Diseño e implementación de arquitectura para control inteligente de tráfico urbano

Alumno: Br. Giovanny González Baltazar

Asesor interno: Dr. Mauricio Gabriel Orozco del Castillo

Asesor externo: Dr. Joel Antonio Trejo Sánchez

Introducción

Los países en vías de desarrollo en innumerables ocasiones han tenido que afrontar problemas a los que otros países ya se han enfrentado antes. Las que antes eran pequeñas urbes, se están sometiendo a un proceso de imparable crecimiento que hace que día con día se asemejen cada vez más a las grandes metrópolis comunes en los países más desarrollados.

No es posible detener la rueda, pero es es una excelente oportunidad para aprovechar la gran base de conocimientos acumulados por otras ciudades y las poderosas herramientas tecnológicas de la modernidad para resolver las problemáticas emergentes de manera innovadora y óptima.

Uno de esos problemas es el tráfico urbano.

En las ciudades, el correcto manejo de los grandes volúmenes de tráfico siempre ha sido un problema. Por ello, a lo largo de la historia han existido diversos acercamientos para intentar solucionar esta problemática. Un caso célebre y funcional es el del semáforo, que comenzó a masificarse a partir de la explosión del uso del *Modelo T* en Estados Unidos. Por muchos años esta ha sido una solución efectiva, pero conforme las urbes crecen más y más, llegan a ser insuficientes para tal cantidad de volumen de tráfico. Es en tiempos más recientes cuando se han comenzado a implementar estrategias que le brindan 'inteligencia' al comportamiento de los mismos, a través el uso de inductores para condicionar el comportamiento del semáforo, o detectores de luces infrarrojas para detectar el patrón de las luces de la sirena de patrullas y ambulancias y priorizar su circulación.

Planteamiento del problema

Una cantidad significativa de la actividad en un área urbana tiene que ver con el movimiento de personas y bienes entre diferentes lugares usando la infraestructura de transporte, y un eficiente y fluido sistema de transporte es esencial para la salud económica y la calidad de vida dentro de las regiones urbanas. (Patriksson, 1994, p. 3)

Conforme crecen las ciudades y se arraiga cada vez más la idea de la necesidad de tener trasporte personal, inevitablemente crece la demanda del sistema de transporte. Este aumento trae consigo problemas serios, como:

- Mayor número de accidentes
- Uso ineficiente del sistema de transporte debido a grandes congestiones.
- Deterioro de la calidad de vida de las zonas adyacentes
- Polución
- Contaminación sonora

(Patriksson, 1994)

> La congestión de tráfico urbano causa considerables costos debido a pérdidas de tiempo, incrementa la posibilidad de accidentes, y los problemas de contaminación en las principales ciudades por lo que tiene un impacto negativo en el ambiente. También es responsable de problemas de salud tales como estrés, ruido y complicaciones similares. La solución de aumentar la dimensión de la red de tráfico urbano no siempre resulta ser la mejor opción además es muy difícil y muy costosa, especialmente en áreas urbanas. (Antonio & Sánchez, 2006, Introducción)

Propuesta de solución

El problema que se afrontará es el de las congestiones usando sistemas inteligentes de control de tráfico, o dicho de otra manera: *semáforos inteligentes*.

El objetivo es que los semáforos actúen similar a los típicos oficiales de tránsito que a veces se encuentran posicionados estratégicamente en las intersecciones más concurridas, pero con esteroides. Dichos policías observan la demanda de tráfico que ocurre en cada carril y en base a ello toman la decisión de a cuál dar el paso para mantener en todo momento un flujo de tráfico óptimo y acortar los tiempos de espera. El problema de usar personas para esta tarea; aparte del costo y la inviabilidad de hacerlo en cada inserción de una ciudad; es mantener el flujo constante, pues ¿de qué sirve que en una intersección un vehículo tenga la vía libre si al llegar a la siguiente no tendrá paso? Entonces el flujo se rompe y se genera una congestión que afecta todos los vehículos que lo suceden. Para solucionar este problema, pensamos que la clave es la coordinación entre los "oficiales de tránsito" y que estos formen una red en todos las intersecciones clave y tengan mentalidad de enjambre para que en todo momento cualquier oficial sepa cuál es la cantidad y dirección del flujo de tráfico en cada una de las intersecciones de la red (o incluso en otras redes adyacentes) y en base a ello tome de decisiones en tiempo real que permitan un flujo de tráfico lo más eficiente posible, y que dé pie a cosas tan útiles como dar paso completamente libre a ambulancias, patrullas y camiones de bomberos. Evidentemente esto no es tarea para un ser humano, pero si para un semáforo inteligente.

> Una de las principales respuestas al problema del control de tráfico urbano es reducir el tiempo de espera de los usuarios en la red de tráfico. Se puede reducir el tiempo de espera de los usuarios en la red de tráfico por medio del cambio dinámico de las señales desplegadas en los semáforos, y que este cambio se realice de acuerdo a la demanda de tráfico y a la coordinación con intersecciones adyacentes. (Trejo, 2006)

La idea de un semáforo que se adapta a la demanda de tráfico no es nueva. Ya existen acercamientos a esta idea, desde la más básica cambiando el ciclo de las luces dependiendo de la hora del día, hasta las más complejas y costosas que usan inductores posicionados estratégicamente antes de llegar a las intersecciones para contabilizar cuantos vehículos están esperando en cada carril para que los semáforos usen esta información para, por ejemplo, no dar paso a un carril que no tiene ningún vehículo esperando. El problema es que implementar la solución anterior es costoso, y aun así tiene mucho margen de mejora, pues la información que puede proporcionar un inductor se limita a si un vehículo pasa encima de él y la hora en la que sucede, lo que complica analizar el flujo real de cada uno de los vehículos. Por ello pretendemos darles uso a las cámaras que muchas veces ya se encuentran en las intersecciones y usar sus imágenes para alimentar algoritmos de machine learning de reconocimiento de objetos, y así saber cuántos, de que tipo y dirección de los vehículos. Ya que esta tarea tiene su propia serie de retos y complicaciones, está siento realizada por otros colegas en el CIMAT. Lo que nos ocupa a nosotros es suponer que contamos con la información que este algoritmo nos brindará y en base a ello desarrollar las estrategias inteligentes de control a usar por los semáforos.

Propuesta de Arquitectura

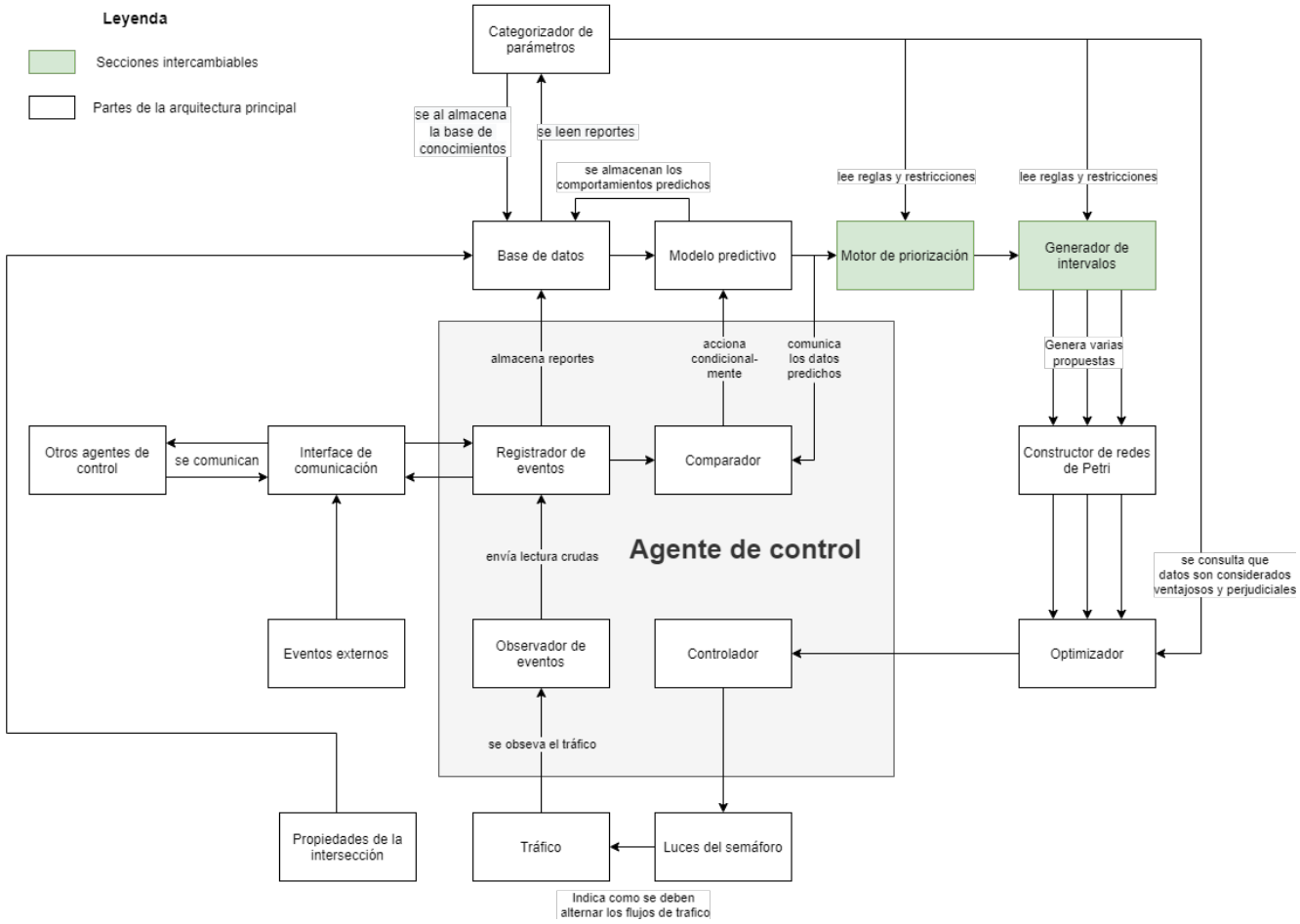


Fig. 1: Arquitectura de un agente de control.

Tráfico

Los vehículos que se mueven en los carriles de la intersección y cruzan de un carril a otro en nodos llamados intersecciones. Sobre las intersecciónes puede haber ubicados semáforos.

Propiedades de la intersección

Son las propiedades intrínsecas de la intersección que no suelen cambiar con el paso del tiempo:

- Número de calles.
- Carriles en cada calle.
- Extensión aproximada de cada carril.
- Estructura de la intersección: todas las posibles conexiones entre carriles.
- Derechos y prioridades de paso.
- Disposición de las luces del semáforo (cuántas hay y a que intersección apuntan).
- Existencia de cruces y puentes peatonales, y si tienen algún botón que permita detener el tráfico.

Observador de eventos

Su única tarea es ser el sentido de la vista del agente de control. Tiene funciones para observar eventos de interés que suceden con regularidad en el mundo real en un momento dado y sus propiedades, tales como:

- El paso de los vehículos, con propiedades como:
  - Ruta que toma (pueden seguir derecho o girar en algún sentido).

  - Velocidad promedio aproximada.
  - Cuanto tiempo permanece detenido.
  - Aceleración aproximada.
- Cambios de luces (en que carril y a que color cambió).

Y también notifica eventos emergentes y sus propiedades, como:

- Aparición de un vehículos de prioridad (de que tipo y en que carril).
- Un accidente (que carril obstruye).

Todo lo observado lo comunica al *Registrador de eventos* indicando el tiempo exacto en el que sucedió (*timestamp*).

Registrador de eventos

Es el encargado de guardar un caché de los datos crudos enviados por el *Observador de eventos* e interpretarlos, para cada hora generar reportes con datos derivados que se almacenan en la base de datos. El reporte generado agrupa los datos por ciclo de semáforo, siendo éstos los siguientes:

- Flujo por carril, es decir, el número de vehículos que pasan por cada carril cada determinado tiempo.
- Tiempo de espera promedio por carril.
- Densidad de cada carril.
- Cola más larga en la señal roja.
- Número de vehículos que pasan en señal verde.
- Tiempo ocioso de la señal verde.
- Cola más larga próxima fase.
- Tiempo de cambio de la cola más larga en señal roja.

También recibe datos al exterior a través de la *Interfaz de comunicación*, que se almacenan selectivamente en la base de datos.

Interfaz de comunicación

Permite enviar y recibir reportes de otros agentes de control, así como notificaciones de eventos externos que pueden alterar el tráfico, como:

- Alertas de desastre natural.
- Eventos públicos y días festivos.
- Manifestaciones.

Comparador

Se encarga de comparar constantemente el comportamiento de los eventos registrados con el comportamiento predicho. Cuando estos difieren de manera significativa, el comparador solicita al *Modelo predictivo* que genere una nueva predicción. También se encarga de solicitar nuevas predicciones cada que un evento externo lo solicite.

Modelo predictivo

Se alimenta del histórico de reportes generados por el *Registrador de eventos* que obtiene de la base de datos para predecir del comportamiento de una intersección durante la siguiente hora. Los datos que genera son los mismos que figuran el los reportes creados por el *Registrador de eventos*. También hace predicciones bajo demanda cuando el *Comparador* lo solicita.

(Incluir aquí gráfica que represente el rango de tiempo en el que son válidas las predicciones, la estimación de pérdida de tiempo efectivo por el tiempo de generación, etc.)

Ya que el proceso de hacer una predicción y su posterior optimización es un proceso intensivo, las peticiones para nuevas predicciones suceden cada hora, pero de manera escalonada. Por ejemplo: si existen 4 agentes de control, uno iniciará una petición de predicción a las 2:00 p.m., mientras que otra los hará a las 2:15 p.m., otro a las 2:30, y así sucesivamente, de tal manera que el tiempo entre cada hora se se reparta lo más igualitariamente posible. De la mano con estas peticiones escalonadas, las peticiones se realizarán de manera asíncrona usando hilos de procesamiento para lograr paralelizar el procesamiento de varias intersecciones a la vez de ser necesario. De esta

manera, se aprovecha también el sistema de encolamiento de hilos nativo de los sistemas operativos, que resulta particularmente útil para no atrasar las predicciones programadas por las realizadas bajo demanda.

Categorizador de parámetros

Se encarga de categorizar la influencia de los parámetro predichos entre sí mismos y como finalmente influyen sobre el comportamiento global del tráfico, para así obtener conclusiones en forma de restricciones y reglas aplicadas para al temporizado de los ciclos del semáforo. Categoriza intervalos de tiempo que presentan una serie de condiciones a términos abstractos, como: "hora pico", "hora sin actividad" o "tráfico esporádico", para así poder usar esas categorías como condiciones para las reglas y restricciones vistas anteriormente. Categoriza como ventajoso o perjudicial el incremento o decremento de un parámetro respecto a parámetros que se busquen incrementar o decrementar (como el tiempo de espera, la generación de colas, etc). De ser necesario, infiere una función de como su variación afecta a estos parámetros.

Ejemplo verbal: si se hacen ciclos en verde un un carril **mayores a** 45 segundos durante **horas pico**, se genera una **cola de espera perjudicial a la fluidez** del carril perpendicular, entonces se infiere una regla que impida que se rebase ese tiempo de ciclo dadas esas condiciones, y en caso de que se tenga que se rebase, se tenga la función que indica que tanto perjudica ese aumento para calcular los tiempos de ciclo que produzcan el mejor costo-beneficio (pues se tienen que tomar en cuenta el resto de los parámetros).

Este módulo analiza constantemente los reportes almacenados en la base de datos para realizar el proceso mencionado anteriormente y todo lo inferido se guarda de manera centralizada como una base de conocimientos central como registros en la base de datos.

Este módulo se inspiro en la llamada *Colección de hechos* usada en Antonio & Sánchez (2006).

> La colección de hechos alberga los datos correspondientes a la aplicación de determinada estrategia de control cuando se presentan ciertas condiciones en el tráfico observado. La colección de hechos puede desempeñar el papel de memoria auxiliar en la cual se registran los razonamientos llevados a cabo. (Antonio & Sánchez, 2006, p. 47)

Motor de priorización

Se encarga de asignar un valor numérico a cada carril, que indica su prioridad. Entre mayor el número, mayor la prioridad. Este número de genera usando como base la demanda predicha, aunque también es posible usar restricciones y reglas definidas por el *Categorizador de parámetros*.

Generador de intervalos

Usa los valores generados por el *Motor de priorización* para generar una proporción de la duración de cada fase del semáforo en la duración total del ciclo. Posteriormente usa como base las reglas y restricciones generados por el *Categorizador de parámetros* para generar varias propuestas de la duración total del ciclo de fases de luces. Se obtiene como salida varios juegos de propuestas de duraciones de fases que pueden haber tomado en cuenta diferentes reglas.

Constructor de redes de Petri

A partir de las propuestas de intervalos generadas por el *Generador de intervalos*, se encarga de construir una red de Petri temporizada para cada una de ellas.

Optimizador

Recibe múltiples propuestas de redes de Petri y a través de algoritmos genéticos, cruza los genes de las diferentes propuestas (que tomaron en cuenta diferente reglas para construirse) buscando maximizar los parámetros considerados como ventajosos (como el flujo) y minimizar los considerados como perjudiciales (como las cosas y los tiempos de espera). La información de cuáles son estos parámetros se obtiene del *Categorizador de parámetros*. Al final da como salida una red de Petri optimizada.

Controlador

Tiene las funciones para necesarias para leer la red de Petri temporizada que recibe y controlar los cambios de luces según lo indique la red.

Luces de semáforo

Son la manera en la que se plasman los intervalos de luces generados y son lo que físicamente indica a los conductores cuando deben circular o detenerse.

Recursos: SUMO

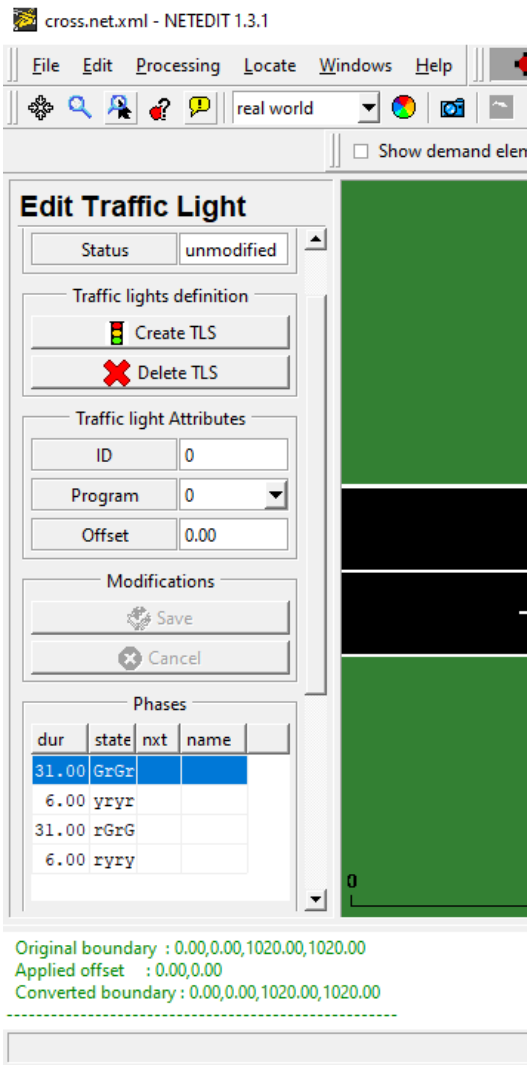
Para probar y desarrollar a detalle la arquitectura, es necesario contar con un entorno de simulación versátil y altamente configurable donde probar los algoritmos creados en base a nuestras hipótesis. El primer acercamiento para resolver reste problema fue desarrollar desde cero un simulador de tráfico, pero ya que esto tiene su propia complejidad y no el objetivo final al que pretendemos llegar, buscamos otras alternativas. De entre todas, la que parece ser la solución definitiva es el simulador de tráfico urbano SUMO. "**S**imulation of **U**rban **MO**bility" (Eclipse SUMO) es un paquete de simulación de tráfico vial de código abierto, altamente portátil, microscópico y continúo diseñado para manejar grandes redes viales. Permite simular cómo una determinada demanda de tráfico que consiste en vehículos individuales se mueve a través de una red de carreteras determinada. La simulación permite abordar un amplio conjunto de temas de gestión del tráfico. Es puramente microscópico: cada vehículo está modelado explícitamente, tiene una ruta propia y se mueve individualmente a través de la red. Las simulaciones son deterministas por defecto, pero hay varias opciones para introducir la aleatoriedad.

Al tratarse de un paquete, la instalación por defecto incluye varias aplicaciones, scripts e interfaces aparte de SUMO. Estas aplicaciones se utilizan para importar y preparar redes de carreteras, así como para procesar datos para su uso en SUMO.

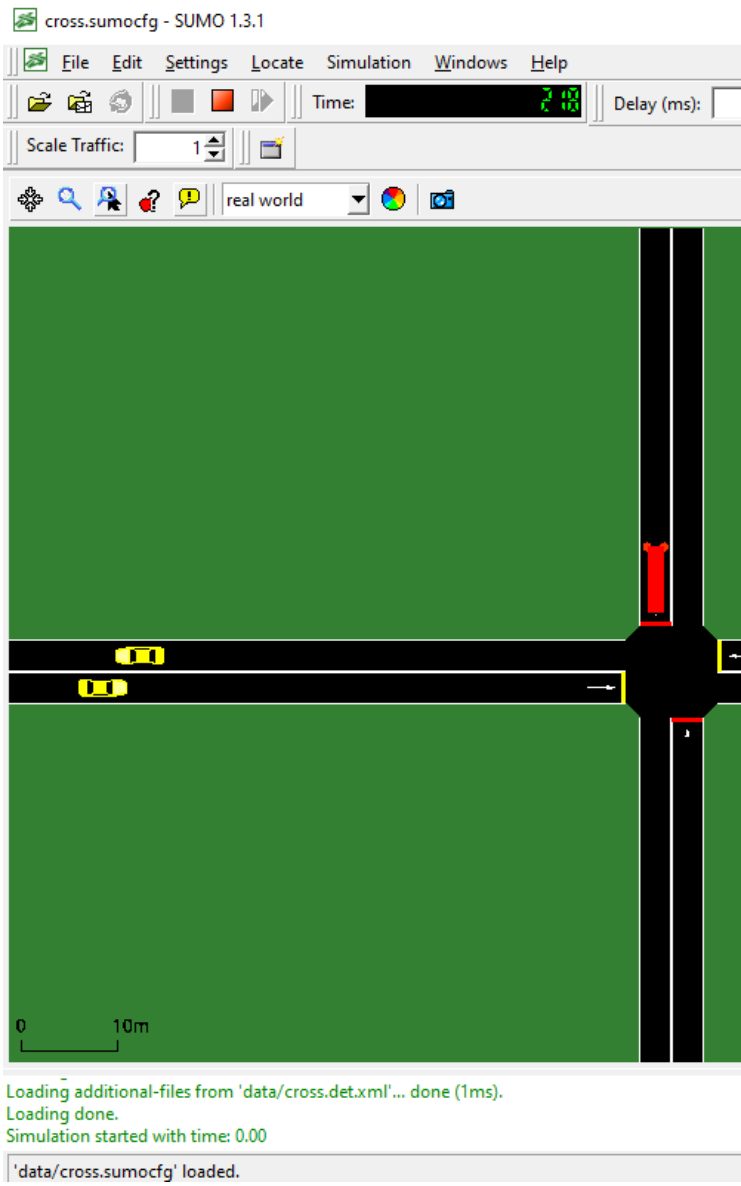
Características relevantes de SUMO para la investigación de técnicas de control de semáforos.

- Incluye todas las aplicaciones necesarias para preparar y realizar una simulación de tráfico
- Permite simular desde una solo intersección hasta ciudades enteras
- Altamente configurable a través de archivos XML
- Documentación completa y actualizada de todas las características, interfaces y librerías que incluye, así como guías, tutoriales y ejemplos de configuración de gran cantidad tópicos.
- Gran cantidad de tipos de vehículos disponibles, entre ellos de emergencia (ambulancias) y de autoridad (patrullas).
- Calles de varios carriles con cambio de carril, carriles configurables para permitir solo el tipo de tráfico especificado.
- Diferentes reglas de derecho de paso, semáforos
- Incluye interfaces en Python para obtener datos de la simulación en tiempo real, así como para controlar aspectos de esta, como los semáforos.
- Incluye editor gráfico de rutas y GUI para el simulador.
- Velocidad de ejecución rápida (hasta 100.000 actualizaciones de vehículo por segundo en una máquina de 1 GHz)
- Código abierto (EPL)

Vista general



Editor gráfico NETEDIT mostrando una inserción y el menú de edición de semáforos.

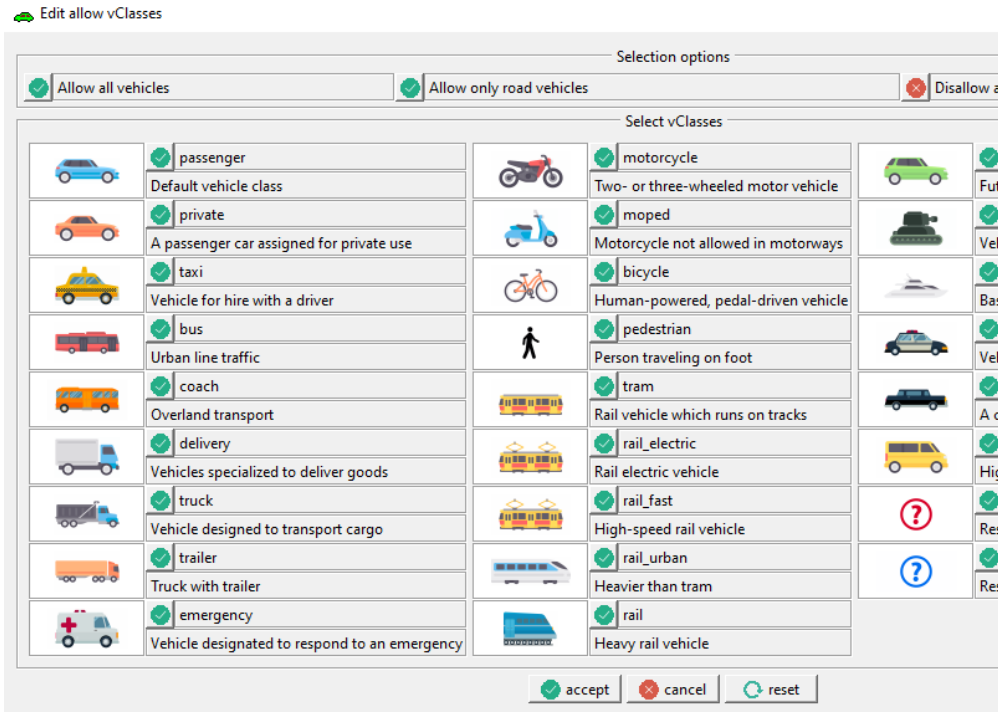


Simulando la intersección anterior de manera gráfica en SumoGui.

```python
def run():
    """execute the TraCI control loop"
    step = 0
    # we start with phase 2 where EW
    traci.trafficlight.setPhase("0",
    while traci.simulation.getMinExpec
        traci.simulationStep()
        if traci.trafficlight.getPhase
            # we are not already swit
            if traci.inductionloop.get
                # there is a vehicle
                traci.trafficlight.set
            else:
                # otherwise try to ke
                traci.trafficlight.set
        step += 1
    traci.close()
    sys.stdout.flush()


def get_options():
    optParser = optparse.OptionParser
    optParser.add_option("--nogui", a
                         default=False
    options, args = optParser.parse_a
```

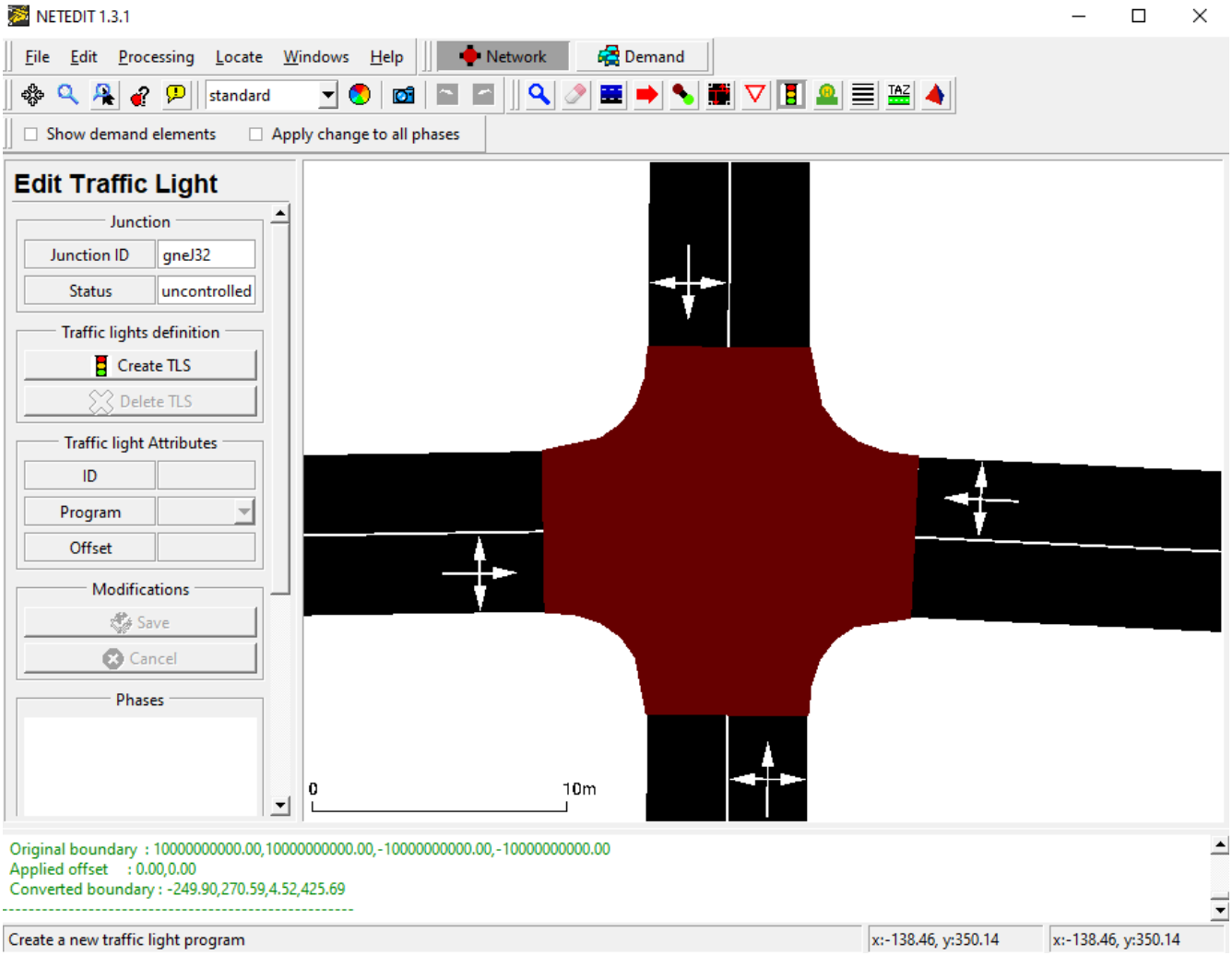
Controlando los semáforos desde Python con TraCI (Traffic Control Interface).

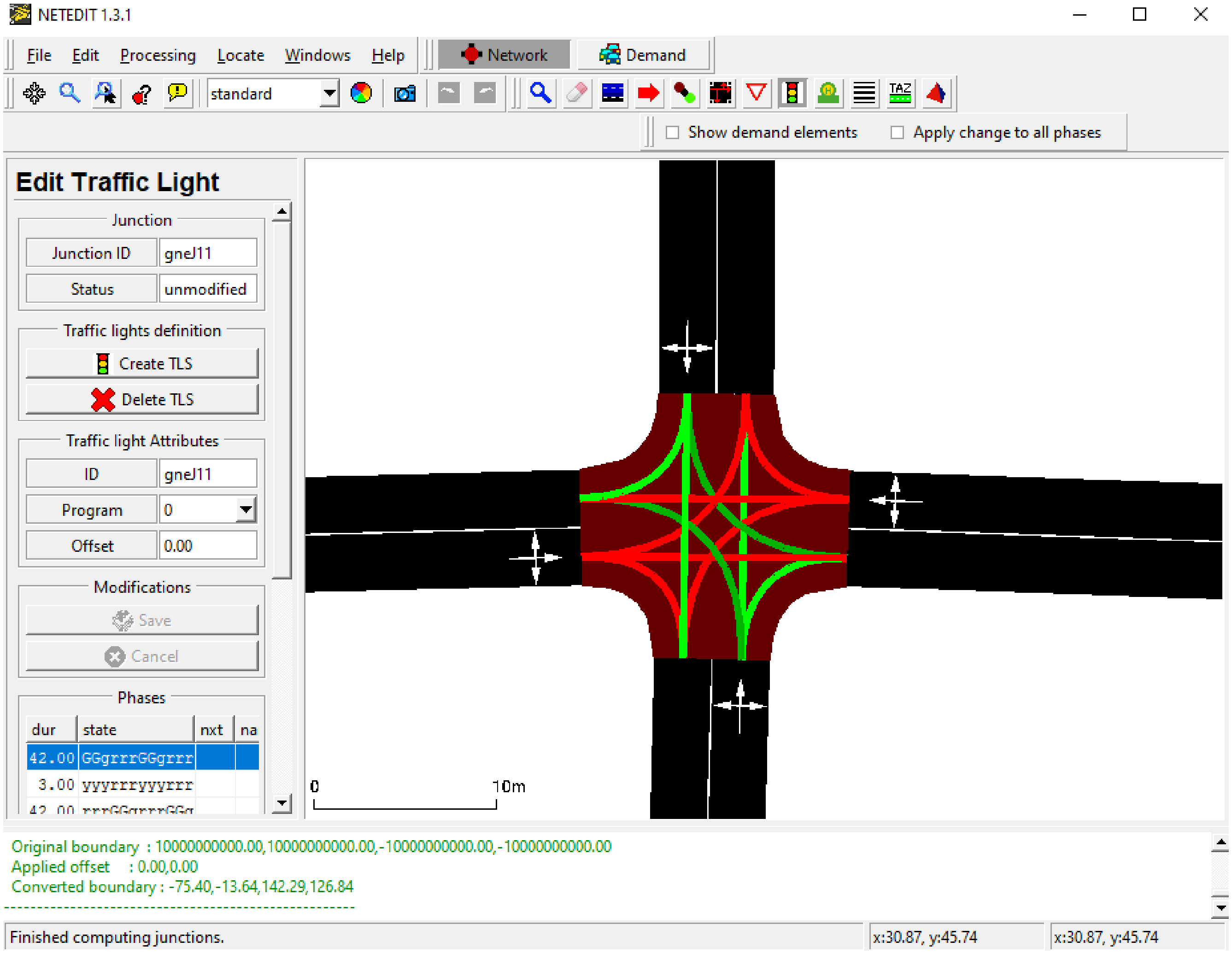


Los diversos tipos de vehículos disponibles.

Simulación de semáforos

Los semáforos (llamados en el simulador TLS - *Traffic Light System*) se pueden crear de manera gráfica en NETDIT y automáticamente generan un programa de control.





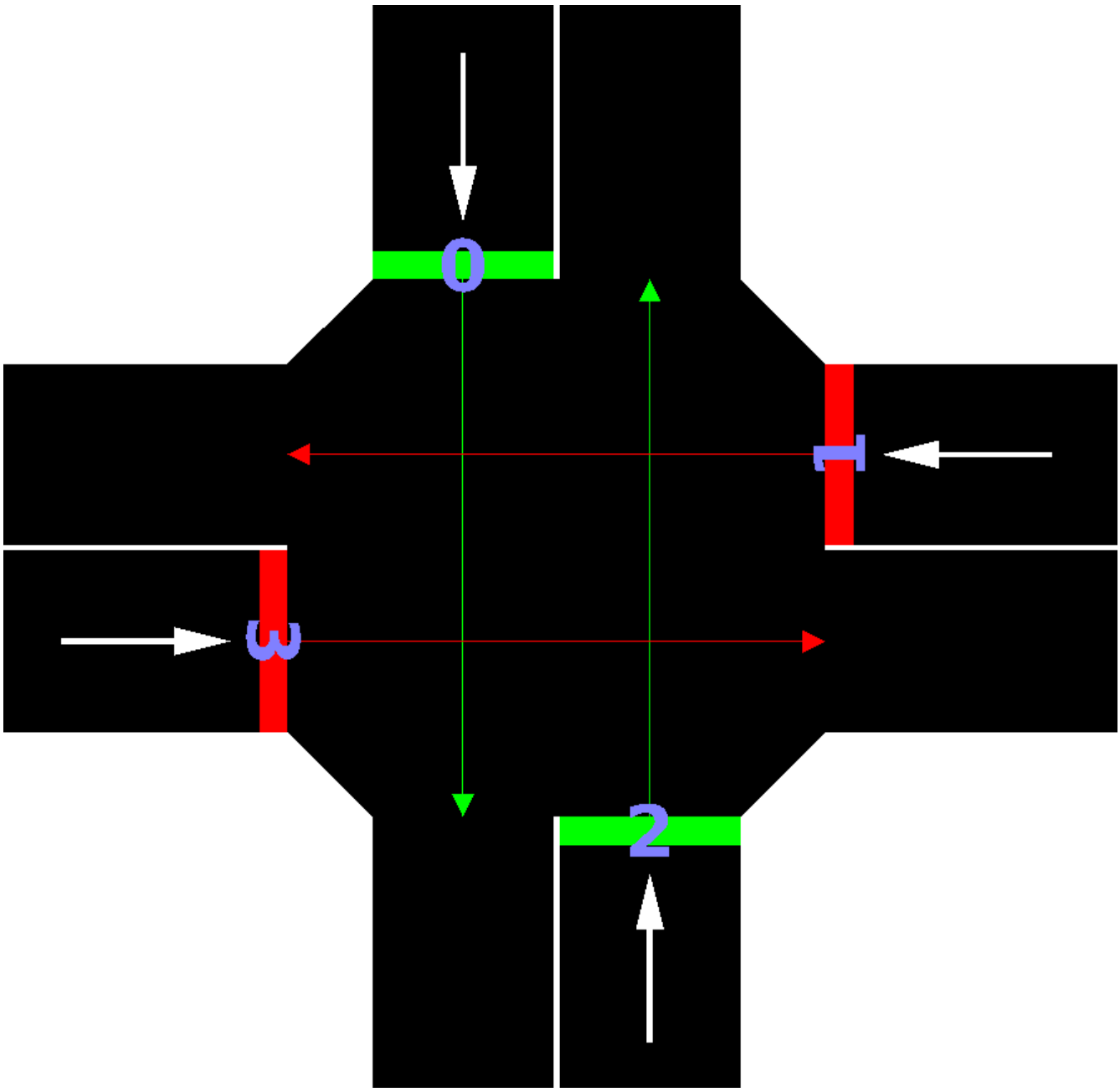
Definición de nuevos programas TLS

Se puede cargar nuevas definiciones para semáforos como parte de un archivo adicional. La definición de un programa de semáforo dentro de un archivo adicional se ve así:

```
<additional>
    <tlLogic id="semaforo_principal" type="static" programID="principal" offset="0">
        <phase duration="40" state="GrGr"/>
        <phase duration="6" state="yryr"/>
        <phase duration="40" state="rGrG"/>
        <phase duration="6" state="ryry"/>
    </tlLogic>
</additional>
```

Cada programa está compuesto de varias fases de cierta duración. En cada una, el atributo *state* define con una cadena de caracteres los colores de todos los semáforos en esa fase. El significado de los caracteres principales se puede ver en la siguiente tabla: | Caracter | Color | Descripción | |———-|———-|——————————————————————————————| | r | rojo | Luz roja: los vehículos deben detenerse. | | y | amarillo | Luz amarilla:los vehículos desacelerarán si están lejos de la insersección, de lo contrario pasarán. | | G | verde | Luz verde de prioridad: los vehículos pasarán.

La posiciónde cada caracter en la cadena se corresponde con las conexiones de la insercción controlada empezando desde arriba en el orden de las manecillas del reloj. Por ejemplo, la siguiente imagen se corresponde con la cadena `state =`



`"GrGr"`.

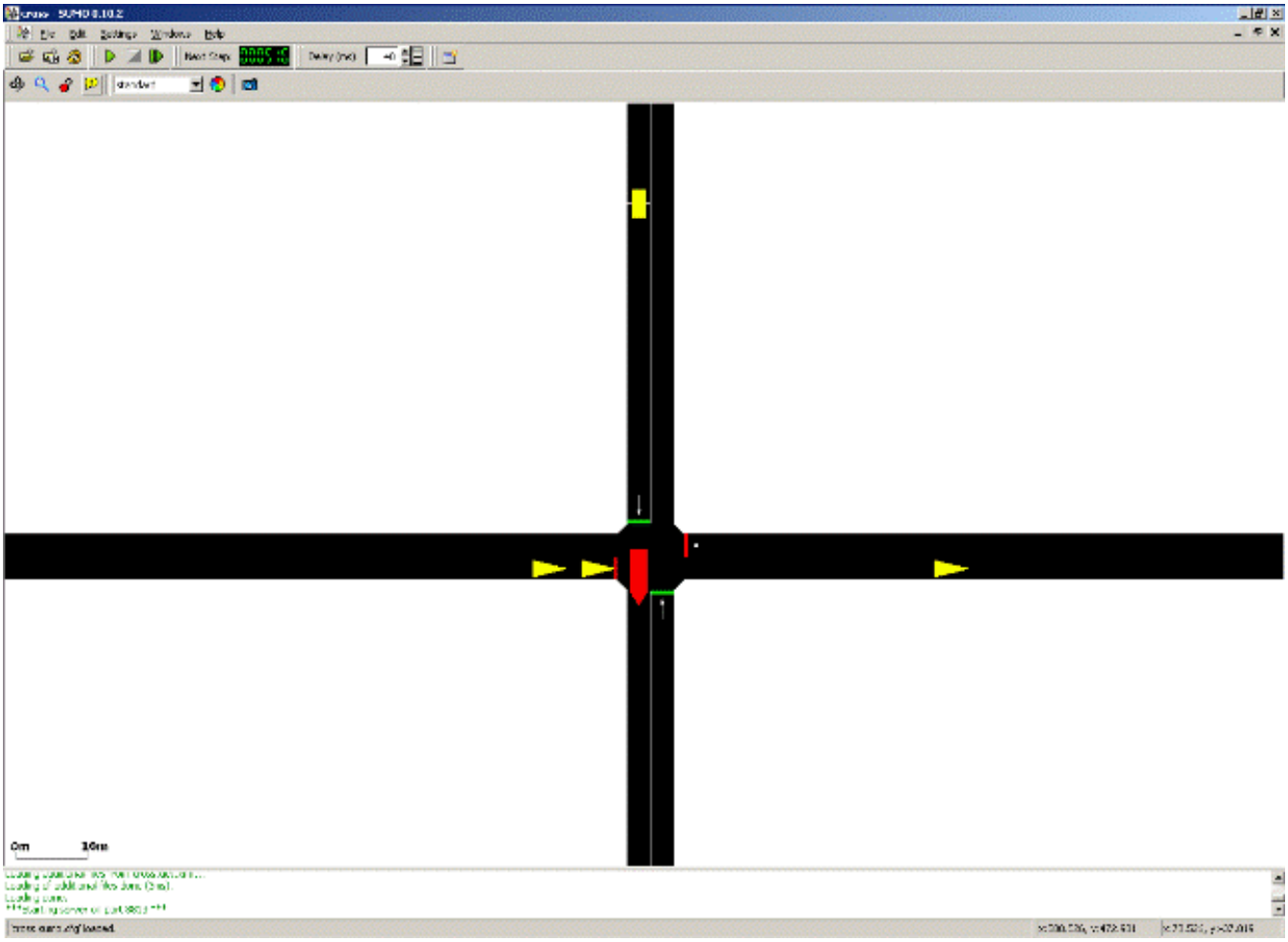
Control de los semáforos con Python

Más importante aún, también es posible controlar los estados de los semáforos programáticamente a través de una interfaz incluida en el paquete de instalación llamada TraCI.

TraCI es la abreviatura de " **Tra**fic **C**ontrol **I**nterface". Al dar acceso a una simulación de tráfico en ejecución, 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 servidor y el código Python con TraCI como cliente.

A continuación, se muestra el código de ejemplo que controla la lógica del semáforo para que se controle de la siguiente manera:



El ejemplo muestra una simple intersección de cuatro vías. Hay tráfico normal en el eje horizontal y vehículos importantes (ambulancias, patrullas, camiones de bomberos, etc.) en el eje vertical de norte a sur. En la vía que viene desde el norte hay un detector para reconocer a los vehículos que van entrando. Mientras que ningún vehículo ingresa desde el norte, siempre damos color verde en el eje horizontal, pero cuando un vehículo ingresa al circuito del detector, cambiamos la señal de inmediato para que el vehículo pueda cruzar la intersección sin detenerse.

```
step = 0
traci.trafficlight.setPhase("0", 2)
while traci.simulation.getMinExpectedNumber() > 0:
    traci.simulationStep()
    if traci.trafficlight.getPhase("0") == 2:
        if traci.inductionloop.getLastStepVehicleNumber("0") > 0:
            traci.trafficlight.setPhase("0", 3)
        else:
            traci.trafficlight.setPhase("0", 2)
    step += 1
traci.close()
```

Creación de simulación

Red de tráfico

Para probar y desarrollar a detalle la arquitectura se usará el simulador de tráfico urbano SUMO, que incluye prácticamente todas las herramientas necesarias.

Se creó la red de tráfico usando un script llamado osmWebWizard, que permite seleccionar un área geográfica real

desde OpenStreetMaps y convertirla en el tipo de archivo que utiliza el simulador.

Para motivos de prueba, se usó una intersección de Mérida conocida, para poder simular flujos a partir de lugares familiares, y en dado caso de ser necesario recolectar datos reales. La intersección en cuestión está ubicada en el sur de la ciudad, donde Circuito Colonias se cruza con la calle 50.

Para corroborar que los datos generados por el script sean certeros, se comparó la generación de la intersección con imágenes reales de Google Maps, y hubo que ajustar el ancho del carril que corresponde a la calle 50 de uno a dos carriles (realmente tiene como 4, pero siempre hay coches estacionados a ambos costados, al final se pueden aprovechar solo 2). Desconozco si las calles adyacentes tienen el ancho correcto, pero considero que solo ésta afectan al objetivo de la simulación.

Los datos recuperados por osmWebWizard, a excepción al numero de carriles de la calle 50, fueron bastante certeros, y la intersección de interés tiene correctamente los derechos de paso del semáforo.

Una vez montada la simulación, fue necesario prepararla para manipularse programáticamente, por lo que el semáforo a controlar se renombró a *semaforo_circuito_colonias* y se modificó el comportamiento de los semáforos para que sean estáticos y solo cambien cuando se les indique manualmente usando código.

Modelado de la arquitectura

Con la red de tráfico lista, el objetivo es programar la arquitectura previamente propuesta en Python, y para ello se utilizará una interfaz para manipular la simulación en tiempo real llamada TraCI. Dicha interfaz ya viene incluído en la instalación por defecto de SUMO.

Los primeros módulos de la arquitectura a programar son el *Observador de eventos* y el *Registrador de eventos*, y para ello se modelaron las propiedades de una intersección en clases que se relacionan entre si.

La más básica es Edge, que es lo equivalente a una calle y contiene sus propiedades asociadas:

- conection_uses_from: relaciona el Edge con una Conection.
- conection_uses_to: relaciona el Edge con una Conection.
- name: nombre o apodo para la calle.
- num_lanes: numero de carriles.
- is_traffic_input: indica si el trafico entra por esta calle.
- associated_detector_name: nombre del detector de trafico asociado a esta calle.
- street_name: nombre real de la calle.
- aprox_length: largo aproximado de la calle.
- aprox_total_width: ancho aproximado de la calle completa que incluye a todos los carriles.

Las calles están conectadas entre si, y esta relación se representa a través de la clase Conection, que indica una conexión simple entre Edges (calles):

- intersection: relaciona que una Intersection puede tener varias Conections.
- from_edge: desde que calle viene el trafico.
- to_edge: hacia que calle viene el trafico.

Las calles se agrupan en intersecciones que tienen conexiones entre ellas y posiblemente un semáforo. Esto se representa en la clase Intersection:

- name: nombre o apodo para la intersección.
- associated_traffic_light_name: el nombre del semáforo asociado a la intersección.
- conections: lista de todas las conexiones entre calles.

Siguiendo la arquitectura, los datos de cada intersección y del estado de la simulación deben almacenarse en algún lugar para posteriormente realizar un reporte que se guardará para su análisis. Se escogió medio de almacenamiento temporal una base de datos en sqlite, y se utilizó la librería Pony ORM para convertir el modelado en clases a tablas de una base de datos.

Actualmente se está trabajando en la implementación del framework Flow, que permite manipular conectar fácilmente SUMO con librerías de Reinforcement Learning, así como incluye métodos para generar tráfico de manera más realista.

Referencias

Antonio, J., & Sánchez, T. (2006). *Control de tráfico urbano basado en sistemas multiagentes Control de tráfico urbano basado en sistemas multiagentes.*

Patriksson, M. (1994). *The Traffic Assignment Problem: Models and Methods.*