

Manual de laboratorio: SUMO

Erick Pérez P.
erickpatriciopp9@gmail.com

Este manual tiene como finalidad ser una guía integral para una implementación de simulación de redes vehiculares. El documento incluye una serie de prácticas detalladas destinadas a ser aplicadas con software de código abierto. Cada práctica proporciona los pasos necesarios para lograr los objetivos establecidos. En última instancia, el objetivo es establecer una simulación de una red vehicular implementando SUMO, TraCI y un modelo de aprendizaje de máquina para entender y predecir el comportamiento del tráfico vehicular.

Índice general

1. Guías de laboratorio	2
1.1. Práctica: Instalación de SUMO mediante archivos binarios	3
1.2. Práctica: Simulación de tráfico vehicular en SUMO	9
1.3. Práctica: Simulación de tráfico vehicular en SUMO. Generación de archivos con WebWizard	18
1.4. Práctica: Generación de estadísticas de la simulación en SUMO	23
1.5. Práctica: Introducción a TraCI con SUMO	39
1.6. Práctica: Simulación de tráfico vehicular TraCI/SUMO	43
1.7. Práctica: Simulación de tráfico vehicular TraCI/SUMO. Contador de vehículos	46
1.8. Práctica: Simulación de tráfico vehicular TraCI/SUMO. Cálculo de velocidad promedio	49
1.9. Práctica: Simulación de tráfico vehicular TraCI/SUMO. Cálculo de emisiones CO ₂	53
1.10. Práctica: Simulación de tráfico vehicular TraCI/SUMO. Cambio de trayectoria de vehículos	57
1.11. Práctica: Simulación de manejo de TLS para tráfico vehicular TraCI/-SUMO. Reinforcement Learning	61
1.12. Práctica: Simulación de tráfico vehicular SUMO. Clasificación de datos con machine learning	74

Capítulo 1

Guías de laboratorio

1.1. Práctica: Instalación de SUMO mediante archivos binarios

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Descargar e instalar el software de Simulación de Movilidad Urbana, SUMO.
- Utilizar los archivos binarios para construir el simulador sobre el sistema operativo Linux en la distribución Ubuntu 22.04.3 LTS.

Introducción

SUMO (Simulación de Movilidad Urbana), es un paquete de simulación de tráfico multimodal continuo, microscópico, altamente portátil y de código abierto diseñado para manejar grandes redes. SUMO es una suite de simulación de tráfico gratuita y de código abierto. Está disponible desde 2001 y permite modelar sistemas de tráfico intermodal, incluidos vehículos de carretera, transporte público y peatones. SUMO incluye una gran cantidad de herramientas de soporte que automatizan tareas centrales para la creación, ejecución y evaluación de simulaciones de tráfico, como la importación de redes, cálculos de rutas, visualización y cálculo de emisiones. SUMO se puede mejorar con modelos personalizados y proporciona varias API para controlar la simulación de forma remota.

SUMO se ha utilizado en varios proyectos para responder una gran variedad de preguntas de investigación:

- Evaluar el desempeño de los semáforos, incluyendo la evaluación de algoritmos modernos hasta la evaluación de planes de cronometraje semanales.
- Se ha investigado la elección de rutas de los vehículos, incluido el desarrollo de nuevos métodos, la evaluación de rutas ecológicas basadas en emisiones contaminantes e investigaciones sobre las influencias en toda la red en la elección de rutas autónomas.
- SUMO se utilizó para proporcionar previsiones de tráfico a las autoridades de la ciudad de Colonia durante la visita del Papa en 2005 y durante la Copa Mundial de Fútbol de 2006.
- SUMO se utilizó para respaldar el comportamiento telefónico simulado en vehículos para evaluar el rendimiento de la vigilancia del tráfico basada en GSM.

- SUMO es ampliamente utilizado por la comunidad V2X para proporcionar seguimientos realistas de vehículos y para evaluar aplicaciones en un bucle en línea con un simulador de red.
- Entrenamiento de IA de planes de semáforos.
- Simulación de los efectos del tráfico de vehículos y peatones autónomos.
- Simulación y validación de la función de conducción autónoma en cooperación con otros simuladores.
- Simulación del tráfico de aparcamientos.
- Simulación del tráfico ferroviario para el despacho de vehículos basado en IA.
- Seguridad vial y análisis de riesgos.
- Cálculo de emisiones (ruido y contaminantes).

Materiales

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

Instrucciones

A continuación, se describen los pasos necesarios para lograr la descarga e instalación de SUMO sobre el sistema operativo Ubuntu 22.04.3 LTS.

1. Actualizar los paquetes del sistema operativo.

```
sudo apt-get update
```

2. Instalar las herramientas y bibliotecas necesarias para el funcionamiento de SUMO. Estas herramientas son:

- Para la infraestructura de construcción necesitará cmake junto con un cliente g++ o clang (o cualquier otro compilador habilitado C-11).
- La biblioteca Xerces-C siempre es necesaria. A utilizar sumo-gui también necesitas Fox Toolkit en versión 1.6.x. Es recomendable también instalar Proj para tener soporte para la geoconversión y referencias. Otro requisito común es Importación de red de shapefile (arcgis). Esto requiere la biblioteca GDAL. Para compilar necesitará las versiones de desarrollo de todos los paquetes. Para openSUSE esto significa instalar libxerces-c-devel, libproj-devel, libgdal-devel, y Foxx16-devel. Para ubuntu la llamada está por encima. La instalación de swig, python3-dev y el jdk permite también la construcción de libsumo mientras que eigen3 es necesario para el modelo overheadwire.

```
sudo apt-get 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
```

- ccache (para acelerar las construcciones)
- ffmpeg-devel (para la salida de vídeo),
- libOpenSceneGraph-devel (para la GUI experimental en 3D),
- gtest (para pruebas de unidad, no utilice 1.13 o posterior)
- gettext (para la internacionalización)
- texttest, xvfb y tkdiff (para los ensayos de aceptación)
- copos, un estilo y autopep para la comprobación de estilo.

```
sudo apt-get install ccache libavformat-dev libswscale-dev
    libopenscenegraph-dev python3-pip python3-setuptools

sudo apt-get install libgtest-dev gettext tkdiff xvfb flake8
    astyle python3-autopep8 pip3 install texttest

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

- Llamar a las herramientas desde netedit requiere una lista de paquetes Python para generar plantillas durante la compilación. Muchos de ellos podrían estar disponibles con el administrador de paquetes de su distribución y la mayoría de las veces que preferimos para usarlos. Para ubuntu esto significa actualmente, usted debe hacer primero:

```
sudo apt-get install python3-pyproj python3-rtree python3-pandas
    python3-flake8 python3-autopep8 python3-scipy python3-pulp
    python3-ezdx
```

3. Una vez instaladas las librerías necesarias, es momento de obtener el código fuente de SUMO. Esto se logra descargando los recursos existentes de sumo en el repositorio de GitHub de Eclipse SUMO.

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

4. Luego, dentro de la carpeta descargada desde el repositorio de GitHub, es necesario realizar la captura adicional de los reemplazos para obtener un historial del proyecto local. Esto se logra con los siguientes comandos.

```
cd sumo
git fetch origin refs/replace/*:refs/replace/*
```

5. Luego instale las piezas restantes utilizando pip. Es necesario estar ubicados dentro de la carpeta sumo que se ha descargado previamente, con el fin de llamar al archivo requirements.txt ubicado dentro de la carpeta de sumo.

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

Para instalar las dependencias más comunes con su gestor de paquetes en Ubuntu, puede utilizar:

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

6. Antes de compilar es recomendable definir la variable de entorno SUMO_HOME. SUMO_HOME debe ser configurado para el SUMO construye sendero desde el paso anterior. Asumiendo que SUMO se encuentra en la carpeta `home/user/sumo-version/`, si quiere definir sólo para la sesión actual el comando a utilizar es el siguiente:

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

En el caso que se encuentre dentro de la carpeta sumo descargada, el comando a utilizar es el siguiente:

```
export SUMO_HOME="$PWD"
```

Para el caso donde se requiera que la variable de entorno quede configurada permanente en el sistema, es necesario modificar el archivo `bash.bashrc` ubicado en la ruta `/etc` y el archivo `.bashrc` ubicado en la ruta `/home/<user>`. Una vez localizados los archivos, es necesario agregar el comando previo al final del archivo. Para lograr esto se pueden utilizar los siguientes comandos:

```
sudo nano /etc/bash.bashrc
sudo nano /home/<user>/.bashrc
```

Luego, en al final del archivo abierto colocar el siguiente comando:

```
export SUMO_HOME="/ruta_de_instalacion_de_sumo"
```

Donde, `ruta_de_instalacion_de_sumo` puede ser `/home/user/sumo-version/`.

```
GNU nano 6.2                               /etc/bash.bashrc
See "man sudo_root" for details.

EOF
fi
esac
fi

# if the command-not-found package is installed, use it
if [ -x /usr/lib/command-not-found -o -x /usr/share/command-not-found/command-not-found ]; then
    function command_not_found_handle {
        # check because c-n-f could've been removed in the meantime
        if [ -x /usr/lib/command-not-found ]; then
            /usr/lib/command-not-found -- "$1"
            return $?
        elif [ -x /usr/share/command-not-found/command-not-found ]; then
            /usr/share/command-not-found/command-not-found -- "$1"
            return $?
        else
            printf "%s: command not found\n" "$1" >&2
            return 127
        fi
    }
fi

export SUMO_HOME="/home/sumo/sumo"
```

Below the terminal window, a menu bar is visible with options like Ayuda, Guardar, Buscar, Cortar, Ejecutar, Salir, Leer fich., Reemplazar, Pegar, Justificar, and Ir a línea.

Figura 1.1: Archivo `bash.bashrc` de `/etc`.

```

GNU nano 6.2                               /home/sumo/.bashrc
# Add an "alert" alias for long running commands. Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$( [ $? = 0 ] && echo terminal || echo error)"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

export SUMO_HOME="/home/sumo/sumo"

```

^G Ayuda ^O Guardar ^W Buscar ^K Cortar ^I Ejecutar ^C Ubicación
^X Salir ^R Leer fich. ^Y Reemplazar ^U Pegar ^J Justificar ^/ Ir a línea

Figura 1.2: Archivo .bashrc de /home/<user>.

- Finalmente, es necesario construir los binarios SUMO con cmake. Para construir con cmake se requiere la versión 3 o superior. Crear una carpeta de construcción para cmake (en la carpeta raíz SUMO o en la descargada desde GitHub) y configurar SUMO con el conjunto completo de opciones disponibles como GDAL y Soporte OpenSceneGraph. El siguiente comando construye los binarios de SUMO:

```
cmake -B build .
```

Después de que esto haya terminado, es necesario correr el siguiente comando:

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

El comando `nproc` le da el número de núcleos lógicos en su ordenador, para que esa marca se inicie de forma paralela construir puestos de trabajo que lo haga construir mucho más rápido.

- Para correr sumo se debe ingresar en la carpeta `bin` y correr cualquiera de los dos opciones: `./sumo` para correr por consola o `./sumo-gui` para correr la interfaz gráfica.

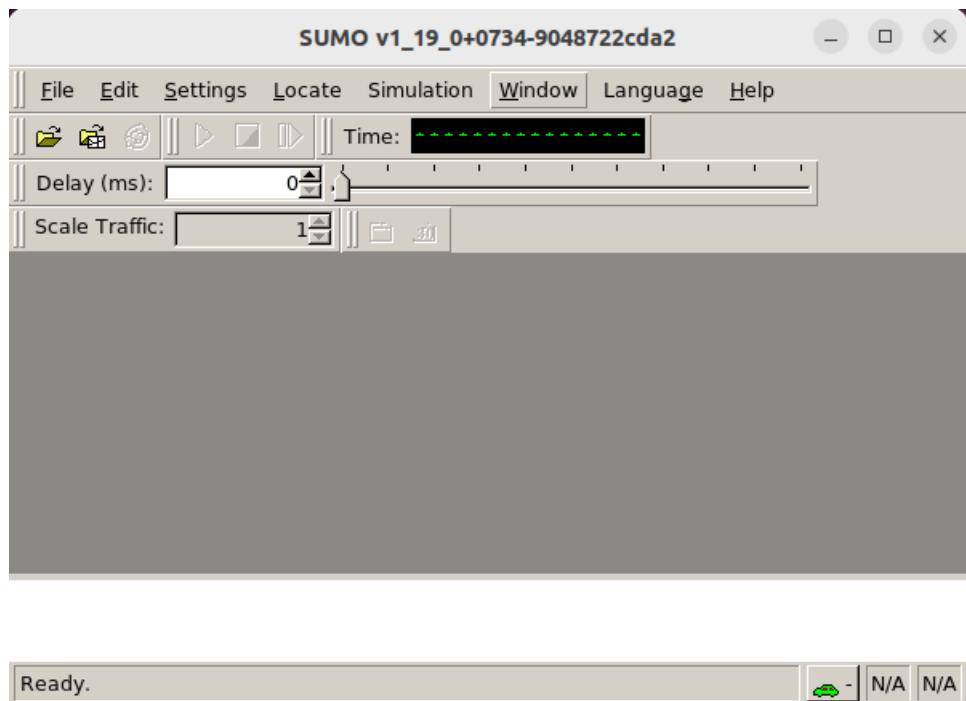


Figura 1.3: SUMO corriendo en la interfaz gráfica.

1.2. Práctica: Simulación de tráfico vehicular en SUMO

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Conocer los archivos necesarios para correr una simulación sobre SUMO.
- Simular una red vehicular sobre SUMO desde la línea de comandos.
- Simular una red vehicular sobre SUMO desde la interfaz gráfica.

Introducción

El primer paso para proceder con una simulación de tráfico vehicular es generar el escenario. SUMO funciona con Open Street Maps (OSM), por lo que la idea es descargar un mapa real desde OSM.

La Figura 1.4 muestra los pasos típicos para realizar una simulación de tráfico. Primero, tenemos que construir redes de carreteras. Luego, se genera la demanda de tráfico (es decir, la movilidad del tráfico). Por último, cuando se hayan generado los archivos requeridos, se puede ejecutar la simulación. Cada paso del flujo de trabajo utiliza una única o un conjunto de herramientas SUMO, como se muestra en la Figura 1.4.

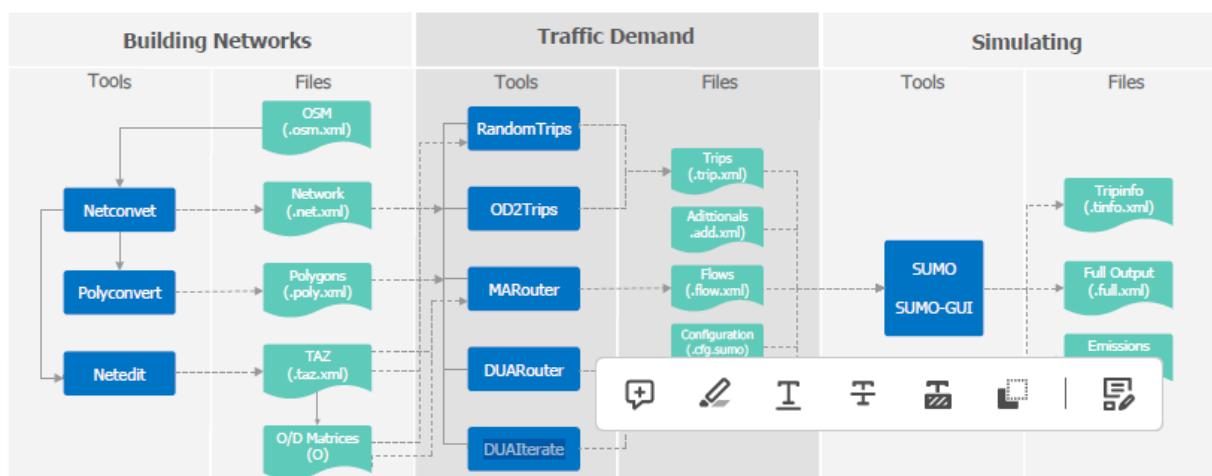


Figura 1.4: Flujo de trabajo de simulación de movilidad de tráfico. Herramientas integradas SUMO y archivos relacionados para realizar simulaciones. Tomado de [1].

Como se observa en la Figura 1.4, la simulación se puede realizar utilizando varias herramientas para construir las rutas y la movilidad. Estas herramientas son: [1]

■ **Building Networks:**

- *Netconvert*: Esta herramienta convierte un archivo de mapa de entrada (.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, sin cambios, uniones de unión, etc.).
- *Polyconvert*: Esta herramienta genera todos los polígonos (por ejemplo, edificios, terrenos, etc.) a partir de la fuente del mapa ingresada (por ejemplo, archivo osm). El archivo de salida (.poly.xml) contiene todas las formas geométricas del mapa.
- *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. Aquí pueden aparecer situaciones de congestión durante la simulación.
- *DUAIterate*: Esta herramienta utiliza un método de asignación llamado iterativo, que intenta calcular el equilibrio del usuario. Esto último significa que las rutas de los vehículos se generan con el mínimo costo (es decir, tiempo de viaje). Esto se hace llamando iterativamente a la herramienta DUARouter. Aquí 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.

Un escenario en SUMO se compone principalmente de tres archivos:

- Archivo de red ("Netconvert"): Este archivo contiene toda la información acerca del mapa, carreteras y calles, codificado para que SUMO lo entienda. Su extensión es .net.

- Archivo de polígonos ("Polyconvert"): Este archivo contiene toda la información acerca de edificios y otros obstáculos como parques que existen en el mapa. Su extensión es .poly.
- Archivo de viajes ("RandomTrips"): Este archivo contiene toda la información referente a los movimientos aleatorios, rutas, de vehículos dentro de la simulación. Su extensión es .rou.

Estos tres archivos están organizados en un archivo de configuración único, con extensión: `sumo.cfg`. En la Figura 1.5, puede ver la organización de los archivos SUMO.

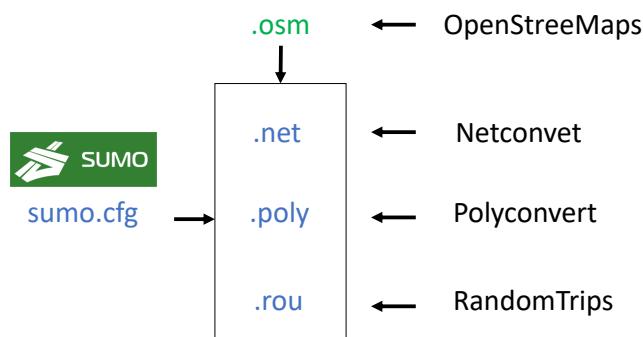


Figura 1.5: Archivos de simulación de SUMO.

Un escenario se puede generar mediante 2 métodos: una forma manual donde los archivos de simulación SUMO también se generan manualmente, y una forma rápida con la herramienta SUMO WebWizard. En la presente práctica se explica el primero de estos dos mecanismos.

Materiales

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

Instrucciones

A continuación se describen los pasos a seguir para obtener los archivos necesarios para correr una simulación.

1. Para obtener los 3 archivos de simulación, primero es necesario descargar un mapa desde la web de OSM.
 - a) Acceder al sitio web de OSM y descargar el mapa.

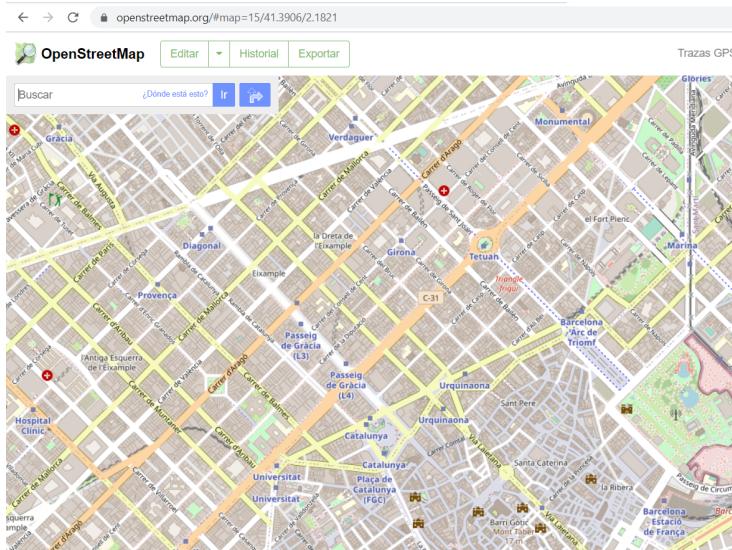


Figura 1.6: Web de OSM.

- b) Haga clic en "Exportar" y luego seleccione la opción: "*Seleccionar manualmente un área diferente*".

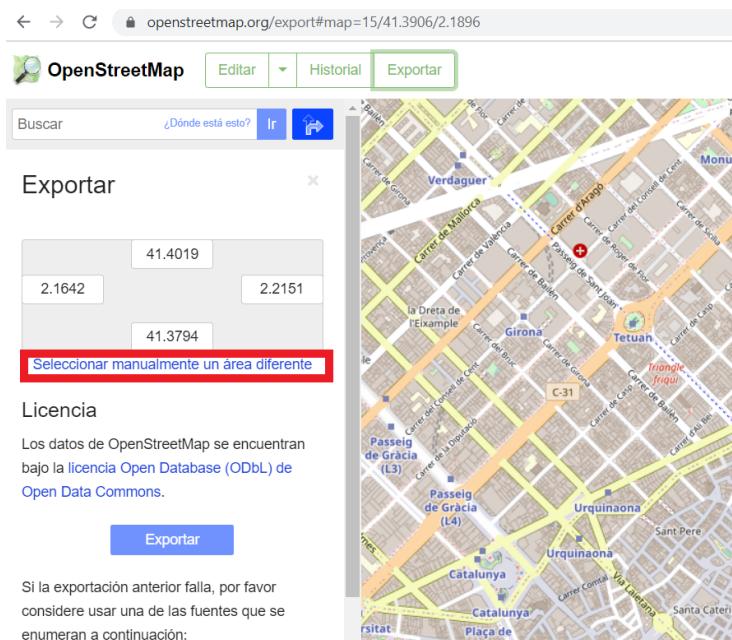


Figura 1.7: Exportando mapa.

- c) Luego se puede seleccionar cualquier área del mapa y finalmente se debe presionar el botón "Exportar". (A través de este método solo se permite exportar un número máximo de nodos, en este caso el máximo es 50000 nodos).

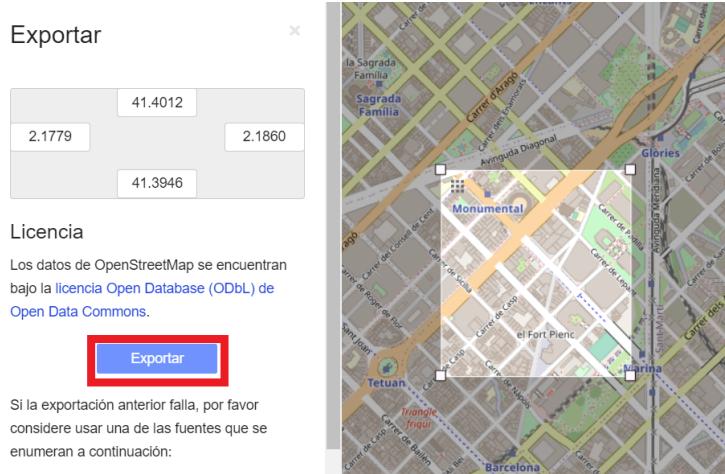


Figura 1.8: Seleccionar área y exportar.

2. Luego, descargar el archivo `osm`. Después de eso, el archivo `osm` debe colocarse en el directorio: `/ruta-de-instalacion/sumo/bin`
3. Ahora, sobre el directorio anterior, se puede invocar la herramienta `netconvert` para generar la Red de Tráfico correspondiente a un escenario urbano:

```
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 bordes que conectan los mismos nodos y están cerca entre sí.
- `-geometry.remove` reemplaza los nodos que solo definen la geometría del borde por puntos de geometría.
- `-osm-files` indica el nombre o ruta del archivo `osm` (descargado desde OSM).
- `-o` indica el nombre del archivo de salida.

La herramienta tiene muchos parámetros que se pueden agregar y modificar siguiendo las guías en [2]. La entrada es el archivo `osm` previamente descargado y como salida se genera un nuevo archivo: `bcn.net.xml`.

4. Luego se invoca la herramienta `polyconvert` (otras opciones se encuentran en [3]):

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

Donde:

- **-net-file** indica el nombre o ruta del archivo de la red vehicular (extensión **.net**).
 - **-o** indica el nombre o ruta del archivo de polígonos (edificios, parques, etc.) de salida con extensión **.poly**.
5. El siguiente archivo a generar es el archivo **RandomTrips**. Para generar este archivo se debe invocar el script de Python **randomTrips.py**. Está ubicado en el directorio: **/ruta-de-instalacion/sumo/tools**. Otras opciones están en [4].

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

Donde:

- **-n** indica el nombre o ruta del archivo de la red vehicular (extensión **.net**).
- **-o** indica el nombre o ruta del archivo de rutas de salida con extensión **.rou**.
- Además, una opción muy utilizada es la **-p**. Esta opción permite generar un tráfico en el que consten n vehículos en un tiempo dado. Este parámetro debe ser calculado con la siguiente ecuación:

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

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. Por lo tanto, reemplazando los valores de simulación en la ecuación 1.1 se obtiene el valor de la ecuación 1.2.

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

Finalmente, el comando sería el siguiente:

```
python3 randomTrips.py -p 36 -n bcn.net.xml -o bcn.rou.xml
```

6. Finalmente, es necesario crear el archivo de configuración de sumo. Como se mencionó previamente, este archivo es el que contiene a todos los archivos necesarios para la simulación. Es decir, a los archivos con extensiones: **.net**, **.poly** y **.rou**.

El archivo **sumo.cfg** como el que se muestra en Código 1.1 es el que manejará la configuración de cada escenario. En el ejemplo que se muestra a continuación vemos los ajustes de configuración para una simulación. Como podemos observar, el expediente se divide en tres secciones: entrada, procesamiento y tiempo. La sección de entrada establecerá cuáles son los archivos necesarios para realizar la simulación, incluidos los archivos de red y de ruta. La sección de procesamiento establecerá valores tales como el de **tiempo de teletransportación** que establece el umbral del tiempo máximo que un vehículo puede permanecer frente a una intersección antes de ser teletransportado al siguiente borde libre de la ruta, en este caso ignoramos los errores que pueden ocasionar las rutas de

ciertos vehículos. Finalmente, la sección de tiempo establece la hora de inicio y finalización y la duración de los pasos de tiempo.

```
<?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>
```

Código 1.1: Archivo **.sumo.cfg.**

Parte 1: Correr simulación desde la línea de comandos.

A continuación, se describen los pasos necesarios para correr una simulación de tráfico sobre SUMO utilizando la línea de comandos.

1. Para correr la simulación de SUMO mediante la línea de comandos, se requiere correr el script denominado **sumo** ubicado en la carpeta **bin** de SUMO.
2. Una vez ubicado el script, es necesario indicar el archivo de configuración con extensión **.sumo.cfg** que deseamos correr. La opción para indicar el archivo de configuración es **-c**. El siguiente comando puede ser utilizado para correr la simulación.

```
./ ruta-de-instalacion-sumo/bin/sumo -c bcn.sumo.cfg
```

O en su defecto, si la variable de entorno ya está debidamente configurada, se puede llamar al script de **sumo** desde cualquier ruta con el siguiente comando:

```
sumo -c /ruta-de-archivo-cfg/bcn.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 '45429539#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 '399760675#0' and edge '256907298#0' found.
Warning: No route for vehicle '42' found.
Warning: No connection between edge '400449634#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 '535309213#0' and edge '48859314#3' found.
Warning: No route for vehicle '132' found.
Warning: No connection between edge '400463706#1' and edge '334948974#2' found.
Warning: No route for vehicle '144' found.
Warning: No connection between edge '640993657#1' and edge '311212627#3' found.
Warning: No route for vehicle '178' found.
Warning: No connection between edge '45429545#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 '-972644144#4' found.
```

Figura 1.9: Simulación en marcha desde la línea de comandos de Linux.

Parte 2: Correr simulación desde la interfaz gráfica.

A continuación, se describen los pasos necesarios para correr una simulación de tráfico sobre SUMO utilizando la interfaz gráfica.

1. Una vez que se cuenta con los 4 archivos necesarios para simular el tráfico vehicular sobre SUMO, lo primero que se tiene que hacer es correr el script con nombre `sumo-gui` ubicado en la carpeta `bin`.
2. Luego, cuando la interfaz gráfica esté abierta es necesario cargar el archivo de configuración de SUMO. Recordando que este archivo tiene la extensión `.sumo.cfg`. En la parte superior izquierda, dar clic en la opción *File/Open Simulation* y seleccionar el archivo de configuración.

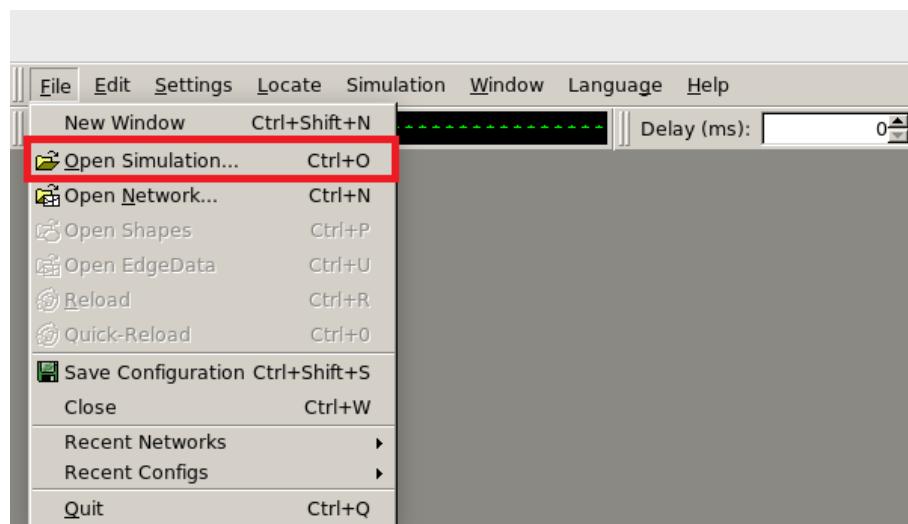


Figura 1.10: Open Simulation.

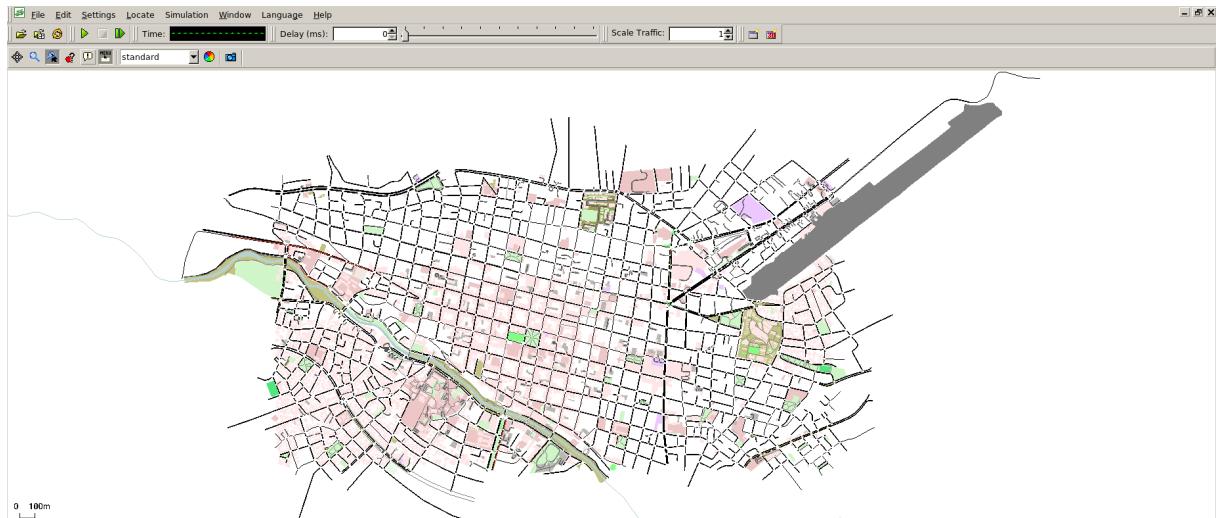


Figura 1.11: Archivo de configuración cargado.

- Finalmente, para correr la simulación se debe tener en cuenta el retardo de la misma. Esto se configura en la parte superior (recuadro rojo de la Figura 1.13) del simulador. Luego, es necesario dar clic en la opción *Run* (recuadro azul de la Figura 1.13) y la simulación estará en marcha.

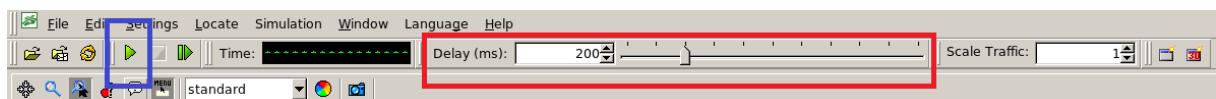


Figura 1.12: Controles del simulador SUMO.

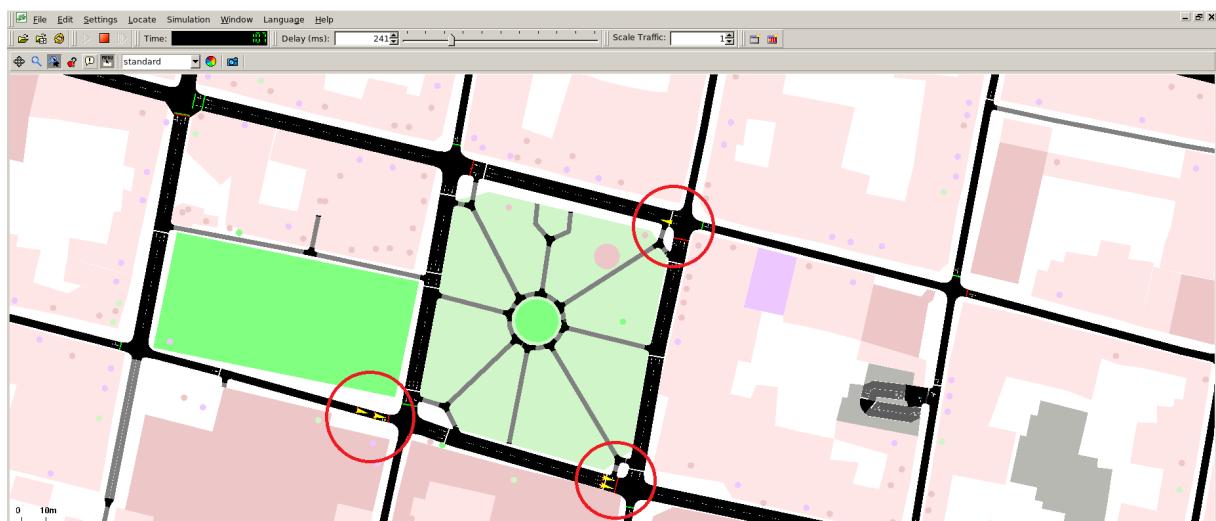


Figura 1.13: Simulación en marcha desde la interfaz gráfica de SUMO.

1.3. Práctica: Simulación de tráfico vehicular en SUMO. Generación de archivos con WebWizard

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Generar los archivos necesarios para correr una simulación sobre SUMO con la herramienta WebWizard.
- Simular una red vehicular sobre SUMO.

Introducción

En el ámbito de las redes vehiculares y la simulación de tráfico, el software SUMO se ha convertido en una herramienta fundamental para investigadores, ingenieros y profesionales que buscan comprender y mejorar la movilidad urbana y vehicular. SUMO proporciona un entorno flexible y poderoso para modelar el flujo de vehículos, peatones y otros elementos de transporte en entornos urbanos y de carretera.

Dentro del ecosistema de SUMO, una herramienta particularmente útil y accesible es el OSMWebWizard. Este componente permite a los usuarios aprovechar los datos geoespaciales disponibles en OpenStreetMap (OSM) para generar escenarios de simulación realistas y detallados. OpenStreetMap es una iniciativa de mapeo colaborativo que proporciona datos geoespaciales de alta calidad y de código abierto, lo que lo convierte en una fuente valiosa para la creación de escenarios de simulación en SUMO.

OSMWebWizard simplifica significativamente el proceso de importación y conversión de datos de OpenStreetMap en formatos compatibles con SUMO, permitiendo a los usuarios generar rápidamente modelos precisos de redes vehiculares y entornos urbanos. Esta herramienta ofrece una interfaz gráfica intuitiva 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.

En esta práctica, exploraremos en cómo utilizar el OSMWebWizard para crear escenarios de redes vehiculares en SUMO a partir de datos de OpenStreetMap. Desde la importación inicial de datos geográficos hasta la configuración de simulaciones específicas, esta herramienta ofrece una solución integral para la creación eficiente de entornos de simulación realistas y precisos.

Materiales

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

Instrucciones

A continuación se describen los pasos a seguir para obtener los archivos necesarios para correr una simulación de tráfico vehicular sobre SUMO. En esta práctica se hace uso de la herramienta WebWizard de SUMO.

- Una forma más sencilla de obtener los archivos de sumo necesarios es también a través del SUMO WebWizard. La herramienta está en la ruta: `ruta-de-instalacion/sumo/tools`.
- Invoque la herramienta: `osmWebWizard.py`. Luego se abrirá un navegador web con el mapa de osm. Se puede modificar la ubicación y luego se debe seleccionar la opción *Seleccionar Área*.

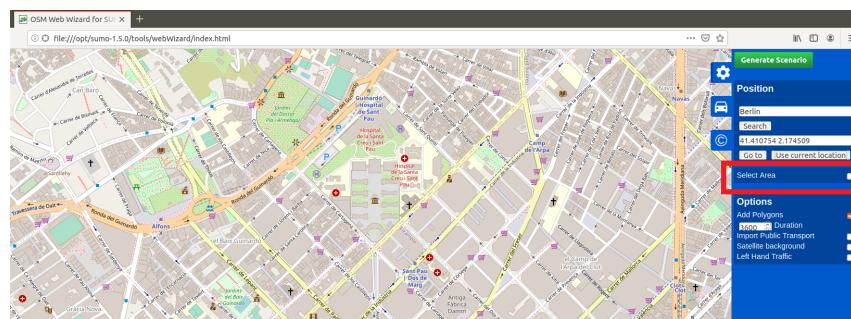


Figura 1.14: osmWebWizard tool.

- Ahora se puede seleccionar un área. Después de eso, se debe presionar el botón verde *Generate scenario*.

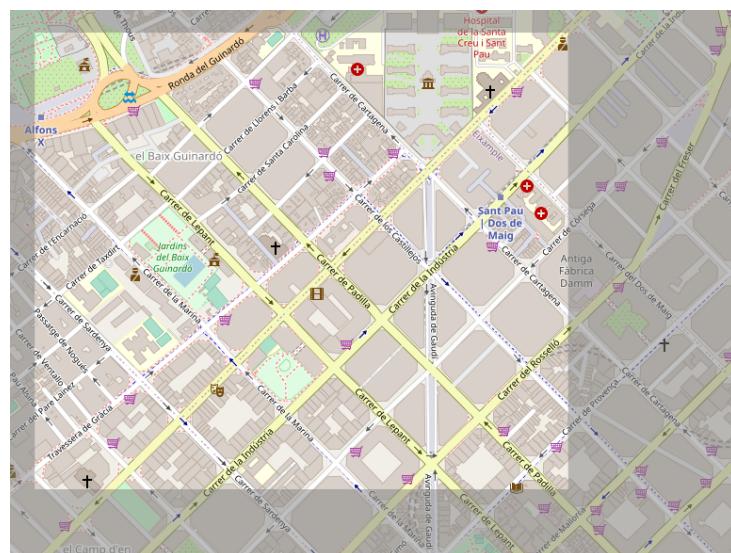


Figura 1.15: Seleccionando área.

- Una vez finalizado el proceso, se agrega una carpeta con la fecha en la que se generaron los archivos como nombre al directorio `/tools`.

```
sumo@sumo-VirtualBox:~/sumo/tools$ ls
2024-02-05-16-21-35  generateParkingLots.py      requirements.txt
assign               generateRailSignalConstraints.py route
averageTripStatistics.py generateRerouters.py    routeSampler.py
build                import                         runSeeds.py
build_config          jtrouter.py                  shapes
contributed          lib                           simpla
countEdgeUsage.py    libsumo                      stateReplay.py
createScreenshotSequence.py libsumo.egg-info     sumolib
createVehTypeDistribution.py libraci                     sumolib.egg-info
detector              libraci.egg-info   tileGet.py
devel                net                           tls
district              osmBuild.py                 tlsCoordinator.py
drt                  osmGet.py                   tlsCycleAdaptation.py
edgesInDistricts.py  osmWebWizard.py            tools.pyproj
emissions             output                      tools.sln
evacuateAreas.py     plot_trajectories.py      traceExporter.py
extractTest.py       ptlines2flows.py           traci
fcdReplay.py         purgatory                 traci.egg-info
findAllRoutes.py    __pycache__                trigger
game                 pyproject.toml            turn-defs
generateBidiDistricts.py randomTrips.py        visualization
generateContinuousRerouters.py README_Contributing.md webWizard
generateParkingAreaRerouters.py req_ci.txt      xml
generateParkingAreas.py req_test_server.txt
sumo@sumo-VirtualBox:~/sumo/tools$
```

Figura 1.16: Carpetas que contienen los archivos SUMO generados.

El contenido de cualquiera de estas carpetas se ve así:

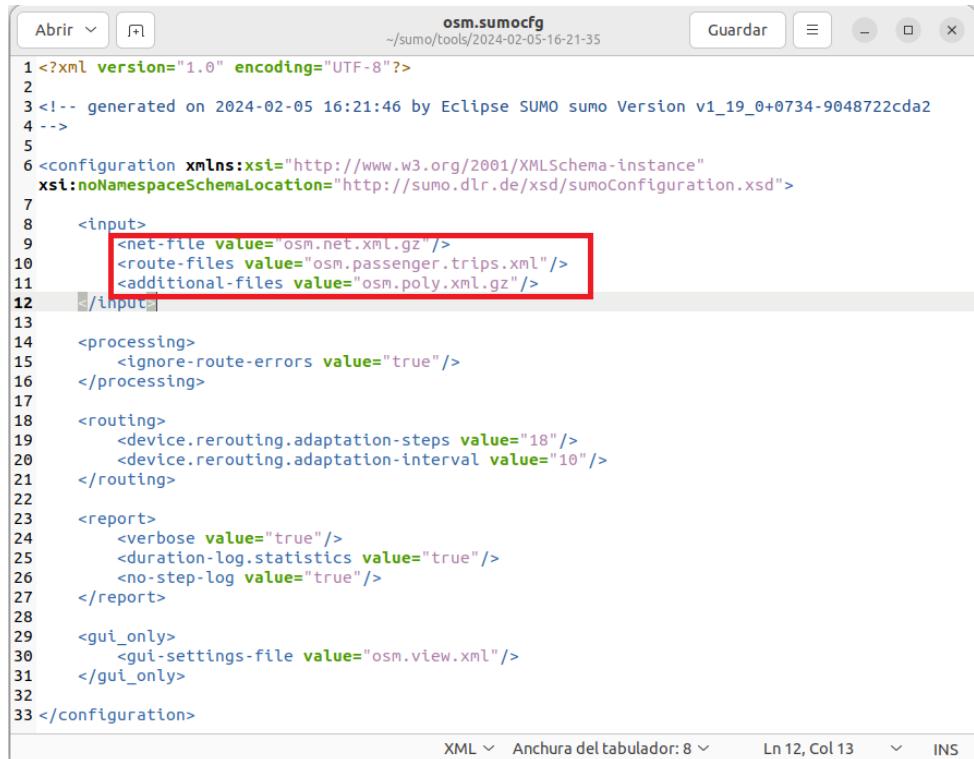
```
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$ ls -l
total 1824
-rwxr-xr-x 1 sumo sumo 573 feb 5 16:21 build.bat
-rw-rw-r-- 1 sumo sumo 813050 feb 5 16:21 osm_bbox.osm.xml.gz
-rw-rw-r-- 1 sumo sumo 1148 feb 5 16:21 osm.netccfg
-rw-rw-r-- 1 sumo sumo 724502 feb 5 16:21 osm.net.xml.gz
-rw-rw-r-- 1 sumo sumo 48588 feb 5 16:21 osm.passenger.trips.xml
-rw-rw-r-- 1 sumo sumo 690 feb 5 16:21 osm.polycfg
-rw-rw-r-- 1 sumo sumo 252912 feb 5 16:21 osm.poly.xml.gz
-rw-rw-r-- 1 sumo sumo 929 feb 5 16:21 osm.sumocfg
-rw-rw-r-- 1 sumo sumo 88 feb 5 16:21 osm.view.xml
-rwxr-xr-x 1 sumo sumo 23 feb 5 16:21 run.bat
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$
```

Figura 1.17: Archivos SUMO generados.

En la Figura 1.17 se ilustra los archivos generados de manera automática desde la herramienta WebWizard. Donde cada uno tiene su función específica:

- **build.bat** es un bash script que contiene el comando para correr la herramienta **randomTrips.py** indicando cada uno de los parámetros necesarios.
- **osm_bbox.osm.xml.gz** es un archivo comprimido que contiene la información de todos los nodos del mapa.
- **osm.netccfg** es el archivo de configuración de las rutas vehiculares. Se tiene como entrada el archivo **osmNetconvert.typ.xml** y el archivo **osm_bbox.osm.xml.gz** para construir el archivo final **.net**.
- **osm.passenger.trips.xml** contiene las rutas de los vehículos (pasajeros).
- **osm.polycfg** contiene la información de los polígonos del mapa.
- **osm.poly.xml.gz** es una carpeta comprimida que contiene el archivo de configuración de los polígonos del mapa. Da como salida el archivo **.poly** final.
- **osm.sumocfg** es el archivo de configuración de la simulación de sumo.
- **osm.view.xml** contiene la configuración del esquema del mapa y el valor del retardo.
- **run.bat** es un script que contiene el comando para correr la simulación de sumo desde la interfaz gráfica.

El archivo más importante siempre será el archivo de configuración, en este caso tiene como nombre `osm.sumocfg`. En este archivo se encuentran la configuración de entrada, procesamiento y salida de la simulación, tal y como se ilustra en la Figura 1.18.



```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!-- generated on 2024-02-05 16:21:46 by Eclipse SUMO sumo Version v1_19_0+0734-9048722cda2
4 -->
5
6 <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">
7
8   <input>
9     <net-file value="osm.net.xml.gz"/>
10    <route-files value="osm.passenger.trips.xml"/>
11    <additional-files value="osm.poly.xml.gz"/>
12  </input>
13
14  <processing>
15    <ignore-route-errors value="true"/>
16  </processing>
17
18  <routing>
19    <device.rerouting.adaptation-steps value="18"/>
20    <device.rerouting.adaptation-interval value="10"/>
21  </routing>
22
23  <report>
24    <verbose value="true"/>
25    <duration-log.statistics value="true"/>
26    <no-step-log value="true"/>
27  </report>
28
29  <gui_only>
30    <gui-settings-file value="osm.view.xml"/>
31  </gui_only>
32
33 </configuration>

```

Figura 1.18: `osm.sumocfg`.

Recordando de la práctica previa, se tenía como entrada los archivos: `.net`, `.rou` y `.poly`. En este caso tenemos los archivos `.net`, `trips` y `.poly`. Donde el primero y tercero redirigen a archivos comprimidos, debido al tamaño de rutas y polígonos existentes en el mapa exportado desde WebWizard. Se observa que ahora se utiliza el archivo `.trips` es cual reemplaza al archivo `.net`, cabe recalcar que se puede utilizar cualquiera de estos. Finalmente, si se requiere ubicar los archivos en otra ruta del sistema operativo, basta con modificar el valor de cada uno de los archivos de entrada. Es decir, no se tendría que poner el nombre específico, sino la ruta en la que se encuentra.

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 modificar el script. En la línea 47, modificar la variable `file` por la variable `name`. Tal y como se indica a continuación:

```
SUMO_HOME = os.environ.get("SUMO_HOME", os.path.join(os.path.dirname(  
os.path.abspath(__name__)), ".."))
```

Este error se debe a que la variable `SUMO_HOME` no está configurada, ya sea que no se realizó la exportación de la misma con el comando `EXPORT` o que no está configurada la ruta en los archivos `.bashrc`.

1.4. Práctica: Generación de estadísticas de la simulación en SUMO

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Simular una red vehicular sobre SUMO.
- Generar los archivos de salida (outputs) de la simulación en SUMO.
- Obtener las estadísticas de la simulación en SUMO.
- Crear scripts en python para graficar las estadísticas de la simulación en SUMO.

Introducción

SUMO permite generar una gran cantidad de medidas diferentes. Todos escriben los valores que recopilan en archivos o en una conexión de socket siguiendo las reglas comunes para escribir archivos. De forma predeterminada, todos están deshabilitados y deben activarse individualmente. Algunas de las salidas disponibles (volcado de posiciones de vehículos sin procesar, información de viaje, información de rutas de vehículos y estadísticas de estado de simulación) se activan mediante opciones de línea de comando, las otras deben definirse dentro de archivos adicionales.

Todos los archivos de salida escritos por SUMO están en formato XML de forma predeterminada. Sin embargo, con la herramienta Python `xml2csv.py` puedes convertir cualquiera de ellos a un formato de archivo plano (CSV) que se puede abrir con la mayoría de los programas de hojas de cálculo. Si necesita una versión binaria más comprimida pero aún “estandarizada”, puede usar `xml2protobuf.py`. Además, todos los archivos se pueden escribir y leer en formato comprimido (gzip), que se activa con la extensión de archivo `.gz`.

Para mantener separadas las salidas de múltiples ejecuciones de simulación, se puede usar la opción `-output-prefix <STRING>` para prefijar todos los nombres de archivos de salida. Al configurar `-output-prefix TIME`, todas las salidas tendrán como prefijo la hora en la que se inició la simulación, lo que las mantiene separadas automáticamente. Otra forma de configurar las salidas, es especificándolas en el archivo `.sumocfg`.

Las salidas disponibles se enumeran a continuación, agrupados en grupos de tipo de tema/agregación. La descripción específica de cada salida se puede encontrar en [5].

- Información basada en vehículos, desagregada

- raw vehicle positions dump:
- emission output
- full output
- vtk output
- **fcd output**
- **trajectories output**
- lanechange output
- surrogate safety measures (SSM)
- vehicle type probe

- Detectores simulados

- Inductive loop detectors (E1)
- Instant induction loops
- Lane area detectors (E2)
- Multi-Entry-Exit detectors (E3)
- Route Detectors

- Valores para bordes o carriles

- edgelane traffic
- aggregated Amitran measures
- edgelane emissions
- edgelane noise
- queue output

- Valores para uniones

- Tools/Output#generateTLSE1Detectors.py scriptt
- Tools/Output#generateTLSE2Detectors.py script
- Tools/Output#generateTLSE3Detectors.py scrip

- **Información basada en el vehículo**

- trip information
- vehicle routes information
- stop output
- battery usage
- collision output

- Información basada en simulación (red)

- simulation state summary statistics

- simulation state person summary statistics
- statistic output
- Información basada en semáforos
 - traffic light states
 - stream-based traffic light switches
 - traffic light states, by switch
 - areal detectors coupled to tls
- Salidas de depuración adicionales

Materiales

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

Instrucciones

A continuación se describen los pasos a seguir para generar los archivos de salida de la simulación. Como se mencionó en la Introducción, existe una gran cantidad de información que se puede obtener de la simulación de SUMO. En la presente práctica nos centraremos en 3 de estas salidas, ya que son muy importantes y contienen información valiosa para obtener estadísticas a analizar. Estas son: *tripinfo*, *fcd* y *trajectories* outputs.

1. El primer paso es generar los archivos necesarios para la simulación. Esto se logra de manera manual con las herramientas de sumo (Práctica 2) o con Web-Wizard (Práctica 3).
2. Luego, es necesario modificar el archivo de configuración de SUMO. Este es el archivo con extensión *.sumocfg* o *.sumo.cfg*. En este archivo es necesario especificar las salidas que se requieren, además de identificar el nombre (ruta) del archivo de salida. Agregando las siguientes líneas al Código 1.1 se logra esto.

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

Donde, se está indicando que se requieren tres salidas: *tripinfo*, *fcd* y *amitran* outputs. En este caso la salida *amitran* hace referencia al archivo de salida de trayectorias de la simulación. Además, se especifica el nombre del archivo de salida. Estos archivos se localizarán en la ruta donde se encuentra en archivo *.sumocfg*. Si se requiere ubicar en otra ruta, basta con especificarla en el valor de la salida. La Figura 1.19 ilustra un ejemplo del archivo de configuración de sumo, especificando las salidas.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!-- generated on 2024-02-05 16:21:46 by Eclipse SUMO sumo Version v1_19_0+0734-9048722cda2
4 -->
5
6 <configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7   xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/sumoConfiguration.xsd">
8
9   <input>
10    <net-file value="osm.net.xml.gz"/>
11    <route-files value="osm.passenger.trips.xml"/>
12    <additional-files value="osm.poly.xml.gz"/>
13  </input>
14
15  <processing>
16    <ignore-route-errors value="true"/>
17  </processing>
18
19  <routing>
20    <device.rerouting.adaptation-steps value="18"/>
21    <device.rerouting.adaptation-interval value="10"/>
22  </routing>
23
24  <report>
25    <verbose value="true"/>
26    <duration-log.statistics value="true"/>
27    <no-step-log value="true"/>
28  </report>
29
30  <gui_only>
31    <gui-settings-file value="osm.view.xml"/>
32  </gui_only>
33
34  <time>
35    <begin value="0"/>
36    <end value="3600"/>
37    <step-length value="1"/>
38  </time>
39
40  <output>
41    <tripinfo-output value="tripinfo.xml"/>
42    <fcd-output value="fcd.xml"/>
43    <amitran-output value="trajectories.xml"/>
44 </configuration>

```

Cargando archivo «/home/sumo/sumo/tools/2024-02... XML ▾ Anchura del tabulador: 8 ▾ Ln 19, Col 30 ▾ INS

Figura 1.19: Archivo .sumocfg especificando las salidas.

3. Luego, correr la simulación de SUMO.

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

4. Luego, en la ruta en la que se corrió la simulación de encontrarán los archivos de salida generados (Figura 1.20).
5. Luego, es posible observar la información que contienen estos archivos xml (Figura 1.21). Sin embargo, no es cómodo trabajar sobre estos archivos, así que es posible utilizar otra herramienta de SUMO para convertir archivos xml en archivos csv. Esto se logra ingresando a la ruta /ruta-de-instalacion-sumo/tools/xml/ y corriendo el script xml2csv.py. Donde se requiere indicar el archivo de entrada, el separador (-s) y el archivo de salida (-o). Por ejemplo:

```
python3 /home/sumo/sumo/tools/xml/xml2csv.py tripinfo.xml -s , -o
tripinfo.csv
```

```
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$ ls -la
total 41384
drwxrwxr-x  2 sumo sumo    4096 feb  6 11:42 .
drwxrwxr-x 36 sumo sumo    4096 feb  5 16:21 ..
-rw-rxr-x  1 sumo sumo     573 feb  5 16:21 build.bat
-rw-rw-r--  1 sumo sumo 22853939 feb  5 20:30 co2.xml
-rw-rw-r--  1 sumo sumo 11590692 feb  6 11:42 fcd.xml
-rw-rw-r--  1 sumo sumo    1152 feb  5 20:30 'osm (copia).sumocfg'
-rw-rw-r--  1 sumo sumo   813050 feb  5 16:21 osm_bbox.osm.xml.gz
-rw-rw-r--  1 sumo sumo    1148 feb  5 16:21 osm.netcfg
-rw-rw-r--  1 sumo sumo   724502 feb  5 16:21 osm.net.xml.gz
-rw-rw-r--  1 sumo sumo    48588 feb  5 16:21 osm.passenger.trips.xml
-rw-rw-r--  1 sumo sumo     690 feb  5 16:21 osm.polycfg
-rw-rw-r--  1 sumo sumo   252912 feb  5 16:21 osm.poly.xml.gz
-rw-rw-r--  1 sumo sumo    1200 feb  6 11:41 osm.sumocfg
-rw-rw-r--  1 sumo sumo      88 feb  5 16:21 osm.view.xml
-rwrxrwxr-x 1 sumo sumo   1681 feb  5 21:16 plots.py
-rwrxr-x  1 sumo sumo      23 feb  5 16:21 run.bat
-rw-rw-r--  1 sumo sumo  5879074 feb  6 11:42 trajectories.xml
-rw-rw-r--  1 sumo sumo  163054 feb  6 11:42 tripinfo.xml
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$
```

Figura 1.20: Archivos output.

```
Abrir ⌂ Guardar ⌂
tripinfo.xml
~/sumo/tools/2024-02-05-16-21-35
44 -->
45
46 <tripinfos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/tripinfo_file.xsd">
47   <tripinfo id="veh3" depart="26.00" departLane="4773029#0_0" departPos="5.10"
    departSpeed="7.82" departDelay="0.85" arrival="133.00" arrivalLane="-120675240#3_0"
    arrivalPos="2.64" arrivalSpeed="7.73" duration="107.00" routeLength="818.45"
    waitingTime="0.00" waitingCount="0" stopTime="0.00" timeLoss="10.89" rerouteNo="1"
    devices="tripinfo_veh3 routing_veh3 fcd_veh3" vType="veh_passenger" speedFactor="0.94"
    vaporized=""/>
48   <tripinfo id="veh5" depart="42.00" departLane="-339201177#5_0" departPos="5.10"
    departSpeed="0.00" departDelay="0.08" arrival="156.00" arrivalLane="169195795#1_1"
    arrivalPos="39.46" arrivalSpeed="14.87" duration="114.00" routeLength="586.03"
    waitingTime="0.00" waitingCount="0" stopTime="0.00" timeLoss="27.29" rerouteNo="1"
    devices="tripinfo_veh5 routing_veh5 fcd_veh5" vType="veh_passenger" speedFactor="1.08"
    vaporized=""/>
```

Figura 1.21: Archivos tripinfo.xml.

6. Además, con los valores obtenidos se pueden graficar las estadísticas que se requiera. Por ejemplo, a continuación se presenta un script en python, el que permite graficar los valores de: tripinfo_routeLength, emissions_CO2_abs y tripinfo_duration, respecto a tripinfo_id. Todo esto, tomando los valores del archivo tripinfo.csv.

```

import pandas as pd
import matplotlib.pyplot as plt

# Lee el archivo CSV
archivo_csv = "tripinfo.csv" # Reemplaza "tu_archivo.csv" con la ruta
                             de tu archivo CSV
datos = pd.read_csv(archivo_csv)

# Extrae las columnas de interes
tripinfo_id = datos['tripinfo_id']
tripinfo_routeLength = datos['tripinfo_routeLength']
emissions_CO2_abs = datos['emissions_CO2_abs']
tripinfo_duration = datos['tripinfo_duration']

# Graficar los datos
plt.figure(figsize=(12, 6))

# Grafica tripinfo_routeLength versus tripinfo_id
plt.subplot(3, 1, 1)
plt.bar(tripinfo_id, tripinfo_routeLength, color='blue')
plt.title('tripinfo_routeLength vs tripinfo_id')
plt.xlabel('tripinfo_id')
plt.ylabel('tripinfo_routeLength')
plt.xticks(rotation=45, ha='right') # Rotar las etiquetas del eje x

# Grafica emissions_CO2_abs versus tripinfo_id
plt.subplot(3, 1, 2)
plt.bar(tripinfo_id, emissions_CO2_abs, color='green')
plt.title('emissions_CO2_abs vs tripinfo_id')
plt.xlabel('tripinfo_id')
plt.ylabel('emissions_CO2_abs')
plt.xticks(rotation=45, ha='right') # Rotar las etiquetas del eje x

# Grafica tripinfo_duration versus tripinfo_id
plt.subplot(3, 1, 3)
plt.bar(tripinfo_id, tripinfo_duration, color='red')
plt.title('tripinfo_duration vs tripinfo_id')
plt.xlabel('tripinfo_id')
plt.ylabel('tripinfo_duration')
plt.xticks(rotation=45, ha='right') # Rotar las etiquetas del eje x

plt.tight_layout()
plt.show()

```

Código 1.2: Script **plots.py** para graficar la información del archivo tripinfo versus tripinfo_id.

7. Correr el script para graficar las estadísticas especificadas dentro del programa.

```
python3 plots.py
```

Está claro que, en el script presentado se especifica la ruta del archivo `csv`. Además, tener en cuenta que para este ejemplo también se está graficando las emisiones de CO2. Para obtener esta salida se debe agregar `<emission-output value="co2.xml"/>` al archivo de configuración de la Figura 1.19.

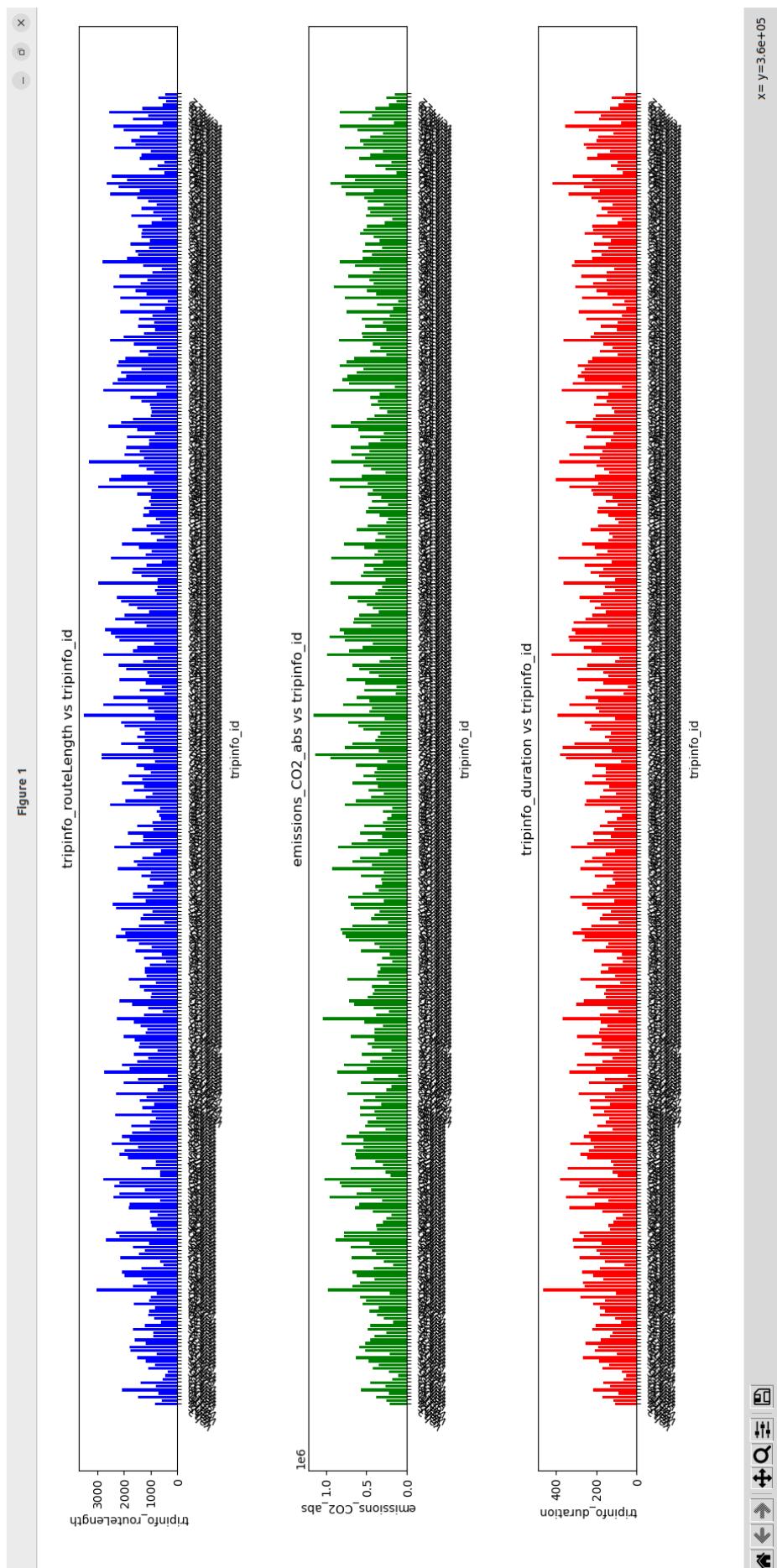


Figura 1.22: Gráficas estadísticas de: Longitud de ruta, Emisiones de CO₂ y Duración de viaje de cada vehículo de la simulación de SUMO.

8. Finalmente, el código presentado en el punto 7 se puede generalizar para que mediante línea de comandos el usuario deba ingresar el archivo `csv`, y las columnas de este archivo que desea graficar. A continuación se muestra el código para lograr esto, teniendo en cuenta que se graficará la columna seleccionada versus la columna `tripinfo_id`.

```

import pandas as pd
import matplotlib.pyplot as plt
import argparse

# Configuración de los argumentos de la línea de comandos
parser = argparse.ArgumentParser(description='Graficar datos desde un archivo CSV')
parser.add_argument('-i', '--input', type=str, help='Nombre del archivo CSV de entrada', required=True)
parser.add_argument('--columnas', type=str, help='Nombres de las columnas a graficar separadas por coma', required=True)
args = parser.parse_args()

# Lee el archivo CSV
archivo_csv = args.input
datos = pd.read_csv(archivo_csv)

# Extrae las columnas de interés
tripinfo_id = datos['tripinfo_id']
columnas_seleccionadas = args.columnas.split(',')

# Si se elige una sola columna, graficar en una sola ventana
if len(columnas_seleccionadas) == 1:
    columna = columnas_seleccionadas[0].strip() # Obtener el nombre de la columna
    datos_columna = datos[columna]

    # Configurar la figura y el gráfico
    plt.figure(figsize=(12, 6))

    # Graficar datos_columna versus tripinfo_id
    plt.bar(tripinfo_id, datos_columna, color='blue')
    plt.title(f'{columna} vs tripinfo_id')
    plt.xlabel('tripinfo_id')
    plt.ylabel(columna)
    plt.xticks(rotation=45) # Rotar las etiquetas del eje x
    plt.tight_layout()
    plt.show()

else:
    # Configura la figura y los subgráficos
    num_columnas = len(columnas_seleccionadas)
    fig, axs = plt.subplots(num_columnas, 1, figsize=(12, 6 * num_columnas), sharex=True)

    # Generar una lista de colores únicos
    colores = plt.cm.tab10.colors[:num_columnas]

    # Graficar los datos en diferentes subgráficos con colores diferentes
    for i, (columna, color) in enumerate(zip(columnas_seleccionadas,

```

```

        colores)):
    columna = columna.strip() # Elimina espacios en blanco al
        inicio y al final
    datos_columna = datos[columna]

    # Grafica datos_columna versus tripinfo_id en el subgrafico
    # correspondiente
    axs[i].bar(tripinfo_id, datos_columna, color=color)
    axs[i].set_title(f'{columna} vs tripinfo_id')
    axs[i].set_ylabel(columna)
    axs[i].tick_params(axis='x', rotation=45) # Rotar las
        etiquetas del eje x

# Ajustar el espacio entre los subgraficos
plt.tight_layout()
plt.show()

```

Código 1.3: Script general **plots-v1-4.py** para graficar la información del archivo tripinfo versus tripinfo_id.

Los comandos para correr el script general, pueden ser los siguientes:

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

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

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

Las Figuras 1.23, 1.24 y 1.25 ilustran los resultados de estos comandos.

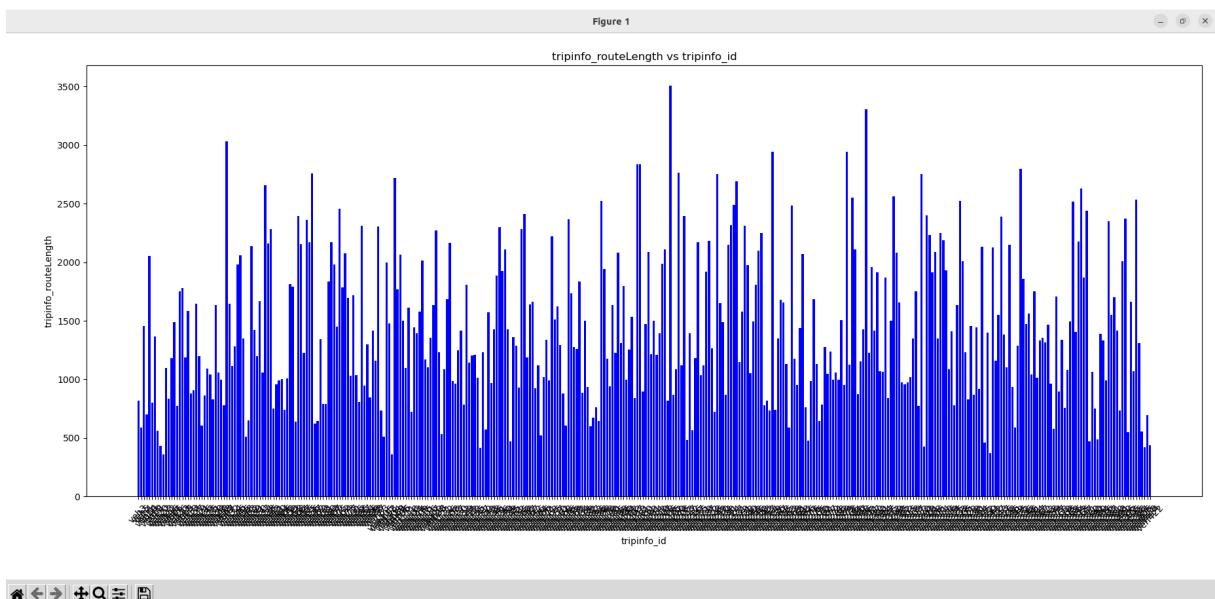


Figura 1.23: Gráfica de tripinfo_routeLength vs tripinfo_id.

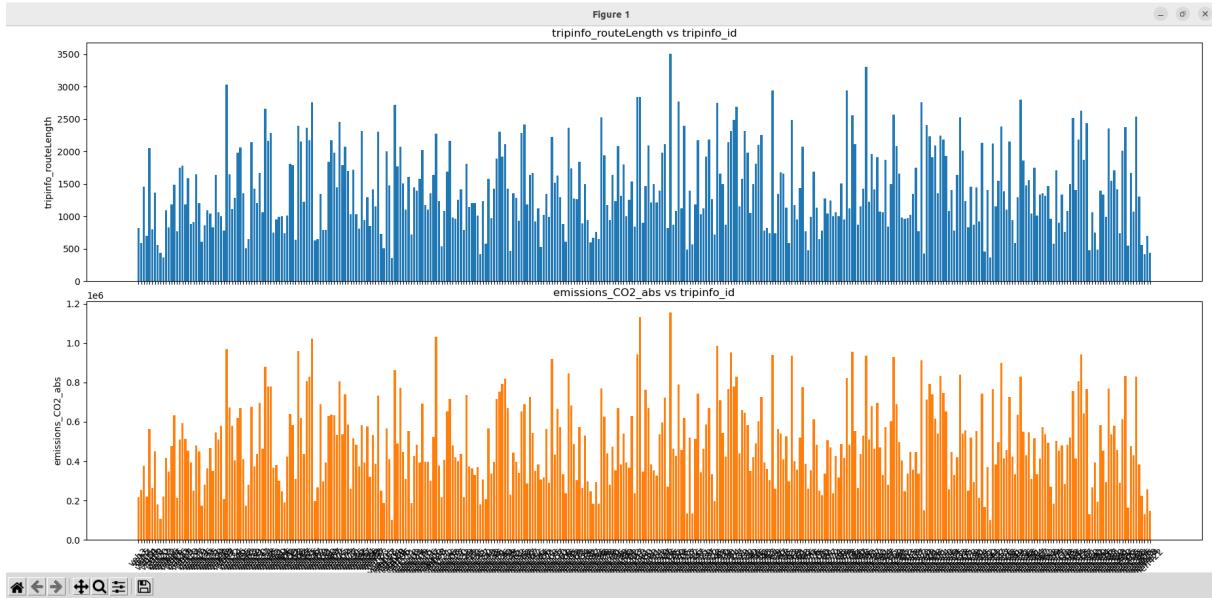


Figura 1.24: Gráficas de: tripinfo_routeLength y emissions_CO2_abs vs tripinfo_id.

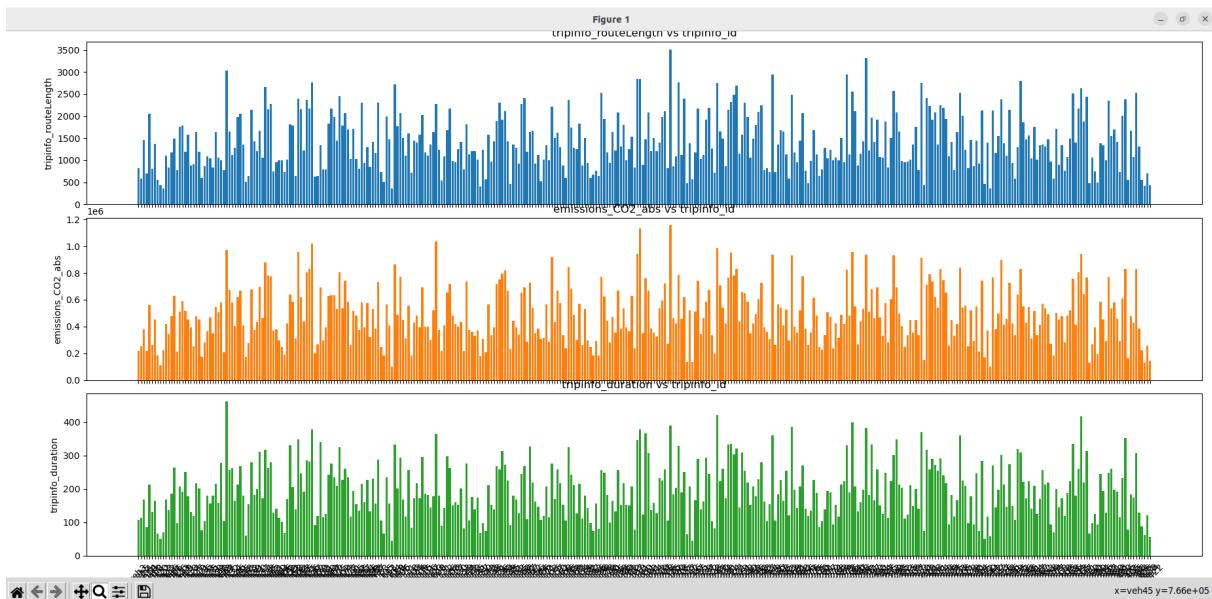


Figura 1.25: Gráficas de: tripinfo_routeLength, emissions_CO2_abs y tripinfo_duration vs tripinfo_id.

9. Dentro de las herramientas de SUMO existen otras herramientas que permiten graficar ciertos aspectos de la simulación. Algunos ejemplos se indica a continuación:

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 por la simulación (Figura 1.19). El comando para lograr esto es el siguiente:

```
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` es hacer un diagrama de dispersión en lugar de un diagrama de líneas.

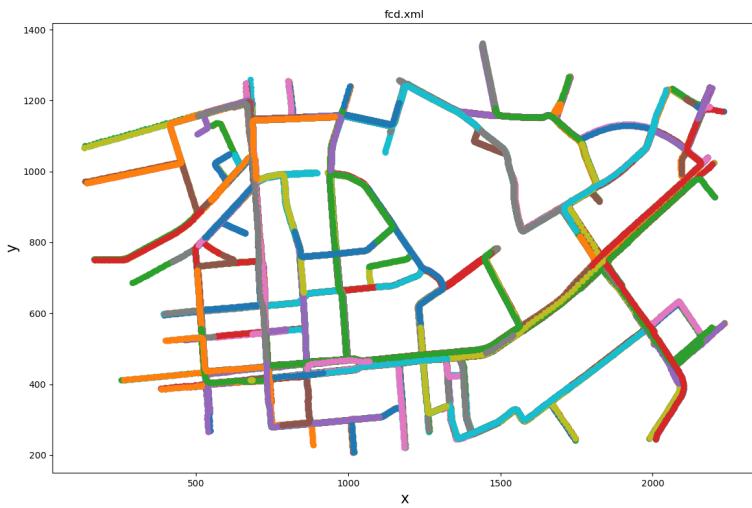


Figura 1.26: Trayectoria graficadas.

La segunda forma es utilizar el script `plot_trajectories.py` que está ubicado en la ruta `tools/`. Para correr este script se requiere como entrada el archivo de salida `fcd_output` generado por la simulación (Figura 1.19). El comando para lograr esto es el siguiente:

```
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.

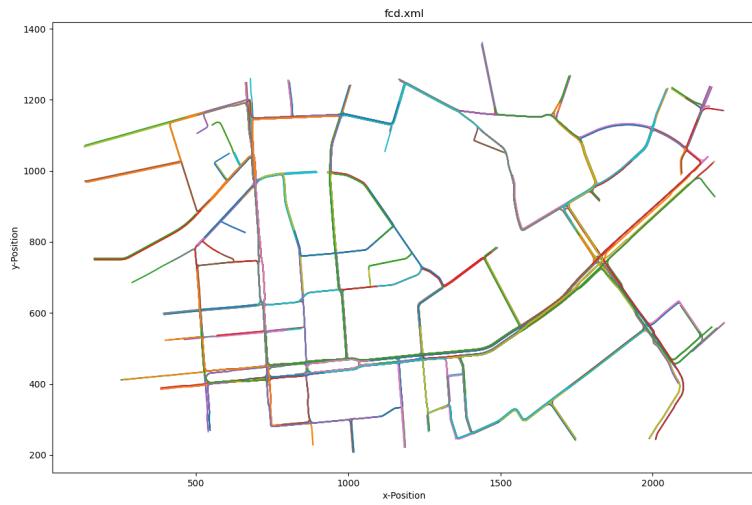
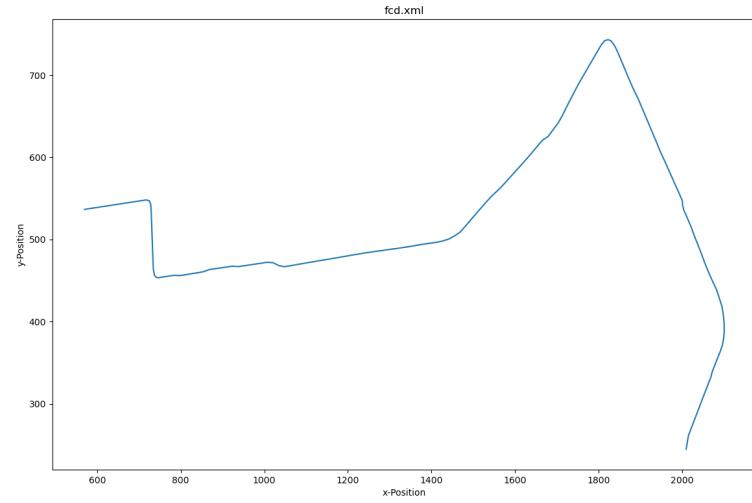


Figura 1.27: Trayectoria graficadas.

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.

Figura 1.28: Trayectoria de *veh1*.

Velocidad:

También es posible graficar y comparar la velocidad de los vehículos en la simulación de SUMO. Para lograr esto se hace uso del script `plot_trajectories.py` ubicado en `/tools`. A continuación se muestra un ejemplo de como usarlo:

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

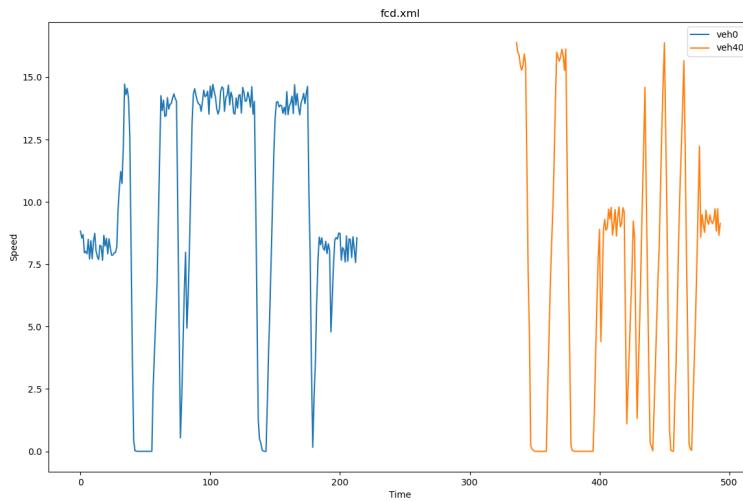


Figura 1.29: Comparación de velocidades de los vehículos *veh1* y *veh40*.

En la Figura 1.29 se ilustra la comparación de las velocidades de los vehículo con IDs 1 y 40. En la Figura se observa que el vehículo 1 comienza su viaje en es segundo 0 y termina en el segundo 220 aproximadamente, con sus variaciones respectivas de velocidad en m/s. Mientras que para el vehículo 40, se observa que su recorrido comienza en el segundo 320 y termina en el segundo 500. En este ejemplo en específico, se logra apreciar que los vehículos tienen su propio trayecto, inicio, fin y velocidad a lo largo de la simulación. Además, con esta gráfica queda claro que los vehículos pueden iniciar su trayecto luego de que inicie la simulación. Así mismo, la simulación puede terminar antes de que un vehículo termine su recorrido. Por tal motivo, es importante configurar de manera adecuada las rutas y tiempos de simulación, o en su defecto, tener en cuenta estos comportamientos al momento de realizar un análisis estadístico.

Densidad de vehículos:

Otra de las herramientas que brinda SUMO es el script `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 dependencia de los atributos de borde definidos. Los pesos de los bordes se leen a partir del archivo “edge-dumps”, tales como: `-edgelane traffic`, `edgelane emissions`, o `edgelane noise`.

Además, es claro que estos valores deben estar dentro de un archivo de salida de simulación. En este caso las medidas de densidad se localizan en el archivo `edge.data.xml`. Estos datos se obtienen agregando una salida en el archivo de

configuración de SUMO .sumocfg (Figura 1.19), en la parte de salidas.

A continuación se indica las salidas utilizadas para este ejemplo.

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

Donde, es claro que existen varias salidas, pero la que nos importa para este caso es la instrucción: <edgedata-output value="edgedata.xml"/>. Donde la densidad de vehículos en cada carretera es medida con la relación: #veh/km

A continuación se muestra un ejemplo de uso de esta herramienta:

```
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.

En el ejemplo previo se indica que se requiere graficar las medidas de densidad en las rutas. Por tal motivo se indica el valor -measures como density,density.

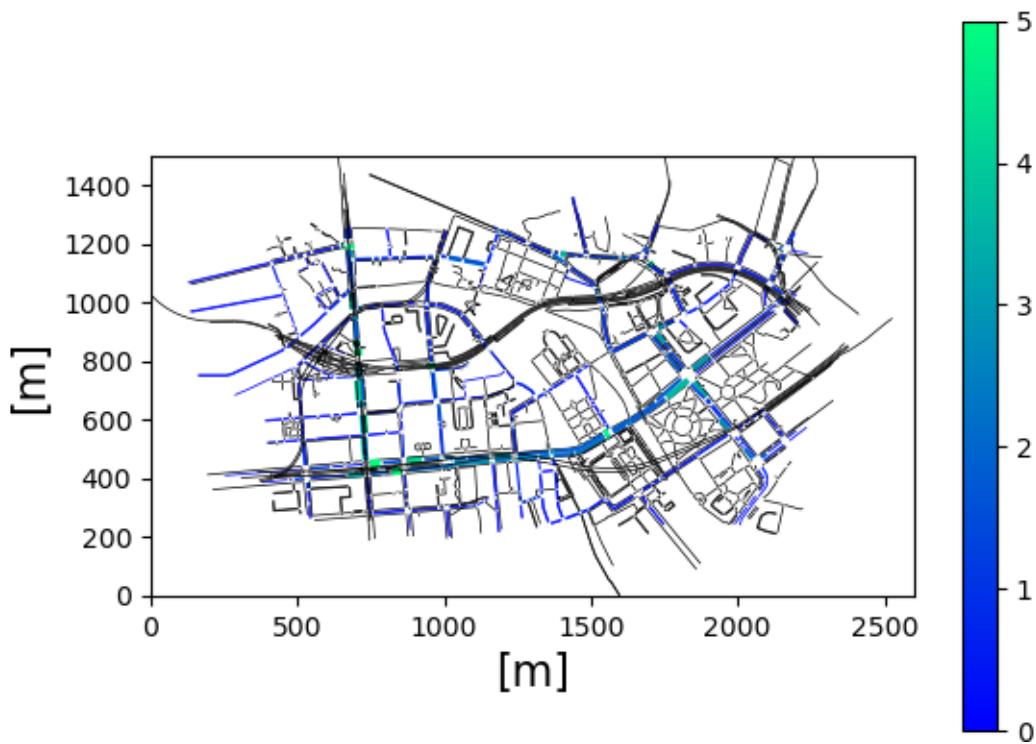


Figura 1.30: Mapa de color de densidad de flujo de vehículos.

Además, es posible graficar el mapa de color de las emisiones medidas en las carreteras del mapa. Para lograr esto, se requiere de una configuración adicional dentro del archivo de configuración de sumo, así como la creación de un nuevo archivo.

El nuevo archivo necesario es de tipo xml referente a los datos que se requiere de las carreteras (`edgeData`). 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 -output-v2.xml" excludeEmpty="true"/>

</additional>
```

Luego, es necesario cargar el nuevo archivo creado al archivo de configuración de sumo. Para esto, 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, input-additional-edgeData
    .xml"/>
</input>
```

Donde `input-additional-edgeData.xml` es el nombre del nuevo archivo creado. A continuación, se indica el comando a utilizar para graficar el mapa de color en base a las emisiones CO₂ medidas en el mapa. Donde los parámetros que han cambiado con respecto al ejemplo anterior son: archivo de entrada (`-i edgeData-output-v2.xml,edgeData-output-v2.xml`) el cual es la salida del nuevo archivo creado; y el valor a graficar (`-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
```

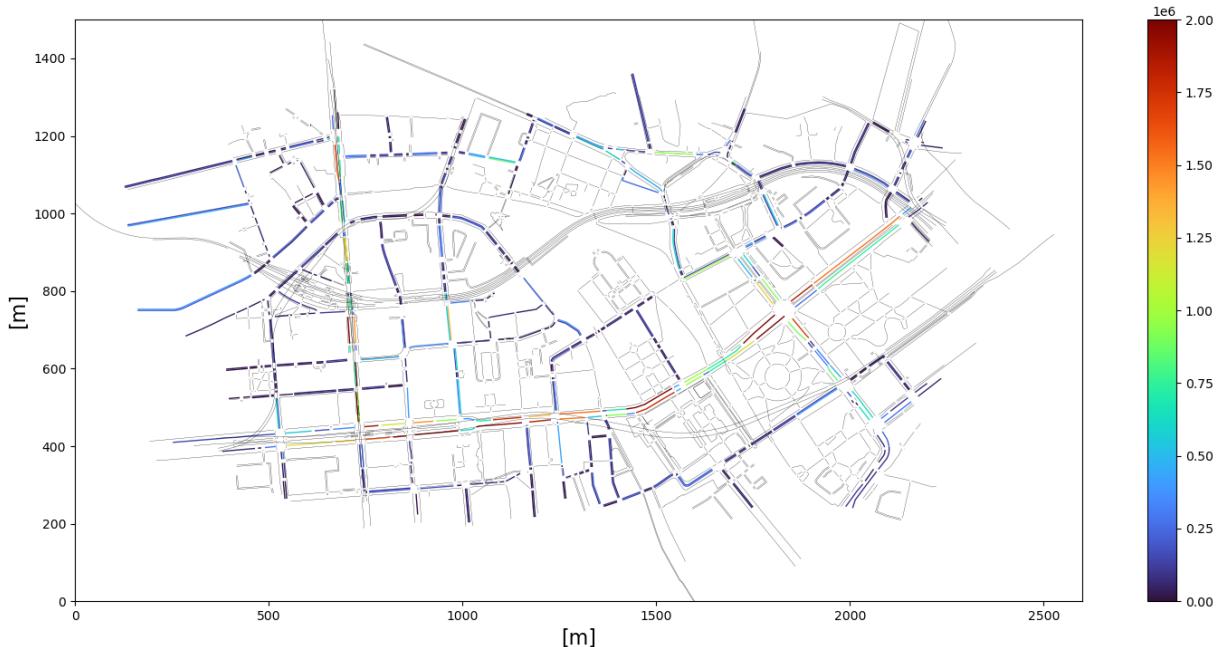


Figura 1.31: Mapa de color de CO₂ absoluto de vehículos.

1.5. Práctica: Introducción a TraCI con SUMO

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Introducir la librería traci para python.
- Verificar la variable de entorno de SUMO con TraCI.

Introducción

TraCI es el término corto para “Interfaz de control de tráfico” (Traffic Control Interface). Al dar acceso a una simulación de tráfico en marcha, permite recuperar valores de objetos simulados y manipular su comportamiento “en línea”.

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. Cuando con esta opción, sumo solo prepara la simulación y espera a que todas 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. Cuando se utiliza sumo-gui como servidor, la simulación debe iniciarse usando el botón de reproducción o configurando la opción `-start` antes de que se procesen los comandos de TraCI.

Después de iniciar sumo, los clientes se conectan a sumo configurando una conexión TCP al puerto de sumo designado. TraCI admite múltiples clientes y ejecuta todos los comandos de un cliente en una secuencia hasta que emite los comandos relacionados con TraCI/Control#Command 0x02: comando Paso de simulación. Para tener un orden de ejecución predefinido, cada cliente debe emitir un comando relacionado con TraCI/Control#Command 0x03: comando SetOrder antes del primer paso de simulación. Esto asigna un número al cliente y los comandos de diferentes clientes que, durante un mismo paso de simulación se ejecutarán en el orden de esa numeración. Cuando se utilizan configuraciones de múltiples clientes, es necesario conocer la cantidad de clientes al iniciar SUMO y todos los clientes deben conectarse antes del primer paso de simulación.

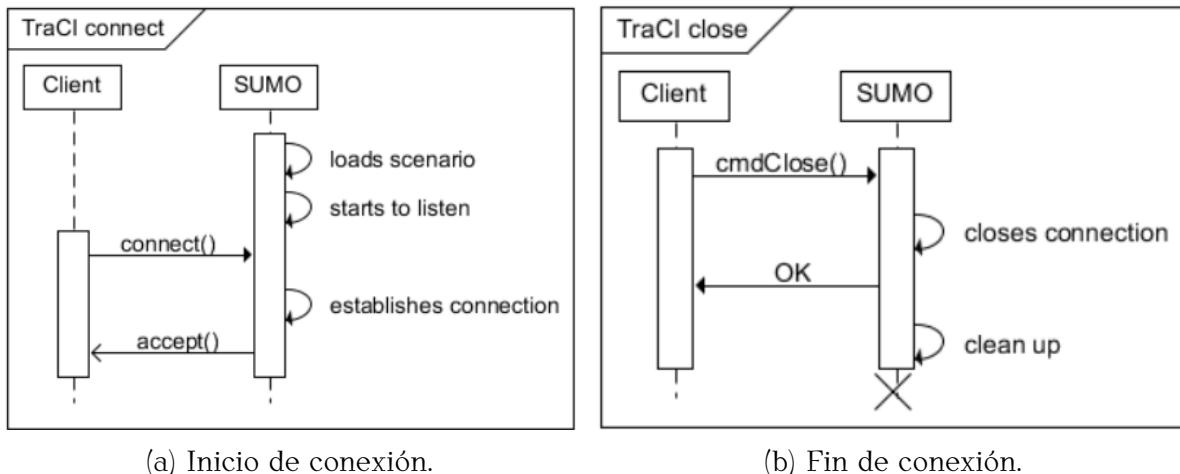


Figura 1.32: Conexión Cliente/SUMO.

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.

El cliente tiene que activar cada paso de simulación en sumo usando los comandos relacionados con TraCI/Control#Command 0x02: comando de paso de simulación. Si se ha realizado alguna suscripción, se devuelven los valores suscritos. La simulación avanzará a la siguiente una vez que todos los clientes hayan enviado el comando del paso de simulación. Actualmente, todos los clientes reciben todos los resultados de la suscripción (incluso si la suscripción ha sido emitida por un cliente diferente).

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.

Materiales

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

Instrucciones

A continuación, se describen los pasos a seguir para lograr la verificación de la variable de entorno de SUMO. Además se utiliza la librería `traci` disponible para python para comprobar la ubicación de los archivos de la librería.

1. El primer paso es instalar la librería `traci` para python.

```
pip3 install traci
```

2. En este punto solo falta codificar en python las acciones que se requieran realizar. Como primer paso es necesario comprobar que la variable de entorno `SUMO_HOME` esté definida. A continuación se presenta el script para realizar esta comprobación.

```

import traci
import sys
import os

def verificar_variable_entorno(variable):
    valor = os.getenv(variable)
    if valor is not None:
        print(f'La variable de entorno {variable} esta definida.')
        print(f'Su valor es: {valor}\n')
    else:
        print(f'La variable de entorno {variable} no esta definida.\n')

if __name__ == "__main__":
    variable_entorno = "SUMO_HOME"
    verificar_variable_entorno(variable_entorno)

    print("LOADPATH:", '\n'.join(sys.path))
    print("TRACIPATH:", traci.__file__)
    sys.exit()

```

Código 1.4: Script **traci_check_sumo_home.py** para verificar la existencia de la variable de entorno.

Donde, se importa la librería traci para verificar la ubicación de los archivos traci. Además, se importa las librerías sys y os para comprobar en el sistema operativo la existencia de la variable de entorno. En el caso de que la variable de entorno no está definida, referirse al punto 6 de la práctica 1.1.

```

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$ []

```

Figura 1.33: Verificación de variable de entorno.

3. Con estos simples pasos, es posible conectar TraCI con SUMO y realizar la codificación para requerimientos específicos, utilizando la librería traci para python. Los comandos pueden lograr lo siguiente:

- **Valores a recuperar:**

- Objetos de tráfico
- Detectores y Salidas
- Red
- Infraestructura
- Varios

■ Cambio de estado:

- Objetos de tráfico
 - Detectores y Salidas
 - Red
 - Infraestructura
 - Varios

■ Suscripciones:

- TraCI/Object Variable Subscription
 - TraCI/Object Context Subscription

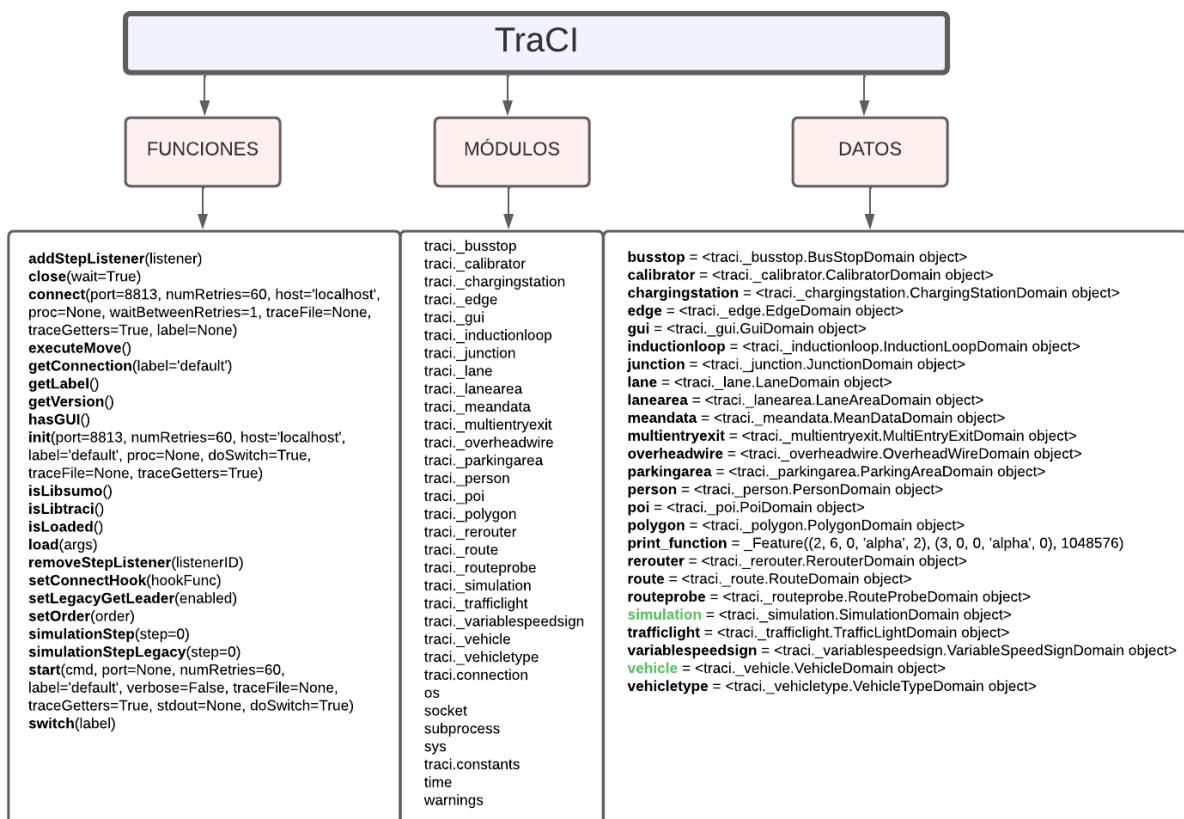


Figura 1.34: Diagrama de funcionamiento de TraCI.py.

1.6. Práctica: Simulación de tráfico vehicular TraCI/SUMO

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Integrar la interfaz TraCI con SUMO.
- Simular una red vehicular sobre SUMO utilizando scripts en python.

Introducción

El tráfico urbano es un fenómeno complejo que requiere de herramientas y simulaciones avanzadas para comprender su dinámica y encontrar soluciones efectivas. SUMO es una plataforma de simulación de tráfico de código abierto que permite modelar y analizar el comportamiento de vehículos, peatones y otros agentes en entornos urbanos. 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. TraCI ofrece a los desarrolladores y usuarios la capacidad de controlar y monitorear de manera dinámica los elementos de la simulación, como vehículos, semáforos, y otros agentes de tráfico.

El objetivo de esta práctica es comprender cómo funciona TraCI y cómo puede integrarse con SUMO para llevar a cabo experimentos avanzados en el campo de la movilidad urbana. En esta práctica en específico, exploraremos los fundamentos de TraCI para correr una simulación de SUMO utilizando un script en python a través de la interfaz TraCI. A lo largo de esta práctica, los participantes aprenderán a utilizar las funciones básicas de TraCI para controlar la simulación de tráfico, así como a desarrollar scripts y programas que aprovechen su potencial para la investigación y desarrollo de soluciones innovadoras en el ámbito de la movilidad urbana.

Materiales

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

Instrucciones

A continuación, se describen los pasos a seguir para lograr correr una simulación de SUMO, utilizando la librería `traci` disponible para python. Esto se logra a través de la configuración de un script en python para correr el archivo de configuración de sumo con el uso de la interfaz TraCI.

1. El primer paso es importar las librerías necesarias. En este caso solo se utilizará `traci`.
2. 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, con el parámetro `-c`.

La función `simulaciónStep(float)` realiza un paso de simulación y simula hasta el segundo dado en el tiempo de simulación. Si el valor dado es 0 o está ausente, se realiza exactamente un paso.

3. Luego, es necesario iniciar el bucle de la simulación. Donde, la condición debe indicar la duración que deseamos de la simulación. Esta condición pueden ser el número de pasos de la simulación, velocidades, ubicaciones, etc. En este caso 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.
4. A continuación, se presente el script utilizado para correr una simulación de SUMO, conectando con TraCI.

```
import traci

traci.start(["sumo", "-c", "/home/sumo/sumo/tools/2024-02-05-16-21-35/
osm.sumocfg"])

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

traci.close()
```

Código 1.5: Script `traci_run_sumo.py` para correr simulación TraCI/SUMO.

```

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
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 0.08s answering queries (0.19ms on average).
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$ █

```

Figura 1.35: Simulación TraCI/SUMO exitosa.

Otra forma de correr la simulación, es indicar los pasos que tiene la simulación. Estos pasos están relacionados con el tiempo de simulación, pero no es el mismo configurado en la Figura 1.19 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 (100, por ejemplo); es decir TraCI toma el control del tiempo de la simulación.

```

import traci

traci.start(["sumo", "-c", "/home/sumo/sumo/tools/2024-02-05-16-21-35/
osm.sumocfg"])
step = 0

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

traci.close()

```

Código 1.6: Variante a script **traci_run_sumo.py** para correr simulación TraCI/SUMO.

1.7. Práctica: Simulación de tráfico vehicular TraCI/SUMO. Contador de vehículos

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Integrar la interfaz TraCI con SUMO.
- Simular una red vehicular sobre SUMO utilizando scripts en python.
- Contar el número de vehículos que circularon en el mapa.

Introducción

El estudio y la gestión del tráfico urbano son aspectos críticos para el desarrollo de ciudades sostenibles y eficientes. En este contexto, SUMO ha surgido como una herramienta poderosa que permite modelar y simular el flujo de vehículos y peatones en entornos urbanos de manera realista. Dentro del conjunto de herramientas que ofrece SUMO, desempeña un papel fundamental al facilitar la comunicación en tiempo real entre simulaciones externas y el simulador SUMO. Una de las capacidades más destacadas de TraCI es su habilidad para contar y rastrear vehículos dentro de la simulación, lo que brinda a los investigadores y planificadores una perspectiva detallada del flujo de tráfico en entornos urbanos.

El propósito principal de esta práctica de laboratorio es explorar cómo funciona TraCI y cómo su capacidad para contar vehículos puede ser aprovechada para analizar y optimizar el tráfico urbano. Mediante ejercicios prácticos, los participantes se familiarizarán con la implementación de scripts y programas que aprovechan la capacidad de TraCI para contar vehículos, lo que les permitirá realizar análisis detallados y tomar decisiones fundamentadas para mejorar la movilidad en entornos urbanos.

Materiales

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

Instrucciones

A continuación, se describen los pasos a seguir para lograr correr una simulación de SUMO, utilizando la librería `traci` disponible para python. Esto se logra a través de la configuración de un script en python para correr el archivo de configuración de sumo con el uso de la interfaz TraCI.

1. El primer paso es importar las librerías necesarias. En este caso solo se utilizará `traci` para la conexión con SUMO y `re` para el manejo de expresiones regulares.
2. Luego, es necesario iniciar la comunicación SUMO/TraCI a través del comando `traci.start(["cmd"])` e iniciar el bucle de la simulación y guardar los identificadores de los vehículos. En este caso se guarda los identificadores en una lista debido a que cada vehículo tiene como nombre `vehID`, donde `ID` indica el número de vehículo de la simulación.
3. Dentro del bucle, es necesario extraer los identificadores de los vehículos, esto son una lista y se obtiene con la instrucción `traci.vehicle.getIDList()`.
4. Finalmente, es necesario crear una lógica con la cual se extraiga los números identificadores de cada vehículo y obtener el número mayor. Con esto estaremos obteniendo el número total de vehículos que se movilizaron sobre el mapa.
5. A continuación, se presenta el script utilizado para correr una simulación de SUMO, conectando con TraCI.

```
import traci
import re

# Iniciar la conexion
traci.start(["sumo", "-c", "/home/sumo/sumo/tools/2024-02-05-16-21-35/osm.sumocfg"])

# Lista de identificadores de vehiculos
lista_veh = []

# Ejecutar la simulacion
while traci.simulation.getMinExpectedNumber() > 0:
    for veh_id in traci.vehicle.getIDList():
        #print(f"ID del Vehiculo: {veh_id}")
        lista_veh.append(veh_id)

    traci.simulationStep()

traci.close()

# Expresion regular para encontrar los numeros en los identificadores de vehiculos
patron = re.compile(r'veh(\d+)')

# Inicializamos el maximo como cero
maximo = 0

# Iteramos sobre la lista de vehiculos
for vehiculo in lista_veh:
```

```

# Buscamos los numeros en los identificadores de vehiculos usando
# expresiones regulares
match = patron.match(vehiculo)
if match:
    numero = int(match.group(1)) # Convertimos el numero a entero
    maximo = max(maximo, numero) # Actualizamos el maximo

print(f"\n En el mapa han circulado un total de {maximo} vehiculos")

```

Código 1.7: Script **traci_cont_veh.py** para correr simulación TraCI/SUMO y contar el número de vehículos.

```

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 vehiculos
sumo@sumo-VirtualBox:~/sumo/tools/2024-02-05-16-21-35$
```

Figura 1.36: Conteo de vehículos con TraCI/SUMO.

1.8. Práctica: Simulación de tráfico vehicular TraCI/SUMO. Cálculo de velocidad promedio

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Integrar la interfaz TraCI con SUMO.
- Simular una red vehicular sobre SUMO utilizando scripts en python.
- Calcular la velocidad promedio de los vehículos que circularon en el mapa.

Introducción

La simulación del tráfico urbano es fundamental para comprender y optimizar la movilidad en entornos urbanos cada vez más complejos. Dentro del conjunto de herramientas proporcionadas por SUMO, TraCI ofrece una gama de funcionalidades, entre las cuales destaca la capacidad para calcular y registrar las velocidades de los vehículos dentro de la simulación.

El objetivo principal de esta práctica de laboratorio es explorar el funcionamiento de TraCI con un énfasis especial en el cálculo de las velocidades de los vehículos en la simulación. A través de ejercicios prácticos, los participantes aprenderán a utilizar las capacidades de TraCI para medir y analizar las velocidades de los vehículos en tiempo real, lo que proporcionará una comprensión más profunda del flujo de tráfico. Durante la práctica, los participantes desarrollarán habilidades para implementar scripts y programas que aprovechen la funcionalidad de cálculo de velocidades de TraCI, lo que les permitirá realizar análisis detallados y tomar decisiones informadas para mejorar la eficiencia y seguridad del tráfico en entornos urbanos.

Materiales

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

Instrucciones

A continuación, se describen los pasos a seguir para lograr correr una simulación de SUMO, utilizando la librería `traci` disponible para python. Esto se logra a través de la configuración de un script en python para correr el archivo de configuración de sumo con el uso de la interfaz TraCI.

1. El primer paso es importar las librerías necesarias. En este caso se utilizará `traci` para la conexión con SUMO. Además de, `csv` para el manejo de archivos csv y `time` para obtener los tiempos de simulación.
2. Luego, es necesario iniciar la comunicación SUMO/TraCI a través del comando `traci.start(["cmd"])`.
3. Luego, indicar la ruta del archivo de salida, abrir el archivo csv e iniciar el bucle de la simulación para guardar los identificadores de los vehículos, velocidades, tiempos de simulación y verificar el nombre de calle (borde) en el que se encuentra. En este caso se 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)
```

4. Finalmente, es necesario crear una lógica para contar el total de vehículos que han circulado en el mapa a lo largo de la simulación. Para esto, se guardan los valores de las velocidades de todos los vehículos en cada paso de la simulación, posteriormente se calcula la velocidad media de cada vehículo y de la simulación general.
5. A continuación, se presenta el script utilizado para correr una simulación de SUMO, conectando con TraCI.

```
import traci
import csv
import time
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Iniciar la conexión
traci.start(["sumo", "-c", "/home/sumo/sumo/tools/2024-02-05-16-21-35/sumo.sumocfg"])

csv_file_path = 'datos_simulacion_vel.csv'

# Función para calcular la velocidad media de los vehículos
def calcular_media():
    # Leer los datos desde el archivo CSV
    df = pd.read_csv(csv_file_path)
    # Calcular la velocidad media de los vehículos
    media_vehiculos = df['speed'].mean()
    media_simulacion = df['speed'].mean()
    return media_vehiculos, media_simulacion
```

```

with open(csv_file_path, 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)

    # Cabecera del CSV
    csv_writer.writerow(['Time', 'VehicleID', 'Speed'])

    # Ejecutar la simulacion
    while traci.simulation.getMinExpectedNumber() > 0:
        current_time = traci.simulation.getCurrentTime() / 1000 # Convertir a segundos

        for veh_id in traci.vehicle.getIDList():
            speed = traci.vehicle.getSpeed(veh_id)
            print(f"Tiempo: {current_time}, ID del Vehiculo: {veh_id}, Velocidad: {speed}")
            csv_writer.writerow([current_time, veh_id, speed])

        traci.simulationStep()

    traci.close()

# Leer el archivo CSV
nombre_archivo_entrada = csv_file_path
datos = pd.read_csv(nombre_archivo_entrada)

# Calcular la velocidad promedio de cada vehiculo
velocidad_promedio_por_vehiculo = datos.groupby('VehicleID')['Speed'].mean()

# Calcular la velocidad promedio general
velocidad_promedio_general = datos['Speed'].mean()

# Calcular el intervalo de confianza del 95% para la velocidad promedio general
intervalo_confianza = stats.sem(datos['Speed']) * 1.96

# Crear la figura y los ejes
plt.figure(figsize=(12, 6))

# Grafica 1: Velocidad promedio de cada vehiculo
plt.subplot(1, 2, 1)
sns.barplot(x=velocidad_promedio_por_vehiculo.index, y=velocidad_promedio_por_vehiculo.values)
plt.title('Velocidad promedio por vehiculo')
plt.xlabel('ID del vehiculo')
plt.ylabel('Velocidad promedio')

# Grafica 2: Velocidad promedio general con intervalo de confianza del 95%
plt.subplot(1, 2, 2)
plt.bar('Velocidad promedio', velocidad_promedio_general, yerr=intervalo_confianza, color='skyblue', capsize=7)
plt.title('Velocidad promedio general con IC 95%')
plt.ylabel('Velocidad promedio')
plt.xticks([])

# Mostrar las graficas

```

```
plt.tight_layout()  
plt.show()
```

Código 1.8: Script **traci_vel.py** para correr simulación TraCI/SUMO y calcular la velocidad promedio.

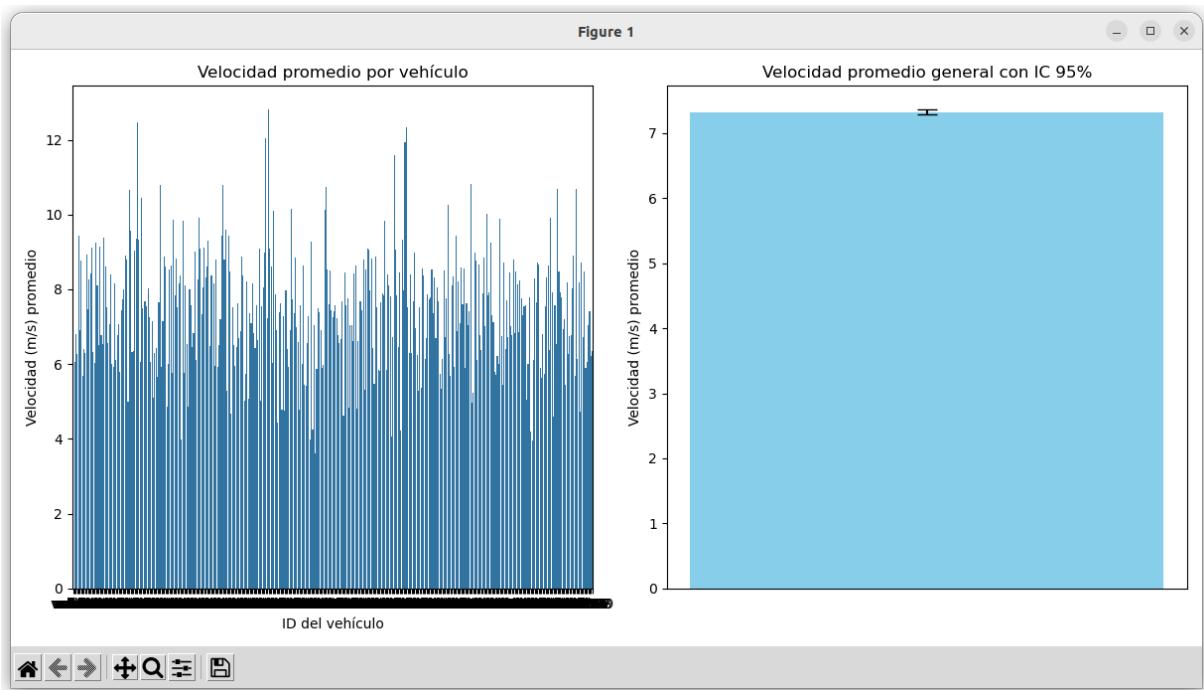


Figura 1.37: Cálculo de velocidades promedio con TraCI/SUMO.

1.9. Práctica: Simulación de tráfico vehicular TraCI/SUMO. Cálculo de emisiones CO₂

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Integrar la interfaz TraCI con SUMO.
- Simular una red vehicular sobre SUMO utilizando scripts en python.
- Calcular las emisiones de CO₂ de los vehículos que circularon en el mapa.

Introducción

Dentro del conjunto de herramientas que ofrece SUMO, TraCI desempeña un papel crucial al permitir la interacción en tiempo real entre simulaciones externas y el simulador SUMO. Una de las capacidades más destacadas de TraCI es su capacidad para calcular y registrar las emisiones de dióxido de carbono (CO₂) de los vehículos en la simulación.

El propósito principal de esta práctica de laboratorio es explorar cómo funciona TraCI, con un énfasis particular en el cálculo de las emisiones de CO₂ de los vehículos en la simulación. A través de ejercicios prácticos, los participantes aprenderán a utilizar las capacidades de TraCI para medir y analizar las emisiones de CO₂ generadas por el tráfico urbano en tiempo real. Durante la práctica, los participantes desarrollarán habilidades para implementar scripts y programas que aprovechen la funcionalidad de cálculo de emisiones de CO₂ de TraCI. Esto les permitirá evaluar el impacto ambiental de diferentes escenarios de tráfico, identificar áreas de alta emisión y explorar estrategias para mitigar el impacto ambiental del transporte en entornos urbanos.

Materiales

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

Instrucciones

A continuación, se describen los pasos a seguir para lograr correr una simulación de SUMO, utilizando la librería `traci` disponible para python. Esto se logra a través de la configuración de un script en python para correr el archivo de configuración de sumo con el uso de la interfaz TraCI.

1. El primer paso es importar las librerías necesarias. En este caso se utilizará `traci` para la conexión con SUMO. Además de, `csv` para el manejo de archivos csv, `time` para obtener los tiempos de simulación, `panda` para manejo de datos, `matplot` para gráficas y `seaborn`, `stats` para estadísticas.
2. Luego, es necesario iniciar la comunicación SUMO/TraCI a través del comando `traci.start(["cmd"])`.
3. Luego, indicar la ruta del archivo de salida, abrir el archivo csv e iniciar el bucle de la simulación para guardar los identificadores de los vehículos, emisiones CO2 y tiempos de simulación. En este caso se 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)
```

4. Finalmente, es necesario crear una lógica con la cual lee la información del archivo de salida csv, para luego obtener las emisiones (**mg/s**) media de cada vehículo y de la simulación en general. Finalmente, se gráfica las emisiones de CO2 obtenidas.
5. A continuación, se presenta el script utilizado para correr una simulación de SUMO, conectando con TraCI.

```
import traci
import csv
import time
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# Iniciar la conexión con SUMO
traci.start(["sumo", "-c", "/home/sumo/sumo/tools/2024-02-05-16-21-35/
osm.sumocfg"])

# Archivo de salida
```

```

csv_file_path = 'datos_simulacion_co2.csv'

with open(csv_file_path, 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)

    # Cabecera del CSV
    csv_writer.writerow(['Time', 'VehicleID', 'CO2'])

    # Ejecutar la simulacion
    while traci.simulation.getMinExpectedNumber() > 0:
        current_time = traci.simulation.getCurrentTime() / 1000 # Convertir a segundos

        for veh_id in traci.vehicle.getIDList():
            co2 = traci.vehicle.getCO2Emission(veh_id)
            print(f"Tiempo: {current_time}, ID del Vehiculo: {veh_id}, CO2: {co2}")
            csv_writer.writerow([current_time, veh_id, co2])

        traci.simulationStep()
    traci.close()

# Leer el archivo CSV
nombre_archivo_entrada = csv_file_path
datos = pd.read_csv(nombre_archivo_entrada)

# Calcular la CO2 promedio de cada vehiculo
co2_promedio_por_vehiculo = datos.groupby('VehicleID')['CO2'].mean()

# Calcular la CO2 promedio general
co2_promedio_general = datos['CO2'].mean()

# Calcular el intervalo de confianza del 95% para el CO2 promedio general
intervalo_confianza = stats.sem(datos['CO2']) * 1.96

# Crear la figura y los ejes
plt.figure(figsize=(12, 6))

# Grafica 1: CO2 promedio de cada vehiculo
plt.subplot(1, 2, 1)
sns.barplot(x=co2_promedio_por_vehiculo.index, y=co2_promedio_por_vehiculo.values)
plt.title('CO2 promedio por vehiculo')
plt.xlabel('ID del vehiculo')
plt.ylabel('CO2 promedio')

# Grafica 2: CO2 promedio general con intervalo de confianza del 95%
plt.subplot(1, 2, 2)
plt.bar('CO2', co2_promedio_general, yerr=intervalo_confianza, color='skyblue', capsize=7)
plt.title('CO2 promedio general con IC 95%')
plt.ylabel('CO2 promedio')
plt.xticks([])

# Mostrar las graficas
plt.tight_layout()

```

```
plt.show()
```

Código 1.9: Script **traci_co2.py** para correr simulación TraCI/SUMO y calcular las emisiones de CO₂.

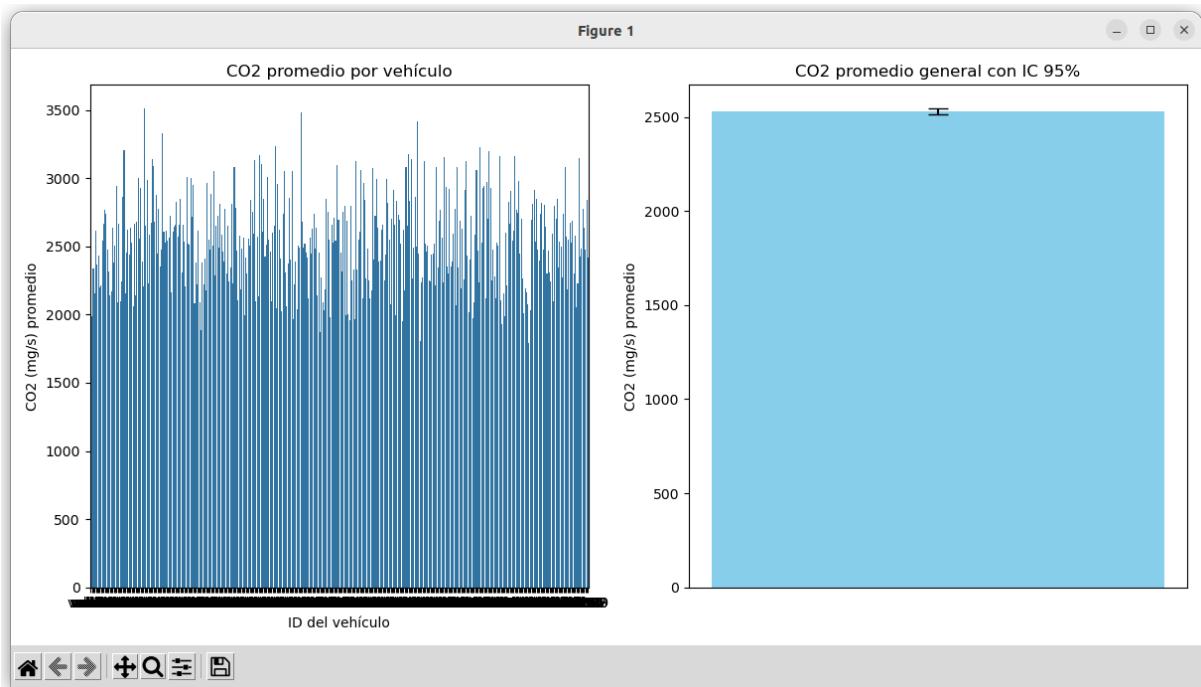


Figura 1.38: Cálculo de emisiones CO₂ promedio con TraCI/SUMO.

1.10. Práctica: Simulación de tráfico vehicular TraCI/SUMO. Cambio de trayectoria de vehículos

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Integrar la interfaz TraCI con SUMO.
- Simular una red vehicular sobre SUMO utilizando scripts en python.
- Cambiar la trayectoria de los vehículos que circularon en el mapa, en base a una lógica dada.

Introducción

Dentro del conjunto de herramientas disponibles en SUMO, TraCI proporciona una interfaz dinámica que permite la interacción en tiempo real con la simulación de tráfico. Una de las características más destacadas de TraCI es su capacidad para calcular nuevas rutas para los vehículos en función de circunstancias específicas, como las velocidades de los vehículos en las calles.

El objetivo principal de esta práctica de laboratorio es explorar el potencial de TraCI para calcular nuevas rutas de vehículos en respuesta a diferentes condiciones de tráfico, especialmente en relación con las velocidades de los vehículos. Por ejemplo, si un vehículo experimenta una velocidad por debajo de cierto umbral, podría ser redirigido a su punto de partida para evitar congestiones o situaciones adversas. A lo largo de esta práctica, los participantes aprenderán a utilizar las capacidades de cálculo de nuevas rutas de TraCI y a desarrollar scripts y programas que aprovechen esta funcionalidad. Esto les permitirá explorar estrategias de gestión de tráfico que optimicen el flujo vehicular y mejoren la eficiencia del transporte urbano, contribuyendo así a la creación de entornos urbanos más seguros y sostenibles.

Materiales

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

Instrucciones

A continuación, se describen los pasos a seguir para lograr correr una simulación de SUMO, utilizando la librería `traci` disponible para python. Esto se logra a través de la configuración de un script en python para correr el archivo de configuración de sumo con el uso de la interfaz TraCI.

1. El primer paso es importar las librerías necesarias. En este caso se utilizará `traci` para la conexión con SUMO y `re` para el manejo de expresiones regulares. Además de, `csv` para el manejo de archivos csv, `time` para obtener los tiempos de simulación, `panda` para manejo de datos, `matplot` para gráficas y `seaborn`, `stats` para estadísticas.
2. Luego, es necesario iniciar la comunicación SUMO/TraCI a través del comando `traci.start(["cmd"])`.
3. Luego, indicar la ruta del archivo de salida, abrir el archivo csv e iniciar el bucle de la simulación para guardar los identificadores de los vehículos, emisiones CO₂ y tiempos de simulación. En este caso se 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) se utiliza:

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

Para cambiar el destino de un vehículos se utiliza:

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

4. Finalmente, es necesario crear una lógica con la cual se verifique la velocidad del vehículo. Si la velocidad es menor a 10 m/s, el destino del vehículo cambia. En este caso se está analizando el vehículo con ID `veh424` y se lo redirige al destino con ID `234463928#1`. 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 y dar clic sobre la carretera en cuestión. Para abrir esta herramienta basta con el comando `netedit` desde la terminal.

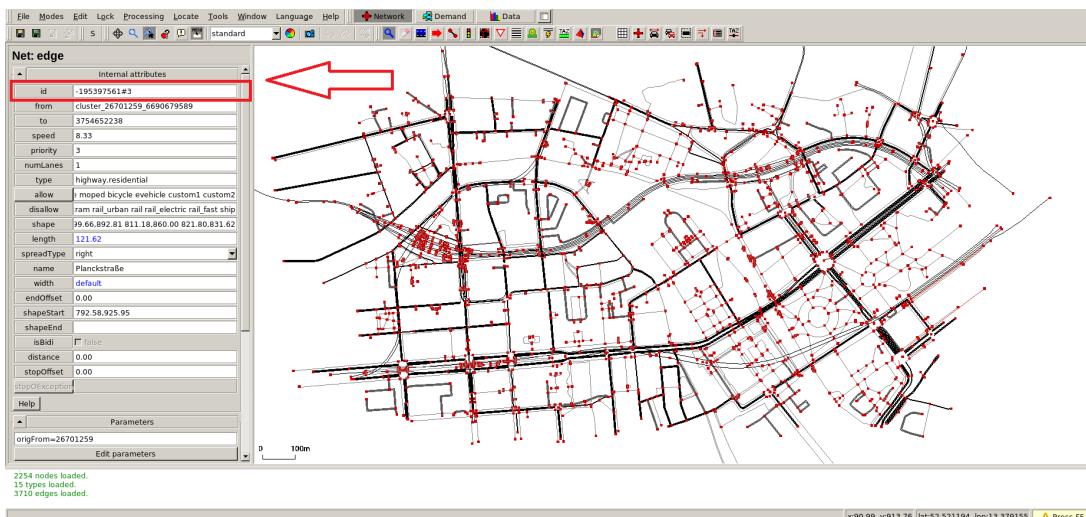


Figura 1.39: Identificador de calle desde netedit.

5. A continuación, se presenta el script utilizado para correr una simulación de SUMO desde la terminal, conectando con TraCI. Si se requiere correr la simulación mediante la interfaz gráfica de SUMO, se tiene que modificar la línea 6 del código y reemplazar `sumo` por `sumo-gui`.

```

import traci
import csv
import time

# Iniciar la conexión
traci.start(["sumo", "-c", "/home/sumo/sumo/tools/2024-02-05-16-21-35/osm.sumocfg"])

csv_file_path = 'datos_change_pos.csv'

with open(csv_file_path, 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)

    # Cabecera del CSV
    csv_writer.writerow(['Time', 'VehicleID', 'Speed', 'EdgeNow'])

    # Ejecutar la simulación
    while traci.simulation.getMinExpectedNumber() > 0:

        current_time = traci.simulation.getCurrentTime() / 1000 # Convertir a segundos

        for veh_id in traci.vehicle.getIDList():
            speed = traci.vehicle.getSpeed(veh_id)
            edgeNow = traci.vehicle.getRoadID(veh_id)

            if veh_id == "veh424" and speed > 10:

                traci.vehicle.changeTarget(veh_id, "234463928#1")
                print(f"**** SE HA CAMBIADO LA RUTA DEL VEHICULO {veh_id} HACIA 234463928#1")

```

```

    print(f"Tiempo: {current_time}, ID del Vehiculo: {veh_id},
          Velocidad: {speed}, EdgeNow: {edgeNow}")

    csv_writer.writerow([current_time, veh_id, speed, edgeNow])

    traci.simulationStep()

traci.close()

```

Código 1.10: Script **traci_change_pos.py** para correr simulación TraCI/SUMO y cambiar el destino de un vehículo.

En la Figura 1.40, se observa que el borde la carretera actual del vehículo *veh424* es 234463928#1, tal y como se esperaba. Además, se logra observar que el simulador realizó el re-enrutamiento del vehículo. Esta conclusión se llega debido a que los bordes de las carreteras previas, tienen el ID diferente. Es decir, el vehículo se movilizó por varias carreteras hasta llegar a su nuevo destino. Cabe mencionar que, por defecto SUMO utiliza el algoritmo Dijkstra para calcular las nuevas rutas.

```

Tiempo: 3904.0, ID del Vehículo: veh424, Velocidad: 15.803129615470938, EdgeNow: :2425782615_5
***** SE HA CAMBIADO LA RUTA DEL VEHICULO veh424 HACIA 234463928#1
Tiempo: 3905.0, ID del Vehículo: veh424, Velocidad: 15.220921826431699, EdgeNow: :1275468904_1
***** SE HA CAMBIADO LA RUTA DEL VEHICULO veh424 HACIA 234463928#1
Tiempo: 3906.0, ID del Vehículo: veh424, Velocidad: 15.466820430428893, EdgeNow: 234463928#1
***** SE HA CAMBIADO LA RUTA DEL VEHICULO veh424 HACIA 234463928#1
Tiempo: 3907.0, ID del Vehículo: veh424, Velocidad: 14.763622594298035, EdgeNow: 234463928#1
Simulation ended at time: 3908.00
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

```

Figura 1.40: Destino cambiado para vehículo *veh424*.

1.11. Práctica: Simulación de manejo de TLS para tráfico vehicular TraCI/SUMO. Reinforcement Learning

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

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.

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.

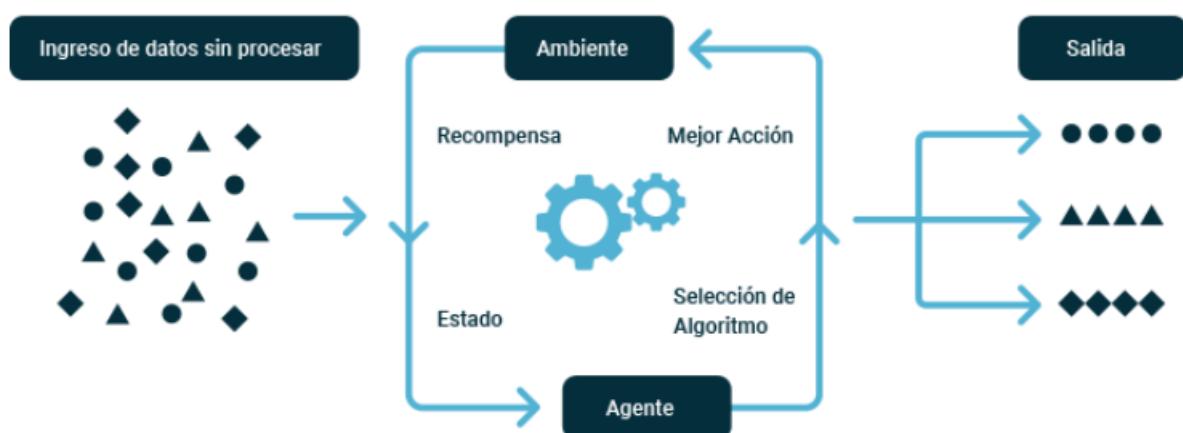


Figura 1.41: Aprendizaje por refuerzo.

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.

Materiales

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

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](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 1.42 con 4 nodos de semáforo: n1, n2, n3 y n4.

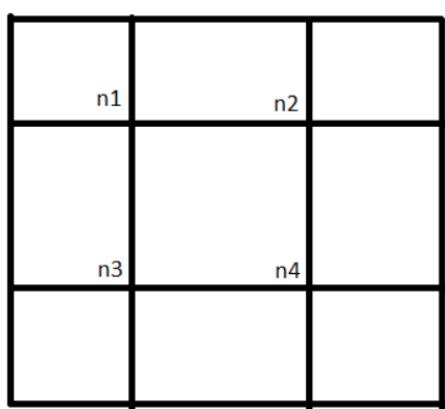


Figura 1.42: Ejemplo de ubicación de semáforos.

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 1.43 indica el diagrama de flujo que sigue el modelo de entrenamiento con reinforcement learning.

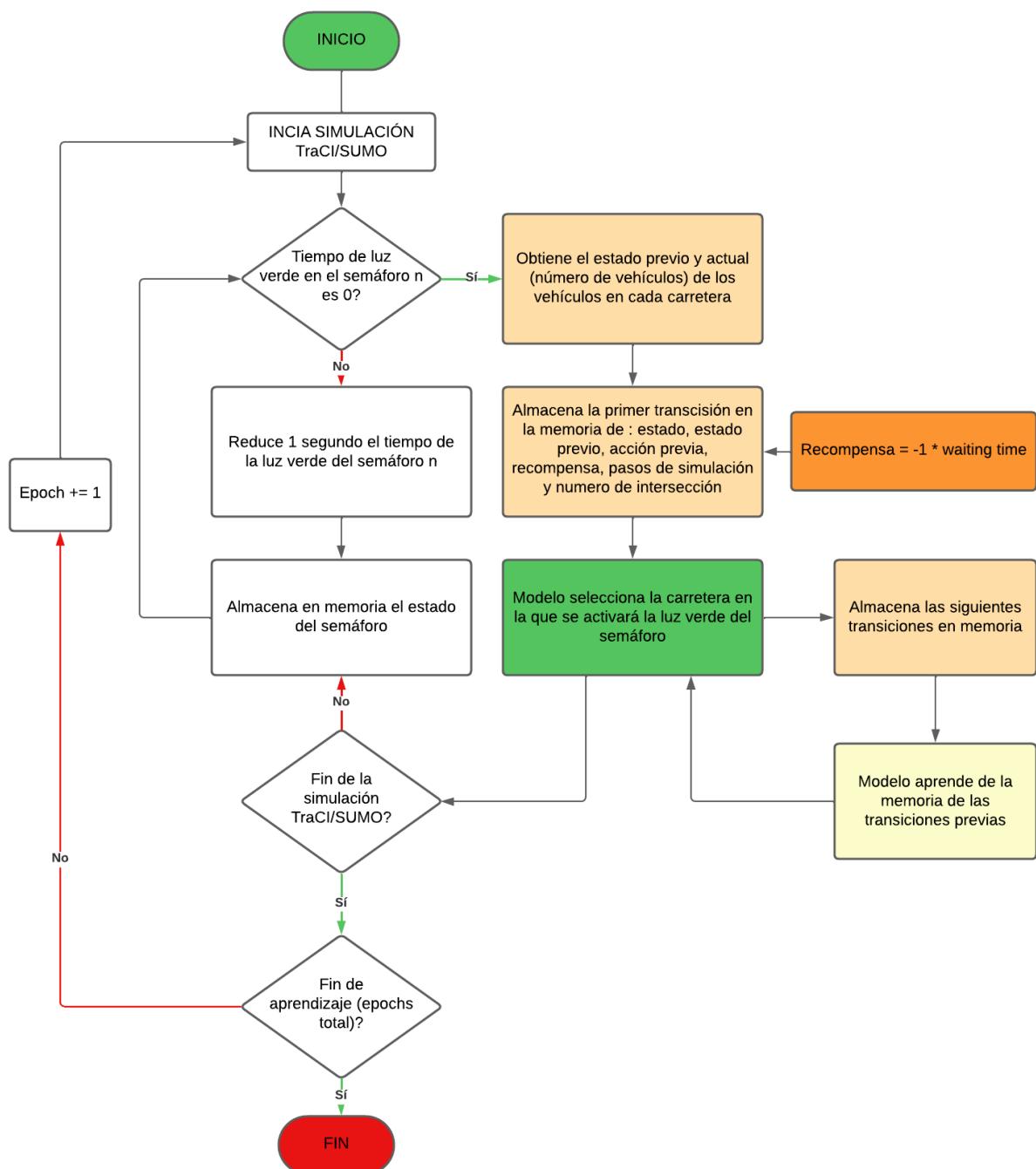


Figura 1.43: Diagrama de flujo de sistema de gestión de semáforo dinámico.

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 1.44 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.

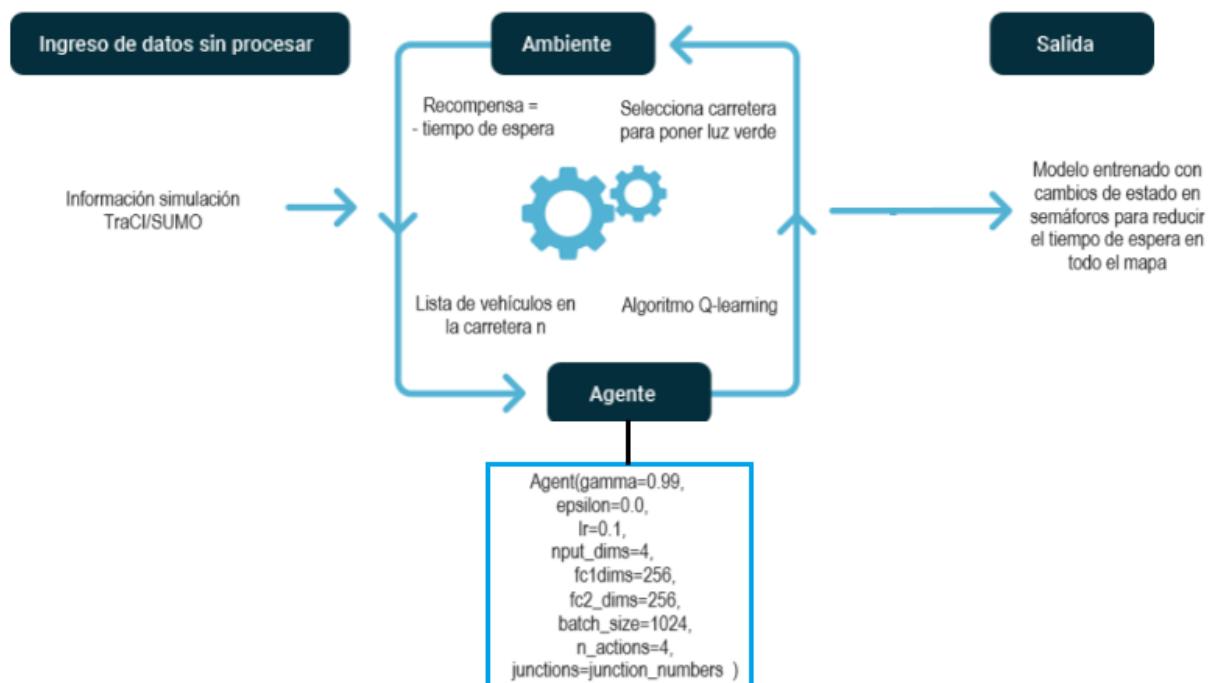


Figura 1.44: Aprendizaje por refuerzo para reducir el timepo de espera en una simulación TraCI/SUMO.

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 1.45.

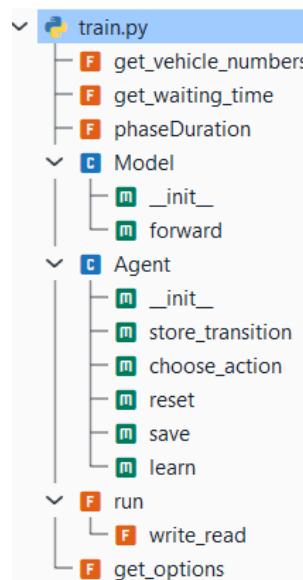


Figura 1.45: Funciones del programa de entrenamiento `train.py`

Donde, se observa que cuenta con tres funciones auxiliares (Figura 1.46), las cuales trabajan con la comunicación TraCI/SUMO para obtener y manejar los datos de la simulación: `get_vehicules_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)

```

Figura 1.46: Funciones auxiliares de `train.py`.

La primera de estas funciones, `get_vehicules_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 1.47 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

```

Figura 1.47: 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 entrenamiento. Donde, en la memoria general creada almacena los daots 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 crear 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 carretera, acción previa y el conjunto de todas las carreteras; y las opciones de configuración de

los semáforos.

```

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,
        iunctions=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"], 
            ["rrrryyrrrrrr", "rrrGGGrrrrrr"], 
            ["rrrrrrrryyrrr", "rrrrrrrGGGrrr"], 
            ["rrrrrrrrrryy", "rrrrrrrrrrGGG"], 
        ]
        ]

        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 calle y obtener todas las calles que son controladas 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 paso de simulación se obtiene las calles que son controladas por un semáforo, así como el tiempo total de espera por calle, 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 calle 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 [6]).

```

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):
        controlled_lanes = traci.trafficlight.getControlledLanes(junction)
        waiting_time = get_waiting_time(controlled_lanes)
        total_time += waiting_time
        if traffic_lights_time[junction] == 0:
            vehicles_per_lane = get_vehicle_numbers(controlled_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`.

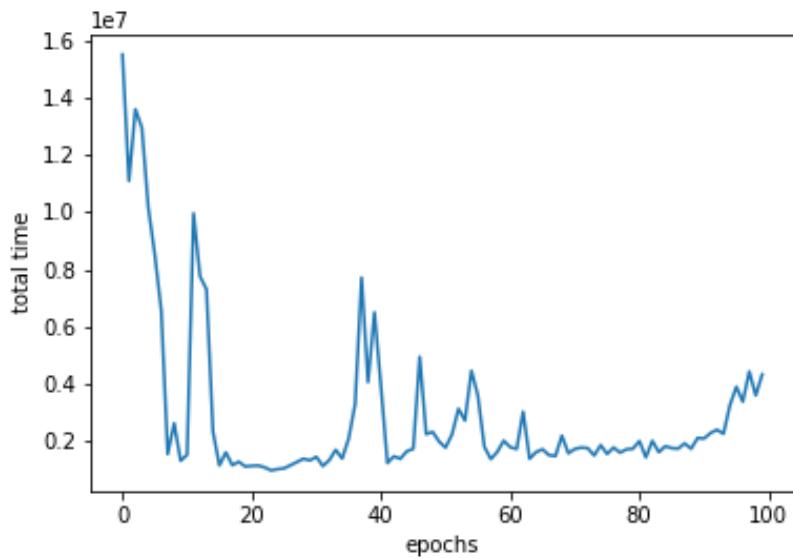


Figura 1.48: Gráfica: Tiempo vs epoch resultado del reinforcement learning en el mapa city1.net.xml.

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.

1.12. Práctica: Simulación de tráfico vehicular SUMO. Clasificación de datos con machine learning

- **Grupo:** Trabajo individual
- **Recursos:** Computador con sistema operativo Linux

Objetivos

- Simular una red vehicular sobre SUMO utilizando scripts en python.
- Obtener estadísticas de la simulación en SUMO.
- Clasificar datos mediante algoritmos de machine learning.

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 invaluable para comprender y tomar decisiones informadas sobre el impacto ambiental, como las emisiones de CO₂. En esta práctica de laboratorio, exploraremos la aplicación de varios algoritmos de clasificación de datos para analizar las emisiones de CO₂ generadas por vehículos en una simulación de tráfico urbano. Los algoritmos que usaremos incluyen:

1. **Regresión Logística (Logistic Regression):** Un algoritmo de clasificación lineal que modela la relación entre las variables de entrada y la probabilidad de pertenecer a una clase específica.
2. **SVC (Support Vector Classifier):** Un algoritmo que encuentra el hiperplano que mejor separa las clases en un espacio de características de alta dimensión.
3. **Random Forest Classifier:** Un 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.

Cada uno de estos algoritmos tiene sus propias ventajas y desventajas, y es crucial comprender cómo funcionan y cuándo es apropiado utilizarlos en diferentes situaciones. 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. Al finalizar la práctica, los participantes habrán adquirido habilidades prácticas en clasificación de datos y podrán aplicar este conocimiento para analizar y comprender mejor el impacto ambiental del tráfico vehicular en entornos urbanos simulados.

Materiales

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

Instrucciones

A continuación, se describen los pasos a seguir para correr dos programas en python que clasifican los datos obtenidos a partir de una simulación vehicular en SUMO.

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 [7] 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. 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>
```

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, intertools, sklearn para clasificar los datos con machine learning y mlxtend para observar las tendencias de la clasificación de datos.

Nota: Se debe tener en cuenta que por objetivos de demostración de funcionamiento de los siguientes algoritmos, se toma los datos de emisiones CO₂ y se los grafica en los ejes x, y. Es claro que se obtendrá una grafica lineal con pendiente positiva. En este punto es tarea del analizador de datos elegir bien los datos a graficar para obtener la clasificación de los mismos.

5. Además, se hace uso de una función propia para obtener el conjunto de datos a clasificar con los algoritmos de `sklearn` y `mlxtend`. Esta función requiere como entrada el archivo csv de salida de la simulación SUMO. Dentro de esta función se toma solo los datos de `tripinfo_id`, `emissions_CO2_abs`, `tripinfo_vType`, luego remplaza los valores de la columna `vType` mediante el siguiente mapeo:
`mapeo = {'taxi': 1, 'motorcycle': 2, 'bus': 3}`. Finalmente, devuelve un array con los valores filtrados. A continuación te indica el código de esta función de filtrado del conjunto de datos a clasificar.

```
import pandas as pd
import numpy as np
from collections import defaultdict

def data_set(archivo_entrada):
    # Nombre del archivo CSV de entrada y salida
    archivo_entrada = archivo_entrada
    archivo_salida = 'dataset-' + archivo_entrada

    # Cargar el archivo CSV en un DataFrame de pandas
    df = pd.read_csv(archivo_entrada)

    # Seleccionar solo las columnas 'vehicle_CO' y 'vehicle_type'
    # columnas_a_mantener = ['timestep_time', 'vehicle_CO', 'vehicle_type']
    columnas_a_mantener = ['tripinfo_id', 'emissions_CO2_abs', 'tripinfo_vType']
    df_filtrado = df[columnas_a_mantener]

    # Reemplazar los valores en la columna 'vType'
    mapeo = {'taxi': 1, 'motorcycle': 2, 'bus': 3}
    df_filtrado['tripinfo_vType'] = df_filtrado['tripinfo_vType'].map(mapeo)

    # Eliminar las dos primeras filas
    df_filtrado = df_filtrado.iloc[2:]

    # Ordenar las filas seg n el valor de la tercera columna (vehicle_type)
    df_filtrado = df_filtrado.sort_values(by='tripinfo_vType', ascending=True)
```

```

# Eliminar las filas que contienen 0 en la segunda columna,
# excepto aquellas con un 3 en la tercera columna
# df_filtrado = df_filtrado[(df_filtrado.iloc[:, 1] != 0) | (
#     df_filtrado.iloc[:, 2] == 3)]
# Eliminar las filas que contienen 0 en la segunda columna
# df_filtrado = df_filtrado[df_filtrado.iloc[:, 1] != 0]

# Guardar el DataFrame filtrado en un nuevo archivo CSV
df_filtrado.to_csv(archivo_salida, index=False, header=False)

# Convertir el DataFrame filtrado en un array de NumPy
resultado_array = df_filtrado.to_numpy()

# Creamos un diccionario para almacenar los valores de x basados
# en los valores de y
X = resultado_array[:,[1]]
y = np.asarray(resultado_array[:,2], dtype = 'int')
valores_por_y = defaultdict(list)

# Iteramos sobre x e y simultaneamente
for valor_x, valor_y in zip(X, y):
    valores_por_y[valor_y].append(valor_x)

# Calculamos la media para cada valor en y
medias_por_y = {}
for valor_y, valores_x in valores_por_y.items():
    medias_por_y[valor_y] = sum(valores_x) / len(valores_x)

# Imprimimos los resultados
print("Se han eliminado las columnas excepto 'timestep_time' ,
      'vehicle_CO' y 'vehicle_type'")
print("Se han reemplazado los valores taxi, motorcycle, bus por
      3,2,1.")
for valor_y, media in medias_por_y.items():
    print(f"Media para y={valor_y}: {media}")

return resultado_array

```

Código 1.11: Script **data_set.py** para filtrar los datos de la simulación SUMO.

6. A continuación, se indica el código utilizado para obtener la clasificación de datos con los algoritmos:

- *LogisticRegression*,
- *SCV*,
- *RandomForestClassifier* y
- *EnsembleVoteClassifier*

```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import itertools
from sklearn.linear_model import LogisticRegression

```

```

from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import EnsembleVoteClassifier
from mlxtend.data import iris_data
from mlxtend.plotting import plot_decision_regions

import data_set

# Initializing Classifiers
clf1 = LogisticRegression(random_state=20)
clf2 = RandomForestClassifier(random_state=0)
clf3 = SVC(random_state=0, probability=True)
eclf = EnsembleVoteClassifier(clfs=[clf1, clf2, clf3], weights=[2, 1,
    1],
    voting='soft')

# Loading some example data
# X, y = iris_data()
# X = X[:, 0]

data = data_set.data_set('tripinfo-emissions.csv')
X = data[:, [1, 2]]
y = np.asarray(data[:, 3], dtype = 'int')

# Plotting Decision Regions
gs = gridspec.GridSpec(2, 2)
fig = plt.figure(figsize=(10, 8))

for clf, lab, grd in zip([clf1, clf2, clf3, eclf],
    ['Logistic Regression (1:bus 2:motorcycle 3:
        taxi)',
     'Random Forest (1:bus 2:motorcycle 3:taxi)',
     'RBF kernel SVM (1:bus 2:motorcycle 3:taxi)'
     '',
     'Ensemble (1:bus 2:motorcycle 3:taxi)'],
    itertools.product([0, 1], repeat=2)):
    clf.fit(X, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X[:-1], y=y, clf=clf)
    plt.title(lab)
    plt.show()

```

Código 1.12: Script **mlxtend.py** para clasificar los datos de la simulación SUMO.

7. El algoritmo clasifica los datos etiquetados que se ingresan a los algoritmos de **sklearn**. Los datos son **X**, **y**, donde **X** representa las medidas de CO2 absolutas medidas en la simulación de cada vehículo sobre ella; **y** representa la etiqueta de datos, es decir, a que tipo de vehículo pertenece cada valor medido. En la Figura 1.49 se ilustra el resultado de clasificar las emisiones CO2 medidas en una simulación de SUMO.

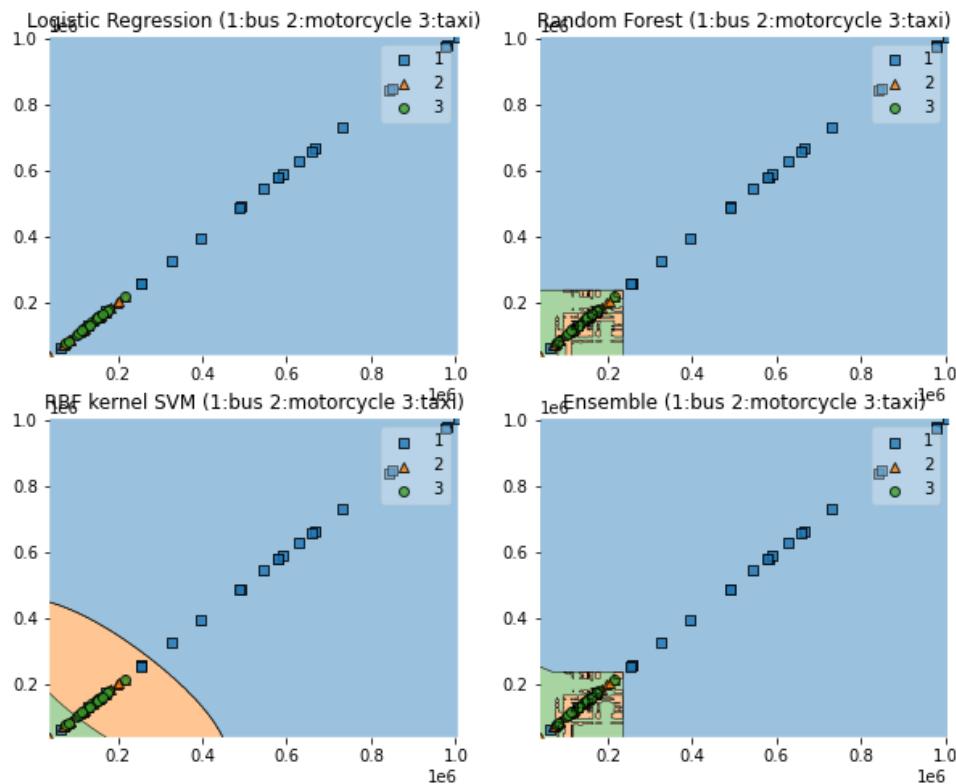


Figura 1.49: Clasificación de emisiones CO₂ etiquetadas de simulación SUMO.

Como se puede observar en la Figura 1.49, se obtiene el mapa de colores de clasificación de emisiones de datos etiquetados. Es decir, a los algoritmos de ingreso información ya etiquetada, donde se tiene que especificar a que tipo de vehículos corresponde el valor medido y como salida se obtiene un mapa de predicción, donde se indica las regiones en las que trabaja cada vehículo.

Sin embargo, hay otro algoritmo de machine learning que permite clasificar datos no etiquetados. Este algoritmo es `kmeans`, el cual calcula los valores centro y a partir de esta se mide la distancia euclíadiana de los valores cercanos para etiquetarlos y clasificarlos.

8. A continuación se indica el código en python, el cual usa datos del archivo de salida 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.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

df = pd.read_csv('tripinfo-emissions.csv')
df.head()

df_sub = df[['emissions_CO2_abs', 'emissions_CO2_abs']]

kmeans = KMeans(n_clusters=3, n_init=10).fit(df_sub.values)
```

```

df['Cluster'] = kmeans.labels_

plt.figure(figsize=(5,5), dpi=100)
colors = ['blue', 'red', 'green']

# graficar puntos de clusters
for cluster in range(kmeans.n_clusters):
    plt.scatter(df[df["Cluster"] == cluster][['emissions_CO2_abs']],
                df[df["Cluster"] == cluster][['emissions_CO2_abs']],
                marker='o', s=5, color=colors[cluster])

# graficar centroides
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            marker='*', s=20, linewidths=3, color='purple', label='Centroides')

plt.title("Clasificación de emisiones CO2 absolutas")
plt.xlabel("emissions_CO2_abs")
plt.ylabel("emissions_CO2_abs")
plt.grid(visible=True)
plt.show()

```

Código 1.13: Script **kmeans.py** para clasificar los datos no etiquetados de la simulación SUMO.

9. A continuación se ilustra el resultado de correr el script **kmeans.py**.

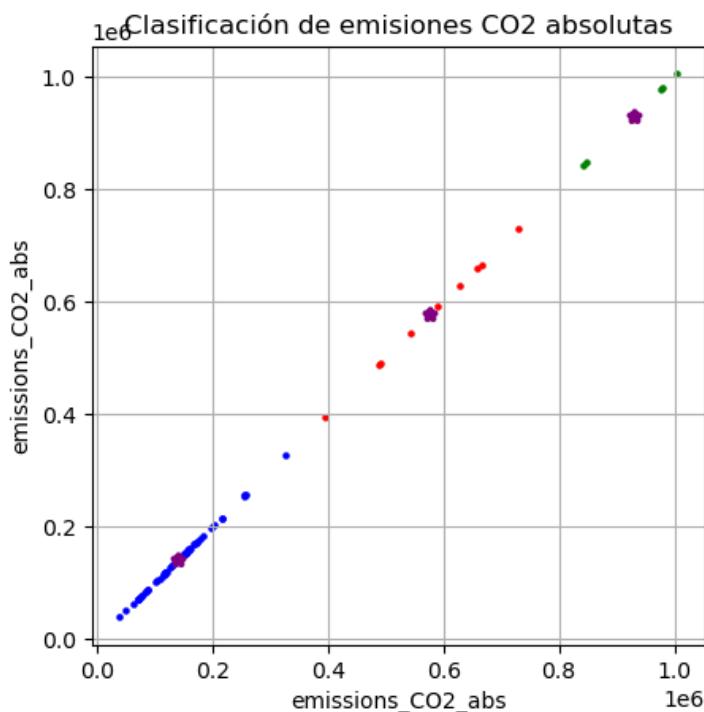


Figura 1.50: Clasificación de emisiones CO2 no etiquetadas de simulación SUMO.

Bibliografía

- [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] "Netconvert," <https://sumo.dlr.de/docs/netconvert.html>, accessed: 2020-10-06.
- [3] "Polyconvert," <https://sumo.dlr.de/docs/polyconvert.html>, accessed: 2020-10-06.
- [4] "Random trips," <https://sumo.dlr.de/docs/Tools/Trip.html>, accessed: 2020-10-06.
- [5] "Outputs," <https://sumo.dlr.de/docs/Simulation/Output/index.html>, accessed: 2024-02-05.
- [6] D. Maunish, "Dynamic-Traffic-light-management-system," 2021. [Online]. Available: <https://github.com/Maunish-dave/Dynamic-Traffic-light-management-system/tree/main>
- [7] SUMO, "Vehicle Type Parameter Defaults," 2024. [Online]. Available: https://sumo.dlr.de/docs/Vehicle_Type_Parameter_Defaults.html