 nodos con dispositivos Arduino, controladores y trancceptores CAN. Modalidad: Trabajo Individual. Recursos: Computador personal con sistema operativo Linux.

7.1 Objetivos

- Integrar la interfaz TraCI con SUMO.
- Simular una red vehicular sobre SUMO utilizando scripts en python.
- Manejar el sistema de señales de semaforización en SUMO a través de un entrenamiento con reinforcement learning.

7.2 Introducción

Las redes vehiculares representan un campo de estudio crítico en el diseño de sistemas de transporte eficientes y sostenibles. En este contexto, el simulador SUMO y su interfaz TraCI desempeñan un papel fundamental al permitir la modelización, simulación y análisis de sistemas de tráfico en entornos urbanos y de carretera. La capacidad de SUMO para simular la interacción dinámica entre vehículos, peatones y otros agentes, junto con la posibilidad de interactuar en tiempo real a través de TraCI, ofrece una plataforma versátil para investigar estrategias de control de tráfico innovadoras. Uno de los aspectos críticos en la gestión del tráfico es el control de semáforos (TLS, por sus siglas en inglés), que desempeña un papel crucial en la regulación del flujo vehicular y la reducción del tiempo de espera de los conductores. La optimización de los tiempos de los semáforos puede mejorar significativamente la fluidez del tráfico, reducir la congestión y disminuir los tiempos de viaje.

En esta práctica de laboratorio, exploraremos la importancia de controlar los TLS como parte de la estrategia para reducir el tiempo de espera de los conductores. Utilizando SUMO y TraCI, se entenderá la optimización de la sincronización de semáforos en entornos urbanos y de carretera. Además, se explorarán técnicas de inteligencia artificial, como el aprendizaje por refuerzo, para mejorar la eficiencia y la adaptabilidad del control de semáforos en tiempo real. Estas técnicas permiten que los semáforos aprendan y se adapten a las condiciones cambiantes del tráfico, mejorando así la experiencia de conducción y la eficiencia del sistema.

de transporte en general.

El aprendizaje por refuerzo es una poderosa técnica de aprendizaje automático que se inspira en la forma en que los seres vivos aprenden a través de la interacción con su entorno. A diferencia de otros enfoques de aprendizaje, el aprendizaje por refuerzo se centra en que un agente aprenda a través de la experiencia, tomando decisiones y recibiendo retroalimentación en forma de recompensas o penalizaciones. En el aprendizaje por refuerzo, un agente interactúa con un entorno, seleccionando acciones con el objetivo de maximizar una señal de recompensa a largo plazo. Esta señal de recompensa puede ser positiva, negativa o neutra, y se utiliza para guiar al agente hacia comportamientos deseables.

Los elementos clave del aprendizaje por refuerzo incluyen:

- **Agente:** Es la entidad que toma decisiones y realiza acciones en un entorno. El agente aprende a través de la interacción con el entorno y la retroalimentación que recibe.
- **Ambiente:** Representa el mundo en el que opera el agente. Puede ser cualquier sistema que responda a las acciones del agente y proporcione retroalimentación en forma de recompensas o penalizaciones.
- **Acciones:** Son las decisiones que el agente puede tomar en un determinado estado del entorno. El conjunto de acciones disponibles depende del problema específico que se esté abordando.
- **Recompensas:** Son señales numéricas que indican qué tan bueno fue el resultado de una acción en un estado particular. El objetivo del agente es maximizar la recompensa acumulada a largo plazo.
- **Política:** Es la estrategia que el agente utiliza para seleccionar acciones en función de su estado actual. La política puede ser determinista o estocástica.



Figure 7.1: Aprendizaje por refuerzo.

Algunos algoritmos comunes de aprendizaje por refuerzo son:

- **Q-Learning:** Un algoritmo de aprendizaje por refuerzo de tipo off-policy que aprende la función de valor óptimo de una política de comportamiento.
- **Sarsa (State-Action-Reward-State-Action):** Otro algoritmo de aprendizaje por refuerzo que actualiza los valores de la función de valor basándose en la acción que el agente realmente toma.
- **Deep Q-Networks (DQN):** Utiliza redes neuronales profundas para aproximar la función de valor en entornos de aprendizaje por refuerzo de alta dimensionalidad.

En la práctica de laboratorio, los participantes aprenderán a implementar y experimentar con estos algoritmos, explorando cómo pueden ser aplicados para optimizar sistemas de transporte, como la sincronización de semáforos, la navegación de vehículos autónomos y la gestión de flotas, contribuyendo así a la eficiencia y seguridad del transporte en entornos urbanos y de carretera. Al comprender y dominar estas herramientas y técnicas, los participantes estarán mejor equipados para abordar los desafíos de la gestión del tráfico y contribuir al desarrollo de sistemas de transporte inteligentes y sostenibles en el futuro.

7.3 Materiales

- **Ordenador** con sistema operativo Linux (Ubuntu 22.04.3 LTS).
- **SUMO** instalado.

7.4 Instrucciones

A continuación, se describen los pasos para utilizar un algoritmo de entrenamiento con reinforcement learning, desarrollado por la Facultad de Ingeniería del Gobierno de Gandhinaga. Este algoritmo de enrutamiento se encuentra en el siguiente repositorio de GitHub.

<https://github.com/Maunish-dave/Dynamic-Traffic-light-management-system/tree/main>

Supongamos que tenemos una cuadrícula de ciudad como se muestra en la Figura 7.2 con 4 nodos de semáforo: n1, n2, n3 y n4.

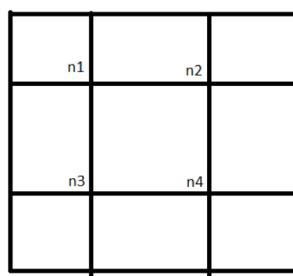


Figure 7.2: Ejemplo de ubicación de semáforos.

Entonces, el modelo de aprendizaje toma 4 decisiones (una para cada nodo) sobre qué lado seleccionar para la señal verde. Donde, es necesario seleccionar un tiempo mínimo (por ejemplo, 30 segundos) para que el modelo no pueda seleccionar un tiempo de luz verde por debajo de ese límite. La tarea principal es minimizar el tiempo que los vehículos tienen que esperar en el semáforo. El tiempo de espera para un semáforo determinado es igual al total de automóviles presentes en el semáforo por el número de segundos. Cada semáforo contará con 4 contadores de tiempo de espera para cada lado de la vía. Entonces, en base a eso, el modelo decidirá qué lado seleccionar para la señal verde. La Figura 7.3 indica el diagrama de flujo que sigue el modelo de entrenamiento con reinforcement learning.

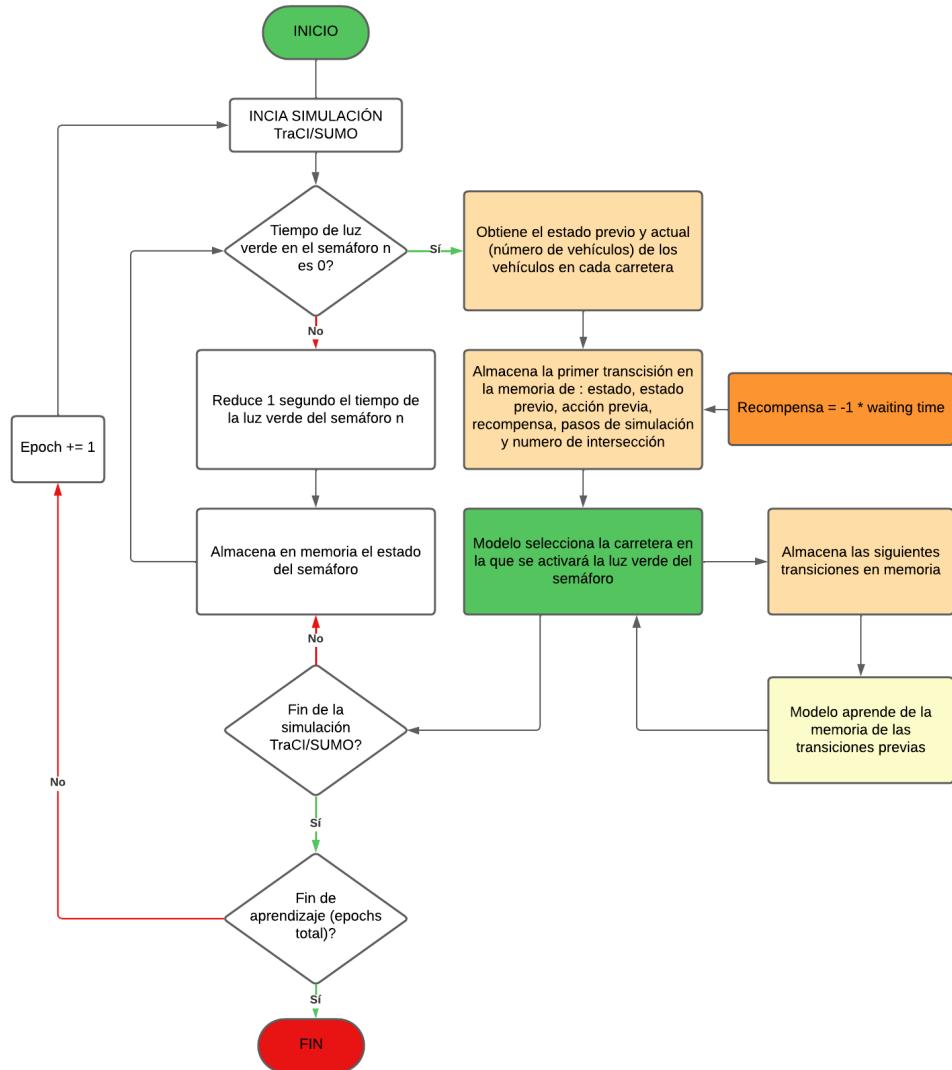


Figure 7.3: Diagrama de flujo de sistema de gestión de semáforo dinámico.

Se ha entrenado al modelo en una serie de eventos. El evento se define como un movimiento fijo en el que los vehículos pasarán a través de nodos de forma fija (pseudoaleatoria). La razón para mantener el evento fijo es que usar un evento aleatorio cada vez dará un resultado aleatorio. Usaremos muchos de estos eventos fijos para entrenar nuestro modelo para que pueda manejar diferentes situaciones. La única información que recibirá nuestro modelo es la cantidad de vehículos presentes en 4 lados de cada nodo de tráfico. y nuestro modelo generará 4 lados, uno para cada nodo. El número de nodos depende del tamaño de la cuadrícula. La Figura 7.4 ilustra las etapas de aprendizaje por refuerzo del modelo de aprendizaje presentado.

Los siguientes pasos describen como correr el programa de entrenamiento de los autores.

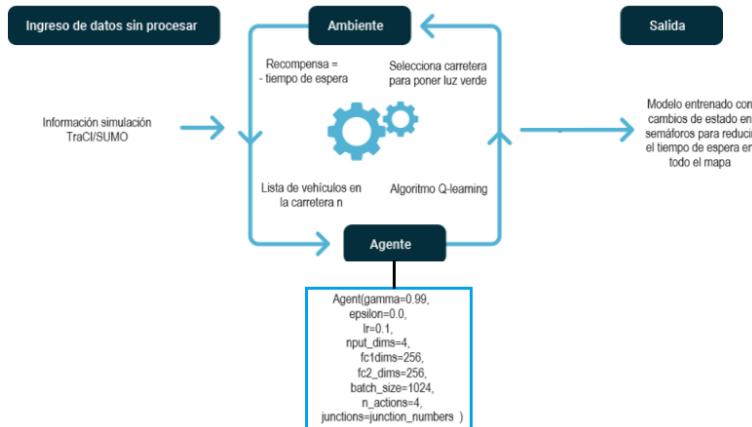


Figure 7.4: Aprendizaje por refuerzo para reducir el timepo de espera en una simulación TraCI/SUMO.

1. Descargar el modelo de entrenamiento del repositorio de GitHub.
2. Extraer la carpeta principal e ingresar a la misma. Instalar los requerimientos necesarios con el siguiente comando:

```
pip3 install -r requirements.txt
```

Dentro de los requerimientos necesarios están: `torch`, `numpy`, `matplotlib`, `sumo`, por lo que si se ha seguido las prácticas previas, solo es necesario instalar `torch` con el siguiente comando:

```
pip3 install torch
```

3. Crear los archivos necesarios para la simulación, estos son los archivos de rutas y de red. Se pueden usar los archivos de las prácticas previas. Para el caso del archivo de red `.net` se puede utilizar el obtenido con `OsmWizard` y tomarlo como entrada para el script `randomTrips.py` y así obtener las rutas. El comando siguiente logra esto:

```
python randomTrips.py -n network.net.xml -r routes.rou.xml -e 500
```

Esto creará un archivo `route.rou.xml` para 500 pasos de simulación para la red `network.net.xml`.

4. Proporcionar los archivos de red y ruta como entrada al archivo de configuración de SUMO con extensión `.sumocfg`. Por ejemplo:

```
<input>
    <net-file value='maps/city1.net.xml'/>
    <route-files value='maps/city1.rou.xml' />
</input>
```

Donde, es necesario indicar la ruta de los archivos que se quiera simular.

5. Finalmente, se puede correr el modelo de entrenamiento. Esto se logra con el script `train.py` y además es necesario indicar el nombre del modelo, que será un archivo de salida binario y además es necesario indicar el número de epoch a correr en la simulación.

```
python3 train.py --train -e 50 -m model_name -s 500
```

Donde, `-e` es para establecer las épocas, `-m` para dar nombre al modelo que se guardará en la carpeta del modelo, `-s` le dice a la simulación que se ejecute durante 500 pasos, `-train` le dice a `train.py` que entrene el modelo. Si no se especifica, cargará `model_name` desde la carpeta del modelo.

El modelo de entrenamiento cuenta con varias funciones dentro de su codificación, tal como se indica en la Figura 7.5.

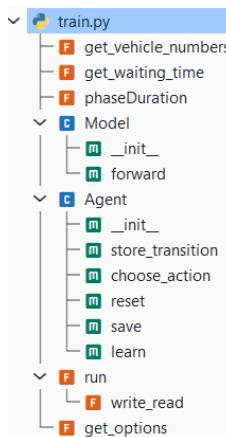


Figure 7.5: Funciones del programa de entrenamiento `train.py`

Donde, se observa que cuenta con tres funciones auxiliares (Figura 7.6), las cuales trabajan con la comunicación TraCI/SUMO para obtener y manejar los datos de la simulación: `get_vehicles_numbers`, `get_waiting_time` y `phase_duration`

```

def get_vehicle_numbers(lanes):
    vehicle_per_lane = dict()
    for l in lanes:
        vehicle_per_lane[l] = 0
        for k in traci.lane.getLastStepVehicleIDs(l):
            if traci.vehicle.getLanePosition(k) > 10:
                vehicle_per_lane[l] += 1
    return vehicle_per_lane

def get_waiting_time(lanes):
    waiting_time = 0
    for lane in lanes:
        waiting_time += traci.lane.getWaitingTime(lane)
    return waiting_time

def phaseDuration(junction, phase_time, phase_state):
    traci.trafficlight.setRedYellowGreenState(junction, phase_state)
    traci.trafficlight.setPhaseDuration(junction, phase_time)

```

Figure 7.6: Funciones auxiliares de train.py.

La primera de estas funciones, `get_vehicles_numbers`, devuelve el número de vehículos que circulan por cada carretera del mapa. Para lograr esto se hace uso de dos funciones disponibles en TraCI para los módulos `lane` y `vehicle`, estas funciones son: `traci.lane.getLastStepVehicleIDs(self, edgeID)`, la cual devuelve los identificadores de los vehículos del último paso de tiempo en el borde dado y; `traci.vehicle.getLanePosition(k)`, la cual retorna la posición del vehículo a lo largo del carril medida en metros.

La segunda función, `get_waiting_time`, devuelve el tiempo total de espera en cada carril del mapa. Para esto, se hace uso de la función `getWaitingTime` del módulo `lane`.

La tercera función, `phase_duration`, se encarga de cambiar la configuración de las TLS del mapa. Donde, es necesario realizar la configuración de las TLS de cada intersección del mapa, para esto se hace uso del módulo `trafficlight` y de sus funciones `setRedYellowGreenState(self, tlsID, state)` y `setPhaseDuration(self, tlsID, phaseDuration)`. La primera establece el estado del TL nombrado como una tupla de definiciones de luz de rugGyYuoO, para rojo, rojo-amarillo, verde, amarillo, apagado, donde las letras minúsculas significan que el flujo tiene que desacelerar. La segunda establece la duración de la fase restante de la fase actual en segundos. Este valor no tiene efecto en las repeticiones posteriores de esta fase.

Luego, dentro del programa se define la función del modelo que se empleará para realizar el entrenamiento de los datos de entrada. La Figura 7.7 ilustra la codificación de este apartado. Donde, se define el valor de las dimensiones de entrada, de la función 1 y 2, así como el número de acciones que tiene el modelo. Además, realiza una transformación lineal a los datos entrantes, en este caso se hace tres transformaciones lineales, desde la dimensión de entrada hasta el número de acciones. Luego, define

el tipo de optimizador que se usará, tipo de pérdidas como error medio cuadrático y define el dispositivo el tensor torch en cuda si está disponible o sino usará el cpu. Finalmente, en la función `forward`, realiza las convoluciones usando la rede neuronal de torch (`torch.nn.functional`), aplicando la función unitaria lineal rectificada por elementos (`F.relu`).

```
class Model(nn.Module):
    def __init__(self, lr, input_dims, fc1_dims, fc2_dims, n_actions):
        super(Model, self).__init__()
        self.lr = lr
        self.input_dims = input_dims
        self.fc1_dims = fc1_dims
        self.fc2_dims = fc2_dims
        self.n_actions = n_actions

        self.linear1 = nn.Linear(self.input_dims, self.fc1_dims)
        self.linear2 = nn.Linear(self.fc1_dims, self.fc2_dims)
        self.linear3 = nn.Linear(self.fc2_dims, self.n_actions)

        self.optimizer = optim.Adam(self.parameters(), lr=self.lr)
        self.loss = nn.MSELoss()
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.to(self.device)

    def forward(self, state):
        x = F.relu(self.linear1(state))
        x = F.relu(self.linear2(x))
        actions = self.linear3(x)
        return actions
```

Figure 7.7: Función del modelo de entrenamiento de `train.py`.

Luego, dentro del programa se define la función del agente que se empleará para realizar el entrenamiento de los datos de entrada. La codificación comienza con la inicialización de las constantes para el modelo den entrenamiento, tales como: gama, epsilon, lr, dimensiones de entrada y de las funciones, tamaño del grupo total, número de acciones, intersecciones y tamaño de memoria. Estos valores los ingresa a la evaluación del modelo (Q learning). Finalmente, es necesario almacenar todos los datos en la memoria del entrenador, utilizando un diccionario (`memory`). Además, es necesario e importante inicializar los vectores de memoria de: estado, nuevo estado, recompensa, acción, termino de acción, y contadores con el tamaño respectivo.

```

class Agent:
    def __init__(
        self,
        gamma,
        epsilon,
        lr,
        input_dims,
        fc1_dims,
        fc2_dims,
        batch_size,
        n_actions,
        junctions,
        max_memory_size=100000,
        epsilon_dec=5e-4,
        epsilon_end=0.05,
    ):
        self.gamma = gamma
        self.epsilon = epsilon
        self.lr = lr
        self.batch_size = batch_size
        self.input_dims = input_dims
        self.fc1_dims = fc1_dims
        self.fc2_dims = fc2_dims
        self.n_actions = n_actions
        self.action_space = [i for i in range(n_actions)]
        self.junctions = junctions
        self.max_mem = max_memory_size
        self.epsilon_dec = epsilon_dec
        self.epsilon_end = epsilon_end
        self.mem_cntr = 0
        self.iter_cntr = 0
        self.replace_target = 100
        self.Q_eval = Model(
            self.lr, self.input_dims, self.fc1_dims, self.fc2_dims,
            self.n_actions
        )
        self.memory = dict()
        for junction in junctions:
            self.memory[junction] = {
                "state_memory": np.zeros(
                    (self.max_mem, self.input_dims), dtype=np.float32
                ),
                "new_state_memory": np.zeros(
                    (self.max_mem, self.input_dims), dtype=np.float32
                ),
                "reward_memory": np.zeros(self.max_mem, dtype=np.float32),
                "action_memory": np.zeros(self.max_mem, dtype=np.int32),
                "terminal_memory": np.zeros(self.max_mem, dtype=np.bool),
                "mem_cntr": 0,
                "iter_cntr": 0,
            }
    }

    def store_transition(self, state, state_, action,reward, done,junction):
        index = self.memory[junction]["mem_cntr"] % self.max_mem
        self.memory[junction][ "state_memory"] [index] = state
        self.memory[junction][ "new_state_memory"] [index] = state_
        self.memory[junction][ "reward_memory"] [index] = reward
        self.memory[junction][ "terminal_memory"] [index] = done
        self.memory[junction][ "action_memory"] [index] = action
        self.memory[junction][ "mem_cntr"] += 1

```

Luego, se tiene una función que almacena los datos de transición del modelo de en-

trenamiento. Donde, en la memoria general creada almacena los datos de: memoria máxima, estado, estado previo, recompensa, si la acción se realizó, la acción y el conteo de memoria.

Luego, se define una función para escoger la acción a realizar. Donde, se tiene como entrada la observación del entrenador, esta observación es la lectura del estado previo. En primera instancia se define la acción como un tensor. Luego verifica el valor de epsilon para poder evaluar el estado de observación, con esto es posible definir la acción a realizar mediante el reenvío (función `forward`) de la acción. Seguido a esto, se tiene las funciones `reset` y `save` para resetear la memoria y guardar el modelo entrenado. Finalmente, se cuenta con la función de aprendizaje del modelo de entrenamiento. Donde, se inicializa el optimizador (Adam), el grupo de intersecciones, el estado del grupo de intersecciones y su nuevo estado, así como la recompensa, y si se realizó la acción. Todos estos parámetros tienen que ser evaluados por el aprendizaje Q, para luego obtener los valores evaluados del estado actual y del próximo estado del grupo de intersecciones, reiniciar el valor del grupo terminado y obtener el objetivo como la recompensa sumado el gamma definido por el valor máximo del valor Q próximo. También obtiene el valor de la pérdida y continúa en un paso el optimizador del algoritmo.

```

def choose_action(self, observation):
    state = torch.tensor([observation],
                         dtype=torch.float).to(self.Q_eval.device)
    if np.random.random() > self.epsilon:
        actions = self.Q_eval.forward(state)
        action = torch.argmax(actions).item()
    else:
        action = np.random.choice(self.action_space)
    return action

def reset(self, junction_numbers):
    for junction_number in junction_numbers:
        self.memory[junction_number]['mem_cntr'] = 0

def save(self, model_name):
    torch.save(self.Q_eval.state_dict(), f'models/{model_name}.bin')

def learn(self, junction):
    self.Q_eval.optimizer.zero_grad()

    batch= np.arange(self.memory[junction]['mem_cntr'], dtype=np.int32)

    state_batch = torch.tensor(self.memory[junction]["state_memory"][[batch]]).to(
        self.Q_eval.device
    )
    new_state_batch = torch.tensor(
        self.memory[junction]["new_state_memory"][[batch]]
    ).to(self.Q_eval.device)
    reward_batch = torch.tensor(
        self.memory[junction]['reward_memory'][[batch]]).to(self.Q_eval.device)
    terminal_batch = torch.tensor(self.memory[junction]['terminal_memory'][[batch]]).to(self.Q_eval.device)
    action_batch = self.memory[junction]['action_memory'][[batch]]
    q_eval = self.Q_eval.forward(state_batch)[batch, action_batch]
    q_next = self.Q_eval.forward(new_state_batch)
    q_next[terminal_batch] = 0.0
    q_target = reward_batch + self.gamma * torch.max(q_next, dim=1)[0]
    loss = self.Q_eval.loss(q_target, q_eval).to(self.Q_eval.device)

    loss.backward()
    self.Q_eval.optimizer.step()

    self.iter_cntr += 1
    self.epsilon = (
        self.epsilon - self.epsilon_dec
        if self.epsilon > self.epsilon_end
        else self.epsilon_end
    )

```

Finalmente, se cuenta con la función run, la cual se encarga de iniciar la comunicación TraCI/SUMO, leyendo el archivo de configuración respectivo. También, es la encargada de extraer los IDs de todas las intersecciones del mapa con la función `getIDList()` del módulo `trafficlight`. En este punto se crea el agente, llamando a la función `agent` e ingresando los valores correspondientes para su creación. Esta función se encarga de verificar si la opción de entrenamiento está activada, si es así, comienza la comunicación TraCI/SUMO e inicializa las variables de tiempo de simulación, así como los diccionarios para: tiempo de semáforos, tiempo de espera previo, vehículos previos en cada calle, acción previa y el conjunto de todas las calles; y las opciones de configuración de los semáforos.

7.4 Instrucciones

```
def run(train=True,model_name="model",epochs=50,steps=500,ard=False):
    if ard:
        arduino = serial.Serial(port='COM4', baudrate=9600, timeout=.1)
        def write_read(x):
            arduino.write(bytes(x, 'utf-8'))
            time.sleep(0.05)
            data = arduino.readline()
            return data
    """execute the TraCI control loop"""
    epochs = epochs
    steps = steps
    best_time = np.inf
    total_time_list = list()
    traci.start(
        [checkBinary("sumo"), "-c", "configuration.sumocfg", "--tripinfo-output", "maps/tripinfo.xml"]
    )
    all_junctions = traci.trafficlight.getIDList()
    junction_numbers = list(range(len(all_junctions)))

    brain = Agent(
        gamma=0.99,
        epsilon=0.0,
        lr=0.1,
        input_dims=4,
        # input_dims = len(all_junctions) * 4,
        fc1_dims=256,
        fc2_dims=256,
        batch_size=1024,
        n_actions=4,
        ijunctions=junction_numbers,
    )
    if not train:
        brain.Q_eval.load_state_dict(torch.load(f'models/{model_name}.bin',map_location=brain.Q_eval.device))

    print(brain.Q_eval.device)
    traci.close()
    for e in range(epochs):
        if train:
            traci.start(
                [checkBinary("sumo"), "-c", "configuration.sumocfg", "--tripinfo-output", "tripinfo.xml"]
            )
        else:
            traci.start(
                [checkBinary("sumo-gui"), "-c", "configuration.sumocfg", "--tripinfo-output", "tripinfo.xml"]
            )

        print(f"epoch: {e}")
        select_lane = [
            ["yyyyyyyyyy", "GGGrrrrrrrr"], 
            ["rrrryyyyyyyy", "rrrGGGrfffff"], 
            ["rrrrrrrryyrr", "rrrrrrGGGrrr"], 
            ["rrrrrrrrrryy", "rrrrrrrrrGGG"]
        ]
        step = 0
        total_time = 0
        min_duration = 5

        traffic_lights_time = dict()
        prev_wait_time = dict()
        prev_vehicles_per_lane = dict()
        prev_action = dict()
        all_lanes = list()
```

También, se encarga de inicializar los vectores para: tiempo de espera previo, acción previa, tiempo de los semáforos, vehículos previos por carretera y obtener todas las car-

reteras que son controlados por semáforos con la función `getControlledLanes(idLane)` del módulo `trafficlight`. Así, con todas las variables configuradas, se inicial el bucle de comunicación TraCI/SUMO, donde para cada pasa de simulación se obtiene las carreteras que son controladas por un semáforo, así como el tiempo total de espera por carretera, utilizando la función auxiliar `get_waiting_time` y acumula el tiempo total de espera de todo el mapa. Luego, si el tiempo del semáforo en análisis es 0, cuenta el número de vehículos en esa carretera y almacena el estado previo y el estado actual, indicando que la recompensa es el negativo del tiempo de espera de esa carretera ($reward = -1 * waiting_time$), además almacena la transición del agente con los valores indicados. Luego, selecciona la nueva acción, basándose en el estado actual, donde indica el tiempo de configuración de los semáforos usando la función auxiliar `phaseDuration`. Al final crea una lógica para configurar el encendido y apagado de unos LEDs a través de un arduino (ver en [7]).

7.4 Instrucciones

```

for junction_number, junction in enumerate(all_junctions):
    prev_wait_time[junction] = 0
    prev_action[junction_number] = 0
    traffic_lights_time[junction] = 0
    prev_vehicles_per_lane[junction_number] = [0] * 4
    # prev_vehicles_per_lane[junction_number] = [0] * (len(all_junctions) * 4)
    all_lanes.extend(list(traci.trafficlight.getControlledLanes(junction)))

while step <= steps:
    traci.simulationStep()
    for junction_number, junction in enumerate(all_junctions):
        controled_lanes = traci.trafficlight.getControlledLanes(junction)
        waiting_time = get_waiting_time(controled_lanes)
        total_time += waiting_time
        if traffic_lights_time[junction] == 0:
            vehicles_per_lane = get_vehicle_numbers(controled_lanes)
            # vehicles_per_lane = get_vehicle_numbers(all_lanes)

        #storing previous state and current state
        reward = -1 * waiting_time
        state_ = list(vehicles_per_lane.values())
        state = prev_vehicles_per_lane[junction_number]
        prev_vehicles_per_lane[junction_number] = state_
        brain.store_transition(state, state_, prev_action[junction_number], reward, (step==steps), junction_number)

        #selecting new action based on current state
        lane = brain.choose_action(state_)
        prev_action[junction_number] = lane
        phaseDuration(junction, 6, select_lane[lane][0])
        phaseDuration(junction, min_duration + 10, select_lane[lane][1])
        if ard:
            ph = str(traci.trafficlight.getPhase("0"))
            value = write_read(ph)

        traffic_lights_time[junction] = min_duration + 10
        if train:
            brain.learn(junction_number)
        else:
            traffic_lights_time[junction] -= 1
        step += 1
    print("total_time",total_time)
    total_time_list.append(total_time)

    if total_time < best_time:
        best_time = total_time
    if train:
        brain.save(model_name)

    traci.close()
    sys.stdout.flush()
    if not train:
        break
if train:
    plt.plot(list(range(len(total_time_list))),total_time_list)
    plt.xlabel("epochs")
    plt.ylabel("total time")
    plt.savefig(f'plots/time_vs_epoch_{model_name}.png')
    plt.show()

```

6. Al final de la simulación, mostrará la gráfica time vs epoch y se guardará en la carpeta plots con el nombre time_vs_epoch_{model_name}.png. En la Figura ?? se ilustra el resultado del algoritmo de reinforcement learning, luego de simular 100 épocas sobre el mapa de prueba city1.net.xml.

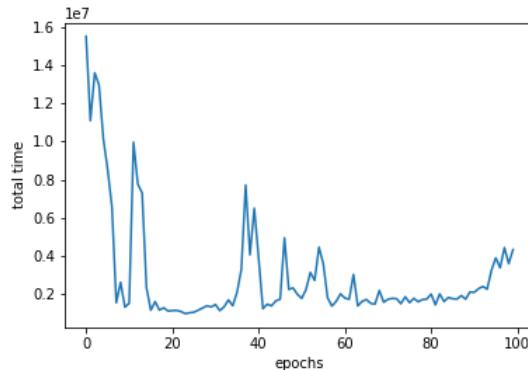


Figure 7.8: Gráfica: Tiempo vs epoch resultado del reinforcement learning en el mapa city1.net.xml.

7. Puede utilizar train.py para ejecutar un modelo previamente entrenado en la GUI.

```
python train.py -m model_name -s 500
```

Esto abrirá la GUI que puede ejecutar para ver cómo funciona su modelo. Para obtener resultados precisos, establezca un valor de -s igual para pruebas y entrenamiento. En la Figura 7.9 se ilustra la diferencia de tiempos de espera logrados en simulaciones con y sin entrenamiento.

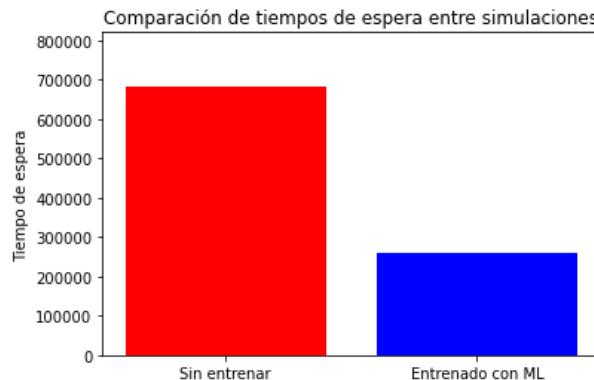
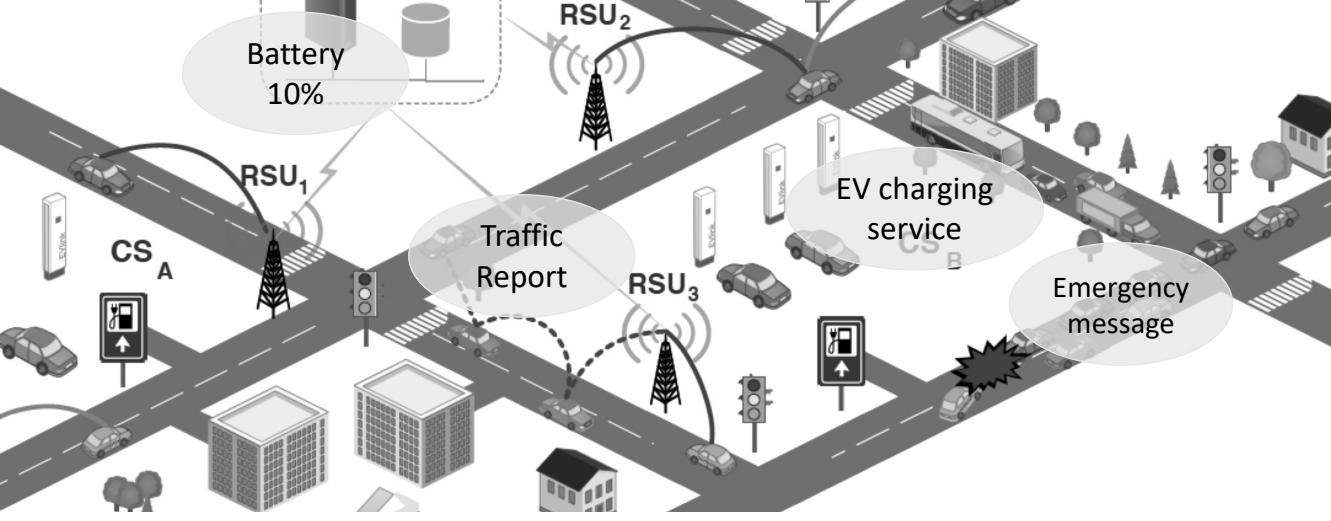


Figure 7.9: Comparación de tiempos de espera de modelos con y sin entrenamiento ML.



Bibliography

- [1] P. Barbecho Bautista, L. F. Urquiza-Aguiar, and M. Aguilar Igartua, “Stgt: Sumo-based traffic mobility generation tool for evaluation of vehicular networks,” in *Proceedings of the 18th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, 2021, pp. 17–24.
- [2] P. Barbecho Bautista, L. Urquiza Aguiar, and M. Aguilar Igartua, “How does the traffic behavior change by using sumo traffic generation tools,” *Computer Communications*, vol. 181, pp. 1–13, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0140366421003601>
- [3] “Netconvert,” <https://sumo.dlr.de/docs/netconvert.html>, accessed: 2020-10-06.
- [4] “Polyconvert,” <https://sumo.dlr.de/docs/polyconvert.html>, accessed: 2020-10-06.
- [5] “Random trips,” <https://sumo.dlr.de/docs/Tools/Trip.html>, accessed: 2020-10-06.
- [6] SUMO, “Vehicle Type Parameter Defaults,” 2024. [Online]. Available: https://sumo.dlr.de/docs/Vehicle_Type_Parameter_Defaults.html
- [7] D. Maunish, “Dynamic-Traffic-light-management-system,” 2021. [Online]. Available: <https://github.com/Maunish-dave/Dynamic-Traffic-light-management-system/tree/main>