Universidad de Alcalá

Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

Mapeado topológico de carreteras para la navegación de vehículos autónomos

Autor: Esther Murciego García

Tutor: Luis Miguel Bergasa Pascual

2017

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Electrónica y Automática Industrial

Trabajo Fin de Grado

Mapeado topológico de carreteras para la navegación de vehículos autónomos

Autor: Esther Murciego García

Tutor: Luis Miguel Bergasa Pascual

TRIBUNAL:

Presidente: Rafael Barea Navarro

Vocal 1º: Miguel Ángel García Garrido

Vocal 2º: Luis Miguel Bergasa Pascual

FECHA: 20/07/2017

Contenido

Índice de Figuras

Índice de Tablas

Índice de Ecuaciones

Resumen

En este Trabajo de Fin de Grado se ha llevado a cabo un estudio de los lanelets, que son elementos de carretera atómicos que permiten definir de forma topológica, la ruta a seguir por un vehículo autónomo.

Se ha demostrado la utilidad de los lanelets implementando un mapa del campus externo de la Universidad de Alcalá.

Además de analizar su eficacia, se han desarrollado unos algoritmos en C++ para obtener en un documento de texto los datos principales del mapeado.

Finalmente se ha validado su comportamiento gracias al simulador V-REP y a los algoritmos de navegación autónoma desarrollados por el grupo GROBIS de la Universidad de Vigo.

Palabras clave: Lanelets, Open Street Maps (OSM), Java OSM (JOSM), Dijkstra, conducción autónoma.

Abstract

In this End of Degree Work we have performed a lanelet´s study, lanelets are atomic road elements that allows us to define the topological form of the autonomous vehicle rute.

It has been demostrated the usefulness of lanelets, implementing a map of The Alcalá de Henares University.

In adittion to analyze its efficiency, algorithms have been developed in C++, to obtain a text document with principal dates of the map.

Finally, its behavior has been validated thanks to the V-REP simulator and the autonomous navegation algorithms developed for the GROBIS group of the Vigo´s University.

Keywords: Lanelets, Open Street Maps (OSM), Java OSM (JOSM), Dijkstra, autonomous Driving.

Conceptos

[1] Lanelet: Un lanelet describe un segmento de carril atómico que se caracteriza por su límite izquierdo y derecho y que están interconectados.

[2] Mapas de campo: Método que consiste en tomar mapas impresos sobre los cuales puede escribir para luego ser subidos y trazados.

[3] Georeferenciación: Técnica de posicionamiento espacial de una entidad en una localización geográfica única y bien definida en un sistema de coordenadas y datum (referencia) específicos.

[4] d3js (D3, Data-Driven Documents): Librería de JavaScript para producir a partir de datos, infogramas dinámicos e interactivos en navegadores web. Utiliza SVG (formato gráfico bidimensional), HTML5 (quinta revisión del lenguaje HTML, que es un lenguaje de programación web) y CSS (es el lenguaje utilizado para describir la presentación de documentos HTML o XML, esto incluye varios lenguajes basados en XML como son XHTML o SVG).

[5] Instituto Geográfico Nacional (IGN): Organismo público encargados de la creación, mantenimiento y comercialización de la cartografía oficial de España.

[6] Datum: Se aplica cuando se hace una relación hacia alguna geometría de referencia importante.

[7] Biblioteca Qt: Multiplataforma orientada a objetos. Utilizado para desarrollar programas que utilicen interfaz gráfica de usuario, aunque también tiene diferentes tipos de herramientas para la línea de comandos y servidores que no necesiten inerfaz gráfica.

[8] Clotoides: Curva tangente al eje de las abscisas en el origen y cuyo radio de curvatura disminuye de manera inversamente proporcional a la distancia recorrida sobre ella. Es por ello que en el punto origen de la curva, el radio es infinito.

[9] Distancia euclídea: En matemáticas, álgebra, geometría y, más específicamente, en análisis real, análisis complejo y geometría analítica, se trata de una función no negativa usada en diversos contextos para calcular la distancia entre dos puntos , primero en el plano y luego en el espacio.

[10] Nodo origen: Aquel que indica el punto de partida de la ruta.

[11] Nodo destino: Aquel que indica el punto de llegada dela ruta.

[12] ROS: Sistema operativo robótico.

[13] V-REP: Es un simulador en 3D con un entorno de desarrollo integrado.

[14] Framework: Es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

[15] Clúster: El inglés cluster, "grupo" o "racimo") se aplica a los conjuntos o conglomerados de ordenadores unidos entre sí normalmente por una red de alta velocidad y que se comportan como si fuesen una única computadora.

[16] OBJ: Es un formato de archivo objeto. Los archivos de objeto son archivos intermedios generados por el compilador antes de crear un ejecutable. El archivo de objeto consiste de una tabla de símbolos y el código C compilado en código de máquina.

[17] Red de Petri: Es una representación matemática o gráfica de un sistema a eventos discretos en el cual se puede describir la topología de un sistema distribuido, paralelo o concurrente. La red de Petri esencial fue definida en la década de los años 1960 por Carl Adam Petri.

Acrónimos

[ROS] ROS (Sitstema operativo robótico): Web page, http://www.ros.org/

[OSM] OSM: Open Street Map

[VREP] V-REP: Simulator Web page, http://www.coppeliarobotics.com/

[SRTM] SRTM (Misión topográfica Radar Shuttle): https://es.wikipedia.org/wiki/Misi%C3%B3n_topogr%C3%A1fica_Radar_Shuttle

[RNDF] RNDF (Archivo de la Definición de la Red de la Ruta**):** Darpa, "Urban challenge route network definition file (RNDF) and mission data file (MDF) formats," Mar. 2007. [Online]. Available: http://archive.darpa.mil/grandchallenge/docs/RNDF MDF Formats 031407.pdf

[UAH] UAH (Universidad de Alcalá de Henares)

[UVIGO] UVIGO (Universidad de Vigo)

[LiDAR] LiDAR: Light Detection and Ranging o Laser Imaging Detection and Ranging es un dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado.

[CVM] CVM (Curvature-velocity method): Establece en tiempo real, cuán lejos puede llegar el vehículo aproximando una trayectoria curva antes de llegar a un obstáculo.

Capítulo 1 Introducción

En estos últimos años, un tema que centra la atención de las empresas de automovilismo, gobiernos e investigadores, es el desarrollo de los coches autónomos, ya que vivimos en un mundo que siempre busca la automatización, buscando siempre la eficacia, y la rentabilidad, pero a la vez, también el confort y seguridad al volante.

Aunque un coche convencional, ya nos ahorra mucho tiempo en viajes y desplazamientos, con los coches autónomos, este tiempo lo podríamos utilizar para otras actividades.

Para que la conducción autónoma sea todavía más eficaz, aparte de incluir las nuevas tecnologías, lo mejor es que esté disponible para todo el mundo. Las tradicionales compañías automovilísticas o la nueva Tesla y otras como Google o Daimler están intentando este propósito, poner a disposición de un público más amplio la tecnología.

Pero antes de llegar a un mundo donde la mayoría de los coches sean autónomos, primero tenemos que perfeccionar los que se están desarrollando en este momento.

El coche autónomo, en el desplazamiento o conduciendo, no puede cometer errores, y para ello, tiene que entender su entorno. Necesita sistemas de percepción, planificación y toma de decisiones.

Hasta ahora todas las decisiones las tomaba el conductor con un mapa o simplemente viendo las señales; con el coche autónomo se trasladan esas responsabilidades a una máquina.

Que el coche pueda entender ese mapa, ese entorno, es el reto de todos los investigadores. El coche tiene que poder identificar el conjunto de carriles, sus interconexiones, la señalización vertical y horizontal, las normas de circulación a cumplir en cada tramo etc…

Ese mapa con todas las indicaciones debe ser completo en sentido geométrico y topológico (posiciones de los elementos, nodos con estados, límites…), además es necesario que el mapa con todos estos datos, esté cargado previamente en el coche, para no tener que estar procesando los datos del entorno continuamente.

En la actualidad no hay mapas tan completos, los mapas usados por los sistemas de navegación GPS no sirven, así que**, el objetivo** de este trabajo de fin de grado ***es utilizar un formato abierto de mapas y mejorarlo, añadiendo esa información topológica, de manera que sea más sencillo integrarlo en el vehículo y fusionarlo con otros mapas como OpenStreetMap o GoogleMap***.

Para ello utilizamos el método Lanelets [1]**.** Desarrollado por el grupo de investigación de una universidad alemana (Die Freie Universität Berlin) liderado por **Raúl Rojas**. Introducir esta información topológica solo era una parte de su proyecto focalizado en la conducción autónoma.

Modelaremos partes relevantes de las zonas conducibles con elementos de lanelet, que son segmentos de carretera atómicos, interconectados y manejables,

representados geométricamente por un enlace izquierdo y derecho. Formando una red de nodos con la información topológica del terreno.

Los lanelets que en este trabajo utilizamos nos indican el camino a seguir con el coche, pero también los límites de dicho carril. Los lanelets también se pueden utilizar usando una única línea en el camino a seguir por el coche, pero en este caso el coche no conoce los límites del carril.

Con los lanelets dibujaremos un mapa de carriles que representen las carreteras del entorno urbano, como es el universitario. Como estos elementos están formados por nodos unidos, al final obtendremos un grafo con su propia ponderación que cubra todo el entorno universitario. Esta ponderación vendrá dada por la longitud de las polilíneas, cuanto más largas más pondera.

Los lanelets no solo representarán carriles, sino también contendrán todos los elementos regulatorios necesarios para que el coche autónomo cumpla todas las normativas de tráfico.

Al tener un grafo ponderado, podemos utilizar alguno de los algoritmos disponibles para obtener un enrutamiento desde un punto origen a uno destino, buscando obtener el camino más corto que pueda seguir el coche.

Para ello utilizaremos el algoritmo **Dijkstra o algoritmo de caminos mínimos,** que a partir de un **nodo origen** [10] y las ponderaciones de los demás nodos, calcula todas las posibilidades de ruta para quedarse con las más cortas a cada nodo etiquetándolas de una manera especial.

Todas estas características las representaremos gracias a la herramienta JOSM, un editor off-line de Open Street Map. Con él podremos, a partir del mapa universitario dibujar los lanelets y añadirle las etiquetas correspondientes que necesitemos en los lugares indicados. Tal como señalizar pasos de cebra, señales para ceder el paso, STOP, roundabout o rotondas, etc.

Además, este editor guardará los datos del mapa dibujado en formato XML, lo que nos hará más sencillo comprobar las relaciones entre lanelets y sus miembros, y hacer el enrutamiento con el algoritmo de caminos mínimos.

Hay que tener en cuenta que el formato XML también aporta portabilidad al trabajo, ya que es independiente de la aplicación que lo utilice.

Se podrá ver, en el documento, un “árbol”, con los nodos al principio, los cuales tienen su propio id, latitud y longitud.

A continuación, se ven los “ways” o caminos con los nodos que los conforman, cada “way” tiene también su propia id.

Y después las relaciones formadas en el mapa: los lanelets y los elementos regulatorios.

Los lanelets tienen su propia id, los “ways” que los conforman, unas etiquetas que caracterizan al lanelet con su límite de velocidad, la capa a la que pertenecen del mapa, y la etiqueta que identifica esa relación como de lanelet. Si forman parte de una rotonda, además también tendrán una etiqueta que indique “roundabout” y a parte de los “ways” tiene adjunto el nodo que contendrá la señal de la rotonda.

Dentro de algunos lanelets, encontramos también, elementos regulatorios. Estas relaciones contendrán un nodo de referencia con la señal, una línea de parada y un nodo de activación para avisar al vehículo de la presencia de una señal para tener en cuenta. Asimismo, contendrá unas etiquetas que indiquen que esta relación no es de lanelets, si no, de elementos regulatorios, otra etiqueta que nos informe de su capa en el mapa, y otra que nos indique de qué señalización se trata.

Encontramos por último la relación “station”, que incluye el nodo que indica una posible parada o destino a la que desplazarse con el vehículo. En nuestro mapa hay 27 posibles paradas por todo el campus.

La validación del mapa se realizará en el entorno de simulación basado en V-REP desarrollado por la **Universidad de Vigo**. Este simulador nos permite obtener un mapa en tres dimensiones a partir del creado y de los datos ya subidos en Open Street Map y probar el algoritmo de navegación. Pudimos ver como el coche sigue los lanelets y obedecía los límites de estos.

Asimismo, se obtendrán unos vídeos de estas simulaciones y unas gráficas con los datos del coche a medida que avanza en la ruta.

De esta manera conseguimos, no solo que el coche conduzca de manera autónoma **cumpliendo** la **normativa** de conducción, sino que además conseguimos una **conducción eficiente** al calcular los caminos más cortos.

Capítulo 2 Open Street Map (OSM)

Antes de empezar a explicar la herramienta utilizada para la realización del mapeado tenemos que introducir otro concepto: Open Street Map.

Nuestro objetivo es que un coche, de manera autónoma, pueda guiarse por las rutas conducibles habilitadas para llegar a su destino correspondiente.

Una alternativa sencilla para lograrlo es la creación de un mapa donde el vehículo se pueda georeferenciar y por ello pensamos en OpenStreetMap.

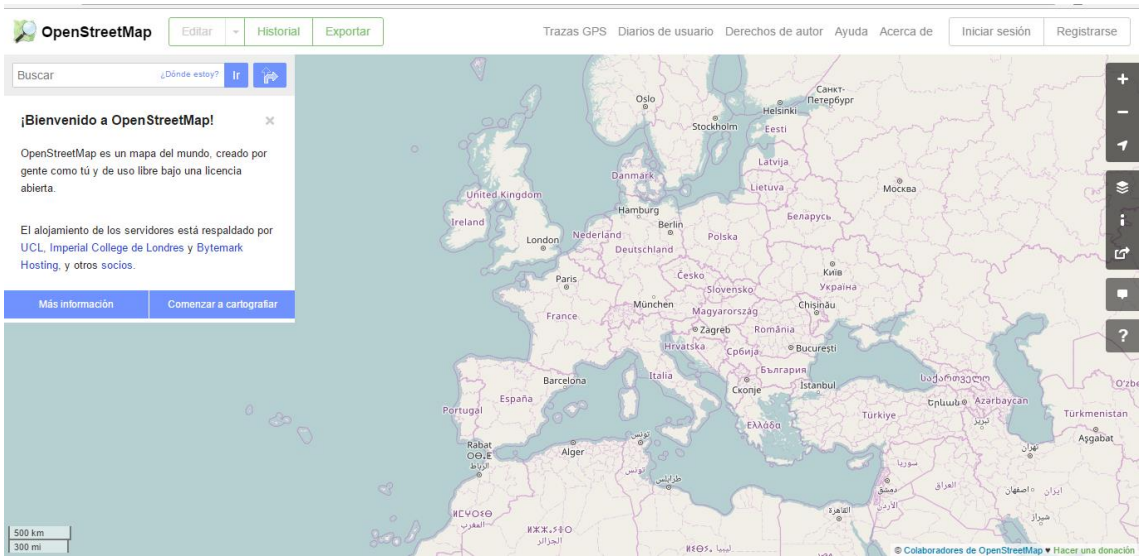


Figura 2.1: Interfaz Open Street Map

OpenStretMap (OSM) es un proyecto colaborativo para crear mapas libres y editables, todo el mundo puede acceder a ellos, así que cualquiera puede contribuir.

¿Por qué se hizo esta herramienta? Porque en la mayoría de los países la información geográfica no es libre. Además, las licencias para su uso restringen su utilización; no se pueden corregir errores ni tampoco introducir datos nuevos.

También hay iniciativas comerciales, como MapShare de Tom Tom, que sí que animan a los usuarios a contribuir, pero todas esas contribuciones pasan a ser propiedad de la empresa.

Así que en Julio de 2004 el inglés Steve Coast decide fundar OpenStreetMap, en concreto, por los altos precios que la empresa Ordance Survey (agencia cartográfica de Gran Bretaña), cobraba por su información geográfica.

En 2006 se registró como fundación, según lo escrito en el registro de Inglaterra y Gales:

"La fundación OpenStreetMap es una organización internacional sin ánimo de lucro dedicada a fomentar el crecimiento, desarrollo y distribución de datos geoespaciales libres y a proveer datos geoespaciales a cualquiera para usar y compartir"

Esta herramienta sigue creciendo hoy, y fue, por ejemplo, de mucha ayuda cuando ocurrió en 2010 el terremoto de Haití.

Voluntarios de OSM y Crisi Commons consiguieron hacer el mapa más completo de Puerto Príncipe (en Haití), en solo dos días, lo que fue de mucha ayuda para las organizaciones que se prestaron a auxiliar a los refugiados.

Al principio la recopilación de datos se realizó gracias a voluntarios mediante "trabajo de campo", a través de GPS, ordenadores portátiles, grabadoras de voz, etc. Más tarde estos datos se incorporaban a OSM.

Ahora que hay otras fuentes de datos comerciales y públicas, (ya que a día de hoy la información geográfica es por obligación, libre), fotografías aéreas y otros medios, obtenemos más datos en menos tiempo y además de mayor precisión.

En España en concreto, el **Instituto Geográfico Nacional (IGN) [5]** modificó en Abril de 2008 la licencia de utilización de sus datos, de manera que una parte pudiera utilizarse gratuitamente para cualquier actividad.

También toda aquella cartografía con derechos de autor caducados, pueden utilizarse en OSM.

2.1 Funcionamiento

Hay distintas maneras de usar OSM **[OSM]** Puedes añadir información a los mapas conduciendo o caminando con dispositivos basados en GPS, con **mapas de campo [2] ,** con aplicaciones de móvil (menos preciso que el GPS), hay algunas aplicaciones con las que puedes navegar usando los datos de OSM, como, por ejemplo: GPS Logger, GPS Essential u OSM Traker, todos disponibles para Android.

Si quieres utilizar el móvil, hay que tener en cuenta qué etiqueta tiene, puede ser GPS o A-GPS. Si tiene la etiqueta A-GPS (GPS asistido), no podrá mapear sin usar conexión a internet, en cambio si solo tiene la etiqueta GPS puede funcionar de manera autónoma sin internet. Y a la hora de elegir la aplicación para mapear solo hay que tener en cuenta que tenga soporte GPX (te permite crear puntos de ruta y personalizar sus registros), tiene que permitir contribuciones a OSM y cargar datos en él sin conexión, que pueda **georeferenciar [3]** archivos multimedia (notas, fotografías y vídeos) y que se encuentre en desarrollo activo para que no quede obsoleta.

Otras herramientas para mapear son iD, Merkaartor y JOSM, esta última es la que utilizamos en este trabajo y la explicaremos de manera más extendida más adelante.

iD es el editor on-line de referencia para incorporar datos en OSM, es fácil, rápido y puedes mapear desde distintas fuentes de datos, como imágenes satélites o aéreas.

Está desarrollado en JavaScript. Utiliza **d3js [4]** para el procesamiento y es un núcleo rápido y modular que gestiona datos de OSM. Su principal modo de representación es el formato SVG.

Merkaartor es un editor off-line multiplataforma de entorno muy cuidado y que utiliza **biblioteca Qt [7]**

JOSM también es un editor off-line. Es una aplicación de escritorio, basado en java y también es multiplataforma. Es el más avanzado y en su desarrollo está involucrado un mayor número de personas.

2.2 Formato de datos

Utiliza una estructura de datos topológica que se almacenan en el **datum [6] WGS84 lat/lon (EPSG:4326) de proyección de Mercator.**

Los elementos básicos de la cartografía de OSM son:

- **Nodos (nodes):** Puntos con una posición geográfica dada.

- **Vías (way):** Lista ordenada de nodos que representan una polilínea o un polígono (si la polilínea empieza y acaba en el mismo punto).

- **Relaciones (relations):** Grupos de nodos, vías u otras relaciones que se le pueden asignar propiedades comunes.

- **Etiquetas (tags):** Se pueden asignar a nodos, caminos o relaciones que constan de una clave (key, característica) y un valor (value, valor de esa característica o define esa característica dicha anteriormente). Ejemplos: speedlimit=40, define que en esta carretera el límite de velocidad está en 40 km/h. Otro ejemplo sería type=lanelet, esta etiqueta se suele poner a polilíneas y establecería que son del tipo lanelet.

Los datos del mapa están en dos dimensiones, no se suele registrar la tercera dimensión, Z. Aunque existen herramientas para tener estos datos en cuenta y superponerlos con los de OSM. Como, por ejemplo: Misión topográfica Radar Shuttle (SRTM) **[SRTM]**

Estos elementos serán utilizados en la creación del mapa.

2.3 Aplicaciones.

Open Street Map no solo se utiliza para hacer mapas de carreteras, sino también para crear mapas de senderismo, de vías ciclables, náuticos…

El cálculo de rutas y navegación en cambio no está tan desarrollado. Ya que hay regiones que no tienen los suficientes datos para ser fiables.

También puede haber problemas, al haber errores en los datos topológicos, o de etiquetado, o calles sin unir, etc producido por los usuarios de manera involuntaria.

OSM facilita herramientas a los usuarios para corregir los errores detectados.

Aun así, existen algunos algoritmos de búsqueda basados en grafos que podemos utilizar, como A* o Dijkstra.

A* es un algoritmo cuyo objetivo es encontrar el camino de menor coste entre un nodo origen y objetivo siempre que se cumplan unas condiciones.

Dijkstra o algoritmo de caminos mínimos, es el que utilizamos en este trabajo para obtener la ruta más corta del origen al objetivo.

2.3.1 DIJKSTRA

Es un algoritmo que determina el camino más corto, dado un vértice origen al resto de vértices, en un grafo con pesos en cada arista. Es decir, el algoritmo explora todos los caminos que llevan del vértice origen a todos los demás nodos del grafo para obtener los caminos más cortos a cada nodo destino **[11]**

EL grafo estará formado por nodos separados entre sí por distintas distancias y cada uno con nombre propio:

[distancia, nodo predecesor](iteraciones).

La longitud de las polilíneas que unen los nodos les otorga a estos una ponderación, que será más alta cuanto más larga sea la distancia.

Para utilizar el algoritmo, una vez tengamos el grafo, hay que seleccionar el nodo origen, que será el primer nodo “permanente” del análisis, el primero que etiquetaremos, y el primero que tomamos de referencia para analizar los siguientes nodos.

Como es el origen, tiene 0 distancia, ningún nodo predecesor y ninguna iteración, lo vemos en la Figura 2.2:

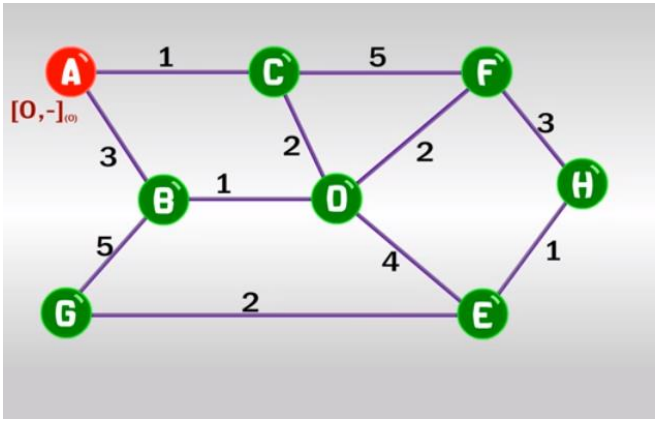


Figura 2.2: Paso 1 en el cálculo de los caminos mínimos

Ahora nos centraríamos en los nodos que están conectados a nuestro nodo origen. En este caso son B y C.

De A a B, la distancia es 3, sumándole la distancia del nodo predecesor y la distancia entre los dos nodos, como A es el nodo origen, su distancia acumulada es 0 y de A a B solo tendríamos 1 iteración.

En C, haríamos el mismo procedimiento, sumamos la distancia que tenía acumulada el nodo predecesor, A, con la distancia que hay entre ellos, en este caso es 1 y tenemos 1 iteración.

El siguiente nodo permanente por elegir, tiene que ser el que tenga menos distancia acumulada, si tienen la misma, se escoge uno de los dos aleatoriamente. Y repetimos el proceso.

Si el nodo permanente está conectado con un nodo que ya se ha analizado previamente, se vuelve a etiquetar y dejamos la etiqueta de menos distancia acumulada.

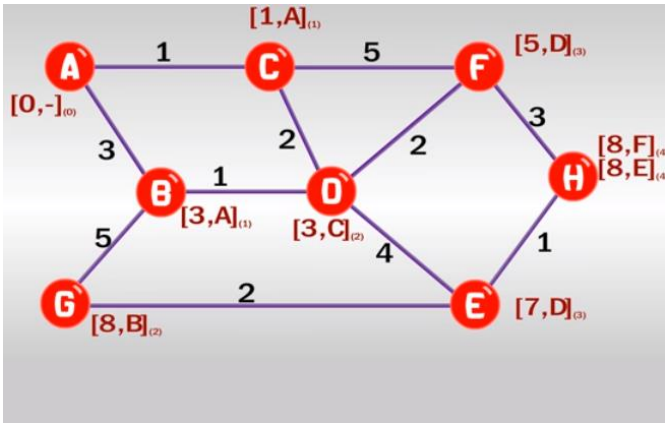


Figura 2.3: Paso 2 en el cálculo de los caminos mínimos

Con el grafo enteramente etiquetado vemos que H tiene dos rutas con la misma distancia acumulada, por lo tanto, hay dos rutas cortas para llegar a H.

Con este algoritmo obtenemos las distancias más cortas a cada nodo del grafo, así, a la hora de hacer la ruta más corta de un nodo a otro es más fácil de obtener.

Para obtener la ruta más corta se mira desde el destino hacia el origen.

Por ejemplo, si queremos ir de A a H, tenemos dos posibles rutas, porque, como hemos dicho antes hay dos rutas igual de cortas para llegar a H.

Una de las rutas sería: llegar a H desde E, según la etiqueta de E, llegamos a ella a partir de D, a D llegamos desde C y a C llegamos desde A, así que la primera ruta más corta que obtenemos de A a H es:

A-C-D-E-H

La segunda ruta más corta sería la siguiente:

A-C-D-F-H

De esta manera se obtiene de una manera fácil y rápida el enrutamiento más corto, una vez etiquetados todos los nodos.

2.3.2 Técnicas de mapeado

Nuestro objetivo es obtener una trayectoria a seguir por el coche autónomo que vaya del punto de inicio al destino lo más corta posible.

Para saber el camino más corto ya tenemos el algoritmo, ahora faltaría el grafo.

Pero también hay que tener en cuenta las normas viales.

Así que, se necesita un mapa, que no solo establezca los carriles por donde circular sino también las normas de tráfico.

También hay que predecir que harán los demás conductores, pero este tipo de problemas no se abordan en este trabajo.

Hay que intentar tener la máxima cantidad de información posible de lo que puede encontrarse el coche, por ejemplo, el coche necesita saber cómo va a ser la carretera antes de entrar a una curva para adaptar su velocidad.

Necesitamos representar el entorno en el que queremos conducir con variables geométricas y topológicas. Necesitamos tener las descripciones de los elementos del entorno y las relaciones entre sí.

Hay distintas técnicas posibles:

2.3.2.1 Enfoques geométricos

K. M. Wurm y su equipo representó el entorno 3-D utilizando octenas que permiten resoluciones arbitrarias donde sea necesario, llamándolo OctoMap**[18]**

El equipo AnnieWay´s representaba un carril como una polilínea con una distancia de vértice de 0,5 m con vértices almacenados y accedidos eficientemente por un árbol kd 2-D. **[19]**

E. D. Dickmanns y su equipo usó **clotoides [8]** para modelar los límites geométricos. **[20]**

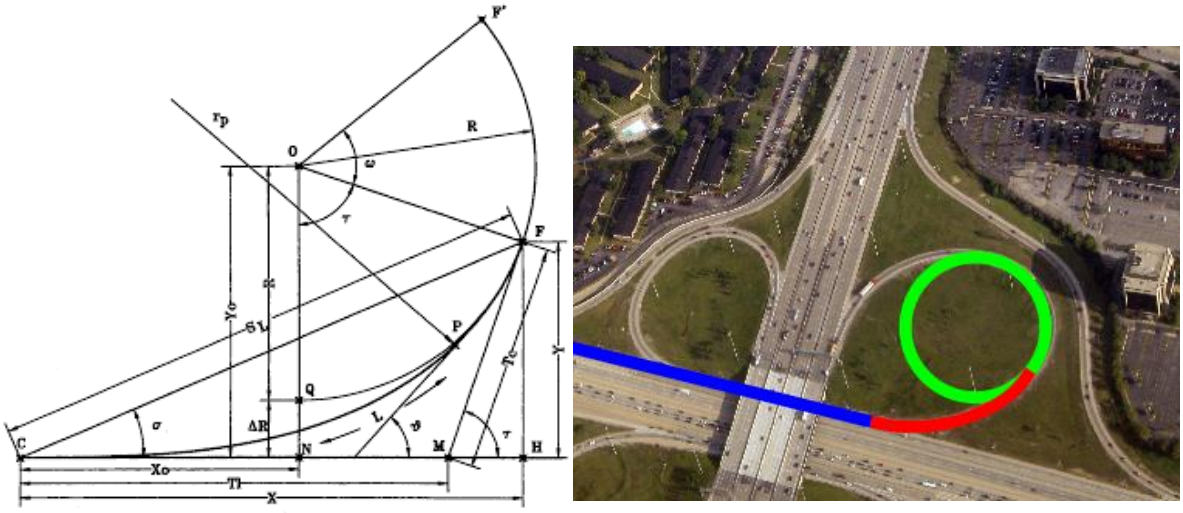


Figura 2.4: Dibujo de un clotoide y un clotoide sobre la carretera

Y. Wang y su equipo introduce el uso de Catmull-Rom splines para modelar límites de carreteras.**[21]**

C. Jung y C. Kelber, hizo un enfoque con diferentes modelos, uno lineal para el campo cercano y uno parabólico para el campo lejano. **[22]**

M. Baer y su equipo presentaron un enfoque para el modelo y el seguimiento de las marcas de carril en un 4-D.**[23]**

Otros enfoques utilizan las llamadas line-snakes o un par de parábolas con curvatura constante. **[24] .**

2.3.2.2 Enfoques topológicos y geométricos

En 2007 se publicó, por Darpa Urban Challenge, el formato de Archivo de la Definición de la Red de la Ruta (RNDF) **[RNDF] .**

Dentro de este formato, las carreteras se dividen en segmentos que comprenden uno o más carriles.

Los carriles son aproximados por un conjunto de waypoints así como un parámetro de anchura opcional. Algunos de los waypoints están etiquetados como waypoints de entrada o salida. De este modo, se representan las conexiones.

Las zonas representan un área libremente manejable aproximada por un polígono delimitador.

Es posible asignar límites de velocidad a segmentos de carretera y zonas.

Algunos autores con este tipo de enfoques son:

M. Brubaker utilizó datos procedentes de OpenStreetMap (OSM). **[25]**

Betaille propuso los Emaps (mapas extendidos), los cuales modelaban el mundo como segmentos de clotoides, líneas y círculos conectados. **[26]**

Hasber y Hensel que usan splines globales y el enfoque Bayesiano para reconstruir y representar la geometría y la topología en el contexto de la localización del rail del vehículo. **[27]**

Y por último tenemos la técnica que utilizamos en este trabajo:

2.3.2.3 Lanelets

A la hora de marcar el camino que debe seguir el coche hay dos opciones: que la polilínea marque el centro del camino o crear dos polilíneas que delimiten un carril en el que tenga que mantenerse el coche.

Con la primera opción surge un problema, el coche no tiene en cuenta los límites del carril.

Si realizamos la segunda opción con dos polilíneas que representen los límites del carril, solventamos ese problema.

Un lanelet describe un segmento de carril atómico que se caracteriza por su límite izquierdo y derecho y que está interconectado con otros.

Los límites son polilíneas (matriz de puntos). El objetivo de cada límite del carril (izquierdo o derecho) es establecer la dirección de conducción, su sentido y delimitar la conducción.

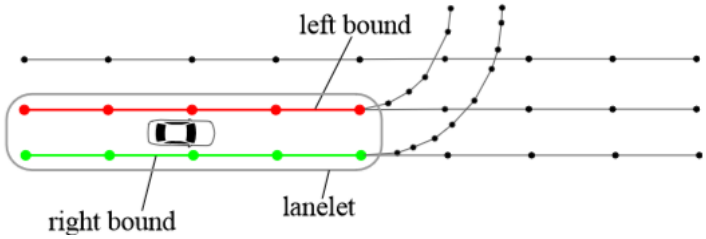


Figura 2.5: Partes del lanelet

Los lanelets como hemos dicho forman carriles, los cuales se consideran adyacentes si los puntos inicial y final son idénticos y la curvatura de las polilíneas determinadas por los bordes izquierdo y derecho está por debajo de cierto umbral. Esto indica que la transición es suave.

Según la longitud del lanelet se le asigna una ponderación.

Con el algoritmo Dijkstra y el grafo que conforma el mapa podemos obtener una ruta.

En nuestro caso algunos carriles lanelet al formar parte de una rotonda se les ha asignado una etiqueta que indica que su rol es el de una rotonda (roundabout), contienen también los elementos regulatorios de las carreteras y, contienen las posibles paradas que puede realizar el coche en el campus universitario.

OSM define un archivo en formato XML para el almacenamiento e intercambio de datos de mapas, así como una arquitectura de servidor para almacenar y validar centralmente datos de mapas.

El programa que utilizamos para comprobar las uniones de los lanelets, recorre el archivo XML y encuentra las agrupaciones que tengan en común los nodos del final con los del principio del siguiente lanelet para corroborar la unión entre ambos, por lo que este formato nos facilita las comprobaciones sobre el mapa.

Así es como se ven los nodos, un lanelet, una estación y un elemento regulatorio en el archivo XML:

```
<?xml version='1.0' encoding='UTF-8'?>

<osm version='0.6' generator='JOSM'>

 <node id='-30473' action='modify' visible='true' lat='40.51294112134' lon='-3.34706049984' />

 <relation id='-43716' action='modify' visible='true'>

  <member type='node' ref='-43667' role='ref' />

  <member type='way' ref='-43668' role='stop_line' />

  <member type='node' ref='-43674' role='activate_point' />
```

```
  <tag k='layer' v='uah_lanelets' />
  <tag k='maneuver' v='pedestrian_crossing' />
  <tag k='type' v='regulatory_element' />
 </relation>
 <relation id='-40357' action='modify' visible='true'>
  <member type='node' ref='-35911' role='ref' />
  <tag k='name' v='Politecnica' />
  <tag k='type' v='station' />
 </relation>
 <relation id='-40365' action='modify' visible='true'>
  <member type='node' ref='-35649' role='roundabout' />
  <member type='way' ref='-38571' role='left' />
  <member type='way' ref='-38563' role='right' />
  <tag k='Role' v='roundabout' />
  <tag k='layer' v='uah_lanelets' />
  <tag k='station' v='40' />
  <tag k='type' v='lanelet' />
 </relation>
```

Utilizamos la librería Liblanelet para leer y analizar los archivos XML, así como la rutina de enrutamiento, implementada como una búsqueda Dijkstra.

La biblioteca oculta la construcción de los carriles, el árbol R (R-Tree) y el gráfico de enrutamiento detrás de una clase LaneletMap.

OSM guarda datos posicionales con una resolución de hasta 10-7 grados, que es de aproximadamente un centímetro en el ecuador.

Las polilíneas que dibujamos en el mapa en JOSM estarán formadas por unos nodos, los cuales tienen su propia id, latitud y longitud y agrupamos estas polilíneas en lanelets para hacer carriles.

Un carril de dos polilíneas es un lanelet.

Para crear dos carriles del mismo sentido, se dibujan 3 polilíneas con el mismo sentido y se etiquetarán, dentro del lanelet que formen, como “left” o “right” según el lado del carril en el que estén y el sentido de este. La polilínea central tendrá entonces dos etiquetas: left y right, se le pondrá una u otra según el lanelet que estemos editando, ya que esta polilínea formará parte de dos lanelets distintos. La polilínea con dos etiquetas se define como **límite compartido**.

Los **límites compartidos** se representan por un elemento de una sola vía, que está presente en todos los lanelet vecinos con el rol apropiado. Esto mantiene la redundancia baja.

Cuando un camino se bifurca hay que definir varios lanelet distintos, donde las bifurcaciones deben salir de los mismos nodos, por ejemplo, si una calle se divide en dos caminos, encontraríamos un lanelet central, y dos caminos que saldrían del final del lanelet central.

La forma de proceder es la que se indica a continuación:

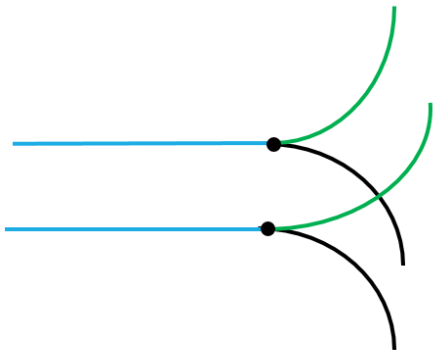


Figura 2.6: Forma correcta de creación de lanelets

Se pueden observar tres lanelet distintos con diferentes colores en la Figura 2.6.

No se podría hacer los siguiente:

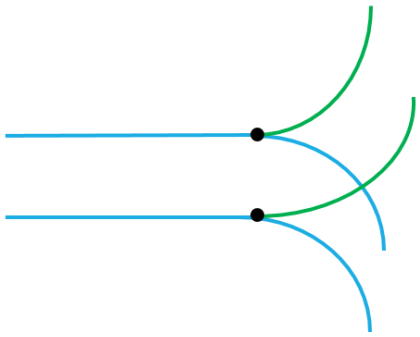


Figura 2.7: Forma incorrecta de creación de lanelets

En la Figura 2.7 se distinguirían solo dos lanelet. El programa con el que comprobamos las conexiones nos devolvería un error si procedemos como en la Figura 2.7, ya que comprueba que el final del lanelet azul no está conectado con el verde, pero el verde sí está con el azul.

Veremos más detalles sobre los procedimientos de creación de carriles en el mapa en el 0.

Como hemos visto para hacer dos carriles del mismo sentido podemos dibujar solo tres polilíneas compartiendo la central. Pero, una polilínea no puede albergar dos sentidos. Por lo tanto, si vamos a dibujar dos carriles de distintos sentidos no podremos compartir la polilínea central, tendrá que haber una distinta para cada lanelet.

2.3.2.4 Relación lanelet

Dentro de la relación lanelet podemos encontrarnos con:

- **Etiquetas**:
    - **Capa (layer):** Con el nombre de la capa a la que pertenece dentro del mapa.
    - **Límite de velocidad (speedlimit):** Establecemos el límite de velocidad de esa carretera.
    - **Tipo(type):** Como hablamos de lanelets, son tipo lanelet. ‘type=lanelet’.
    - **Rol (Role):** Esta etiqueta la tienen todos aquellos que contengan un nodo ‘roundabout’, e indica que ese lanelet pertenece a una rotonda.
    - **Parada (station):** Es un nodo cuya etiqueta de tipo es la de estaciones. Representan los lugares del campus a los que puede ir el coche o desde los que puede salir.
- **Miembros:**
    - **Polilíneas**: Etiquetadas con un nombre según el lado del carril al que pertenezcan: right para la derecha y left para la izquierda.
    - **Otras relaciones**: Como elementos regulatorios.
    - **Nodo rotonda**: Que se llamará “roundabout”, este nodo en el mapa tiene el icono de rotonda. Todos los lanelet que tengan este nodo también tendrán la etiqueta: ‘Role=roundabout’.

2.3.2.5 Elementos regulatorios

La idea es incluir en las líneas esas características reguladoras de las carreteras como son las señales y los semáforos. Ya que el coche tiene que cumplirlas.
Los elementos regulatorios contienen la siguiente información:

- **Etiquetas (tag)**: Tienen las siguientes:
    - **Tipo**: El tipo en este caso es el de elemento regulatorio. 'type=regulatory_element'.
    - **Maniobra** (**maneuver**): Establece la "maniobra" que tiene que realizar el coche al encontrarse con este elemento regulatorio. Puede ser:
        - Ceda el paso(give_way)
        - Paso de peatones (pedestrian_crossing)
        - STOP
        - Semáforos.
    - **Capa (layer):** Con el nombre de la capa a la que pertenece dentro del mapa.
- **Miembros (member)**: Son los elementos que conforman el elemento regulatorio.
    - **Línea de parada (stop line):** Establece el límite en el que el coche debe pararse. La tienen todos los elementos regulatorios.
    - **Referencia**: Es el nodo en el que establecemos qué señal representa este elemento regulatorio.
      Si se trata de un semáforo tendrá más de un punto de referencia, tiene el semáforo propio de ese carril y los demás puntos de referencia que representen los otros semáforos con los que está sincronizado.
    - **Punto de activación (activate_point)**: Punto en el cual el coche es avisado de que a pocos metros de él se encuentra una señal que debe respetar.
    - **Punto final (end_point)**: Solo lo tienen los semáforos. Es un punto que sobre pasa mínimamente a la línea de parada, y que indica el punto que no puede sobrepasar si por alguna razón sobrepasara la línea de parada.

Estos elementos regulatorios no solo indican las posiciones de las señales físicas, también encontramos en el mapa, por ejemplo, el "ceda el paso" antes de entrar a una rotonda o cuando tienes que ceder el paso a alguien que se aproxime por tu derecha. Es decir, también está representadas las "reglas no escritas" de la conducción.

2.3.2.6 Paradas o estaciones

Es el último tipo de relación que nos podemos encontrar y que también se incluye dentro de los lanelets. Está formada por:

- **Etiquetas**:
    - **Tipo (type):** Establece el tipo de relación, como en las otras, en este caso es tipo “station” o estación.
    - **Name (nombre):** Establece el nombre de la parada. En total hay 27 paradas a lo largo del mapa del campus:

        Allergy_Therapeutics_Iberica_SL

        AnovaIT_Consulting

        Apeadero

        Aryse_Comunicaciones

        Asociacion_Empresarios_del_Henares

        Biblioteca_Nacional_de_España

        Biologia

        BTSA_Biotecnologias_Aplicadas

        CDG_Telefonica

        Ciencias_Ambientales

        DeportesUAH

        Farmacia

        Global_Line

        Grupo_Epelsa

        HospitalUniversitario_PrincipedeAsturias

        Inmunotek_SL

        Mecanizados_Escribano

        Medicina

        Mytra

Politecnica

PolitecnicaEnd

PolitecnicaStart

Residencia_Universitaria

TiendaUAH

Varlion_Oficial

Veracetics_Aloe_Vera_Puro

Vidacord

- **Miembro:** En esta relación solo tenemos un miembro que es el nodo que representa la parada.

Capítulo 3 Herramienta JOSM (Java Open Street Maps)

JOSM es el editor off-line de OpenStreetMaps que utilizamos para este trabajo. El creador de esta herramienta fue Immanuel Scholz, actualmente es mantenido por Dirk Stöcker.

Tiene el formato de una aplicación para PC y su página principal es: josm.openstreetmap.de

El editor lo descargué mediante la terminal de Linux (Ubuntu) siguiendo los pasos del siguiente enlace:

https://josm.openstreetmap.de/wiki/Download

En mi equipo en Windows 8.1, instalé una máquina virtual de Oracle para Windows con el que cargue el sistema operativo Linux (Ubuntu), por las restricciones del código que tendría que usar, ya que JOSM está disponible también para Windows, pero el código que tenía disponible solo es compatible con Linux.

Para su funcionamiento es necesario tener instalado al menos Java 8, que también me lo descargué desde la página principal de java, para su instalación.

Al ser off-line, dos personas distintas pueden editar la misma zona a la vez, ya que, el procedimiento para editar una zona, es descargarla de internet, editarla desde la aplicación y después se suben los datos a Open Street Map. JOSM se encarga de mediar si varias personas suben datos de la misma zona, fusionará los cambios realizados por los usuarios.

Es muy útil para cambiar un conjunto grande de datos. Si solo quieres cambiar algún detalle, como por ejemplo el nombre de una calle, es más rápido utilizar el editor on-line iD.

3.1 Interfaz

Al abrir el editor de JOSM, podemos ver lo siguiente:

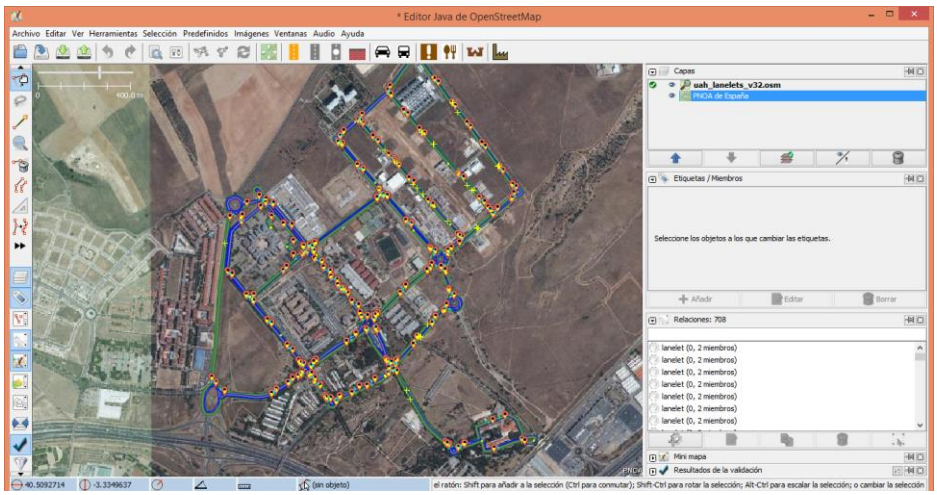


Figura 3.1: Interfaz JOSM

Recordemos que este editor es off-line, no necesita internet para editar el mapa, incluso hay imágenes disponibles en la aplicación para no descargarnos la zona del mapa que queremos editar, pero no están actualizadas. Para obtener imágenes más actualizadas, sí necesitamos internet, por ejemplo, se puede utilizar las imágenes de PNOA Spain que están más actualizadas que las que vienen en la aplicación, para editar con más precisión.

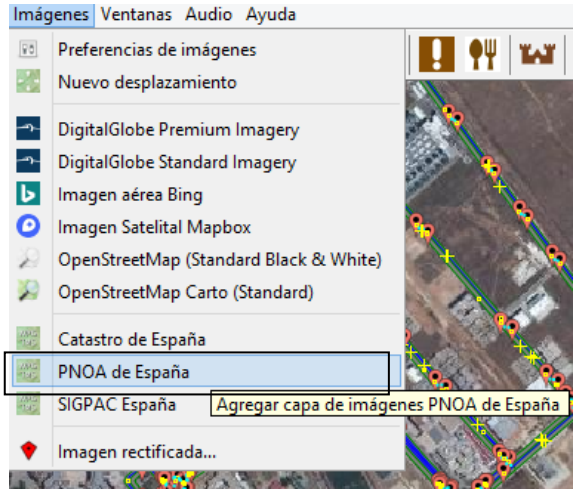


Figura 3.2: Imágenes disponibles en JOSM

No están tan actualizadas, ya que no es lo mismo que descargarse una zona desde Open Street Map.

En la barra de la parte superior vemos las distintas opciones que tenemos para la administración de los proyectos. Podemos guardar un proyecto, subirlo, bajarnos algún proyre o abrir uno que tengamos guardado:

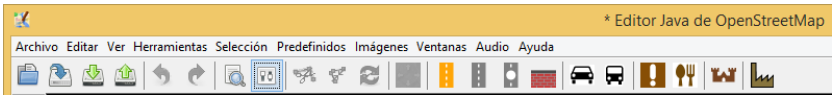


Figura 3.3: Barra superior de la interfaz de JOSM

Iconos principales de la barra superior

Icono	Descripción
	Abrir proyecto
	Guardar proyecto
	Bajar datos
	Subir datos
	Deshacer
	Restaurar
	Preferencias

Tabla 3.1: Iconos barra superior

Los iconos utilizados que se pueden ver en la barra izquierda:

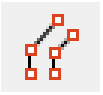
Icono	Descripción
	Seleccionar, mover, escalar y girar objetos.
	Dibujar nodos, al unirlos obtenemos las polilíneas.
	Hace copias paralelas de vías.

Tabla 3.2: Iconos principales de la barra izquierda

A la derecha de la Figura 3.1 podemos ver las siguientes ventanas:

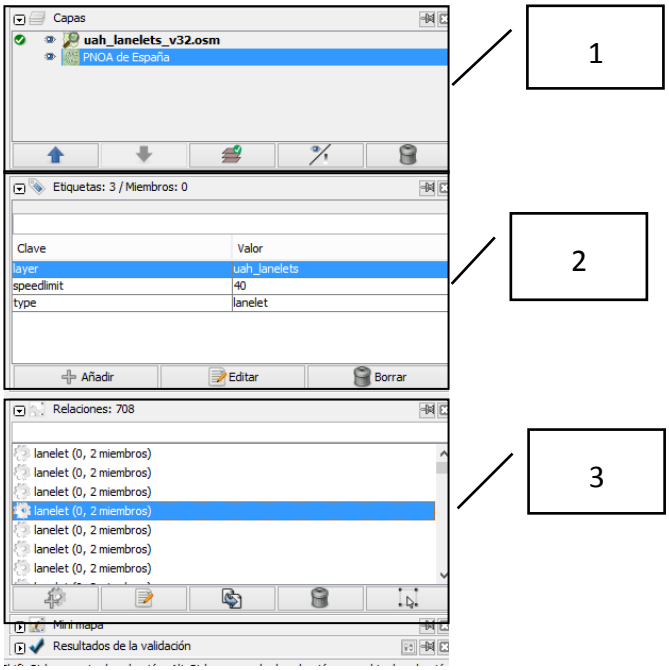


Figura 3.4: Parte derecha de la interfaz de JOSM

1. **Capas:** Aquí observamos las distintas capas que conformarán el mapa, en principio son dos, la capa donde se guardan los elementos dibujados y otra capa con las imágenes del mapa, en las que nos basamos a la hora de dibujar las polilíneas.

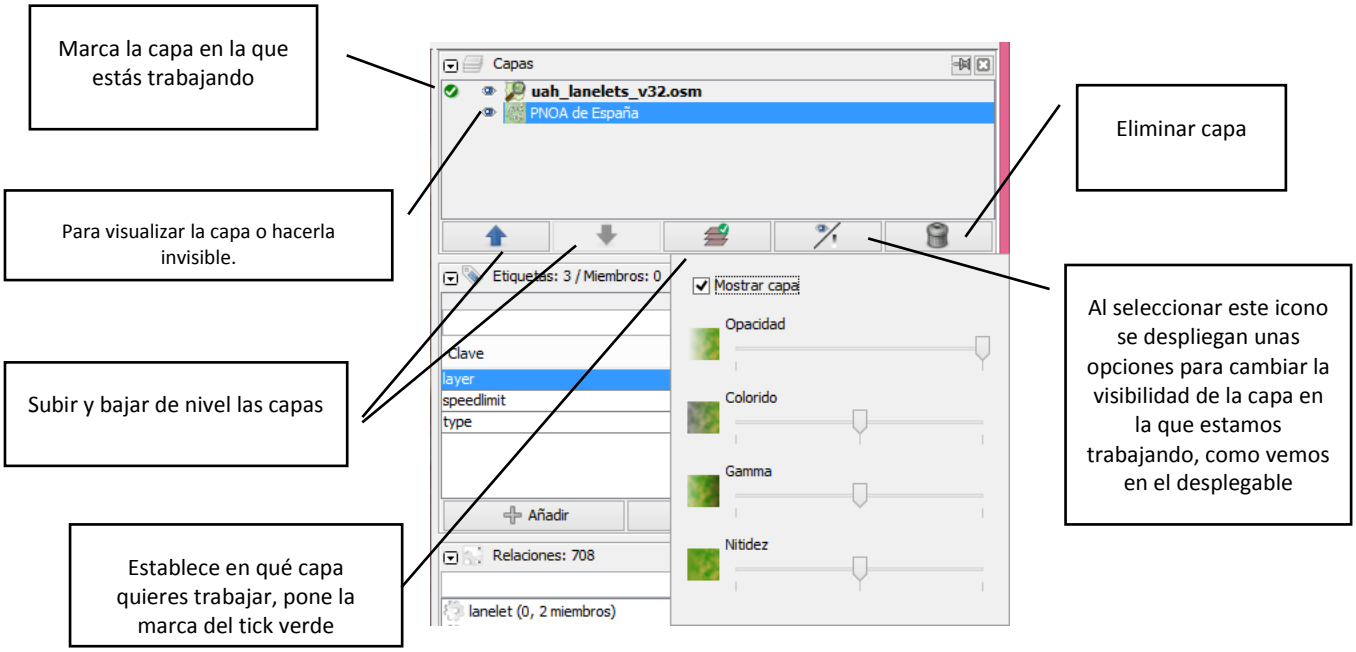


Figura 3.1: Apartado de "Capas" en la interfaz de JOSM

2. **Etiquetas/miembros:** Este apartado contiene una información distinta dependiendo de lo que selecciones:
   Si se selecciona una relación, podemos visualizar las etiquetas de la relación, en el caso que mostramos la relación es de lanelet:

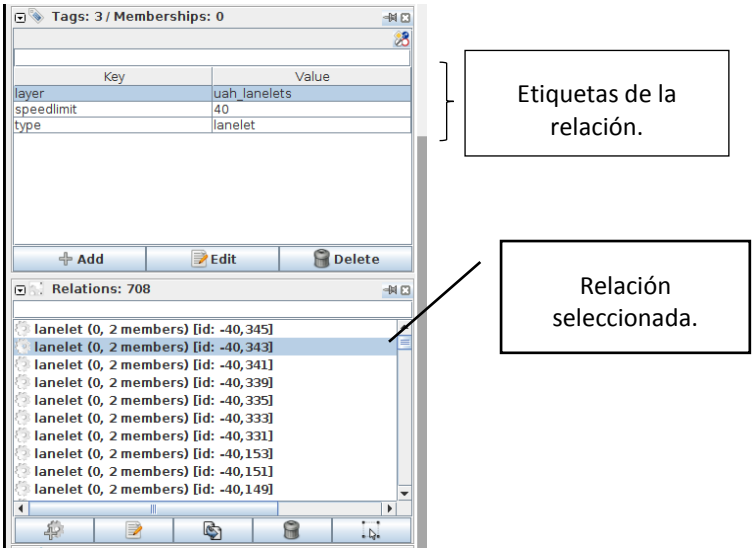


Figura 3.5: Apartados de Relaciones y etiquetas en JOSM

Al seleccionar un elemento de la relación se pueden ver sus etiquetas y las relaciones a las que pertenece:

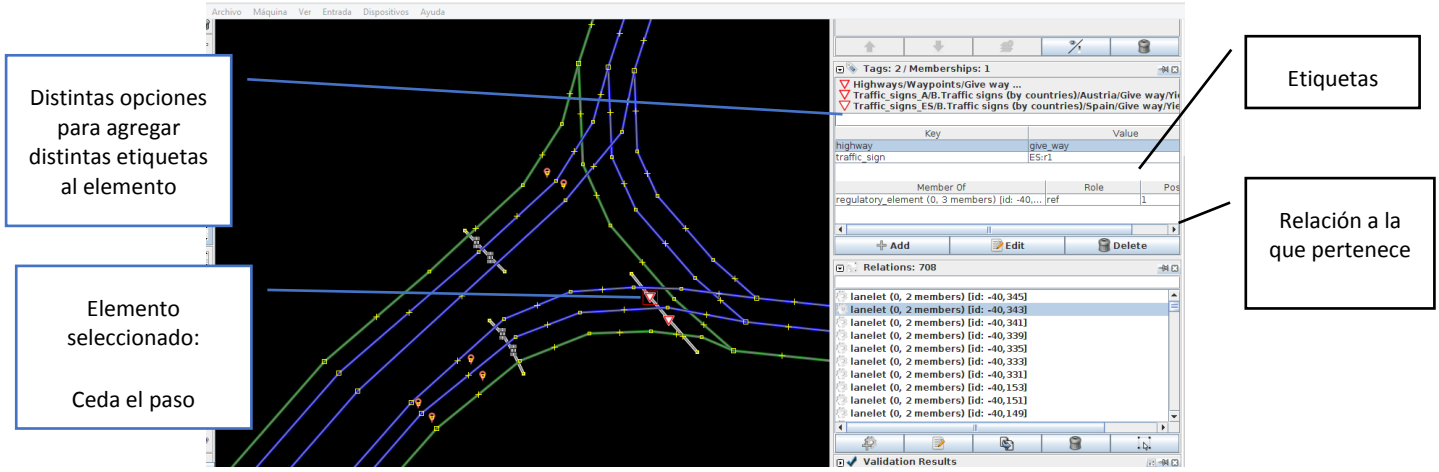


Figura 3.6: Características de un elemento en JOSM

En la ventana que se puede ver en la Figura 3.7, vemos las demás funcionalidades de la ventana de las Etiquetas/Miembros.

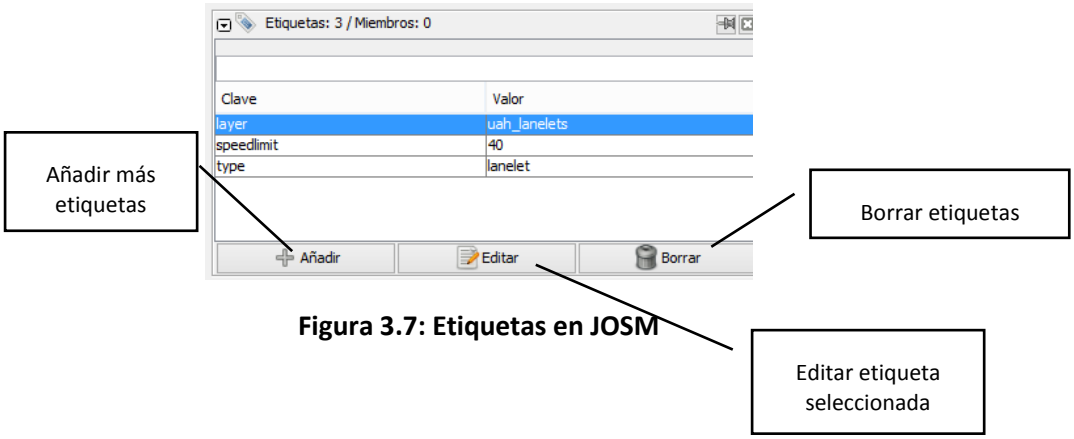


Figura 3.7: Etiquetas en JOSM

3. **Relaciones:** enseña una lista de todas las relaciones incluidas en el mapa, junto con su id.
   Estas ids cambian cada vez que cargamos el mapa, ya que trabajamos off-line, el programa los crea y con ello cambian en el xml.
   Si trabajáramos online, subiendo los datos, siempre tendríamos las mismas.
   Para que se puedan ver las ids hay que seleccionar:
   Preferences ->Advanced preferences

   Y buscar la opción "osm-primitives.showid" y establecerla en true:

Figura 3.8: Opción para que aparezcan las ids

Así que en la ventana de relaciones podemos ver:

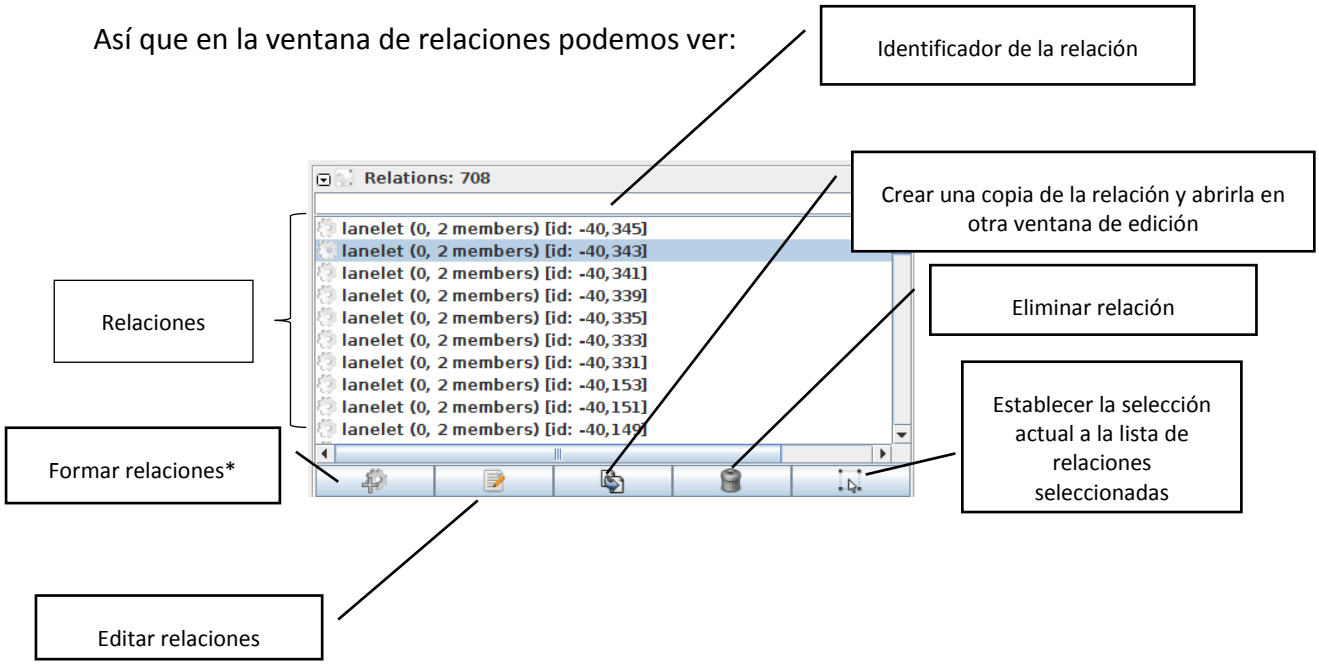


Figura 3.9: Relaciones en JOSM

*Para formar relaciones seleccionamos varios elementos del mapa y seleccionamos el icono señalado, se abre entonces, esta ventana:

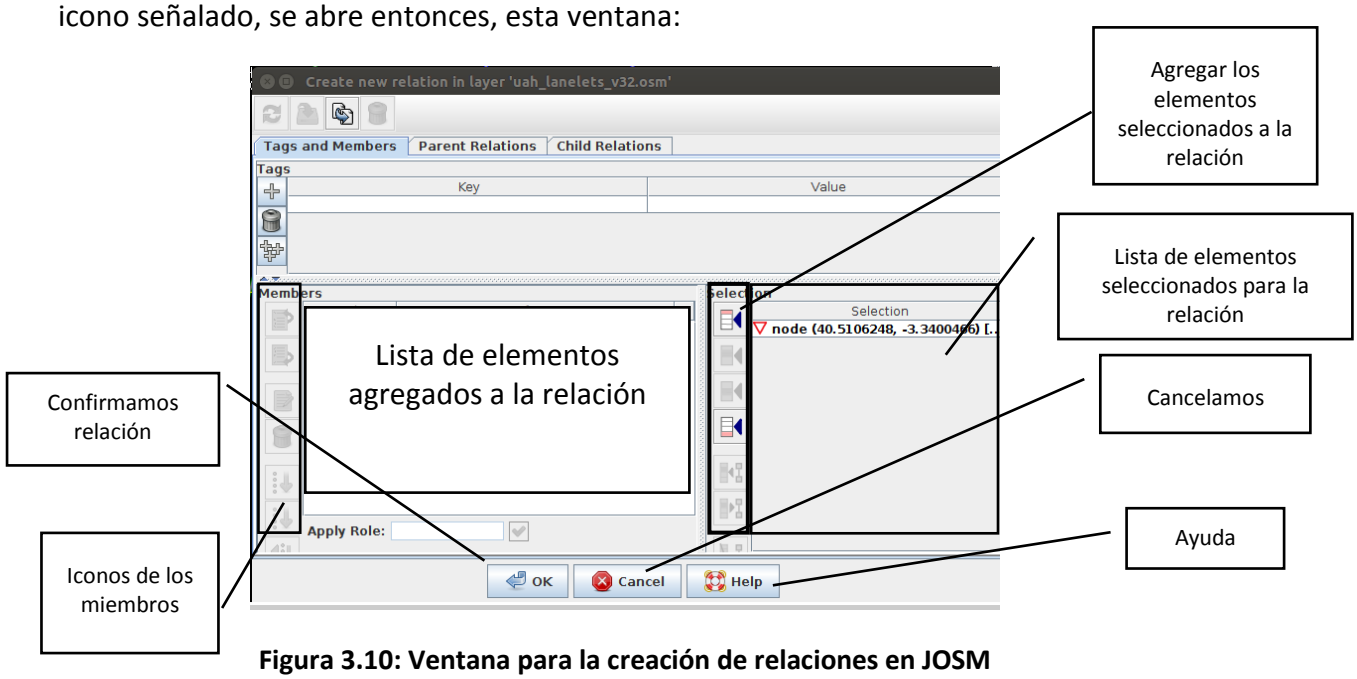


Figura 3.10: Ventana para la creación de relaciones en JOSM

Los iconos que podemos ver en la ventana de la Figura 3.10:

Icono	Descripción
	Desplaza una posición hacia arriba el elemento seleccionado
	Desplaza una posición hacia abajo el elemento seleccionado
	Editar la relación del elemento actualmente seleccionado.
	Eliminar elemento
	Ordenar todos los elementos de la relación.
	Ordenar los elementos seleccionados y los demás, debajo.
	Invertir orden de los miembros de la relación.

Icono	Descripción
	Descargar miembros incompletos
	Descargar miembros incompletos seleccionados

Tabla 3.3: Iconos de la ventana de miembros

Iconos para agregar los elementos a la relación, que se pueden ver en la Figura 3.10:

Icono	Descripción
	Añadir todos los elementos actuales seleccionados en el conjunto actual antes del primer miembro
	Añadir todos los elementos actuales seleccionados en el conjunto actual antes del primer miembro seleccionado.
	Añadir todos los elementos actuales seleccionados en el conjunto actual detrás del último miembro seleccionado.
	Añadir todos los elementos actuales seleccionados en el conjunto actual antes del último miembro.
	Seleccionar los miembros de la relación que se refieren a dos objetos de la selección actual.
	Seleccionar los objetos para los miembros de la relación indicados

Tabla 3.4: Iconos para agregar elementos a la relación

Para los iconos de las señales de tráfico instalamos el siguiente plug-in desde la página oficial de JOSM:

https://josm.openstreetmap.de/wiki/Plugins

Y elegimos el plug-in Road sign.

Podemos ver el icono en la ventana de Etiquetas/Miembros si seleccionamos un nodo:

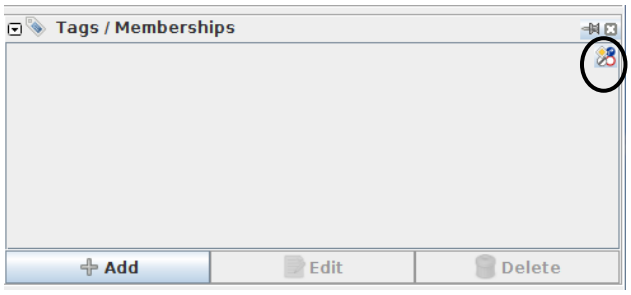


Figura 3.11: Icono para insertar señales

Si hacemos click se abre la ventana que vemos en la Figura 3.12 donde podemos elegir qué señal queremos adjudicar al nodo seleccionado y las etiquetas:

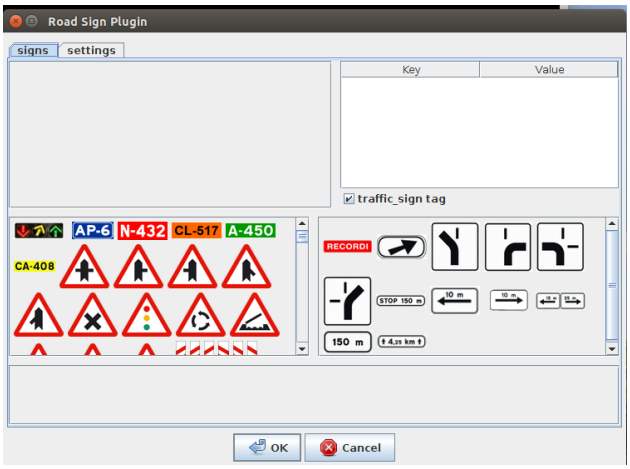


Figura 3.12: Lista de señales a elegir

Hemos utilizado este plug-in para los: cedas el paso, semáforos, STOP y rotondas.

Capítulo 4 Algoritmos

Utilizamos principalmente dos algoritmos, uno para saber si las conexiones del mapa son correctas, con el nombre de demo.cpp, y otro para obtener las rutas, llamado enrutamiento.cpp.

4.1 Algoritmo Demo.cpp

Este algoritmo se encarga de la comprobación de las conexiones entre los lanelets.

El argumento que hay que introducir es la ubicación en nuestro ordenador del mapa, del cual, queremos comprobar las conexiones entre los lanelets.

La primera parte del código de este programa contiene funciones que pertenecen al proyecto nombrado en la introducción, de la universidad alemana Die Freie Universität Berlin, liderado por Raúl Rojas.

Este código lo utilizamos para comprobar que existen todas las relaciones entre los lanelets. Obtenemos un listado en el que podemos ver qué lanelet está conectado con cuál.

El código recorre los nodos de los lanelets y se centra en los que conforman los límites de cada lanelet, es decir, los que están al principio y al final del lanelet.

Cuando recorre esos nodos límite comprueba para qué lanelet ese nodo forma parte del principio y para qué lanelet ese nodo forma parte del final, eso significa que esos lanelets están conectados.

Complementamos el código con unas líneas para poder comprobar que todos los lanelets tienen al menos dos conexiones. Esta parte del código siempre nos devolverá dos ids, de los lanelets de las siguientes paradas: PolitecnicaStart y PolitécnicaEnd.

Ya que PoliteccnicaStart solo puede ser una parada origen y PolitecnicaEnd solo puede ser una parada destino. Están dibujados en el mapa de manera que solo tienen una conexión con otro lanelet, así que la función, si el mapa está bien, nos tendría que devolver solo la id de los dos lanelet que corresponden con estas dos paradas.

Ya que si devuelve la id de una lanelet que no es ni PolitecnicaStart o PolitecnicaEnd eso significa que ese lanelet solo está conectado con un lanelet más, cuando lo normal es que tenga una lanelet delante y otro detrás formando el camino.

Así detectamos rápidamente dónde están los lanelets mal conectados. Si hay lanelets mal conectados no podemos ejecutar el siguiente algoritmo, el de enrutamiento.

A continuación, mostramos un diagrama de bloques del funcionamiento del algoritmo.

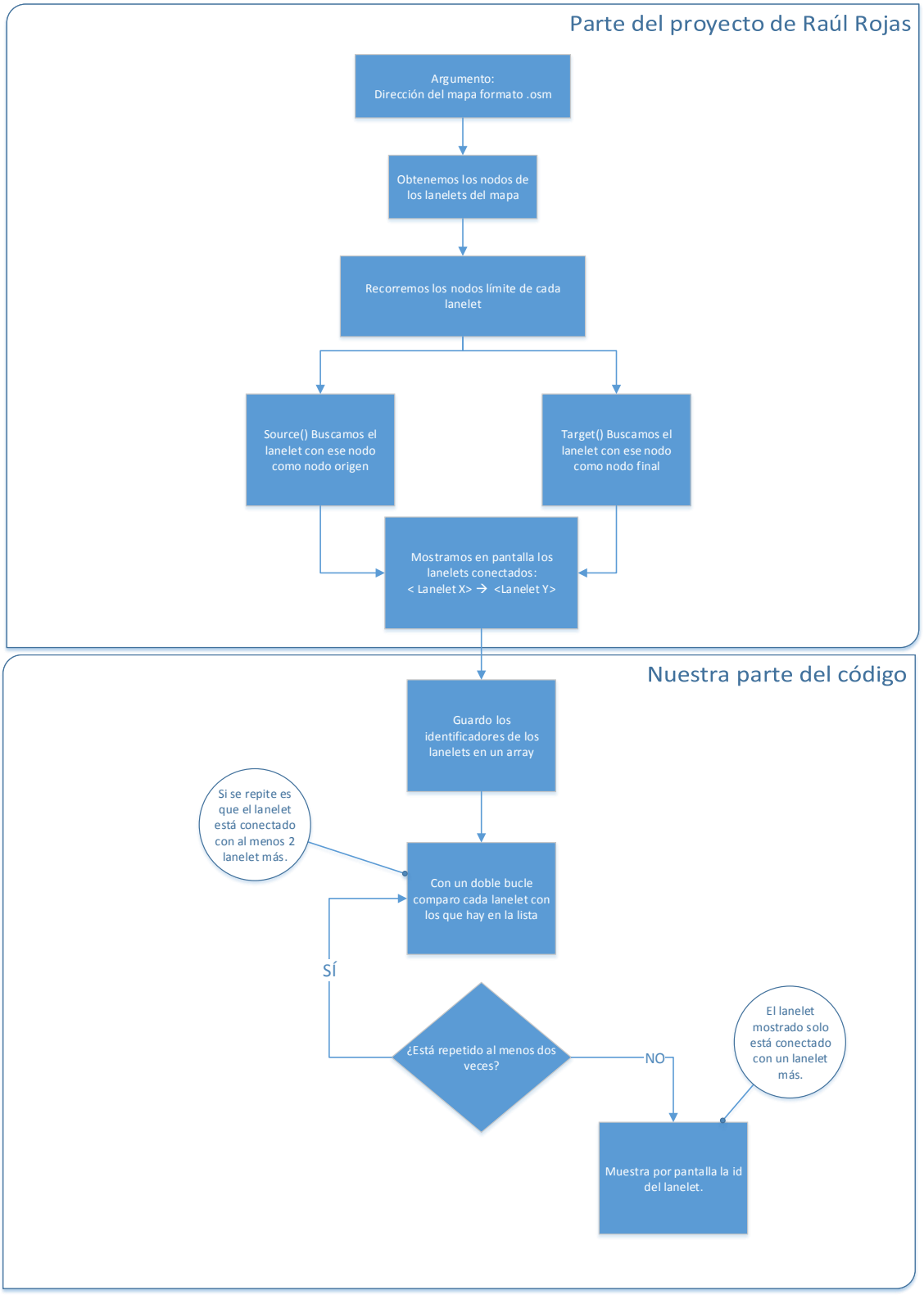


Figura 4.1: Diagrama del algoritmo Demo.cpp

4.2 Algoritmo Enrutamiento.cpp

Antes de comenzar con el algoritmo de enrutamiento.cpp, es necesario introducir el concepto de **distancia euclídea [9]** ya que es utilizado.

4.2.1 Distancia euclídea

La distancia euclídea es utilizada en matemáticas, geometría, análisis real, análisis complejo y geometría analítica. Se trata de una función no negativa usada en diversos contextos para calcular la distancia entre dos puntos, primero en el plano y luego en el espacio.

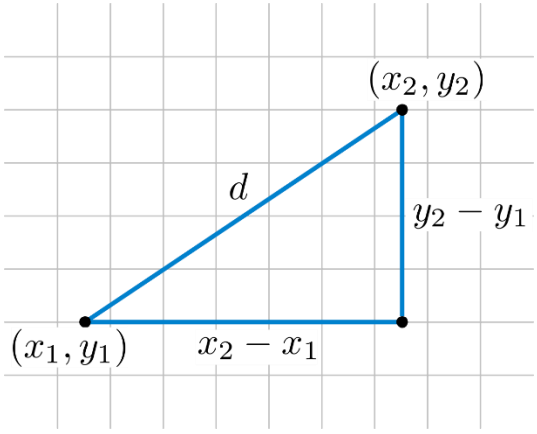


Figura 4.2: Distancia euclídea y el teorema de Pitágoras

Está basada en el teorema de Pitágoras. En dos dimensiones su expresión es la siguiente, si quieres calcular la distancia euclídea desde el punto P1 al P2 siendo las coordenada de P1(x1, y1) y las de P2 (x2,y2), y siendo el espacio de 2 dimensiones, ya que no introducimos ninguna altura, en nuestro caso:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ecuación 4.1: Distancia euclídea

Comparando distancias euclídeas en el programa, podremos establecer el nodo más cercano al nodo origen o destino, y con ello podremos saber el lanelet que hay que establecer como origen y el lanelet que hay que establecer como destino.

Puede surgir la pregunta de, si sabemos en qué lanelet están las paradas, por qué no buscarlas entre los lanelets y establecer así los lanelet de origen y destino, o asignar según la parada escrita, su lanelet, en vez de comparar su latitud y longitud con los nodos de los lanelet para buscar el más cercano.

Esto lo hacemos porque en principio el programa iba a utilizarse para cualquier tipo de coordenada y también porque el programa cada vez que cargamos el mapa en la

herramienta JOSM, como hemos dicho anteriormente, cambiará las ids de los lanelets, por lo tanto, si se quisiera cargar el mapa, ya sea para cambiar una etiqueta, corregir un error, etc, luego en el programa habría que cambiar a mano las "ids" de los lanelets que corresponderían a cada parada.

Además, encontramos el problema de que era fácil acceder a las etiquetas que conformaban el lanelet, pero no a sus miembros (esta parte del código no fue creada por nosotros), por lo que tampoco podríamos recorrer los lanelet buscando al miembro station, que a su vez es una relación y habría que acceder a sus datos internos.

Por lo que la opción más sencilla, que ya teníamos, además en marcha, era calcular la distancia euclídea entre los nodos de los lanelet comparándolos con los de origen y destino.

4.2.2 Descripción del algoritmo

Solo algunas funciones de este código son del proyecto de Raúl Rojas.

Este programa tiene como argumentos el nombre de la parada de origen y el nombre de la parada destino en la línea siguiente.

Dentro del programa se indica el mapa de donde obtenemos los datos, y según el nombre de la parada escrita en los argumentos, les corresponden unas coordenadas indicadas en el programa con funciones condicionales, por lo que si alguna parada es movida en el mapa habría que cambiarle las coordenadas en el código.

A continuación, obtendríamos los lanelet de origen y destino, a partir del nodo origen y el nodo destino.

Lo que se explica a continuación se aplica tanto al **nodo destino,** como al **nodo origen**.

Recorreremos todos los lanelet del mapa. Necesitaremos dos bucles, uno para el lado derecho de los lanelet del mapa y otro para el lado izquierdo del lanelet. Con lado derecho e izquierdo nos referimos a la polilínea etiquetada como "right" (derecha) o "left" (izquierda) del lanelet.

En cada bucle lo que se realiza es lo siguiente:

Del lanelet que recorramos guardaremos en un array los nodos del lado derecho (o del izquierdo según en qué bucle nos encontremos).

De cada nodo obtendremos su longitud y su latitud, cada una en una variable, para poder calcular la **distancia euclídea** con el nodo origen y el destino:

$$Distancia_{E.Origen,derecho} = \sqrt{\left(Latitud_{Origen} - Latitud_{Nodo\,derecho}\right)^2 + \left(Longitud_{Origen} - Longitud_{Nodo\,derecho}\right)^2}$$

Ecuación 4.2: Distancia euclídea con el nodo origen, lado derecho

$$Distancia_{E.Destino,derecho} = \sqrt{(Latitud_{Destino} - Latitud_{Nodo\,derecho})^2 + (Longitud_{Destino} - Longitud_{Nodo\,derecho})^2}$$

Ecuación 4.3: Distancia euclídea con el nodo destino, lado izquierdo

Vemos las ecuaciones utilizadas en Ecuación 4.2 y Ecuación 4.3.

Comparamos la distancia obtenida con una variable en la que iremos guardando las distancias más pequeñas obtenidas, por separado del origen y del destino, hasta que hayamos comparado con todos los nodos del lado derecho y guardemos el valor de la distancia más pequeña y del id al que pertenece el nodo con la distancia más pequeña.

A continuación, se realizaría lo mismo, pero con los nodos del lado izquierdo. También guardaríamos la distancia más reducida que hayamos obtenido individualmente para el origen y para el destino y la id de su correspondiente lanelet.

Después, se compara la distancia euclídea más pequeña obtenida del nodo origen con los nodos del lado derecho de los lanelets y con la distancia más pequeña obtenida para el nodo origen también, pero del lado izquierdo de los lanelets del mapa.

Según cual sea más pequeña de las dos, elegiríamos la id del lanelet correspondiente. Lo mismo con el nodo destino.

Como los datos del mapa están guardados en una variable de la clase LaneletMap, utilizaremos una función de esta clase para obtener la ruta más corta: **shortest_path.**

En esta función son necesarios dos argumentos, el lanelet origen y destino para la ruta. Los cuales ya tenemos.

La ruta viene dada con el siguiente formato:

<Lanelet X><Lanelet Y><Lanelet Z>…

(Siendo X, Y y Z ids de los lanelets que conforman la ruta en el orden a seguir.

Sabiendo los lanelets de la ruta, plasmaremos en un documento de texto, la ruta a seguir y las características más importantes de cada lanelet.

De cada lanelet mostraremos el tipo (“type” = “lanelet”), la velocidad límite (“speedlimit”), la capa a la que pertenece (“layer”), el rol que se le establece si lo tiene

(el único existente es el de rotonda (“roundabout”)), los nodos de la polilínea derecha del lanelet y los nodos de la polilínea izquierda.

También mostraremos sus elementos regulatorios, de los cuales mostraremos, las coordenadas y el identificador de la referencia (nodo etiquetado como “ref”), la maniobra que se realiza en este elemento regulatorio (etiquetada como “maneuver”) y las coordenadas y el identificador del nodo de activación (etiquetado como “activate_point”).

De esta manera tenemos en el mismo documento todos los datos que podríamos necesitar para simularlo.

A continuación, se muestra un diagrama de bloques del algoritmo.

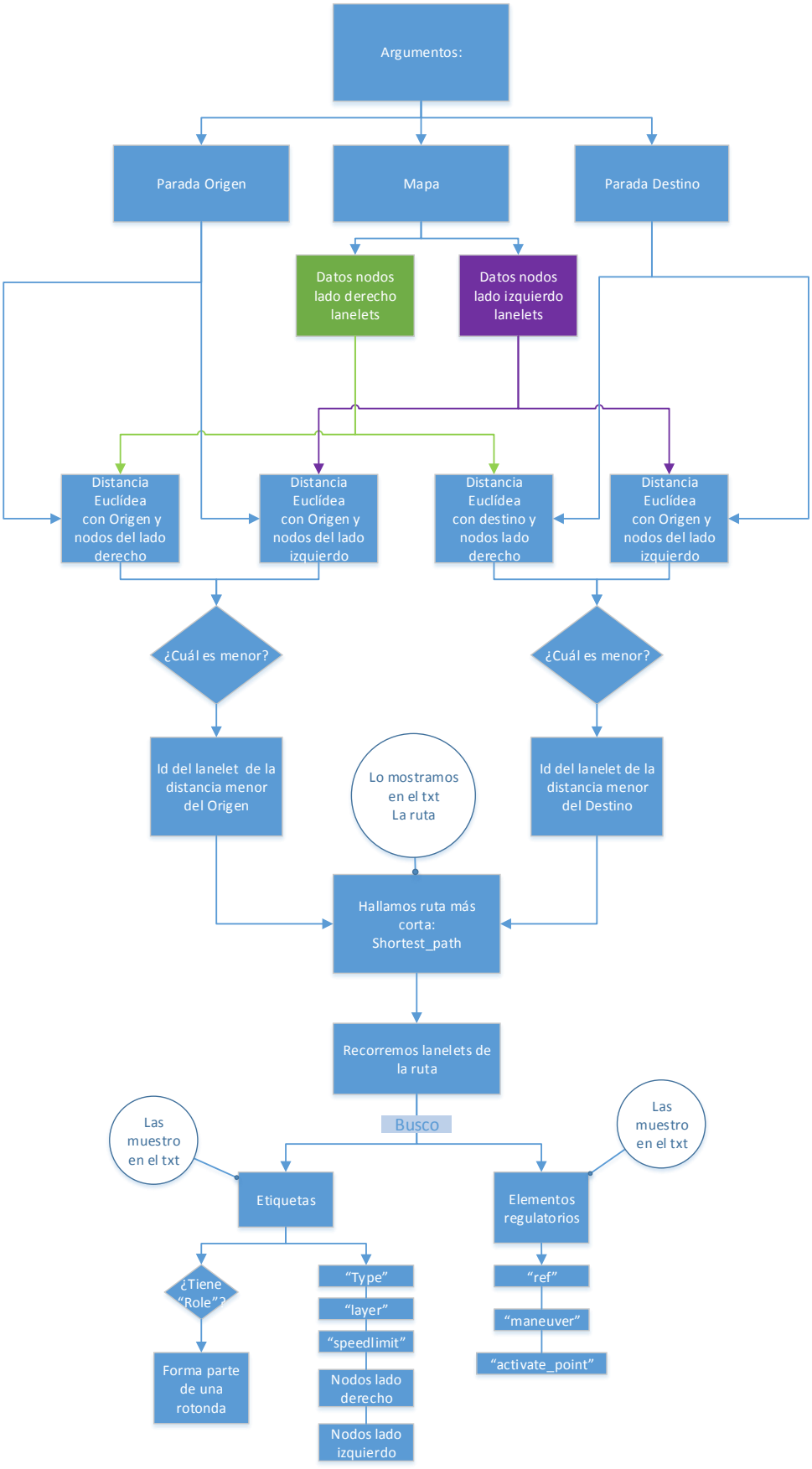


Figura 4.3: Diagrama algoritmo Enrutamiento.cpp

Capítulo 5 Simulación

Este Trabajo de fin de grado forma parte del proyecto SmartElderlyCar **[28]** realizado entre la Universidad de Alcalá (UAH) **[UAH]** y la Universidad de Vigo (UVIGO) **[UVIGO]** . Se utilizó el entorno de simulación que desarrollaron, SmartCar, del que hablaremos más adelante, para la obtención de las simulaciones en imágenes y vídeo del coche circulando por el mapa creado con la herramienta JOSM.

El simulador del proyecto es el **V-REP [VREP]** en el entorno **ROS [12] .** Se planteó utilizar el simulador Gazebo en vez del de VREP ya que, Gazebo ya soportaba de forma nativa ROS y los dos tenían un rendimiento muy parecido. Pero finalmente se eligió VREP por su facilidad para crear nuevos modelos y modificar los existentes.

ROS significa sistema operativo robótico, es de código abierto, un **framework [14]** para el desarrollo de software de robots que provee la funcionalidad de un sistema operativo en un **clúster [15]** variado. Proporciona un sistema de comunicación entre diferentes programas del sistema que conforma el proyecto que realizaron.

Como V-REP no soporta nativamente la comunicación con nodos ROS necesitan del *plugin RosInterface.*

V-REP permite importar formatos de mallas, por lo que necesitamos que el mapa sea un archivo **OBJ [16] .**Por lo que, nuestro mapa que está en formato OSM debe ser convertido a formato OBJ y para ello se utiliza la herramienta de código abierto también **OSM2World**.

Al importarlo establecemos el eje Z apuntando hacia arriba y en el código del archivo se puede ver el origen de coordenadas.

Aunque en la simulación las coordenadas están en WGS 84 (que son las del formato OSM), hay que pasarlas a coordenadas cartesianas (UTM). Para ello, como bien podemos leer en el paper de **[30]** , se utilizan las librerías implementadas en el paquete *ROS geodesy* y, restándole un desfase dado por la coordenada de origen del mapa, convirtiéndola previamente al sistema cartesiano (UTS).

Si una vez importado el entorno se han creado demasiados detalles, V-REP tiene opción de eliminar los detalles sobrantes.

La parte visual del coche que se puede ver en la simulación, y que podemos ver en la Figura 5.1, ha sido importada de un repositorio de CAD, dándole las formas adecuadas para que pudiera verse como un coche más real.

La parte con propiedades dinámicas está formada por cuboids (paralelepípedos rectos) a los que se les da una masa para simular la inercia del vehículo. Además, está dotado de cuatro ruedas con amortiguación, motor propulsor, frenos, dirección y sensores.

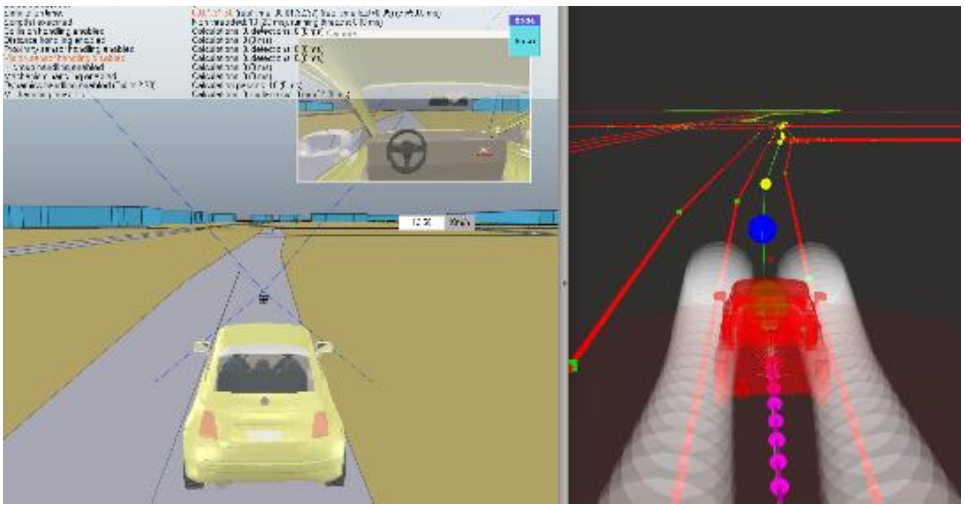


Figura 5.1: Imagen del coche de la simulación recorriendo los carriles del mapa

Para obtener una simulación más realista y que el movimiento del coche se parezca más a un coche real autónomo, hay que controlar su sistema de dirección, en el que hay que tener en cuenta que debe tener inercia y fricción como un volante real, por lo que se simuló en V-REP como un sistema dinámico, con un motor controlado por la posición. Internamente se controla en el V-REP con un controlador PID (controlador proporcional, integral y derivativo).

La dirección del coche es, entonces, como la de un coche real, la posición del volante determina el ángulo de dirección de las ruedas delanteras (mediante un modelo matemático que relaciona estas variables).

Para ajustar el modelo se procedió a mover el coche manualmente mediante un joystick, con un mando de la consola PlayStation PS3 y un volante con conector USB, pudiendo probar las velocidades y/o ángulos deseados.

También se crearon modelos de los sensores que iba a llevar el coche real para la percepción de su entorno (LiDAR **[LiDAR]** y cámara estéreo).

El módulo base para el funcionamiento del coche está implementado en un child script de V-REP y funciona como una interfaz para comunicar el sistema y el vehículo.

A partir de los comandos de velocidad (que pueden ser manuales o automáticos) obtiene la orientación y velocidad que deben de tener las ruedas delanteras. Y para poder girar las ruedas delanteras se obtienen el ángulo de giro que debe tener el volante y la velocidad lineal que queremos para el coche.

En principio V-REP no muestra el tiempo de simulación, pero mediante el plugin *RosInterface* y unas configuraciones propias solucionamos este problema.

5.1 Entorno de simulación

El entorno desarrollado para la simulación lo llamaron **SmartCar** y está formado por diversas capas:

- **Capa de interfaz:** el nivel más alto. Con las interfaces de usuario, donde se establecen por ejemplo los objetivos destino del coche.

- **Capa de control:** gestiona el mapa, los comportamientos, la percepción del entorno y la planificación.

- **Capa de servidor hardware/simulación:** contiene los controladores que manejan la capa de hardware/simulación.

- **Capa hardware/simulación:** capa más baja. Si nos referimos a capa hardware hablamos del coche real si no, es el simulado.

Con este simulador se pueden planificar rutas. Como ya tenemos el mapa de la zona a recorrer no utilizamos un planificador global de rutas, ya que con el mapa tenemos bien definidas las restricciones de movimiento y podemos utilizar un planificador de rutas que necesite las posiciones de inicio, meta y la GPS.

Como resultado obtenemos una lista de nodos que debe recorrer el coche para llegar a su destino.

Con esa lista de nodos trazamos una trayectoria a seguir mediante una lista de puntos de paso objetivo en el centro de los carriles. Podemos ver los puntos objetivo en la Figura 5.1, el siguiente punto objetivo al que se dirige se puede ver de color azul.

Para el seguimiento de la trayectoria se utiliza el **Pure Pursuit Controller [31]**. Es un algoritmo de seguimiento de rutas que utiliza tres elementos que podemos ver en la siguiente figura:

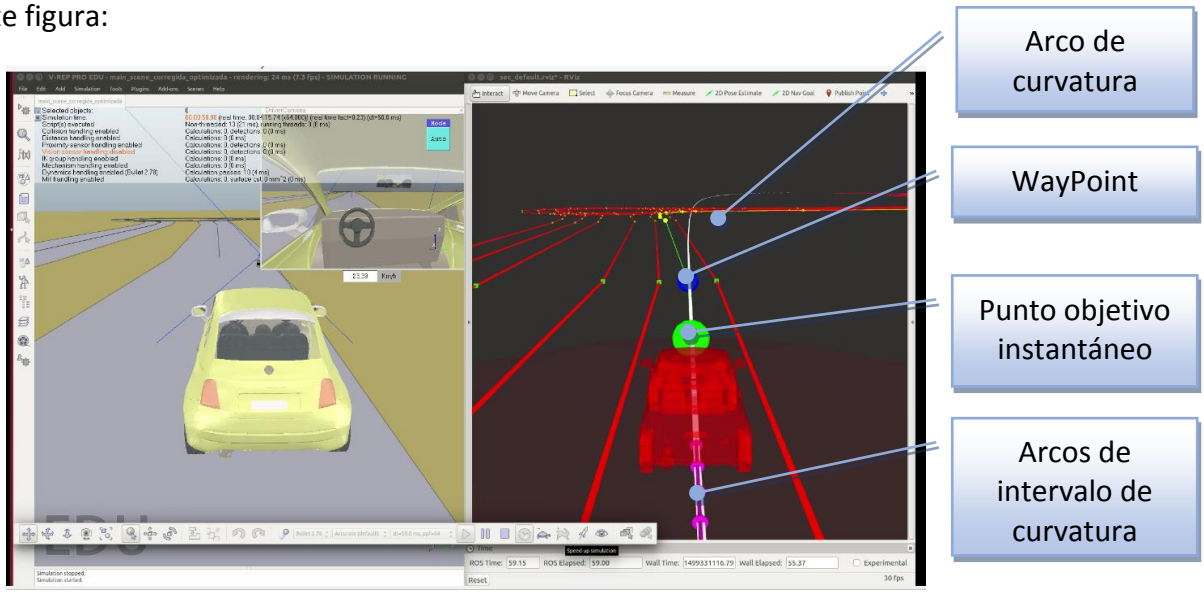


Figura 5.2: Elementos del entorno de simulación para el seguimiento de la trayectoria.

Y son:

- **Punto Lookahead**: de color rojo. Establece como de lejos debe "mirar" el coche con respecto a su situación actual para calcular los comandos de velocidad. Podemos verlo en el dibujo esquemático de la Figura 5.3.

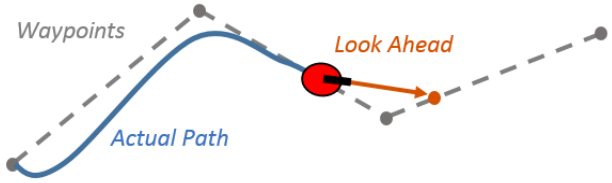


Figura 5.3: Puntos para el seguimiento de la trayectoria

- **Waypoint:** punto de color azul.indica el próximo waypoint o punto del camino al que se dirige.

- **Punto objetivo instantáneo**: de color verde. Está interpolado linealmente a partir de los waypoints.

- **Arco de curvatura:** es un arco de color blanco que indica el arco de curvatura instantáneo calculado por el pure pursuit para que el coche alcance el Punto objetivo instantáneo. Siempre tiene una amplitud de 45º.

- **Arcos de intervalo de curvatura:** son unos arcos de color fucsia que delimitan el intervalo de mejor curvatura posible a seguir a cada instante. Se pueden ver levemente en la Figura 5.4.

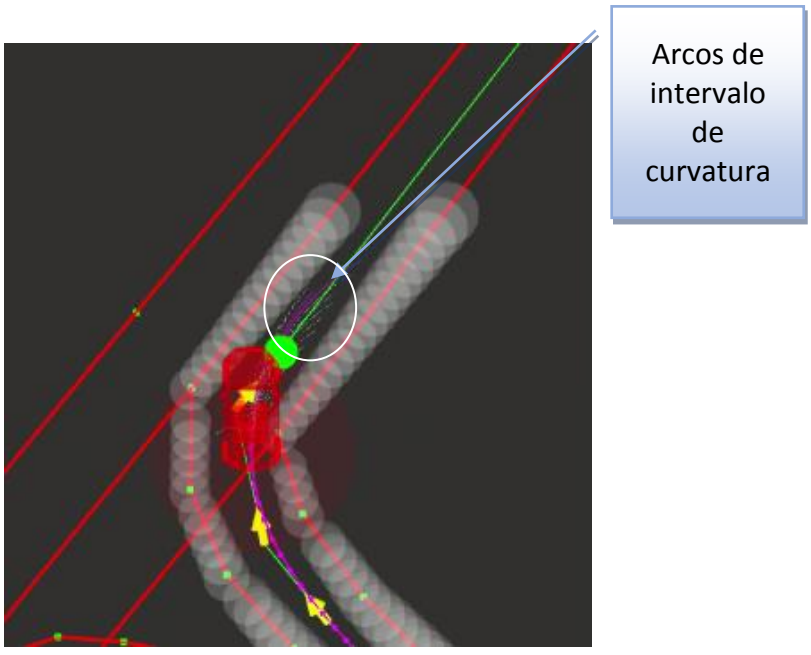


Figura 5.4: Arcos de intervalo de curvatura.

Además, tenemos dos tipos de navegación a la hora de seguir la trayectoria:

- **Navegación reactiva:** consiste en que el coche tenga en cuenta y reaccione ante los obstáculos. En nuestro caso no hay obstáculos como tal, no hay coches aparcados u objetos caídos en la carretera, pero están los límites que conforman los lanelets, que se pueden considerar "obstáculos virtuales", el coche debe mantenerse entre ellos. Utiliza el Curvature-Velocity Method (CVM) **[CVM]** , dibujando unos arcos blancos con las posibles trayectorias tangentes a los obstáculos que se pueden ver levemente delante del vehículo en la Figura 5.4.

- **Navegación no reactiva:** solo ejecuta el algoritmo de seguimiento, sin saber dónde está el borde del carril. No se sale del carril porque el punto lookahead se ajusta dinámicamente y la aceleración lateral se limita para que siga bien la trayectoria, ya que tampoco usa CVM.

A continuación, se hablará del comportamiento vial.

5.2 Comportamiento vial

El comportamiento del coche frente a cruces con preferencias, semáforos y demás señales de tráfico está implementado mediante **Redes de Petri[17]** con una herramienta de la Universidad de Vigo llamada: *RoboGraph*. Cada comportamiento se define por una o varias Redes de Petri.

En el caso de este trabajo los comportamientos que debe tener el vehículo en el tráfico se obtienen por la definición en el mapa de señales con una etiqueta sobre el comportamiento que tiene que llevar a cabo ("maneuver"), y las coordenadas sobre dónde pararse ("stop_line" y "ref") y dónde debe el coche empezar a tener en cuenta esa señal ("activate_point").

El comportamiento vial frente a obstáculos dinámicos o imprevistos no están contenidos en este trabajo.

5.3 Algoritmo de navegación

Utiliza dos archivos cpp: map_manager_node y map_manager_base.

En map_manager_node se realiza la llamada a las funciones, definidas en map_manager_base, necesarias para realizar el enrutamiento.

A la hora de obtener el enrutamiento utiliza unos algoritmos parecidos a los de enrutamiento.cpp.

Carga el mapa definido por lanelets y tiene de argumentos, la latitud y longitud del punto de origen y del punto de destino, recorre los lanelets y los guarda en un array de lanelets.

A continuación, esta parte del código es la que se diferencia con enrutamiento.cpp.

En enrutamiento.cpp calculábamos qué lanelet era más cercano a las coordenadas de origen y destino mediante la distancia euclídea a los otros nodos, pero no se tiene en cuenta que, si el lanelet es muy largo o no tiene nodos cerca de las coordenadas a investigar, la distancia a los lanelets calculada, puede dar lugar a que las coordenadas están más cerca de un lanelet que otro cuando no es así. Podemos verlo en la Figura 5.5.

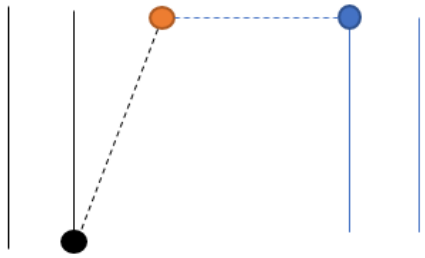


Figura 5.5: Distancias a los lanelets

Como se ve en la Figura 5.5, como se toman distancias a los nodos de los lanelets, la distancia al lanelet negro puede dar mayor que a la del lanelet azul a pesar de que el punto naranja está cerca del lanelet de color negro.

En cambio, en el simulador se hacen las proyecciones de las coordenadas dadas de forma perpendicular a los lanelets, como se ve en la Figura 5.6 y se guarda la distancia obtenida al lado izquierdo y derecho del lanelet.

Se suman esas dos distancias y la que sea más pequeña corresponde a la del lanelet más cercano.

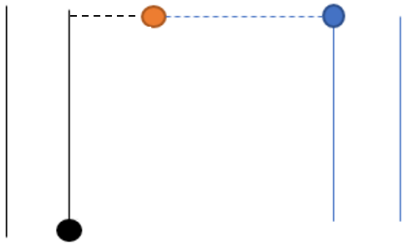


Figura 5.6: Proyecciones del algoritmo de navegación del simulador.

Cuando ya se tienen los lanelets a los que pertenecen los puntos origen y destino, se utiliza la función shortest_path, como en enrutamiento.cpp, para obtener la ruta más corta de lanelets.

En la Figura 5.6, podemos ver el diagrama de bloques del algoritmo de enrutamiento del simulador.

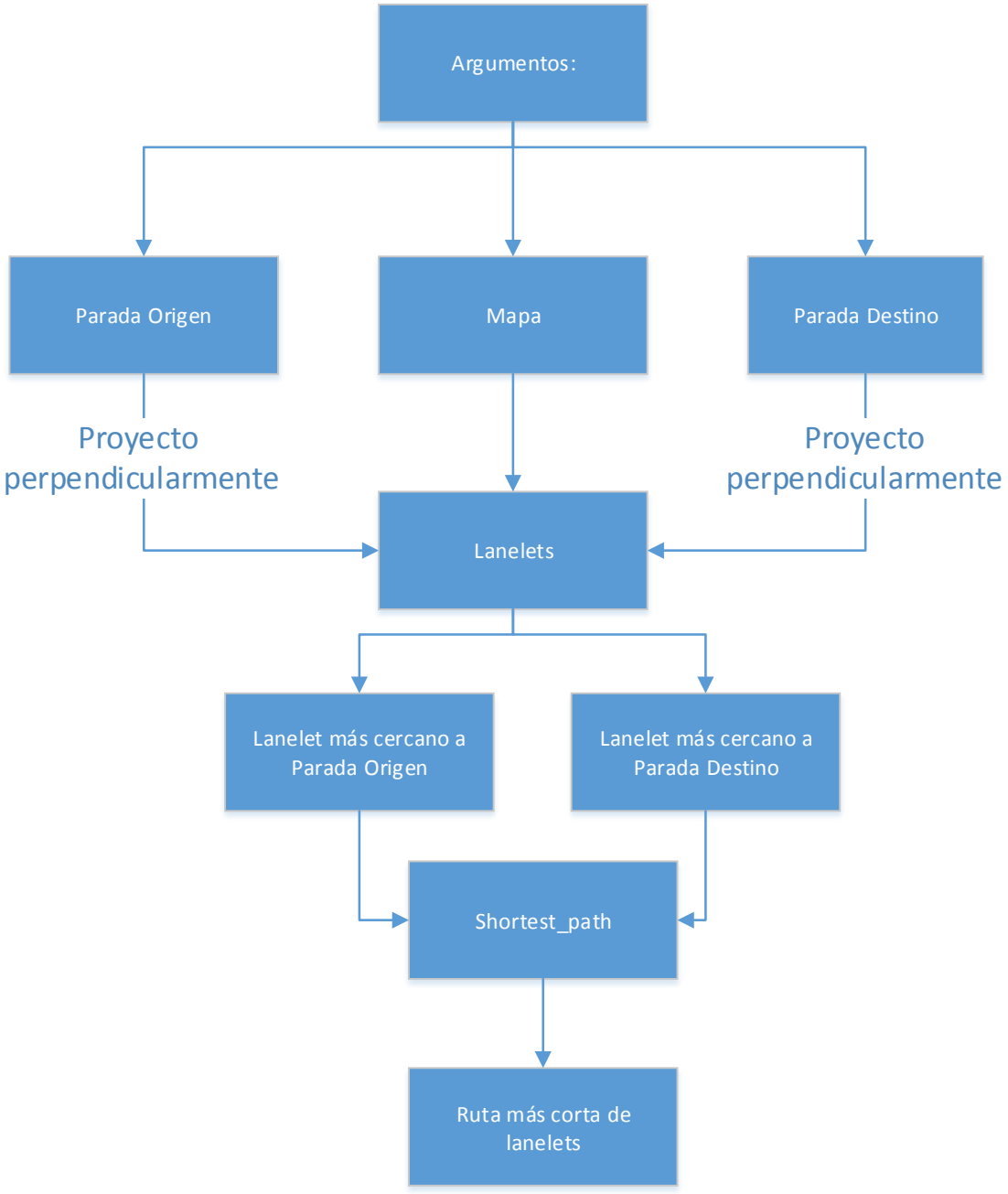


Figura 5.7: Diagrama del enrutamiento del simulador

Con este simulador pudimos demostrar la eficacia de los lanelets y el enrutamiento del vehículo a partir de ellos.

Capítulo 6 Resultados

A partir del mapa obtenido mediante la herramienta JOSM, formado por lanelets, y el simulador V-REP de la universidad de Vigo con su entorno SmartCar se han obtenido los siguientes resultados:

6.1 Mapa

Comenzamos presentando el mapa del entorno de trabajo.

Este mapa está en formato OSM ya que ha sido creado a partir de Open Street Map, tiene formato xml, en el que podemos ver claramente el árbol de nodos que conforman todos los carriles que recorren el campus.

La zona mapeada está al noroeste de Alcalá de Henares, delimitada por la autopista A-2 (Madrid-Barcelona) por el sur y la Biblioteca Nacional por el norte. Por el este, delimitado por el pueblo de Meco y por el oeste delimitado por la carretera comarcal M-121 (Alcalá de Henares – Meco).

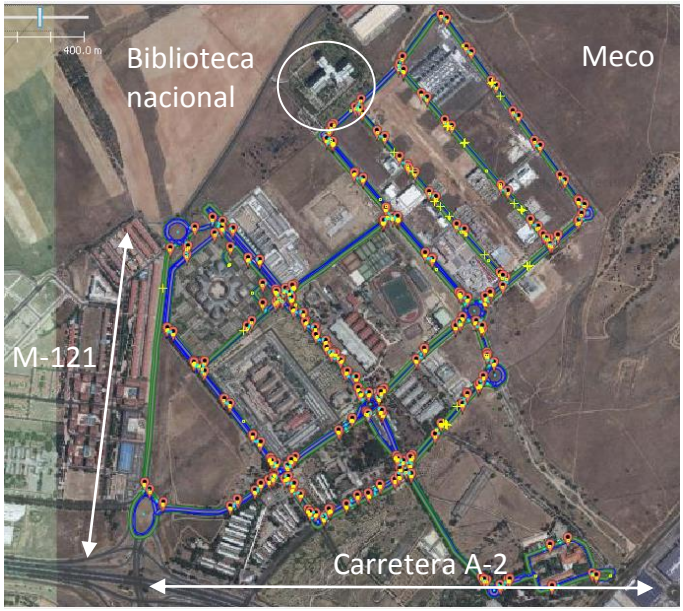


Figura 6.1: Mapa de JOSM con lanelets

Se pueden ver en las zonas mapeadas más de cerca a continuación, en la Figura 6.2.

Figura 6.2: Politécnica

Debajo de la Politécnica nos encontramos el Hospital Universitario Príncipe de Asturias y la facultad de Medicina. Y más abajo, se ve la facultad de Farmacia:

Figura 6.3: Hospital, facultad de Medicina y farmacia

A la derecha de Medicina encontramos el campus deportivo de la universidad y una gran zona con diversas empresas como Telefónica, y la Biblioteca nacional.

Figura 6.4: Campus deportivo, biblioteca y zona empresarial

Se puede ver en la Figura 6.1 y en las imágenes de cada zona del campus, los lanelets dibujados en el mapa.

Los lanelets son los carriles dibujados y etiquetados como tipo “lanelet”, los cuales nos permiten no solo marcar el camino, si no también delimitarlo y establecer las normas de tráfico que debe seguir el vehículo, como hemos visto en 0.

Las estructuras creadas en el mapa para establecer el comportamiento vial del coche son dos: el rol de rotonda y el de elementos regulatorios. De los que hablaremos en los siguientes apartados.

El mapa ya creado en Open Street Map, tiene una forma muy distinta, no solo porque se indican también edificios sino porque los caminos solo tienen una polilínea central, no como con los lanelets que tenemos una por cada límite del camino.

Figura 6.5: Mapa de Open Street Map

En la Figura 6.5, se puede ver el mapa que estaba subido a Open Street Map, con su polilínea por carril.

Diferencia entre los carriles de Open Street Map (OSM) y los lanelets de nuestro mapa:

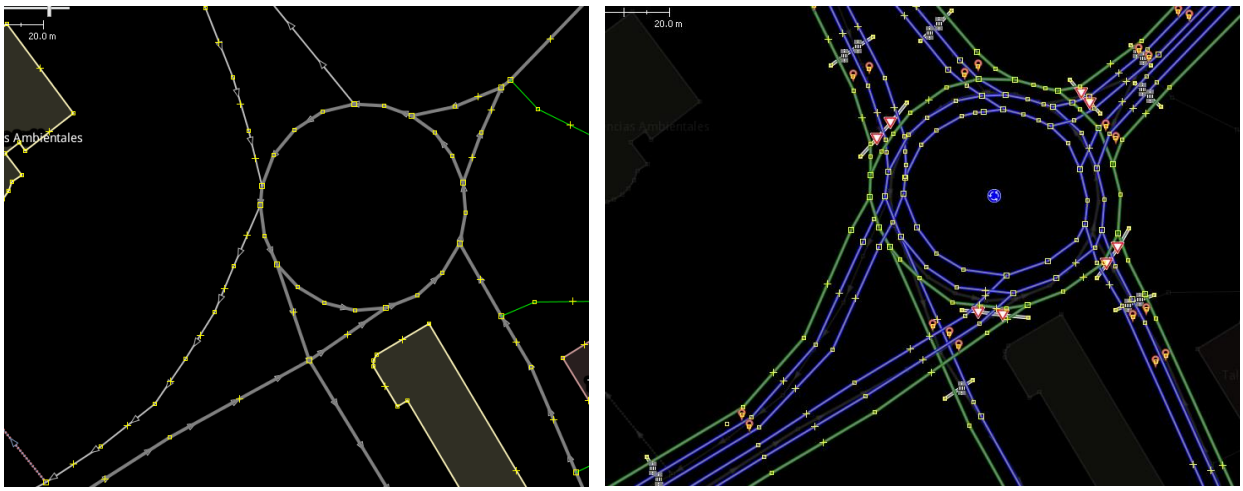


Figura 6.6: A la izquierda vemos los carriles de OSM y la derecha los de los lanelets de JOSM

La diferencia entre los dos mapas se ve muy clara en la Figura 6.6, a la izquierda vemos el de Open Street Map y a la derecha el de lanelets en JOSM, se pueden apreciar muy bien los carriles en el mapa de lanelets y en el de OSM, en el cual incluso si hay dos carriles solo utilizan una línea central.

6.1.1 Elementos regulatorios (type=regulatory_element)

Encontramos 3 tipos:

6.1.1.1 Pasos de cebra (maneuver = pedestrian_crossing)

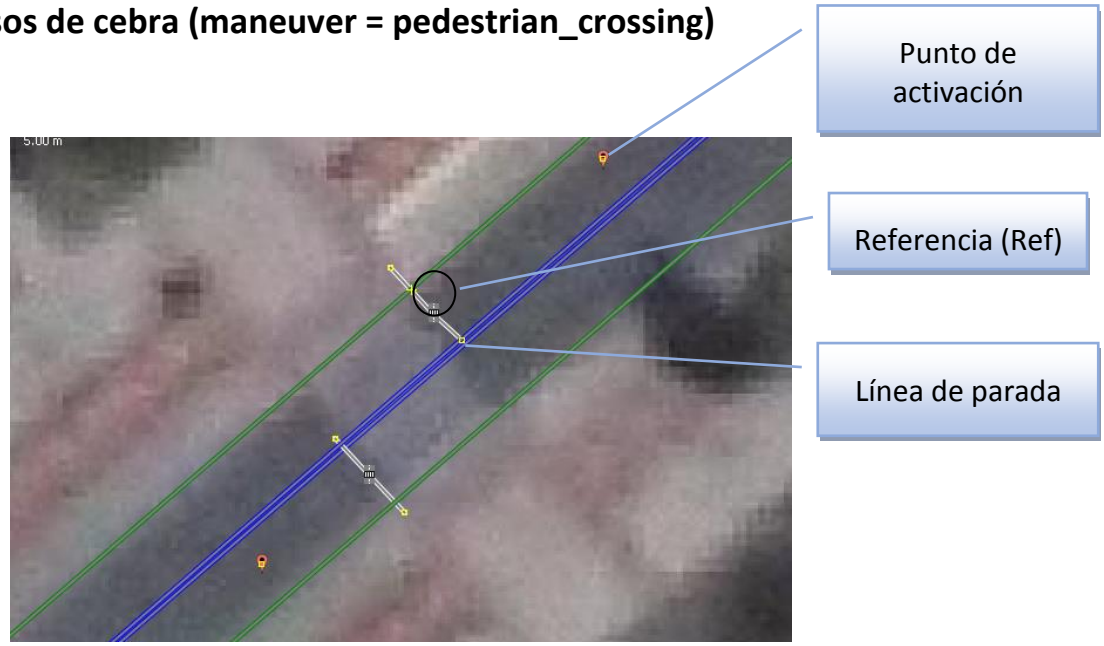


Figura 6.7: Paso de cebra en el mapa de JOSM

Podemos ver en la Figura 6.7 cómo es la representación de los pasos de cebra en los mapas.

También podemos ver que se crea para cada sentido y cada carril (ya que independientemente del sentido debes respetar los pasos de cebra) los tres elementos de los que ya hablamos anteriormente en el Capítulo 1.

Los parámetros del elemento regulatorio que podemos ver en el mapa son:

- **Stop_line**: La línea de parada, línea que no puede sobrepasar el coche.

- **Activate_point:** Punto en el cual el coche es avisado de la existencia de este elemento regulatorio.

- **Ref**: Está en la línea de parada, sujeta el símbolo que representa a este elemento regulatorio. Con el plug in que introdujimos, JOSM nos permite establecer si tiene el paso de cebra pintado o no, si tiene señal o no y si esta lleva luces o no.

En el xml esta relación se define de esta manera:

```
<relation id='-40945' action='modify' visible='true'>
  <member type='node' ref='-37783' role='ref' />
  <member type='way' ref='-39529' role='stop_line' />
  <member type='node' ref='-37791' role='activate_point' />
  <tag k='layer' v='uah_lanelets' />
  <tag k='maneuver' v='pedestrian_crossing' />
  <tag k='type' v='regulatory_element' />
</relation>
```

6.1.1.2 Ceda el paso (maneuver = give_way)

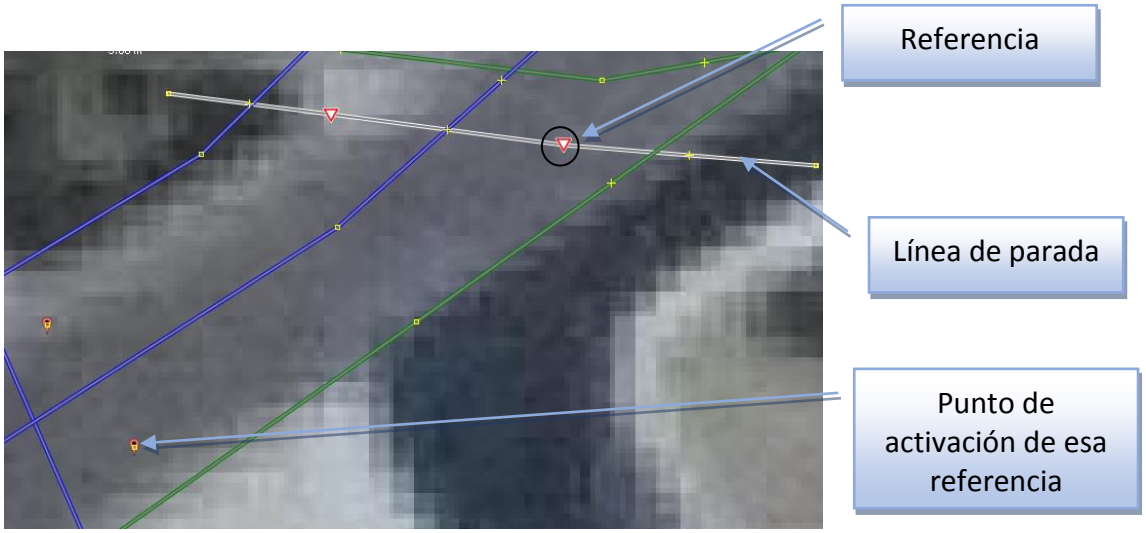


Figura 6.8: Ceda el paso en JOSM

En la Figura 6.8 podemos ver la representación de una ceda el paso en el mapa.

Creamos un ceda el paso por cada carril en el que sea necesario. No ocurre como en los pasos de cebra, no hay que ponerlos en todos los carriles en todos los sentidos.

Los ceda el paso se establecen al entrar a las rotondas, siempre que se haya una señal física y si no hay señal física, cada vez que nos encontremos en un cruce, hay que ceder el paso al vehículo que provenga de nuestra derecha.

También está formado por los mismos elementos que el paso de cebra:

- ✓ **Stop_line**
- ✓ **Activate_point**
- ✓ **Ref**: Se encuentra, en el xml, de esta manera:

```
<node id='-36685' action='modify' visible='true' lat='40.50543281759' lon='-3.33629690782'>
  <tag k='highway' v='give_way' />
  <tag k='traffic_sign' v='ES:r1' />
</node>
```

En el xml esta relación se define de esta manera:

```
<relation id='-39565' action='modify' visible='true'>
  <member type='node' ref='-35191' role='activate_point' />
  <member type='way' ref='-37875' role='stop_line' />
  <member type='node' ref='-30921' role='ref' />
  <tag k='layer' v='uah_lanelets' />
  <tag k='maneuver' v='give_way' />
  <tag k='type' v='regulatory_element' />
</relation>
```

6.1.1.3 Stop (maneuver=stop)

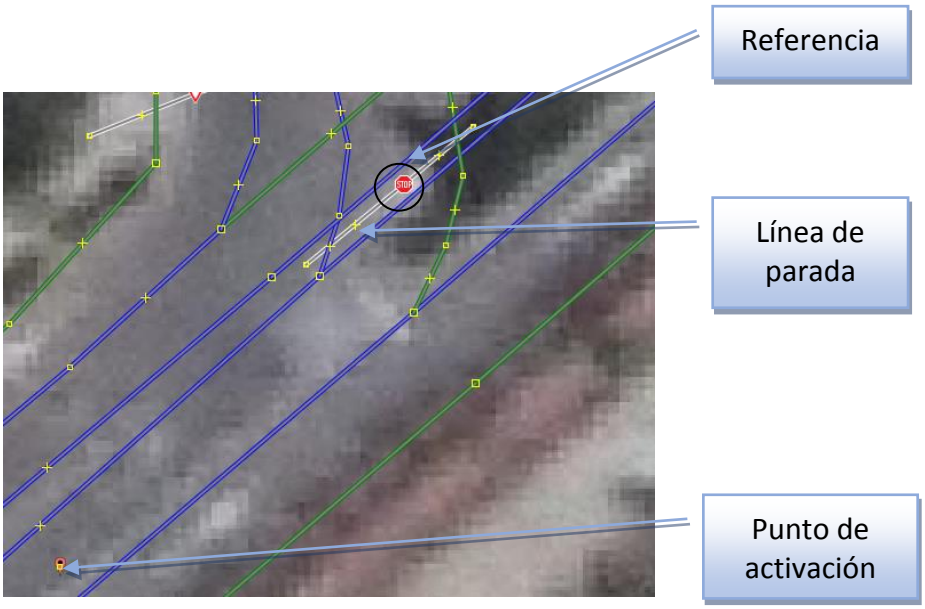


Figura 6.9: STOP en JOSM

En la Figura 6.9 podemos ver la representación de un Stop en el mapa.

En este caso, se crea solo en los carriles donde esté indicado el STOP en forma de señal física y/o pintada en el suelo.

También está formado por los mismos elementos que el paso de cebra y el ceda el paso:

- ✓ **Stop_line**
- ✓ **Activate_point**
- ✓ **Ref**. La estructura de este nodo se puede ver en el xml, de esta manera:

```
<node id='-35265' action='modify' visible='true' lat='40.51438247837' lon='-3.33778321371'>
  <tag k='highway' v='stop' />
  <tag k='traffic_sign' v='ES:r2' />
</node>
```

En el xml esta relación se define de esta manera:

```
<relation id='-40179' action='modify' visible='true'>
  <member type='way' ref='-39027' role='stop_line' />
  <member type='node' ref='-35267' role='activate_point' />
  <member type='node' ref='-35265' role='ref' />
  <tag k='layer' v='uah_lanelets' />
  <tag k='maneuver' v='stop' />
  <tag k='type' v='regulatory_element' />
</relation>
```

6.1.1.4 Rotonda (Rol=roundabout)

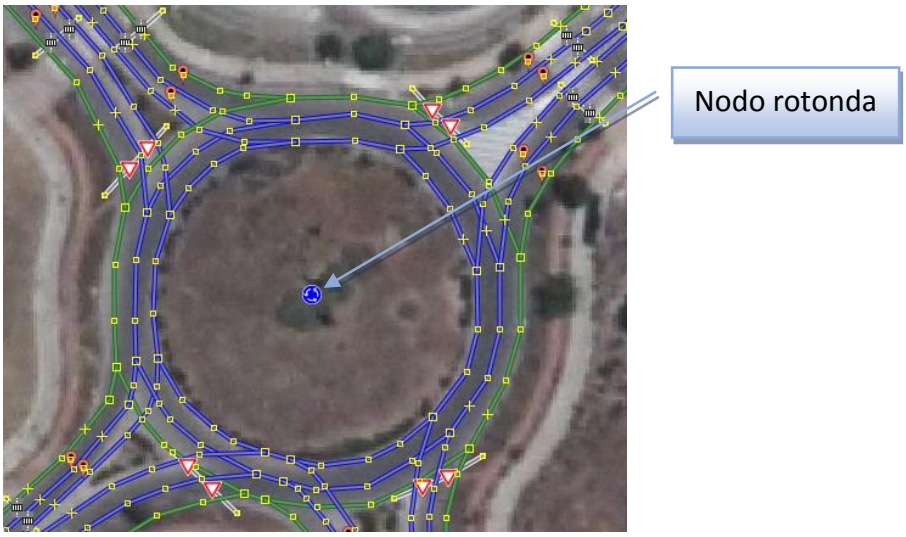


Figura6.10: Rotonda en JOSM

En la Figura6.10, podemos ver la representación de una rotonda en el mapa.

En cada rotonda hay un nodo central con la señal de rotonda.

Todos los lanelets que formen parte de una rotonda contendrán entre sus miembros el nodo central de su rotonda y además una etiqueta que establezca que forman parte de una rotonda, les asigna un rol.

Esto se debe a que en principio se pensó en buscar ese nodo en los lanelets, para enseñar en el documento del que hablaremos en el segundo apartado de este capítulo, que ese lanelet forma parte de una rotonda. Pero era mucho más sencillo acceder a las etiquetas, así que, todos los lanelets que conformen rotondas contendrán los dos elementos; una etiqueta y el nodo central de la rotonda.

En el xml esos nodos se encuentran de esta manera:

```
<node id='-35657' action='modify' visible='true' lat='40.51731430083' lon='-3.34582955873'>
  <tag k='junction' v='roundabout' />
  <tag k='traffic_sign' v='ES:r402' />
</node>
```

Y en la relación:

```
<relation id='-39941' action='modify' visible='true'>
  <member type='node' ref='-35651' role='roundabout' />
  <member type='way' ref='-38671' role='right' />
  <member type='way' ref='-38661' role='left' />
  <tag k='Role' v='roundabout' />
  <tag k='layer' v='uah_lanelets' />
  <tag k='speedlimit' v='40' />
  <tag k='type' v='lanelet' />
</relation>
```

6.1.1.5 Semáforos (maneuver = Traffic_lights)

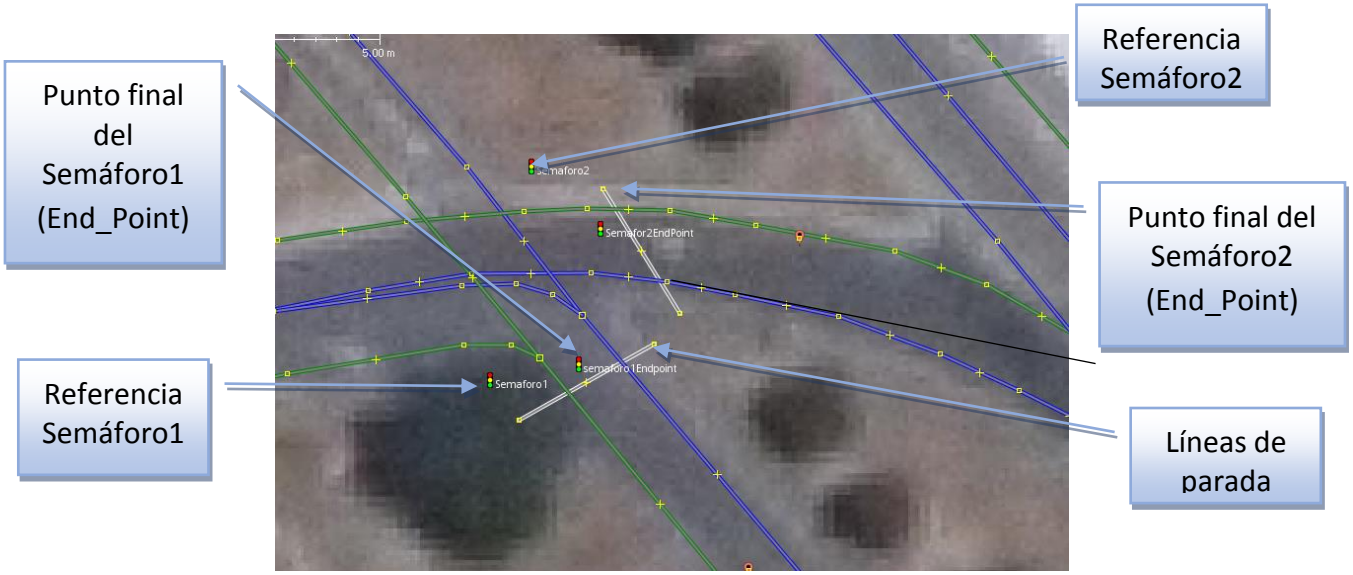


Figura 6.1: Semáforos en JOSM. Semáforo1 es la referencia del semáforo de su carril, pero también está incluido como referencia en el elemento regulatorio del Semáforo2, y viceversa.

En ninguna parte del campus hay semáforos físicos, pero decidimos incluir un ejemplo en un cruce de carreteras del mapa, cerca de la Politécnica, para probar el comportamiento del coche con ellos, para futuros proyectos.

Está formado por los siguientes elementos:

- ✓ **Stop_line**
- ✓ **Activate_point**
- ✓ **End_Point**. En el xml se puede ver de esta manera:

```
<node id='-35243' action='modify' visible='true' lat='40.51486695811' lon='-3.34958265689'>
  <tag k='crossing' v='no' />
  <tag k='highway' v='traffic_signals' />
  <tag k='name' v='semaforo1Endpoint' />
</node>
```

- **Ref**: tendrá dos referencias; el propio semáforo del carril y el otro semáforo con el que se tiene que coordinar. Si se diera el caso de un cruce con más de dos semáforos, tendría de referencia todos los demás semáforos con los que debe coordinarse. La estructura de este nodo se puede ver en el xml, de esta manera:

```
<node id='-35247' action='modify' visible='true' lat='40.51486004743' lon='-3.34963214774'>
  <tag k='crossing' v='no' />
  <tag k='highway' v='traffic_signals' />
  <tag k='name' v='Semaforo1' />
 </node>
```

La relación se define en el xml de esta manera:

```
<relation id='-40171' action='modify' visible='true'>
  <member type='node' ref='-35249' role='ref' />
  <member type='way' ref='-39023' role='stop_line' />
  <member type='node' ref='-35243' role='end_point' />
  <member type='node' ref='-35247' role='ref' />
  <member type='node' ref='-35239' role='activate_point' />
  <tag k='layer' v='uah_lanelets' />
  <tag k='maneuver' v='Traffic_lights' />
  <tag k='type' v='regulatory_element' />
</relation>
```

6.1.2 Estaciones(type=station)

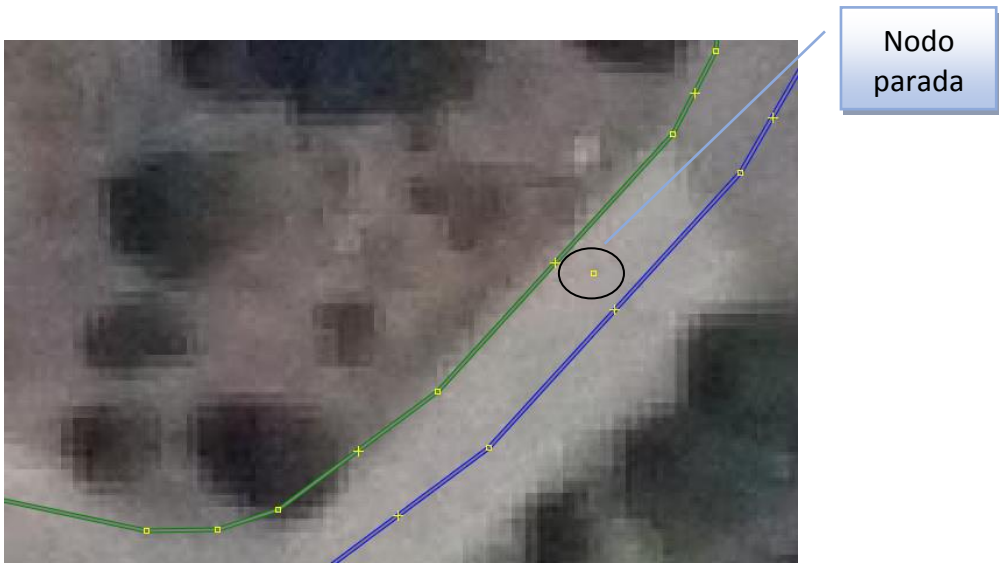


Figura 6.11: Parada del Apeadero

Podemos ver en la Figura 6.11. cómo se ven las estaciones en el mapa.

Son aquellos nodos que formen parte de la relación Station.

Esta relación se compone de los siguientes elementos:

- El nodo de la parada.
- Etiqueta Name, para establecer el nombre de la estación.

En el xml se puede ver de esta manera:

```
<relation id='-40357' action='modify' visible='true'>
  <member type='node' ref='-35911' role='ref' />
  <tag k='name' v='Politecnica' />
  <tag k='type' v='station' />
</relation>
```

6.2 Documento

Como ya explicamos en el capítulo de los Algoritmos, creamos un programa para obtener los datos más importantes en un documento de texto. Se puede ver una parte del documento en la Figura 6.12.

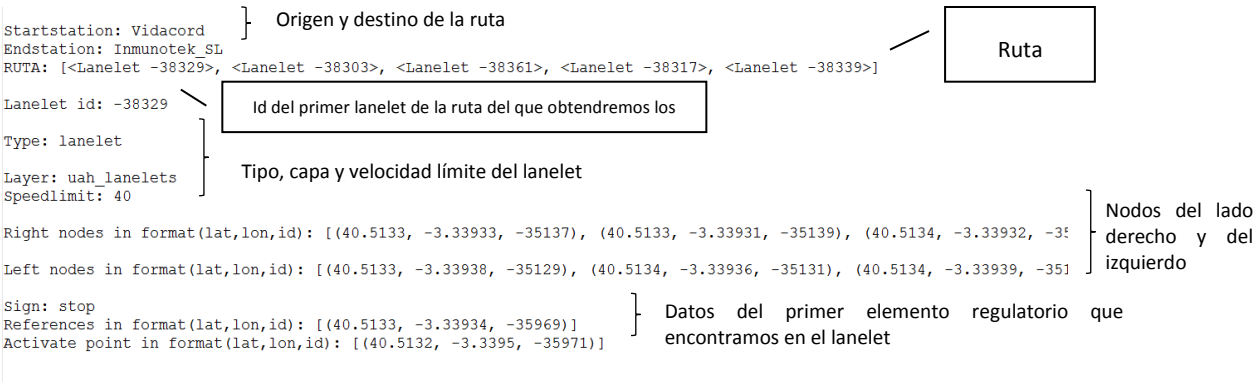


Figura 6.12: Parte del documento de texto que obtenemos al establecer una ruta

El documento incluye los siguientes parámetros:

- Los argumentos de la ruta. Ejemplo
  Origen: Politecnica
  Destino: Medicina

- La ruta de lanelet. En el siguiente formato:

  <Lanelet -35260><Lanelet -6598>…

- De los lanelet:
  - Tipo
  - Velocidad límite
  - Capa
  - Rol
  - Nodos derecha e izquierda (coordenadas e identificador)
- De los elementos regulatorios de los lanelet
  - Tipo
  - Referencias (coordenadas e identificador)
  - Maniobra
  - Punto de activación (coordenadas e identificador)

Con este documento tenemos agrupados en el mismo sitio todos los parámetros importantes del enrutamiento.

6.3 Simulación

En el Capítulo 5 queda explicado el funcionamiento del simulador V-REP utilizado por la Universidad de Vigo para probar un coche circulando por el mapa creado de la Universidad de Alcalá.

Se han simulado diversas rutas y se ha obtenido un vídeo por cada una de ellas, de los que a continuación se mostrarán imágenes, en los que podemos ver al coche circulando por el campus gracias a los lanelets, demostrando así su eficacia.

La circulación del coche por el mapa podía tener navegación reactiva o no como explicamos en el apartado 5.1. Decidimos que fuera no reactiva ya que suponía menos tiempo de simulación, ya que se tienen muchos menos parámetros en cuenta.

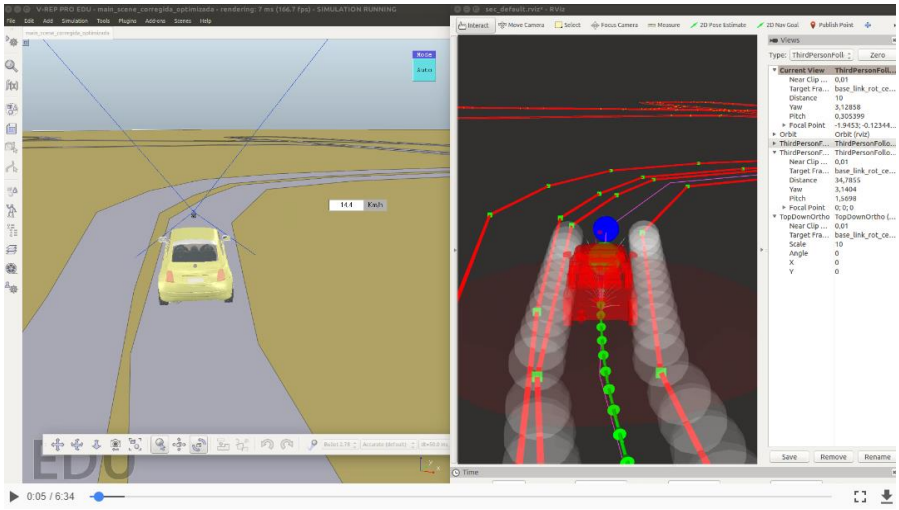


Figura 6.13: Navegación no reactiva

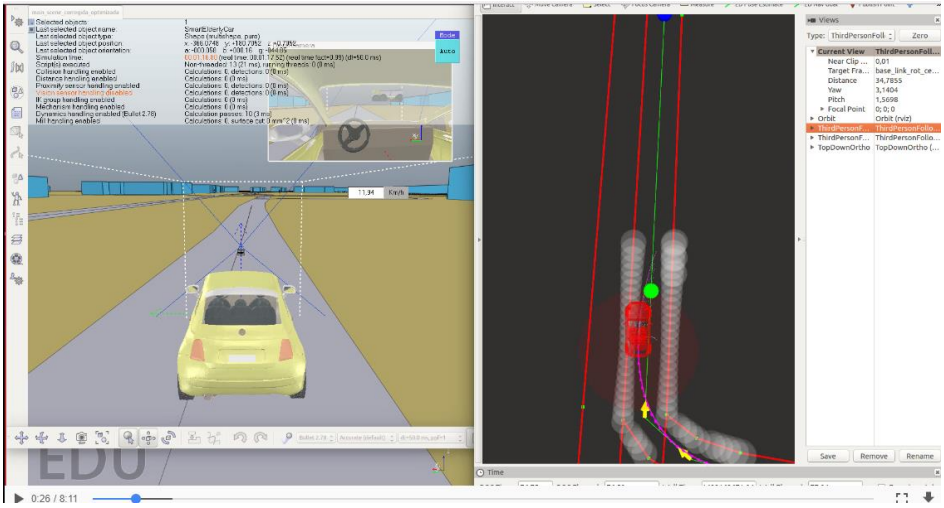


Figura 6.14: Navegación reactiva

Como podemos ver en las Figura 6.13 y Figura 6.14, a primera vista parece que apenas no hay diferencia, pero en la Figura 6.14, se observan parámetros en la pantalla que no están en la imagen de la no reactiva, ya que como ya explicamos en el apartado 5.1, en la reactiva, tiene en cuenta os límites de los lanelets y se mueve calculando curvaturas y velocidades con CVM. Mientras que en la no reactiva solamente sigue los puntos objetivo en orden, a una velocidad moderada, y por ello no se sale de los límites establecidos.

Las rutas de las que hemos obtenido simulación son las siguientes:

Origen	Destino
PolitecnicaStart	Apeadero
Apeadero	PolitecnicaEnd
Biología	Hospital
Residencia_Universitaria	CDG_Telefonica
Biblioteca_Nacional_de_España	HospitalUniversitario_PrincipedeAsturias
Farmacia	Politécnica

Tabla 6.1: Ejemplos de rutas

A continuación, mostraremos imágenes de distintas rutas, incluyendo la lista de lanelets que recorren con las ids de la última versión del mapa.

6.3.1 Ruta: PolitécnicaStart → Apeadero

Las estaciones y la ruta de lanelets seguida es:

Startstation: PolitecnicaStart

Endstation: Apeadero

RUTA: [<Lanelet -51855>, <Lanelet -51851>, <Lanelet -51837>, <Lanelet -51051>, <Lanelet -51221>, <Lanelet -51087>, <Lanelet -51083>, <Lanelet -51085>, <Lanelet -51217>, <Lanelet -51069>, <Lanelet -51139>, <Lanelet -51149>, <Lanelet -51203>, <Lanelet -51205>, <Lanelet -51451>, <Lanelet -51457>, <Lanelet -51475>, <Lanelet -51561>, <Lanelet -51531>, <Lanelet -51533>]

La Figura 6.15 muestra una imagen del vehículo recorriendo los lanelets de esta ruta:

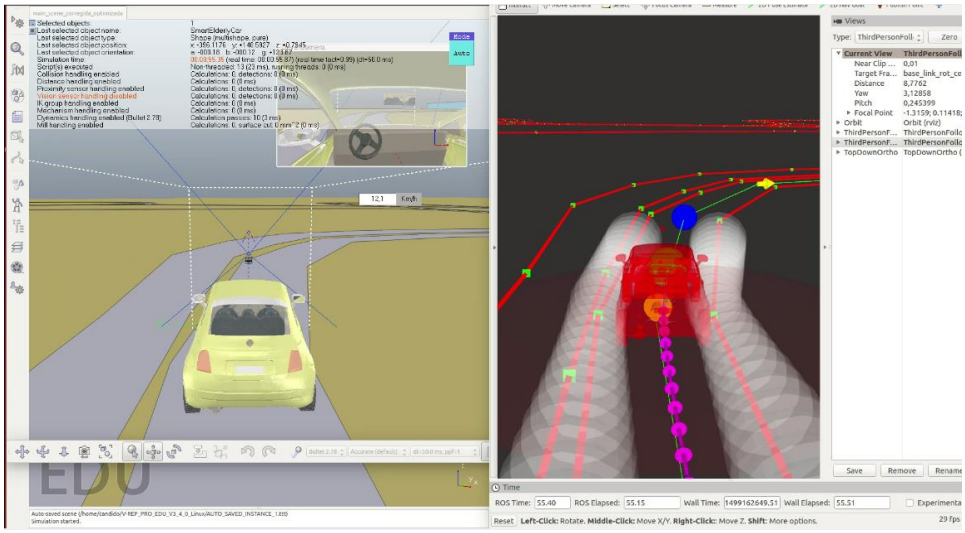


Figura 6.15: Vista posterior en la ruta Politecnica-Apeadero

Se puede ver más de cerca las líneas de curvatura, de color fucsia, los puntos objetivo de color azul y la ruta a seguir en verde.

En la siguiente Figura 6.16 podemos ver al vehículo, visto desde arriba, en una de las rotondas de la ruta. Se pueden ver mejor los puntos objetivo, de color azul.

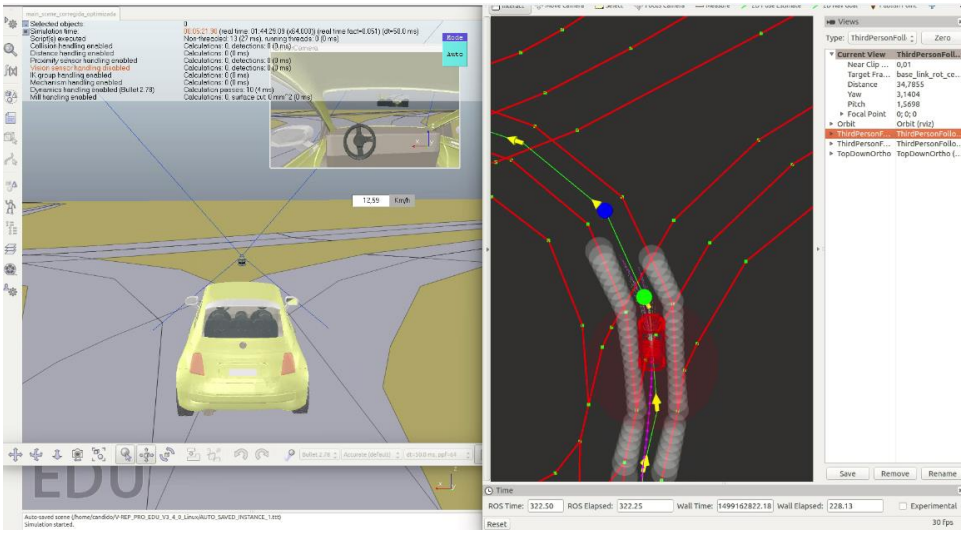


Figura 6.16: Vista desde arriba recorriendo la ruta PolitecnicaStart-Apeadero

En la Figura 6.17, vemos la ruta que ha seguido el vehículo a lo largo de todo el campus, mediante el simulador.

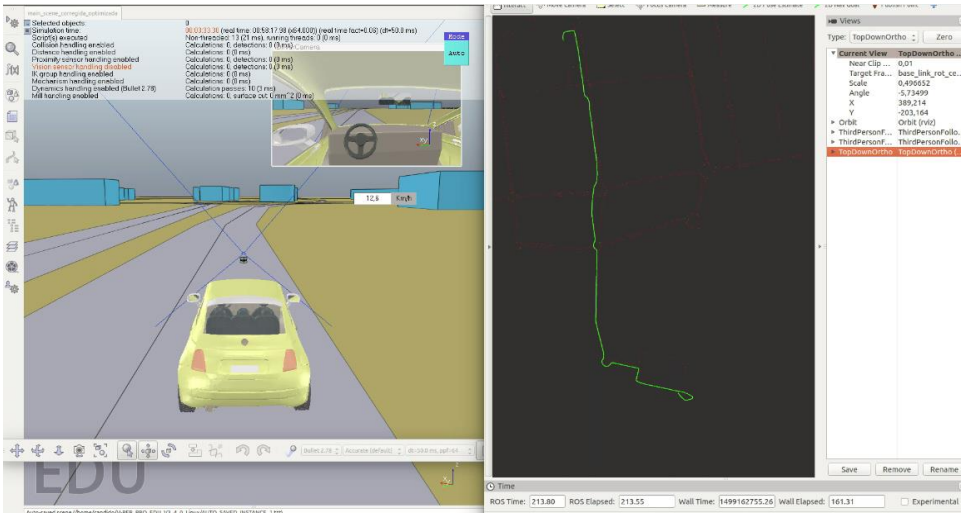


Figura 6.17: Vista de la ruta por el simulador

Como no se ve el mapa del campus de fondo en la Figura 6.17, en el simulador, podemos ver la ruta seguida más claramente en la Figura 6.18.

Figura 6.18: Ruta PolitecnicaStart-Apeadero en el mapa de JOSM

Se puede ver que efectivamente, ha escogido la ruta más corta.

Esta ruta ha sido simulada usando la navegación reactiva, cuyo tiempo de proceso es mayor que la no reactiva debido a que no tiene que ir calculando los obstáculos que aparecen en la ruta para definir su camino y su velocidad. En este caso la duración de la ruta es de 8 minutos y 11 segundos y se recoge en un vídeo obtenido desde el simulador

6.3.2 Ruta: Apeadero →PolitecnicaEnd

La ruta de lanelets seguida para este caso y con los ids de la última versión del mapa es:

Startstation: Apeadero

Endstation: PolitecnicaEnd

RUTA: [<Lanelet -51533>, <Lanelet -51535>, <Lanelet -51529>, <Lanelet -51543>, <Lanelet -51547>, <Lanelet -51549>, <Lanelet -51527>, <Lanelet -51467>, <Lanelet -51463>, <Lanelet -51459>, <Lanelet -51445>, <Lanelet -51451>, <Lanelet -51477>, <Lanelet -51511>, <Lanelet -51503>, <Lanelet -51495>, <Lanelet -51479>, <Lanelet -51135>, <Lanelet -51125>, <Lanelet -51099>, <Lanelet -51101>, <Lanelet -51185>, <Lanelet -51073>, <Lanelet -52027>, <Lanelet -51123>, <Lanelet -51045>, <Lanelet -51047>, <Lanelet -51049>, <Lanelet -51193>, <Lanelet -51225>, <Lanelet -51843>, <Lanelet -51853>]

En la Figura 6.19, podemos ver desde atrás cómo circula el coche por la ruta de este caso en la salida del Apeadero, con esta curva tan cerrada podemos ver que la trayectoria de puntos objetivo (la trayectoria verde de la Figura 6.19) es menos suave que la llevada a cabo por el vehículo.

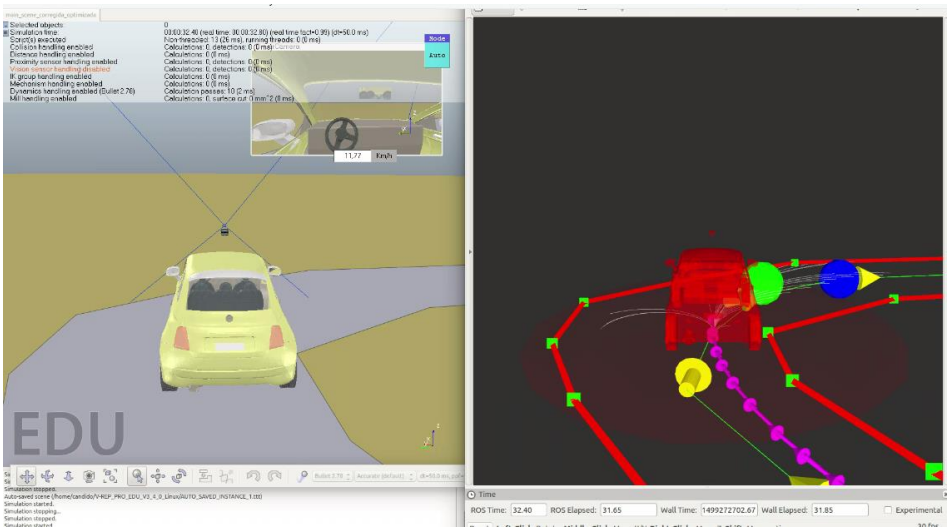


Figura 6.19: Vista posterior del vehículo en la ruta del Apeadero-PolitecnicaEnd

En la siguiente imagen, en la Figura 6.20, vemos desde arriba el coche en una de las rotondas.

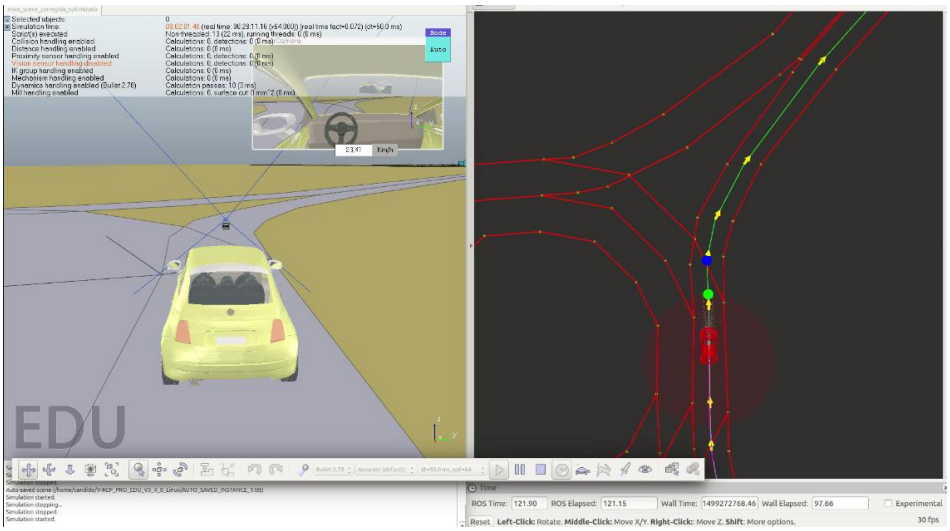


Figura 6.20: Vista desde arriba en la ruta del Apeadero-PolitecnicaEnd

La ruta en el simulador la podemos ver en la Figura 6.21.

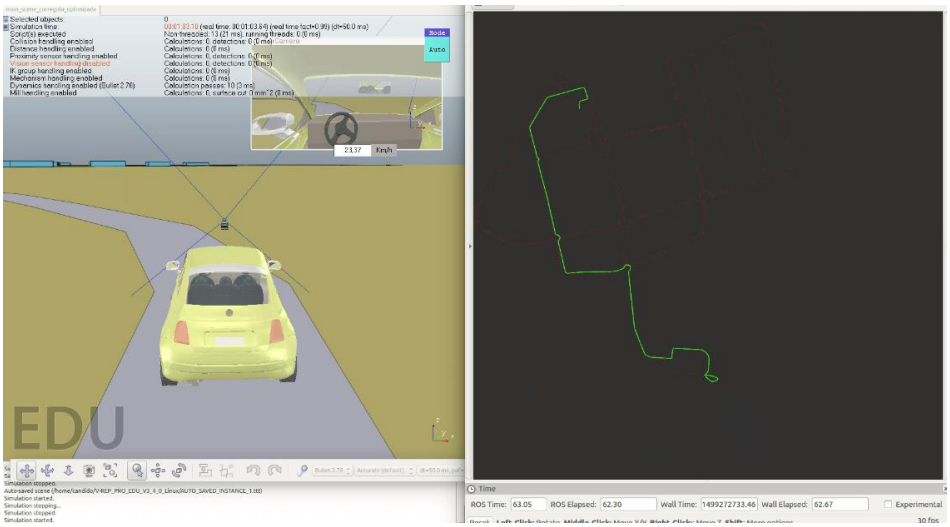


Figura 6.21: Ruta del simulador Apeadero-PolitecnicaEnd

No se ve el mapa universitario de fondo, así que, en la siguiente figura, podemos ver la ruta que vemos en el simulador, encima del mapa universitario:

Figura 6.22: Ruta Apeadero-PolitecnicaEnd en JOSM

Esta ruta la obtuvimos sin navegación reactiva, como las siguientes, por lo que los vídeos del simulador grabando la trayectoria seguida, son más cortos.

El video de este caso duró 4 minutos con 26 segundos.

Como se observa, la simulación no reactiva puede hacer que la simulación dure la mitad de tiempo a pesar de ser rutas muy parecidas en distancia, por las causas comentadas en el apartado anterior.

Aunque a primera vista parezca la misma parada PolitecnicaStart y PolitecnicaEnd, no lo son, están al lado, pero no son el mismo punto. Lo podemos ver en la figura 9.14

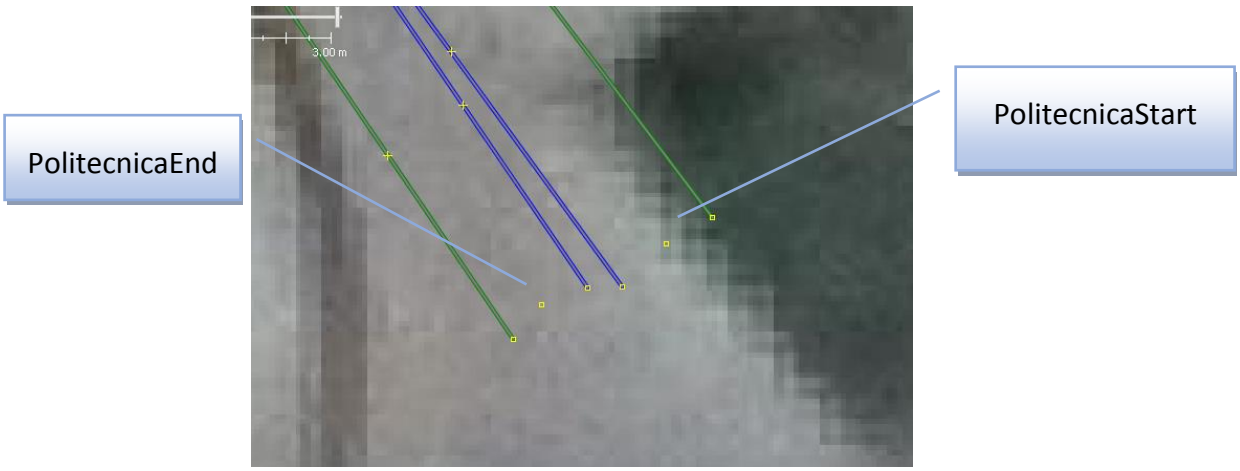


Figura 6.23: Paradas PolitecnicaStart y PolitecnicaEnd

6.3.3 Ruta: Biologia → HospitalUniversitario_PrincipedeAsturias

La ruta de lanelets seguidas, junto con las paradas, son:

- ✓ Startstation: Biologia
- ✓ Endstation: Hospital

Nombre de destino incorrecto

RUTA: [<Lanelet -51539>, <Lanelet -51537>, <Lanelet -51529>, <Lanelet -51543>, <Lanelet -51547>, <Lanelet -51549>, <Lanelet -51553>, <Lanelet -51561>]

A continuación, mostramos una vista posterior del vehículo, donde vemos más de cerca como toma las curvas:

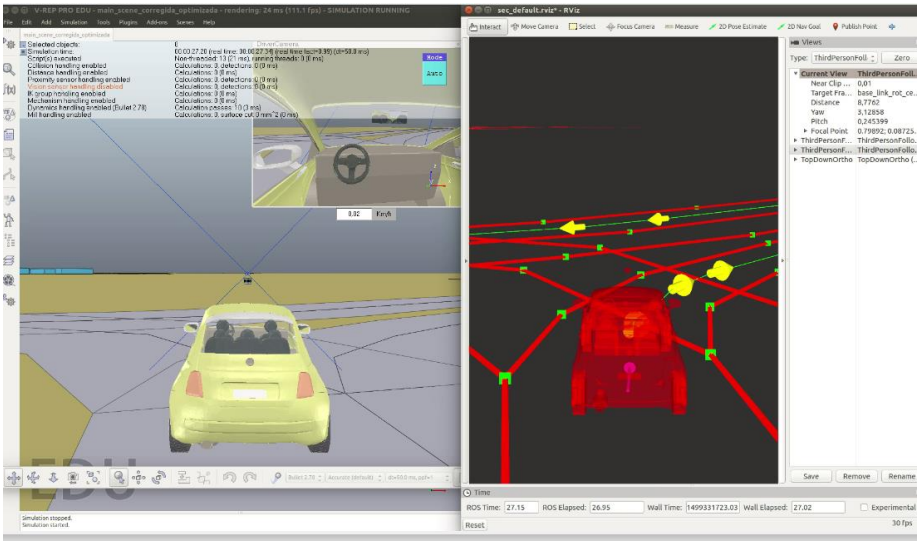


Figura 6.24: Vista posterior del vehículo recorriendo la ruta Biología-Hospital

En la siguiente Figura 6.25, vemos una vista desde arriba cruzando una de las rotondas de la ruta, se puede ver marcada la trayectoria que seguirá el coche, perfectamente centrada en el carril de la rotonda.

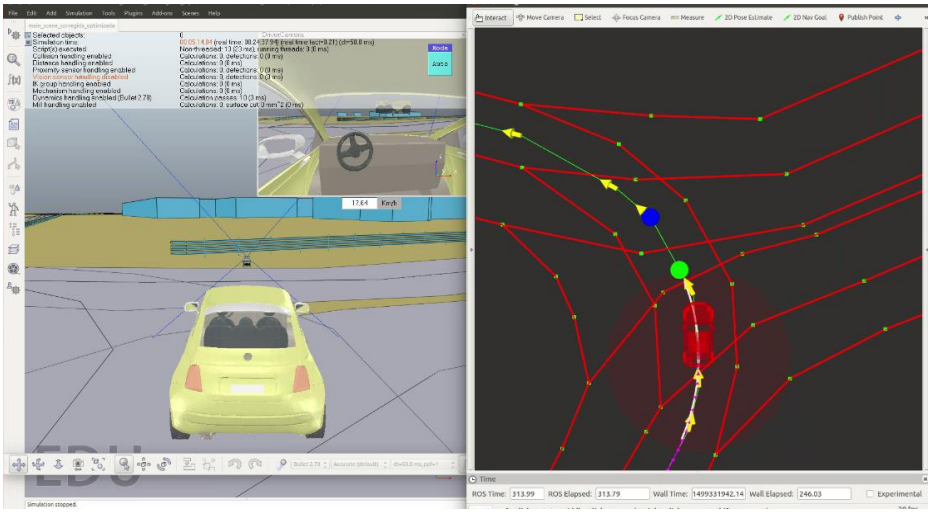


Figura 6.25: Vista desde arriba recorriendo la ruta de Biología-Hospital

A continuación, mostramos la ruta del simulador en la Figura 6.26, en este caso está orientada al revés que la de la Figura 6.27. Pero es la misma ruta

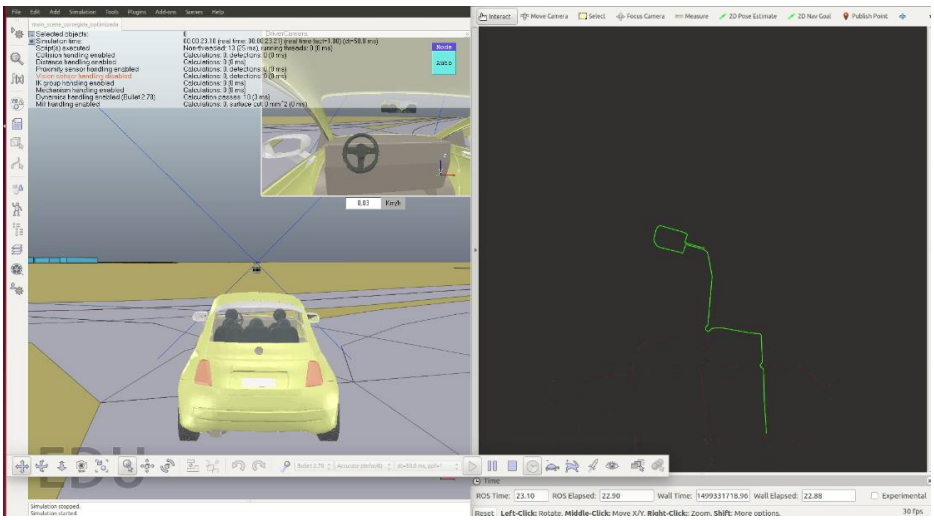


Figura 6.26: Ruta en el simulador de Biología – Hospital

Vemos más claramente en la Figura 6.27, la ruta seguida:

Figura 6.27: Ruta de Biología-Hospital en JOSM

Seguimos con navegación no reactiva, en este caso el vídeo duró 4 minutos con 30 segundos.

6.3.4 Ruta: Residencia_Universitaria → CDG_Telefonica

Las paradas de la ruta y la lista de lanelets que se han seguido son las siguientes:

Startstation: Residencia_Universitaria

Endstation: CDG_Telefonica

RUTA: [<Lanelet -51511>, <Lanelet -51503>, <Lanelet -51495>, <Lanelet -51479>, <Lanelet -51135>, <Lanelet -51125>, <Lanelet -51099>, <Lanelet -51101>, <Lanelet -51185>, <Lanelet -51187>, <Lanelet -51831>, <Lanelet -51057>, <Lanelet -51035>, <Lanelet -51333>, <Lanelet -51337>, <Lanelet -51357>, <Lanelet -51565>, <Lanelet -51569>, <Lanelet -51639>, <Lanelet -51591>, <Lanelet -51643>, <Lanelet -51601>, <Lanelet -51647>]

En la Figura 6.28 vemos el vehículo desde atrás llevando a cabo una recta de la ruta siguiendo los puntos objetivo:

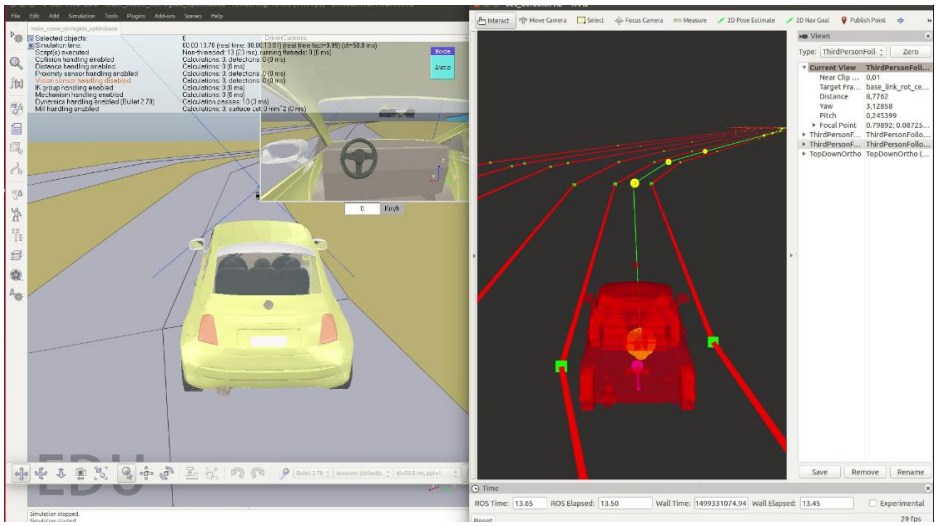


Figura 6.28: Vista posterior en la ruta Residencia-Telefonica

En la Figura 6.29 vemos al vehículo atravesando una rotonda, manteniéndose en el centro del carril y siguiendo perfectamente los puntos objetivo de esta ruta:

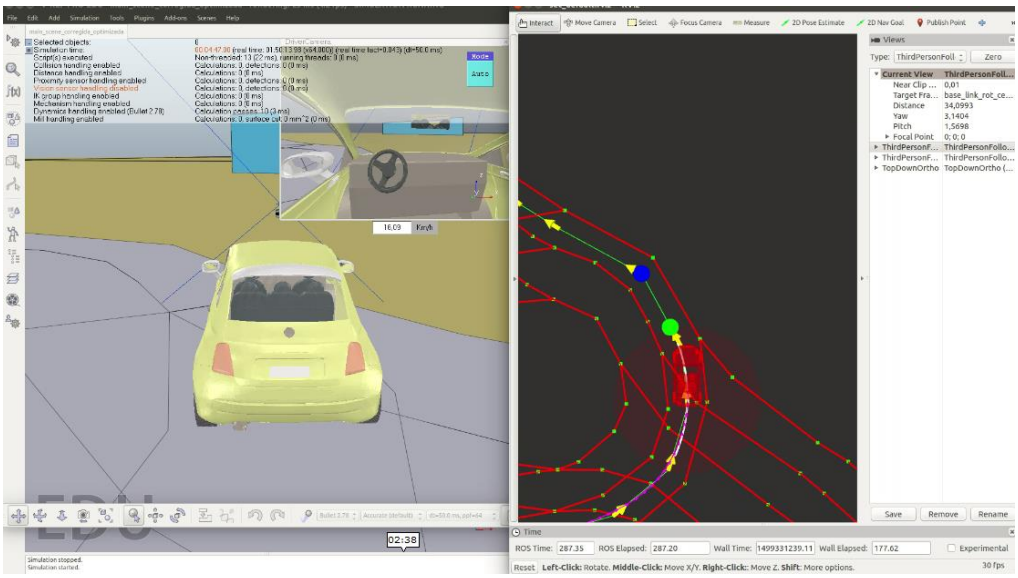


Figura 6.29: Vista desde arriba en la ruta Residencia-Telefónica

A continuación, en la Figura 6.30 vemos la trayectoria seguida por el vehículo para esta ruta, como en las anteriores, no se ve en el fondo de la imagen el mapa universitario,

así que ilustraremos en la siguiente figura, la Figura 6.31, la ruta, dibujándola sobre el mapa universitario:

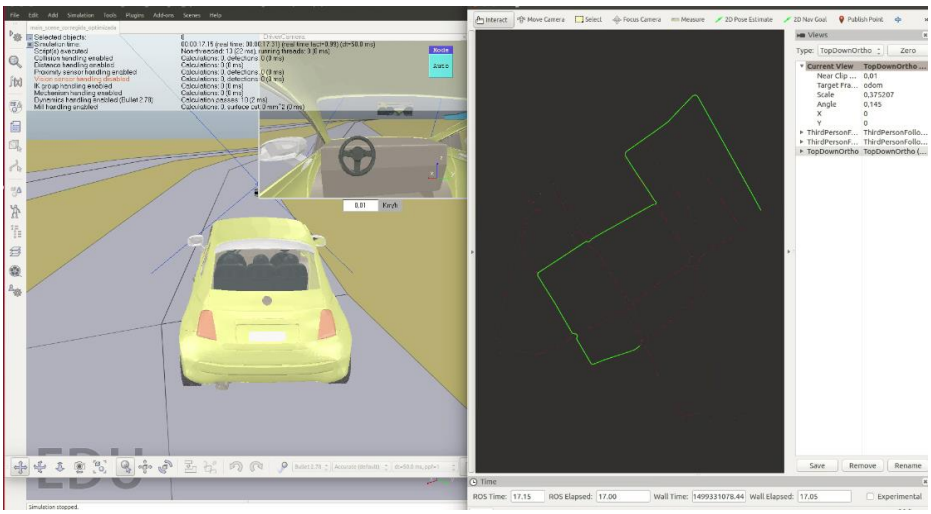


Figura 6.30: Ruta en el simulador de Residencia-Telefonica

Como hemos dicho en la Figura 6.31 tenemos la ruta dibujada sobre el mapa universitario:

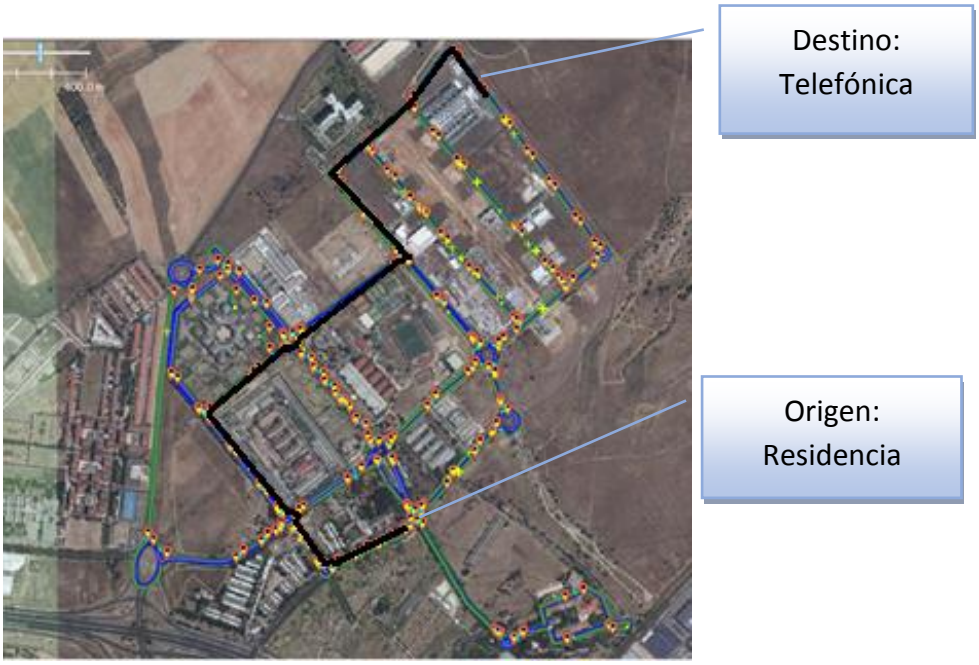


Figura 6.31: Ruta de Residencia-Telefónica en JOSM

En este caso el vídeo duró 6 minutos con 28 segundos.

6.3.5 Ruta: Biblioteca_Nacional_de_España → HospitalUniversitario_PrincipedeAsturias

Las paradas de la ruta y la lista de lanelets que se han seguido son las siguientes:

Startstation: Biblioteca_Nacional_de_España

Endstation: HospitalUniversitario_PrincipedeAsturias

RUTA: [<Lanelet -51641>, <Lanelet -51571>, <Lanelet -51581>, <Lanelet -51355>, <Lanelet -51345>, <Lanelet -51351>, <Lanelet -51041>, <Lanelet -51367>, <Lanelet -51039>, <Lanelet -51059>, <Lanelet -51079>, <Lanelet -51065>, <Lanelet -51139>, <Lanelet -51141>, <Lanelet -51133>, <Lanelet -51125>, <Lanelet -51099>, <Lanelet -51101>]

Vemos a continuación, en la Figura 6.32, la vista desde atrás del vehículo, mientras recorre la trayectoria de la ruta:

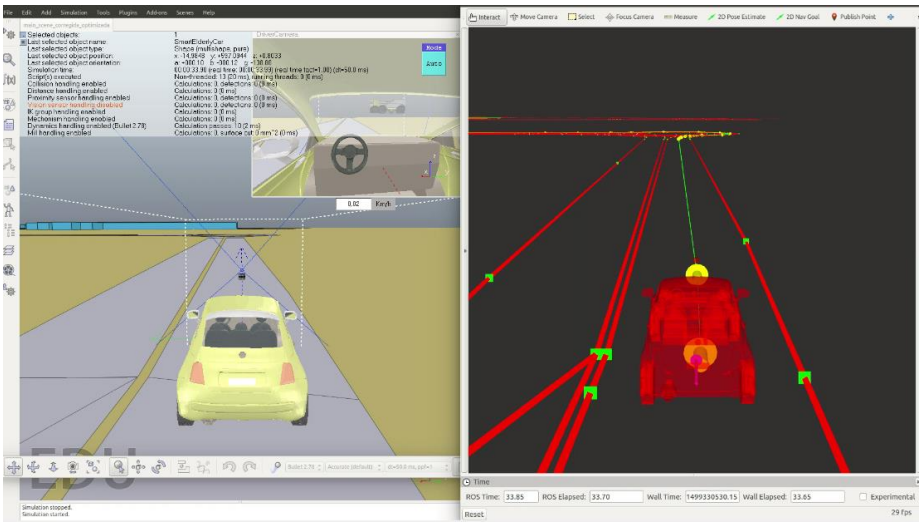


Figura 6.32: Vista posterior en la ruta Biblioteca-Hospital

En la siguiente figura, la Figura 6.33 podemos ver desde arriba al vehículo atravesando una de las rotondas de la ruta sin salirse del carril.

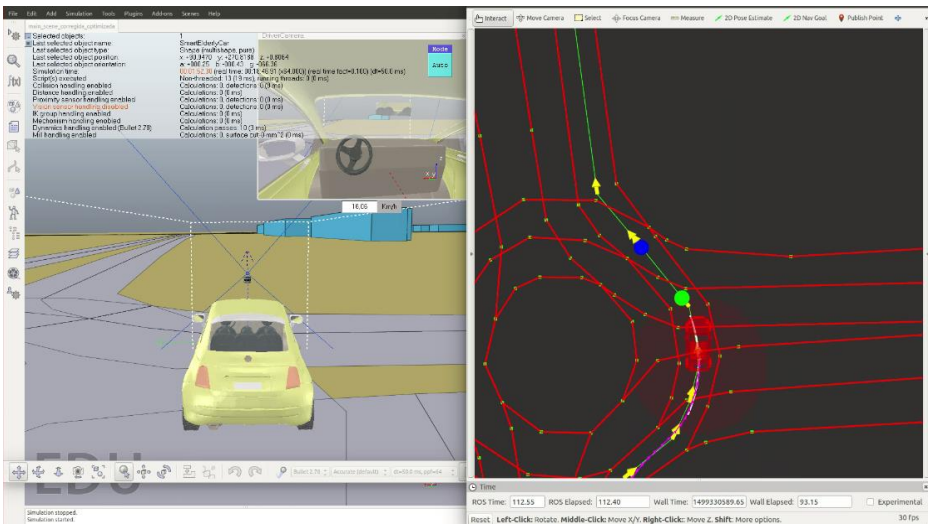


Figura 6.33: Vista desde arriba en la ruta Biblioteca-Hospital

En Figura 6.34, podemos ver la ruta seguida por el vehículo en un fondo negro, el mapa de la universidad está pintado, pero con líneas roja que son las que se ven en las fotos de vistas más cercanas.

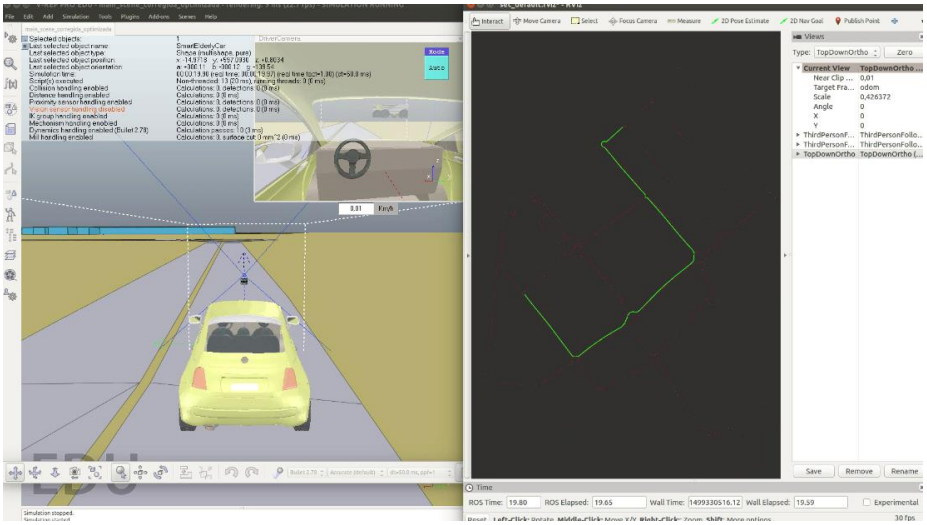


Figura 6.34: Ruta en el simulador de Biblioteca – Hospital

Para dejar más claro la trayectoria llevada, la dibujamos sobre el mapa universitario:

Figura 6.35: Ruta de la Biblioteca - Hospital en JOSM

En este caso el video duró 4 minutos y 30 segundos.

6.3.6 Ruta: Farmacia → Politecnica

Las paradas de la ruta y la lista de lanelets que se han seguido son las siguientes:

Startstation: Farmacia

Endstation: Politecnica

RUTA: [<Lanelet -51417>, <Lanelet -51459>, <Lanelet -51445>, <Lanelet -51451>, <Lanelet -51477>, <Lanelet -51511>, <Lanelet -51503>, <Lanelet -51495>, <Lanelet -51479>, <Lanelet -51135>, <Lanelet -51125>, <Lanelet -51099>, <Lanelet -51101>, <Lanelet -51185>, <Lanelet -51073>, <Lanelet -52027>, <Lanelet -51123>, <Lanelet -51045>, <Lanelet -51047>, <Lanelet -51049>, <Lanelet -51193>, <Lanelet -51225>, <Lanelet -51843>, <Lanelet -51847>, <Lanelet -51851>, <Lanelet -51869>]

En la Figura 6.36 podemos ver el vehículo desde atrás recorriendo los carriles, pero en este caso también vemos una línea blanca, a parte de la fucsia y la verde con los puntos objetivo.

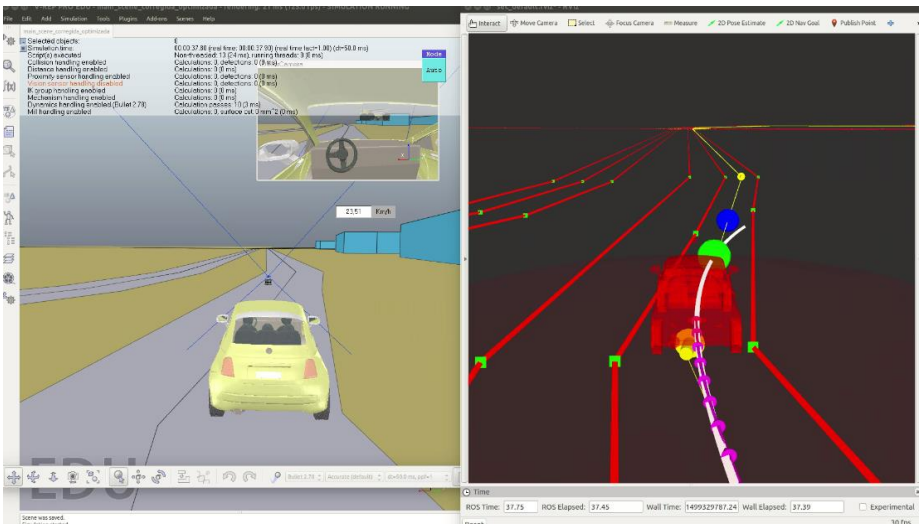


Figura 6.36: Vista posterior en la ruta Farmacia-Politecnica

Esa línea marca dibuja la curvatura que ha tiene que seguir el vehículo para seguir avanzando correctamente.

A continuación, en la Figura 6.37, tenemos una vista desde arriba, del vehículo atravesando las curvas de una rotonda.

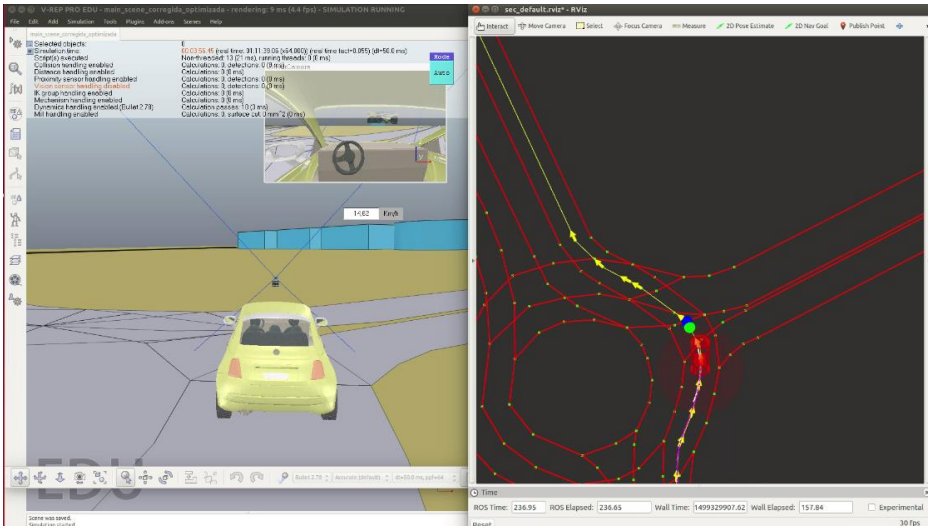


Figura 6.37: Vista desde arriba en la ruta Farmacia – Politecnica

Para esta ruta, conseguimos mostrar el dibujo de la ruta en la pantalla entera del simulador. Podemos verlo en la Figura 6.38.

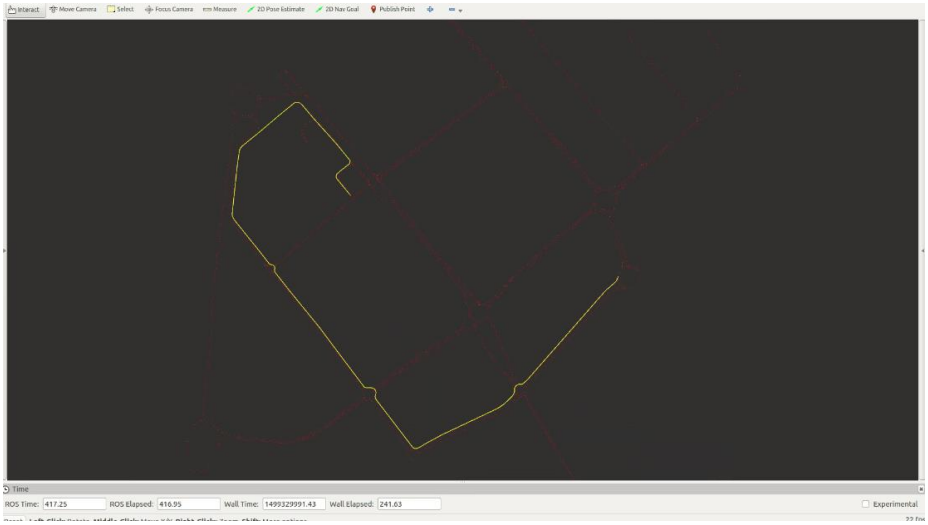


Figura 6.38: Ruta en el simulador de Farmacia – Politecnica

Y, como en la Figura 6.38 no se distingue el mapa de la universidad en el fondo, se puede ver dibujado en la Figura 6.39:

Figura 6.39: Ruta de Farmacia - Politecnica en JOSM

En este caso el vídeo duró 5minutos con 35 segundos.

Capítulo 7 Conclusiones

La implementación del mapa basado en lanelets del campus externo de la Universidad de Alcalá ha contribuido al objetivo final del proyecto SmartElderlyCar consistente en el desarrollo de un vehículo eléctrico capaz de conducir de forma autónoma por nuestro campus. Manteniéndose dentro de los límites de conducción y cumpliendo las normas viales.

El mapa de Open Street Map no proporcionaba los suficientes datos para ello, ya que utiliza solo una línea por carril, mientras que con los lanelets delimitan el carril con dos polilíneas evitando así que el coche se salga de los límites establecidos por la carretera.

Además, con el método de los lanelets podemos incluir en el mismo mapa elementos regulatorios para el coche de manera que cumpla así las normas viales.

Hemos conseguido también obtener, a partir del mapa, un documento de texto con los datos más útiles del entorno del mapa: las señales, las coordenadas más importantes de la señalización como cuándo debe tener en cuenta el coche que hay una señal y dónde debe pararse, las coordenadas de los nodos que forman los lanelets, las coordenadas de las paradas para las rutas y la ruta más corta que debe seguir el vehículo para llegar a su destino.

La utilidad del mapa ha sido validada en las simulaciones realizadas con la ayuda de la Universidad de Vigo; donde se comprueba el cálculo correcto de la ruta más corta, así como la navegación dentro de los límites de las carreteras.

La utilidad de los elementos regulatorios podremos verla en el futuro desarrollo del proyecto, donde podremos simular un coche que tiene que interactuar con más elementos en el entorno ya sean estáticos, utilizando los elementos regulatorios, o dinámicos, como personas u otros coches.

También hay que tener en cuenta la gran portabilidad del mapa ya que utiliza un formato XML.

Por lo tanto, se demuestra que el método de los lanelets a la hora de crear un mapa para la circulación, es muy completo, ya que conseguimos solucionar la mayor parte de los problemas de la navegación de una sola vez y será muy útil en el el proyecto SmartElderlyCar de conducción autónoma.

Capítulo 8 Presupuesto

En este capítulo se adjuntan todas las partidas presupuestarias necesarias para la ejecución del proyecto. Se adjuntan tanto los presupuestos para materiales como los de mano de obra.

8.1 Material:

En la Tabla 5 se adjunta el desglose de costes de material utilizado para el proyecto:

Concepto	Precio	Amortización	Tiempo de uso	Coste
Ordenador ASUS i7	700€	4 años	6 meses	87,5 €
Ratón óptico	9€	4 años	6 meses	1,125€
Conexión ADSL	10€/mes	-	6 meses	60 €
			TOTAL	148,625 €

Tabla 8.1: Costes de material

8.2 Software:

En la Tabla 6 se adjunta el desglose de costes de material utilizado para el proyecto:

Concepto	Precio	Amortización	Tiempo de uso	Coste
Máquina virtual	0€	N/A	6 meses	0€
Ubuntu 14.04.3	0€	N/A	6 meses	0€
JOSM	0€	N/A	6 meses	0€
Liblanelet	0€	N/A	6 meses	0€
Eclipse Indigo	0€	N/A	6 meses	0€
			TOTAL	0€

Tabla 8.2: Costes de software

8.3 Otros conceptos:

En la tabla 7 se adjunta el desglose de todos los costes adicionales requeridos en este trabajo:

Concepto	Consumo	Tiempo de uso	Coste
Transporte	26€/mes	6 meses	156€
Material de oficina			10€
		TOTAL	166€

Tabla 8.3: Otros costes

8.4 Mano de obra:

En la Tabla 8 se adjunta el desglose de todos los costes de mano de obra requeridos en este trabajo:

CONCEPTO	PRECIO	HORAS	COSTE
Ingeniería/programación	20€/h	360 h	7200 €
Mecanografía	15€/h	70 h	1050 €
		TOTAL	8250 €

Tabla 8.4: Costes de mano de obra

8.5 Presupuesto total:

En la tabla 9 se adjunta el desglose de todos los costes totales requeridos en este trabajo son:

Concepto	Costes
Costes de material	148,625 €
Costes de software	0 €
Costes de mano de obra	8250 €
Otros costes	166€
TOTAL	8564,625 €
Impuestos (21%)	1798,57€
Total	10363,195€

Tabla 8.5: Costes totales

Capítulo 9 Pliego de condiciones

A continuación, describiremos las características de hardware y software necesarias para la realización de este proyecto.

9.1 Requisitos Hardware

En la tabla 10 se pueden ver los requisitos hardware necesarios para el desarrollo de este proyecto, que se ha desarrollado sobre un ordenador portátil ASUS i7.

Componente	Características
RAM	8GB
Disco duro	258GB
Tarjeta gráfica	GEFORCE
Procesador	Intel core i7
Sistema operativo	64 bits
Ratón	Ratón óptico Easterntimes Tech

Tabla 9.1: Requisitos Hardware del trabajo

9.2 Requisitos Software

En la tabla 11 se pueden ver los requisitos Software necesarios para el desarrollo de este proyecto, que se ha desarrollado sobre un ordenador portátil ASUS i7.

Componente	Características
Máquina virtual	Oracle VM
SO de la máquina virtual	Ubuntu 14.04.3
Eclipse Indigo	Entorno de desarrollo
Librerías y código	Liblanelet: https://github.com/phbender/liblanelet De aquí obtenemos el código usado en el proyecto de Raúl Rojas. Java versión 8 actualización 121 Librerías básicas de C++

Tabla 9.2: Requisitos software

Capítulo 10 Bibliografía

http://learnosm.org/es/beginner/introduction/

http://wiki.openstreetmap.org/wiki/History_of_OpenStreetMap

http://wiki.openstreetmap.org/wiki/ES:JOSM

http://wiki.openstreetmap.org/wiki/ES:ID

http://learnosm.org/es/josm/start-josm/

https://es.wikipedia.org/wiki/OpenStreetMap

https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda_A*

https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra

https://www.youtube.com/watch?v=LLx0QVMZVkk

https://www.ecured.cu/Distancia_eucl%C3%ADdea

https://github.com/phbender/liblanelet

https://es.wikipedia.org/wiki/Framework

https://es.wikipedia.org/wiki/Cl%C3%BAster_(inform%C3%A1tica)

https://es.wikipedia.org/wiki/Sistema_Operativo_Rob%C3%B3tico

http://ri.cmu.edu/pub_files/pub1/simmons_reid_1996_1/simmons_reid_1996_1.pdf

C. Otero, E. Paz, R. Sanz y J. López, R. Barea, E. Romera, E. Molinos, R. Arroyo, L.M. Bergasa, E. López, "SIMULACIÓN DE VEHÍCULOS AUTÓNOMOS USANDO V-REP BAJO ROS", Aceptado para su publicación en la conferencia JA2017 (XXXVIII Jornadas de Automática), Gijón, España, 6-8 de septiembre de 2017.

Paul Czerwionka, "A Three Dimensional Map Format for Autonomous Vehicle", Thesis, Institute of Computer Science, Freie Universität Berlin, Germany, September 29, 2014.

[18] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: a probabilistic, flexible, and compact 3D map representation for robotic systems," in Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation, vol. 2, 2010.

[19] A. Geiger, M. Lauer, F. Moosmann, B. Ranft, H. Rapp, C. Stiller, and J. Ziegler, "Team AnnieWAY's entry to the 2011 grand cooperative driving challenge," IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 3, pp. 1008–1017, 2012.

[20] E. D. Dickmanns and B. D. Mysliwetz, "Recursive 3-d road and relative ego-state recognition," IEEE Transactions on pattern analysis and machine intelligence, vol. 14, no. 2, p. 199213, 1992.

[21] Y. Wang, D. Shen, and E. K. Teoh, "Lane detection using spline model," Pattern Recognition Letters, vol. 21, no. 8, pp. 677–689, Jul. 2000

[22] C. Jung and C. Kelber, "A robust linear-parabolic model for lane following," in 17th Brazilian Symposium on Computer Graphics and Image Processing, 2004. Proceedings, 2004, pp. 72–79.

[23] M. Baer, U. Hofmann, and S. Gies, "Multiple track 4D-road representation," in 2010 IEEE Intelligent Vehicles Symposium (IV), Jun. 2010, pp. 319–324.

[24] D. J. Kang, J. W. Choi, and I.-S. Kweon, "Finding and tracking road lanes using line-snakes," in , Proceedings of the 1996 IEEE Intelligent Vehicles Symposium, 1996, 1996, pp. 189–194.

[25] M. Brubaker, A. Geiger, and R. Urtasun, "Lost! leveraging the crowd for probabilistic visual self-localization," in 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.

[26] D. Betaille and R. Toledo-Moreo, "Creating enhanced maps for lanelevelvehiclenavigation,"IEEETransactionsonIntelligent

[27] C. Hasberg, S. Hensel, and C. Stiller, "Simultaneous localization and mapping for path-constrained motion," IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 2, pp. 541–552, 2012.

[28] SmartEderlyCar: http://www.robesafe.es/index.php/en/smartelderlycar

[29] CommonRoad: Composable Benchmarks for Motion Planning on Roads

[30] Simulación de vehículos autónomos usando V-REP bajo ROS.

[31] https://es.mathworks.com/help/robotics/ug/pure-pursuit-controller.html

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá