Ciencias de la Computación
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS UNIVERSIDAD DE CHILE

Informe de Práctica

CC5901 – Práctica Profesional II

Empresa:	Departamento de Ingeniería Eléctrica Universidad de Chile
Supervisor:	Sandra Céspedes, Ph.D.
Alumno:	Manuel Olguín
Carrera:	Ingeniería Civil en Computación
RUT:	18.274.982 – 6
E-Mail:	molguin@dcc.uchile.cl
Tel:	+56 9 7463 6997

5 de abril de 2017
Santiago, Chile.

1. Certificado de la Empresa

2. Observaciones

Índice

Índice de figuras

3. Resumen

El presente documento detalla el trabajo realizado entre Agosto y Octubre del 2016 por el alumno Manuel Olguín en el Grupo de Investigación en Wireless Networking del Departamento de Ingeniería Eléctrica de la Universidad de Chile. Dicho trabajo se realizó bajo la supervisión de la profesora Sandra Céspedes, y consistió en la modelación de un sistema de alarma temprana para ciclistas en el contexto de un Sistema de Transporte Inteligente.

4. Introducción

4.1. Lugar de Trabajo

El Grupo de Investigación en Wireless Networking (en adelante, *el Grupo*) es un conjunto de estudiantes y profesionales del área de Tecnologías de la Información y Comunicaciones (en adelante, *TICs* [1]) liderado por la profesora Sandra Céspedes [2], docente del Departamento de Ingeniería Eléctrica de la Universidad de Chile (en adelante, *DIE* [3]). El Grupo desarrolla sus labores de investigación y desarrollo en el Laboratorio de Comunicaciones Avanzadas, ubicado en el 5º piso del departamento del DIE ubicado en Av. Tupper 2007, Santiago.

4.2. Equipo de Trabajo

Los miembros del grupo son estudiantes de pre- y posgrado, además de profesionales ligados a las TICs, de diversos orígenes, especialidades e incluso nacionalidades. En específico, el trabajo realizado por el practicante fue desarrollado bajo la supervisión de la doctora Sandra Céspedes y en colaboración con un estudiante de pregrado del DIE.

4.3. Software y Conceptos Importantes

El trabajo se realizó principalmente en C++ y Python, utilizando software especializado para la modelación y estudio de redes inalámbricas, redes de transporte y la integración de ambas en Sistemas de Transporte Inteligente.

4.3.1. Sistemas de Transporte Inteligente

Los sistemas de transporte conforman la columna vertebral de nuestras ciudades, contribuyendo directamente al desarrollo de la sociedad urbana. Un sistema de transporte bien diseñado y eficiente permite el desplazamiento rápido y cómodo de personas y bienes; en cambio, uno ineficiente genera grandes problemas, alargando los tiempos de viaje y aumentando la contaminación atmosférica.

Los Sistemas de Transporte Inteligente (en adelante *ITS*, por sus siglas en inglés – *Intelligent Transportation Systems*) surgen como una respuesta a la necesidad de optimización y modernización de los sistemas de transporte existentes. La Unión Europea define a los ITS como aplicaciones avanzadas que, sin incorporar inteligencia como tal, pretenden proveer servicios innovadores relacionados con distintos modos de transporte y de administración de tráfico, que además otorgan información a los usuarios, permitiéndoles utilizar el sistema de transporte de manera más segura, coordinada e inteligente [4]. De acuerdo al Departamento de Transportes de los EEUU, estos sistemas se pueden dividir en dos grandes categorías [5]:

Sistemas de Infraestructura Inteligente Tienen como enfoque el manejo de los sistemas de transporte a niveles macro, y la transmisión de información oportuna a los usuarios. Esta categoría incluye, entre otros, sistemas de advertencia y señalización dinámica en ruta (ya sea a través de pantallas o sistemas de comunicación inalámbrica), sistemas de pago electrónico y de coordinación del flujo de tráfico.

Sistemas de Vehículos Inteligentes Engloba todo aquello relacionado con la automatización y optimización de la operación de un vehículo. Dentro de esta categoría se incluyen sistemas de advertencia y prevención de colisiones, de asistencia al conductor — por ejemplo, sistemas de navegación — y control autónomo de vehículos.

El factor común entre ambas categorías es la necesidad de extraer información en tiempo real desde el entorno, la cual debe procesarse y en muchos casos generar una respuesta a transmitir al usuario. Para este fin, actualmente se utilizan tecnologías de comunicación inalámbricas, tanto de área local (los estándares incluídos en la familia WLAN, IEEE 802.11), como de área personal (WPAN, IEEE 802.15) [6, 7, 8].

4.3.2. OMNeT++

Objective Modular Network Testbed in C++, mejor conocido como OMNeT++ [9], es un *framework* y librería para la simulación de sistemas de eventos discretos, principalmente utilizado para la simulación y análisis de redes de comunicaciones. Consiste en un conjunto de módulos y librerías en C++, además de un entorno de desarrollo integrado (en adelante, *IDE* por sus siglas en inglés – *Integrated Development Environment*) y una interfaz gráfica, los cuales posibilitan la construcción de simulaciones dinámicas y extensibles tanto de sistemas cableados como inalámbricos.

OMNeT++ es software libre, de fuente abierta, y altamente popular en la comunidad académica.

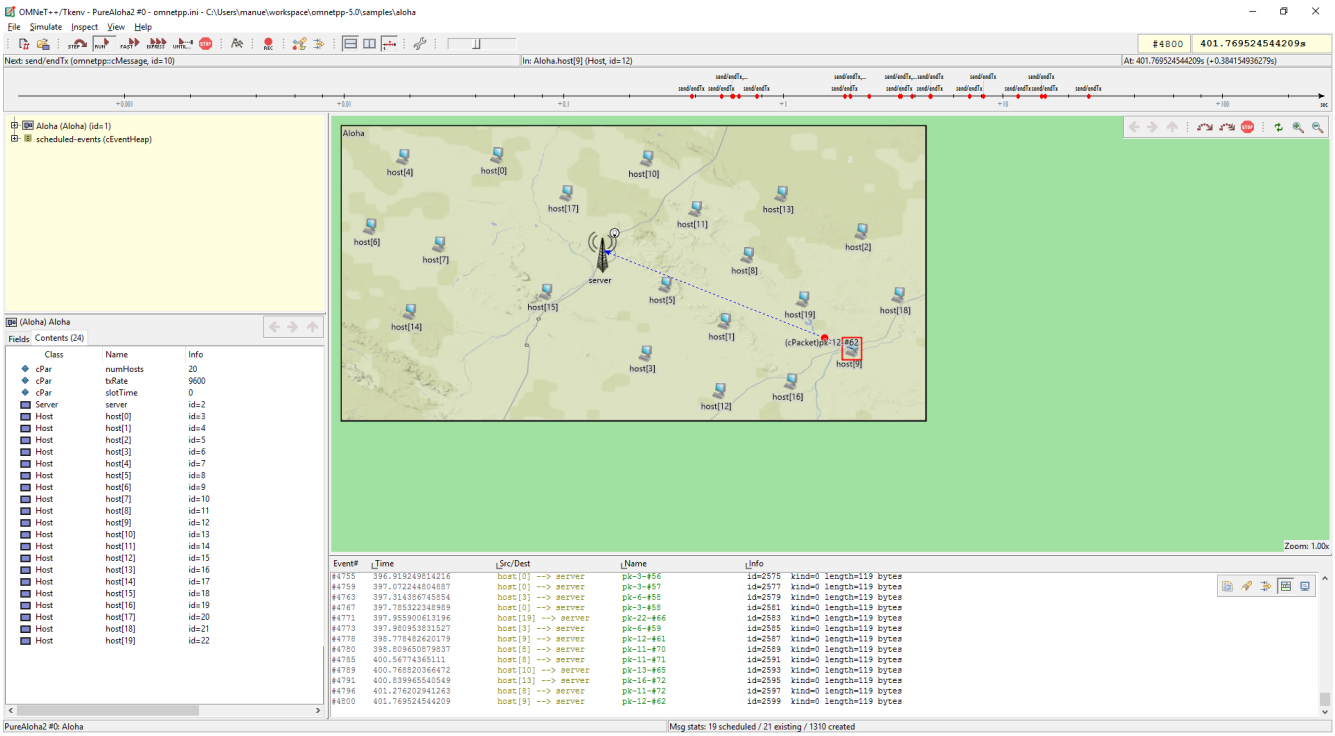

Figura 1: Entorno gráfico de simulación de OMNeT++

4.3.3. SUMO

SUMO (*Simulation of Urban MObility*) es un simulador de tráfico vehicular de fuente abierta desarrollado originalmente por el Instituto de Sistemas de Transporte del Centro Aeroespacial Alemán (DLR, por sus siglas en alemán – *Deutsches Zentrum für Luft- und Raumfahrt e.V.*) [10]. Es un simulador microscópico, en el cual cada vehículo es simulado de manera particular, moviéndose individualmente por la red.

4.3.4. VEINS

VEhicles In Network Simulation, *VEINS*, es un framework de fuente abierta para la simulación, tanto en su aspecto de comunicaciones como de transporte, de Sistemas de Transporte Inteligente.

4.4. Situación Previa

* hablar un poco sobre Javiera *

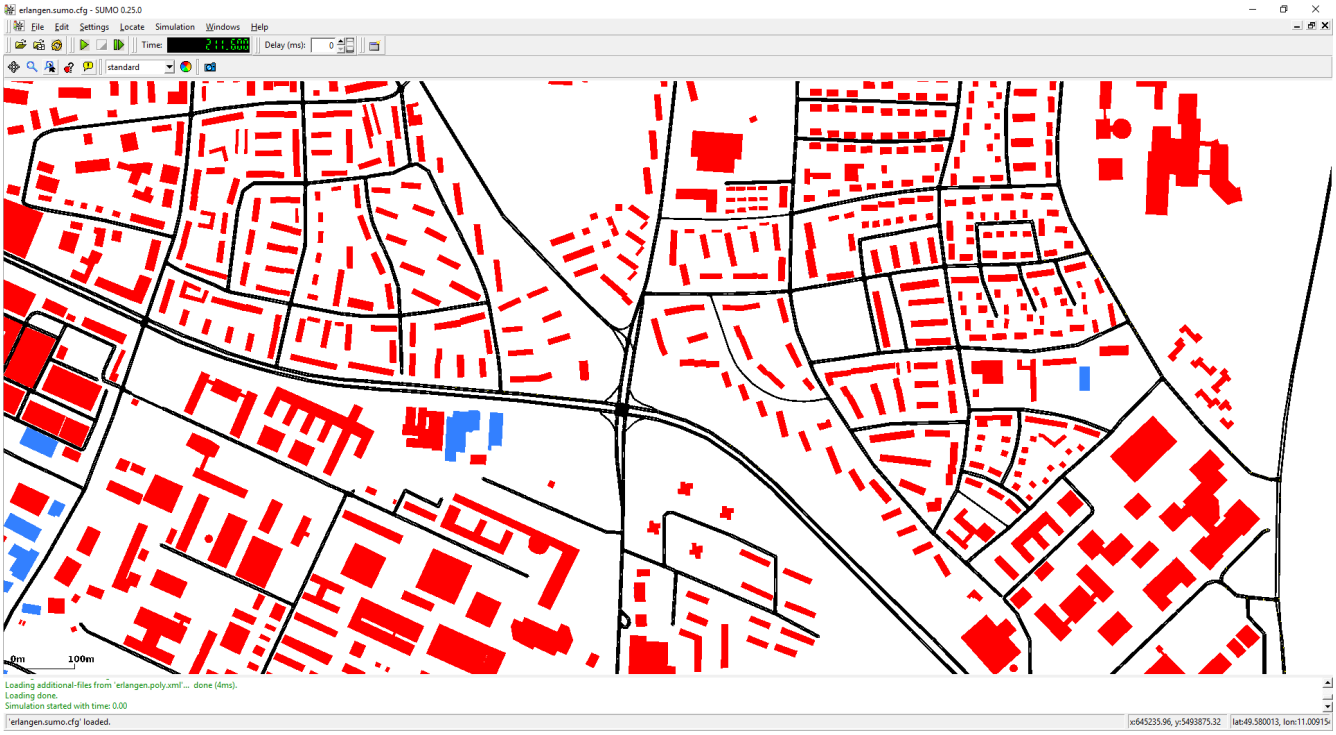

Figura 2: Simulación de una red en SUMO

5. Trabajo Realizado

El trabajo realizado tenía como fin último la elaboración de un prototipo de un sistema de alerta temprana para ciclistas en un ITS. El fin de este sistema era el de predecir, mediante un modelo de probabilidades bayesianas descrito por Liebner *et al.* en [11], la probabilidad de un giro sorpresivo de un vehículo en un cruce, y notificar esto a ciclistas cercanos a dicho cruce (se asumieron ciclistas y vehículos en el contexto de un ITS, dotados de capacidades de comunicación inalámbrica).

Este trabajo se dividió entonces en dos partes:

1. Estudio y comprensión del software especializado OMNeT++, SUMO y VEINS, y la posterior elaboración de un prototipo simple para ilustrar la factibilidad del proyecto.
2. Implementación del sistema, utilizando como base el prototipo anterior y utilizando registros de tráfico real.

Cabe destacar que las labores del practicante abarcaron la implementación del modelo vehicular y la integración del modelo predictivo a este. El estudio y la implementación del modelo descrito en [11] quedó a cargo de Sebastián Piña M., alumno del DIE y compañero de equipo del practicante.

5.1. Estudio del software y elaboración de un prototipo inicial simple

El primer mes de la práctica se enfocó totalmente en el estudio del software especializado, concentrado principalmente en VEINS y la elaboración de modelos de sistemas de transporte inteligentes utilizando el framework. Como un *bonus* adicional, esto significó el aprendizaje de C++ por parte del practicante. Con el fin de realizar un aprendizaje didáctico y aplicado, se implementó además un prototipo simple de un cruce vehicular con carriles para ciclistas, el cual se describe a continuación.

5.1.1. Prototipo inicial

El prototipo inicial se construyó en base al trabajo previamente realizado por Javiera Born (descrito en la sección 4.4). Este consistió en una simulación muy simple, compuesta por un vehículo motorizado y una bicicleta, los cuales interactúan en un cruce. Si bien esta simulación presenta un escaso valor práctico y académico, fue de gran ayuda para que el practicante adquiriera las habilidades necesarias para el manejo del software especializado.

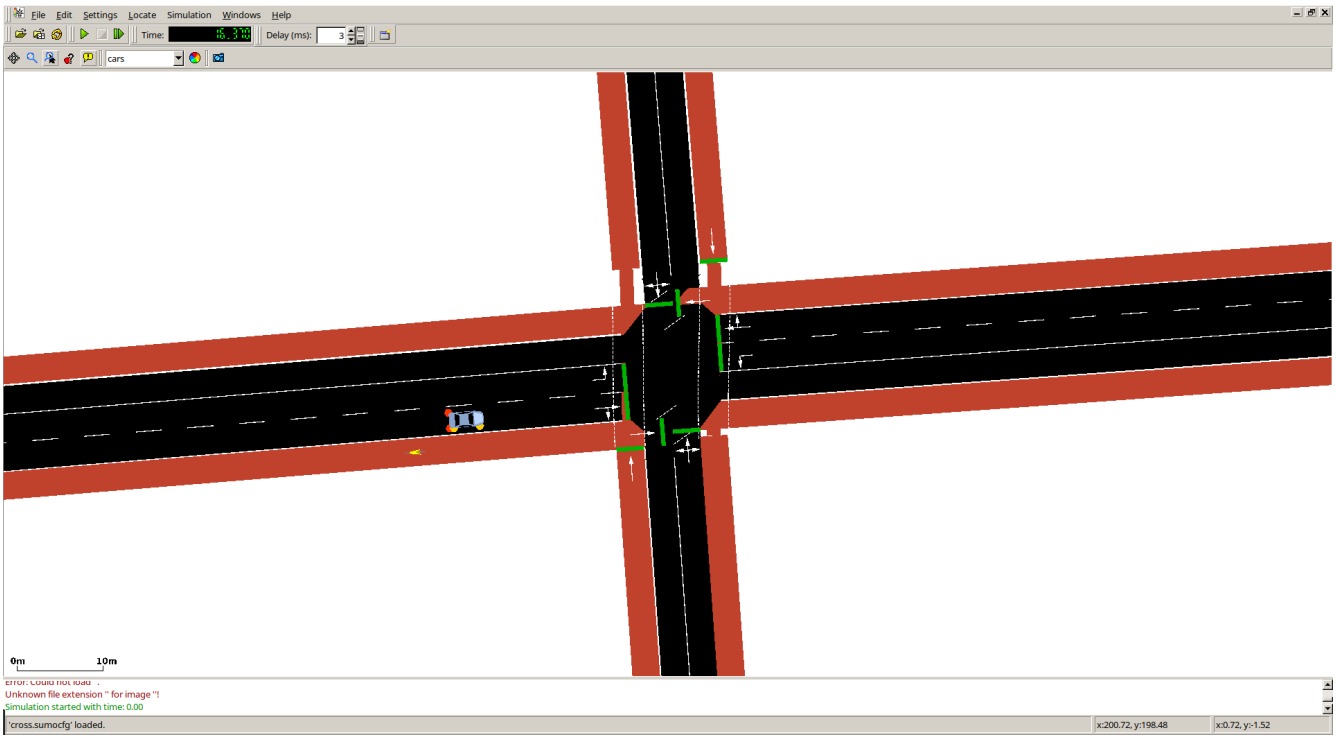

Figura 3: Prototipo inicial en ejecución

5.1.2. Parámetros físicos de la simulación

El aspecto de transporte de la simulación no presenta grandes cambios respecto a lo desarrollado anteriormente por J. Born – se reutilzó el cruce vehicular simple compuesto por una vía de tres pistas y otra de dos, ambas con ciclovías a los costados. Se modificaron los flujos vehiculares para que consistieran únicamente en

un automóvil y una bicicleta, y para sincronizar su llegada al cruce y poder simular la detención de la bici en reacción a un mensaje de viraje emitido por el automóvil (detalles de la implementación de esto en la sección 5.1.3).

5.1.3. Simulación de comunicación inalámbrica y lógica?

La lógica de la simulación se implementó en un módulo de OMNeT++ llamado *BikeManeuver*. El comportamiento de ambos vehículos fue implementado en este único módulo; al iniciar la simulación, se verifica (entre otros parámetros) el identificador del vehículo, en base a lo cual se asigna un valor a un *flag* que determina el comportamiento del módulo en el resto de la simulación.

Cada vehículo fue modelado como una máquina de estados finita, los cuales cambian de estado en respuesta a estímulos exteriores y/o interiores (la figura 4 ilustra estos diagramas de estado). A continuación se detalla el comportamiento de ambos, y el código 4 expone un extracto del código del módulo.

Bicicleta El comportamiento de esta es extremadamente simple. El módulo espera hasta recibir un mensaje de advertencia desde el vehículo; al recibirlo, verifica la distancia actual de la bicicleta al cruce. De ser esta menor a una distancia D establecida, la bicicleta se detiene hasta recibir un segundo mensaje del automóvil, el cual confirma que este último ha finalizado su giro. Finalmente, la bicicleta resume su trayectoria hasta terminar la simulación.

Automóvil El automóvil tiene un comportamiente ligeramente más complicado, ya que debe constantmente estar monitoreando su distancia al cruce para determinar cuando emitir los mensajes que espera la bicicleta. Para esto, al principio de la simulación se programa un "automensaje" (*selfbeacon* en inglés) a ser enviado por el módulo a si mismo en el siguiente instante de simulación ($0{,}1$s más tarde). Al recibir dicho mensaje, se programa un nuevo *selfbeacon* y se verifica si la distancia del vehículo al cruce es menor o igual a la distancia D previamente mencionada. De ser así, se emite el primer mensaje de advertencia, y se cambia el estado a "girando", hasta detectar (mediante la misma técnica anterior de mediciones periódicas) que se encuentra nuevamente fuera del cruce (distancia $\geq D$), momento que se emite el segundo y último mensaje y se vuelve a la operación normal.

Cabe notar que la técnica de uso de *selfbeacons* para realizar operaciones periódicas en OMNeT++ es altamente común, y fue uno de los principales aprendizajes de esta estapa del trabajo de práctica [12].

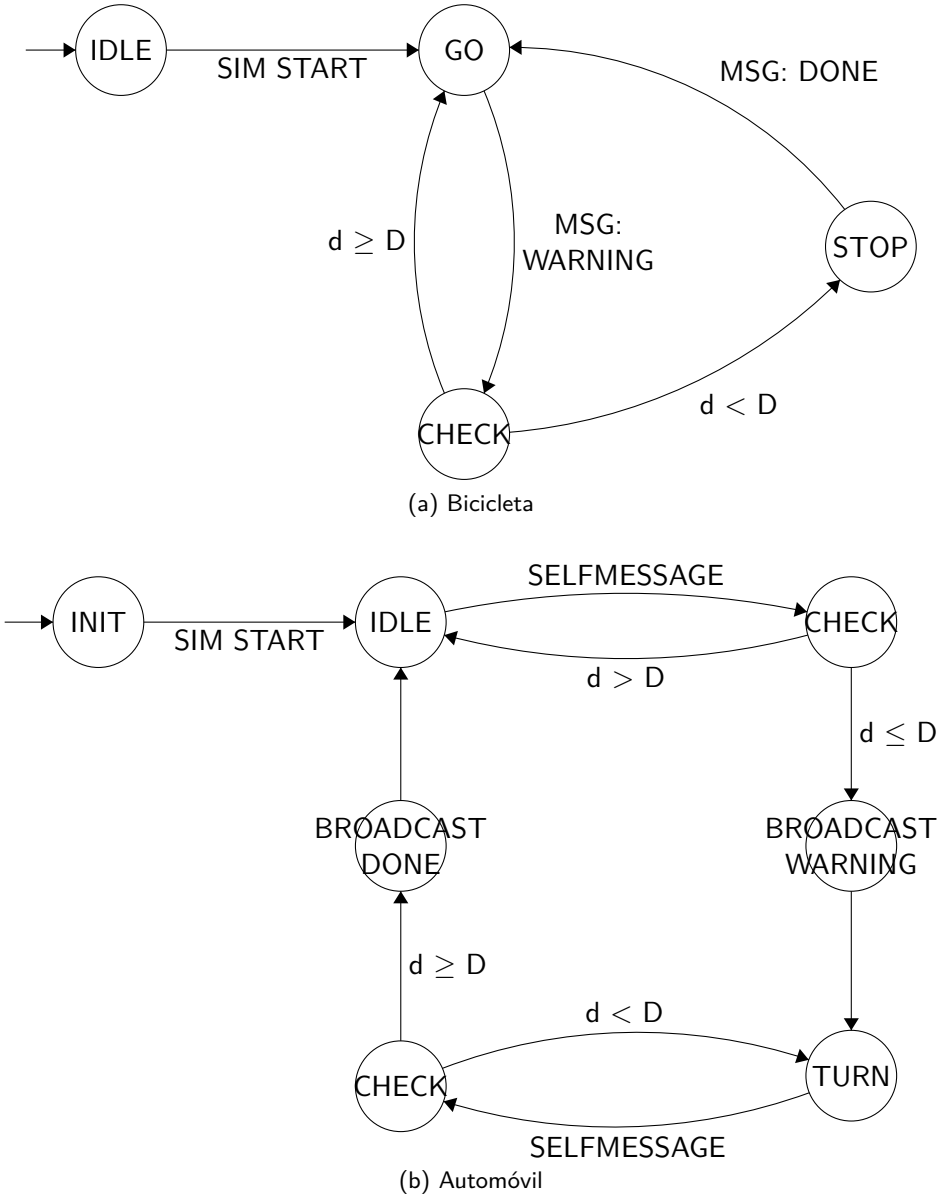

Figura 4: Diagramas de estado de bicicleta y automóvil. Notar que se representan también estados transitorios, para los cuales no se estimó necesario crear un estado "real" en la implementación.

5.2. Modelación del sistema de alarma temprana

Como se mencionó anteriormente, la segunda etapa del trabajo contemplaba la implementación de un modelo basado en el trabajo de Liebner *et al.*, descrito en [11]. En esta publicación, los autores detallan un modelo predictivo del comportamiento de vehículos en intersecciones urbanas, basado en la teoría de probabilidades bayesianas *explicar algoritmo*.

5.2.1. Modelo físico de la simulación

Se optó por utilizar un escenario de SUMO con trazas vehiculares reales; es decir, un registro real del tráfico en una ciudad. De esta manera se asegura un alto grado de realismo del comportamiento de los vehículos en la simulación, lo cual es un requisito fundamental para la validación de los resultados obtenidos.

El escenario *TAPAS Cologne* [13] fue el escogido para este fin, dada la calidad y la cantidad de datos que incluye. Este escenario describe el flujo vehicular en la ciudad de Cologne, en Alemania, durante un día completo, tanto para vehículos particulares como para el transporte público, en un mapa que abarca la ciudad en su totalidad. Por conveniencia, incluye dos sub-escenarios distintos, uno que describe completamente 24 horas de tráfico vehicular, y otro que contiene información de sólo dos horas (entre 6:00 y 8:00 a.m.). Fue este último el utilizado para el desarrollo del sistema de alarma temprana, el cual además se adaptó a la naturaleza del sistema a modelar.

Dado que el sistema contempla el monitoreo de una intersección, se decidió escoger un cruce apropiado y sólo incluir las trazas de movimiento vehicular que lo incluyeran en su trayectoria. La elección del cruce se hizo de manera manual, ya que no requiere condiciones estrictas, siguiendo los siguientes pasos:

1. Se seleccionó primero un grupo semi-arbritrario de intersecciones a considerar, sin semáforos y de cantidad de pistas reducidas. La primera condición es de carácter funcional – la existencia de un semáforo se asume que elimina la necesidad de un sistema predictivo de giros sorpresivos. La segunda es de carácter más práctico; una intersección con muchas pistas agrega complejidad innecesaria a un modela en tan temprana etapa de desarrollo.
2. En segundo lugar, se analizaron los flujos vehiculares de cada uno de los cruces seleccionados, descartando aquellos que presentaran nula o poca presencia vehicular a lo largo de la simulación.
3. Finalmente se hizo la elección final, en base a la simplicidad total del cruce.

La intersección finalmente seleccionada para la simulación fue el cruce 55063174, compuesto por la intersección de los *arcos* (calles) 132530717, 132530803, 7651840 y 132530803, caracterizada por ser un cruce entre dos calles de una pista cada una, lo cual tiene la ventaja de que limita los virajes a considerar a únicamente dos direcciones distintas. Este cruce fue modificado levemente además para incluir ciclovías al costado de ambas calles, ya que el escenario *TAPAS Cologne* no incluye registros de este medio de transporte.

Por otro lado, las trazas vehiculares incluídas en el escenario también debieron ser alteradas y filtradas. Dichas trazas están descritas en archivos XML (ver código 1), compuestos por dos tipos esenciales de elementos; vehículos y rutas. Una ruta consiste en una serie de identificadores de *arcos* que forman un recorrido desde una posición inicial A hasta un destino B. Estas se asocian luego a vehículos, describiendo de esta manera el movimiento de éstos últimos por la red vehicular. Puesto que, para los fines de la simulación en cuestión, sólo interesaban aquellos vehículos que se desplazaran por la intersección seleccionada anteriormente, se optó

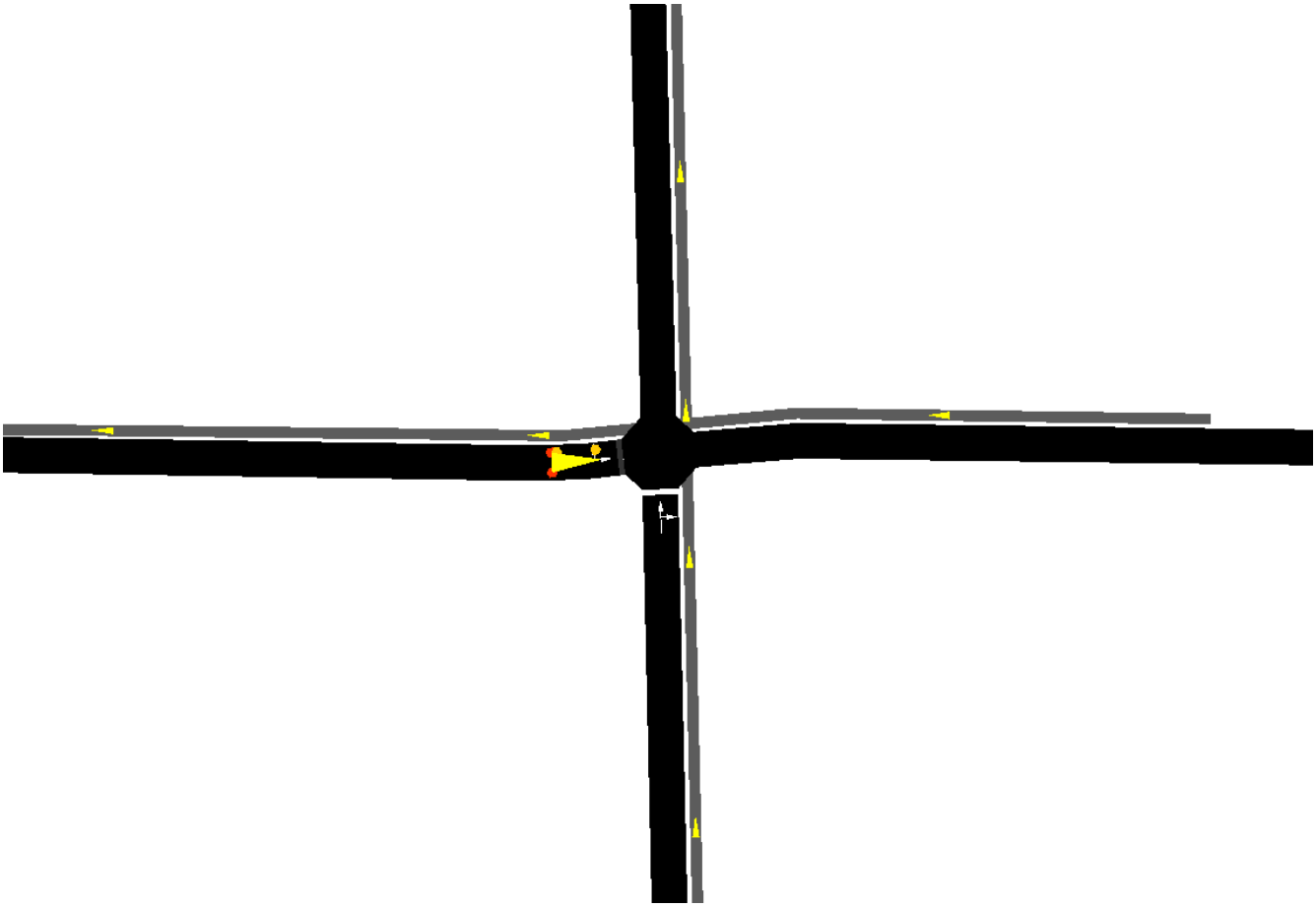

Figura 5: *Zoom* del cruce 55063174, escogido para la implementación del sistema de alarma temprana.

por filtrar el archivo de rutas, eliminando todos aquellos vehículos no relevantes para el trabajo. Esto se efectuó utilizando un script escrito en Python (ver código 2), y redujo el tamaño del archivo de rutas de aproximadamente 200 MB a sólo 1.5 MB, lo cual aceleró de manera bastante el tiempo de carga y de inicio del escenario en SUMO.

```
<vehicle id="37937_37937_358_0" type="pkw" depart="21600.00">
    <route edges="328974180 133852448-AddedOnRampEdge 133852448 7854060 ..."/>
</vehicle>
<vehicle id="38648_38648_359_0" type="pkw" depart="21600.00">
    <route edges="51793889 -155592914#2 -155592914#1 155592907#2 155592972 ..."/>
</vehicle>
<vehicle id="38750_38750_359_0" type="pkw" depart="21600.00">
    <route edges="8437608#3 -8437608#6 -8437608#2 -189913664#3 -209682829#1 ..."/>
</vehicle>
<vehicle id="41314_41314_361_0" type="pkw" depart="21600.00">
    <route edges="25326879#0 24609080#3 118427328#0 118427328#2 118427328#3 ..."/>
</vehicle>
<vehicle id="41690_41690_361_0" type="pkw" depart="21600.00">
    <route edges="29337995#1 29337995#3 29337995#5 29337995#7 9250113#2 ..."/>
</vehicle>
<vehicle id="43455_43455_362_0" type="pkw" depart="21600.00">
    <route edges="340575954#3 99714977#3 99714977#4 196122462#0 27892605#0 ..."/>
</vehicle>
```

Código 1: Extracto de trazas vehiculares del escenario TAPAS Cologne. Cada vehículo tiene asociada una ruta, la cual describe el camino que recorre este desde su origen hasta su destino.

```
import xmltodict

EDGES = ['132530717#0', '132530803#1', '7651840#0', '132530803#0']

with open('cologne6to8.rou.xml', 'r') as infile, \
        open('cologne6to8.filtered.rou.xml', 'w') as outfile:
    orig_xml = xmltodict.parse(infile.read())
    vehicles = orig_xml['routes']['vehicle']
    relevant_vehicles = []

    for v in vehicles:
        edges = v['route']['@edges'].split(' ')
        for e in EDGES:
            if e in edges and v not in relevant_vehicles:
                relevant_vehicles.append(v)

    orig_xml['routes']['vehicle'] = relevant_vehicles
    outfile.write(xmltodict.unparse(orig_xml, pretty=True))

    print(
        '''
Done. Filtered on edges: {e}.
Final results: {i} cars match the desired edges.
        '''.format(e=EDGES, i=len(relevant_vehicles))
    )
```

Código 2: Código de extracción de vehículos relevantes.

5.2.2. Lógica de la simulación

El escenario contempla tres tipos de entidades diferentes que interactúan en el cruce:

RSU La unidad de infraestructura inteligente, *RSU* (por sus siglas en inglés, *RoadSide Unit*), no "existe" en el escenario físico, sino que es creada y trasladada al centro de la intersección por OMNeT++ al iniciarse la simulación. La RSU tiene como labor la interpretación del estado del sistema en cada instante de tiempo; recibe datos desde los vehículos presentes y es la entidad que detecta un viraje sorpresivo y emite la advertencia correspondiente.

La funcionalidad de esta fue separada en múltiples archivos. En primer lugar, la clase *BikeWarningRSU* extiende un módulo básico de OMNeT++ equipado con interfaces inalámbricas. En este se implementó la comunicación del módulo con el resto de las entidades de la simulación; en específico, la RSU recibe *beacons* periódicos desde los automóviles, los cuales contienen información posicional de los vehículos, como su velocidad y posición. Esta información luego es proporcionada como parámetros al algoritmo de decisión – si la determinación de éste último es positiva, la RSU emite una advertencia en modo *broadcast* a todos los vehículos presentes en la simulación. En caso contrario se descarta el resultado.

El algoritmo de decisión en sí fue implementado en Python por S. Piña, y fue integrado a la simulación mediante la clase de C++ *DecisionAlgorithm*, utilizando código Python embebido (ver código 5). Esta clase inicializa la interfaz con Python, y presenta funciones de conveniencia para interactuar directamente con la implementación del algoritmo desde C++.

Automóviles Los automóviles en esta simulación se consideran "tontos", en el sentido que no cuentan con inteligencia propia más allá de la lógica de recolección de información que luego es enviada a la RSU. Su funcionalidad se implementó en el módulo de OMNeT++ *BikeWarningApplication*.

Esta clase utiliza el mismo método de *selfbeacons* detallado en la etapa anterior de la práctica para periódicamente medir los parámetros físicos del vehículo (posición, velocidad y calle actual) y enviarlos a la RSU. Para este fin se utilizó el formato de mensaje por defecto del protocolo *WAVE* (*Wireless Access for Vehicular Environments*, una variante del estándar IEEE802.11 para comunicaciones en contextos de sistemas de transporte inteligentes [8]). La implementación de este mensaje en OMNeT++ sólo cuenta con un campo de texto para el *payload*, por lo cual la información requerida debió codificarse en *JSON* antes de ser enviada. El código 3 ilustra el método encargado de la recolección y envío de datos en el módulo.

Bicicletas Finalmente, las bicicletas también se implementaron en *BikeManeuver*, sin embargo su comportamiento es aún más simple que el de los automóviles. Simplemente monitorean de manera pasiva las emisiones del RSU, y al recibir una advertencia se detienen por un periodo de tiempo predeterminado antes de seguir su trayectoria normal. Se decidió optar por esta simplicidad para priorizar primero la validación del funcionamiento fundamental de la teoría antes de implementar un modelo más complejo.

```
void BikeWarningApplication::carHandleSelfMsg(cMessage *msg)
{
    switch (msg->getKind()) {
            case SEND_BEACON_EVT: {
                //std::cout << "Sending beacon: ";
                WaveShortMessage* wsm = prepareWSM("beacon", beaconLengthBits,
                                                   type_CCH, beaconPriority,
                                                   -1, -1);

                // get current position, speed, lane and id
                Coord pos = mobility->getCurrentPosition();
                double speed = mobility->getSpeed();
                std::string id = mobility->getExternalId();
                std::string lane_id = traciVehicle->getLaneId();

                // encode into JSON for easy packing into WAVE short message.
                json data_j = {
                        {"id", id},
                        {"vel", speed},
                        {"pos_x", pos.x},
                        {"pos_y", pos.y},
                        {"lane_id", lane_id}
                };

                std::string jdata = data_j.dump();
                //std::cout << jdata << std::endl;

                // load payload into message and send it
                wsm->setWsmData(jdata.c_str());
                sendWSM(wsm);
                scheduleAt(simTime() + par("beaconInterval").doubleValue(), sendBeaconEvt);
                break;
            }
            default: {
                if (msg)
                    DBG << "APP: Error: Got Self Message of unknown kind! Name: " <<
    msg->getName() << endl;
                break;
            }
        }
}
```

Código 3: Código encargado de la medición periódica y envío de datos a la RSU en *BikeWarningApplication*.

6. Conclusiones

Desafortunadamente, a pesar del trabajo realizado por el practicante y su colega, la simulación no fue del todo fructuosa y hasta el día de hoy se encuentra congelado su desarrollo. El principal problema surge de que la teoría predictiva está basada en el estudio del comportamiento de los vehículos dada la existencia de otro vehículo en la pista, frente al primero, y la adaptación de este modelo a uno basado únicamente en los datos individuales de cada vehículo presentó problemas inesperados.

Sin embargo, de todos modos se lograron resultados parciales, los cuales fueron presentados en el *Taller de Inteligencia Urbana* en Santa Cruz, Región de O'Higgins, en Noviembre del 2016 por S. Piña. Además, se presentó un borrador inicial del trabajo al profesor Falko Dressler (uno de los creadores de VEINS, y un reconocido investigador en el área de Sistemas de Transporte Inteligente) en su visita a Chile a principios de Agosto.

Finalmente, el trabajo generó un interés profundo en el practicante, específicamente en el tema de simuladores para ITS. Actualmente, este interés se ha visto concretado en un trabajo de memoria a desarrollarse durante el presente semestre.

Referencias

[1] *Tecnologías de la Información y Comunicaciones* - https://es.wikipedia.org/wiki/Tecnolog%C3%ADas_de_la_informaci%C3%B3n_y_la_comunicaci%C3%B3n

[2] *Sandra Céspedes U.* - http://www.cec.uchile.cl/~scespedes/

[3] *Departamento de Ingeniería Eléctrica* - http://die.cl/

[4] Directive 2010/40/EU of the European Parliament and of the Council *on the framework for the deployment of Intelligent Transport Systems in the field of road transport and for interfaces with other modes of transport*, 2010 O.J. L 207/1

[5] U.S. Department of Transportation *Office of the Assistant Secretary for Research and Technology (OST-R)* http://www.itsoverview.its.dot.gov/ (*04/2017*)

[6] D. J. Dailey, K. McFarland and J. L. Garrison *Experimental study of 802.11 based networking for vehicular management and safety*, Intelligent Vehicles Symposium (IV), 2010 IEEE, San Diego, CA, 2010, pp. 1209-1213.
doi: 10.1109/IVS.2010.5547955

[7] W. Xiong; X. Hu; T. Jiang. *Measurement and Characterization of Link Quality for IEEE 802.15.4-compliant Wireless Sensor Networks in Vehicular Communications*, IEEE Transactions on Industrial Informatics , vol.PP, no.99, pp.1-1
doi: 10.1109/TII.2015.2499121

[8] D. Jiang and L. Delgrossi. *IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments*, Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE, Singapore, 2008, pp. 2036-2040.
doi: 10.1109/VETECS.2008.458

[9] OMNeT++ Discrete Event Simulator. *An extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators*.
https://OMNeTpp.org/ (*04/2017*)

[10] SUMO – Simulation of Urban MObility. *A free and open traffic simulation suite which is available since 2001.*
http://sumo.dlr.de (*4/2017*)

[11] M. Liebner, F. Klanner, M. Baumann, C. Ruhhammer and C. Stiller. *Velocity-Based Driver Intent Inference at Urban Intersections in the Presence of Preceding Vehicles*, IEEE Intelligent Transportation Systems Magazine, vol. 5, no. 2, pp. 10-21, Summer 2013.
doi: 10.1109/MITS.2013.2246291

[12] OMNeT++ Simulation Manual *4.7.1 Self-Messages*
https://omnetpp.org/doc/omnetpp/manual/#sec:simple-modules:self-messages (*4/2017*)

[13] TAPAS Cologne *Scenario describing the traffic in the city of Cologne, Germany, during a whole day.*
http://sumo.dlr.de/wiki/Data/Scenarios/TAPASCologne (04/2017)

7. Anexos

```
FSM_Switch(carFSM)
{
case FSM_Exit(CAR_INIT):
{
    // init simulation: go to car idle state and schedule selfbeacon for
    // periodic checks of distance to junction
    FSM_Goto(carFSM, CAR_IDLE);
    scheduleAt(simTime() + ping_interval, selfbeacon);
    break;
}
case FSM_Exit(CAR_IDLE):
{
    // got a selfbeacon?
    if (msg == selfbeacon)
    {
        // "transitory" state: check distance to crossing.
        // if we are too far away then do nothing apart from scheduling another
        // selfbeacon, otherwise send warning and change state
        if (traci->getDistance(
                traci->junction("cluster_0__0_0__2_0__4_0__6").getPosition(),
                mobility->getCurrentPosition(), false) > 30)
        {
            // too far away, schedule next check
            scheduleAt(simTime() + ping_interval, selfbeacon);
            break;
        }

        // close to the crossing, send warning and start turn
        t_channel channel = dataOnSch ? type_SCH : type_CCH;
        turnmsg = prepareWSM("data", dataLengthBits, channel, dataPriority, -1, 2);
        turnmsg->setWsmData(warning);
        sendWSM(turnmsg);
        scheduleAt(simTime() + ping_interval, selfbeacon);
        FSM_Goto(carFSM, CAR_TURNING);
    }
    break;
}
case FSM_Exit(CAR_TURNING):
{
    // got selfbeacon?
    if (msg == selfbeacon)
    {
        // "transitory" state: check distance to crossing.
        // are we still turning? then do nothing apart from scheduling selfbeacon
        // otherwise, broadcast and return to normal operation
        if (traci->getDistance(
                traci->junction("cluster_0__0_0__2_0__4_0__6").getPosition(),
                mobility->getCurrentPosition(), false) < 30)
        {
```

```
            // still turning, do nothing and schedule next check
            scheduleAt(simTime() + ping_interval, selfbeacon);
            break;
        }

        // donde turning, let the bike know!
        t_channel channel = dataOnSch ? type_SCH : type_CCH;
        turnmsg = prepareWSM("data", dataLengthBits, channel, dataPriority, -1, 2);
        turnmsg->setWsmData(warning);
        sendWSM(turnmsg);
        scheduleAt(simTime() + ping_interval, selfbeacon);
        FSM_Goto(carFSM, CAR_IDLE);
    }
    break;
}
}
```

Código 4: Extracto del código encargado del comportamiento del automóvil.

```
DecisionAlgorithm::DecisionAlgorithm() {
    Py_Initialize();
    PyRun_SimpleString( "import os, sys, random, traceback\n"
                        "sys.path.append(os.getcwd())\n");
    pName = PyUnicode_DecodeFSDefault(DECISION_ALGO_SCRIPT);
    pModule = PyImport_Import(pName);
    if(pModule == NULL)
    {
        std::cout << "Null pModule" << std::endl;
        throw -1;
    }
    pFunc = PyObject_GetAttrString(pModule, DECISION_ALGO_FUNC);
    pAddEntry = PyObject_GetAttrString(pModule, DECISION_ALGO_ADDENTRYLANE);
    pAddExit = PyObject_GetAttrString(pModule, DECISION_ALGO_ADDEXITLANE);

    if(!(pFunc && PyCallable_Check(pFunc)))
    {
        std::cout << "Null Decision Function" << std::endl;
        throw -1;
    }
    else if(!(pAddEntry && PyCallable_Check(pAddEntry)))
    {
        std::cout << "Null Entry Lane Function" << std::endl;
        throw -1;
    }
    else if(!(pAddExit && PyCallable_Check(pAddExit)))
    {
        std::cout << "Null Exit Lane Function" << std::endl;
        throw -1;
    }

}
```

```
33
34 double DecisionAlgorithm::checkTurn(double velocity, double pos_x, double pos_y, std::string
       car_id, std::string lane_id)
35 {
36
37     pArgs = PyTuple_New(5);
38     PyTuple_SetItem(pArgs, 0, PyFloat_FromDouble(velocity));
39     PyTuple_SetItem(pArgs, 1, PyFloat_FromDouble(pos_x));
40     PyTuple_SetItem(pArgs, 2, PyFloat_FromDouble(pos_y));
41     PyTuple_SetItem(pArgs, 3, PyByteArray_FromStringAndSize(car_id.c_str(), car_id.length()));
42     PyTuple_SetItem(pArgs, 4, PyByteArray_FromStringAndSize(lane_id.c_str(),
       lane_id.length()));
43
44
45     PyObject* result = PyObject_CallObject(pFunc, pArgs);
46     if(result == NULL)
47     {
48         std::cerr << "null return, something went wrong" << std::endl;
49         throw -1;
50     }
51
52     //long final = PyLong_AsLong(result);
53     double final = PyFloat_AsDouble(result);
54     Py_DECREF(pArgs);
55
56     return final;// == final;
57 }
```

Código 5: Llamado a código Python externo desde C++ en el módulo *DecisionAlgorithm*