UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

INCORPORACIÓN DE SENSORES REALISTAS EN LA SIMULACIÓN EN CARLA DE LA NAVEGACIÓN DE COCHES AUTÓNOMOS

Grado en Ingeniería en Electrónica, Robótica y Mecatrónica

Autor: DANIEL STEVEN GAMBA CORREA

Tutor: JESÚS MORALES RODRIGUEZ

Cotutor JORGE LUIS MARTINEZ RODRIGUEZ

Departamento de Ingeniería de Sistemas y Automática

MÁLAGA, Junio de 2.023

Agradecimientos

A Dios, que me ha guiado a lo largo de toda mi vida

A mis padres, que desde muy joven me incentivaron a estudiar y desde entonces me han acompañado en cada etapa de mi vida. Todo lo que he logrado no hubiese sido posible sin ellos

A mis hermanos, que me han acompañado durante todos estos años, apoyándome y dándome el ánimo necesario para seguir adelante

Resumen

Este trabajo de fin de grado (TFG) se ocupa del desarrollo, la configuración e incorporación de sensores reales en vehículos autónomos en *CARLA Simulator*. Entre los sensores modelados se incluyen un LiDAR tridimensional (3D), diferentes tipos de cámaras, así como GNSS e IMU.

También desarrolla simulaciones realistas totalmente controladas y personalizadas donde se elaboran rutas específicas para el vehículo y donde actúan diferentes agentes que se encuentran en vías urbanas, como otros automóviles, transporte público, ciclistas y peatones.

Por último, se proporciona una interfaz con ROS 2 donde se dispone de toda la información registrada y se puede analizar posteriormente.

Palabras clave: LiDAR, GNSS, sensor IMU, cámara RGB, cámara de profundidad, cámara térmica, CARLA, ROS 2, conducción autónoma

Abstract

This thesis deals with the development, configuration, and integration of real sensors in autonomous vehicles in CARLA Simulator. The modeled sensors include a three-dimensional (3D) LiDAR, various types of cameras, as well as GNSS and IMU.

It also develops realistic and fully controlled simulations where specific routes are created for the vehicle, and different agents such as other cars, public transportation, cyclists, and pedestrians interact in urban environments.

Finally, an interface with ROS 2 is provided, allowing access to all recorded information for subsequent analysis.

Keywords: LiDAR, GNSS, IMU, RGB camera, depth camera, termal camera, CARLA, ROS 2, autonomous driving

Índice

Acrónimos

- 3D: Tridimensional
- API: *Application Programming Interface*
- FOV: *Field of View*
- FPS: *Frames per Second*
- GNSS: *Global Navigation Satellite System*
- IMU: *Inercial Measurement Unit*
- MPU: Multiple Process Unit
- LiDAR: *Light Detection and Ranging*
- NIST: *National Institute of Standars and Technology*
- NPC: *Non Player Character*
- RCL: *ROS 2 Client Library*
- RGB: *Red, Green and Blue*
- ROS: *Robot Operating System*
- RTK: *Real Time Kinematics*
- SAE: *Society of Automotive Engineers*
- TFG: Trabajo de Fin de Grado
- TFM: Trabajo de Fin de Máster

1 Introducción

1.1 Motivación

En la actualidad, el desarrollo de vehículos autónomos es una línea de investigación importante en el sector de la automoción. Aunque algunos coches tienen la capacidad de percibir su entorno y tomar decisiones similares a las de un conductor humano, la autonomía total del vehículo sin intervención humana aún requiere más investigación y esfuerzo. Actualmente, los sistemas comerciales más avanzados, como el *autopilot* de Tesla, se encuentran en el nivel 3, de los 6 niveles (ver Figura 1.1) que establece la Sociedad de Ingenieros de la Automoción (SAE) [1]. En este nivel el conductor debe estar preparado para intervenir cuando el sistema lo requiera.

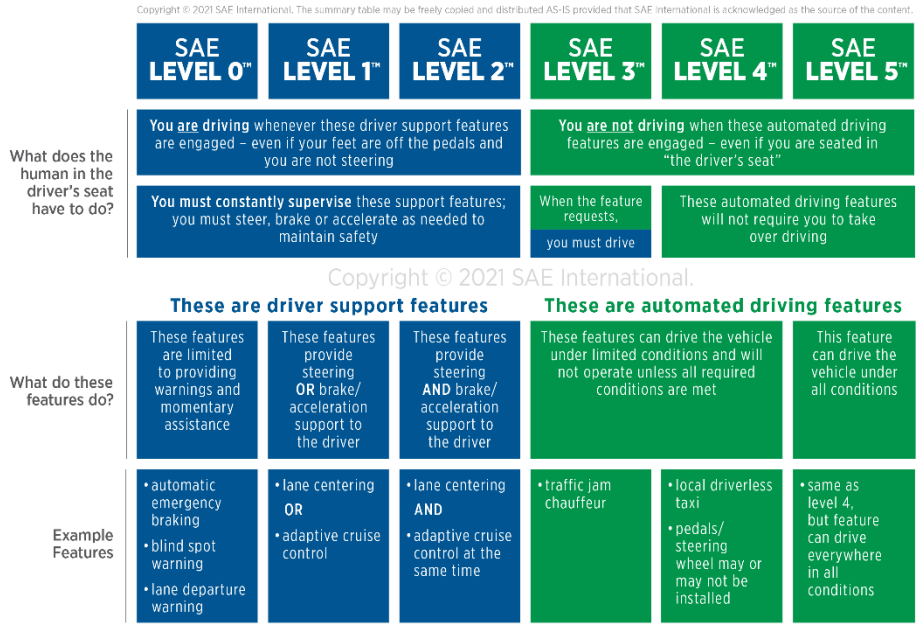


Figura 1.1. Los 6 niveles de conducción autónoma [1]

Aunque se ha llegado a alcanzar este nivel, es cierto que la mayoría de las pruebas que se hacen en la actualidad son en carreteras. Sin embargo, las aplicaciones en áreas urbanas, como los centros de la ciudad, con zonas peatonales, carriles para bicicletas y denso tráfico, son más exigentes para el desarrollo de estos sistemas inteligentes. En este tipo de entorno, existe un patrón de tráfico en el que principalmente peatones, patinadores y ciclistas comparten el mismo espacio con vehículos privados y de carga. Esto puede comprometer su seguridad, incluso teniendo en cuenta que los límites de velocidad en estas áreas urbanas suelen ser más bajos que en otras zonas.

1.2 Marco de Realización

Este TFG se enmarca dentro de las labores de investigación del departamento de Ingeniería de Sistemas y Automática. Concretamente se desarrolla en el proyecto "Predicción del movimiento de los participantes del tráfico para la integración segura del vehículo autónomo en áreas urbanas", a partir de ahora referido en este documento como proyecto PREMOVE [2], con el objetivo de desarrollar técnicas para detectar y predecir el movimiento de los participantes del tráfico, y así poder evitar colisiones y facilitar la integración segura de los vehículos autónomos en áreas urbanas restringidas.

Además, el autor ha obtenido una beca de colaboración con departamentos del Ministerio de Educación y Formación Profesional para la realización de su TFG.

1.3 Objetivos

El objetivo de este trabajo es la incorporación de sensores realistas para la navegación autónoma haciendo uso de la última versión del simulador CARLA. Para ello, se instalará la versión *source* [3]. Esto nos permitirá tener libre acceso a una plataforma en la que se pueden desarrollar sistemas de percepción y controlar los agentes implicados en el tráfico. Además, la versión *source* hace posible obtener un mayor conocimiento de cómo son las medidas y las físicas de todos los agentes implicados en la simulación, incluyendo personalizar los entornos de simulación. En este trabajo también se ha hecho uso de la versión *package* del simulador, debido a que esta versión consume menos recursos del equipo empleado, permitiendo una ejecución más rápida y efectiva.

Una vez seleccionados los sensores que se utilizarán, se realizará el estudio de sus características y se relacionarán con los parámetros que se disponen en el simulador. De este modo, se planificará la disposición de los sensores en el vehículo simulado.

También se pretende llevar a cabo una monitorización de los agentes que intervienen en la simulación por medio de la integración de sensores realistas en estos agentes. Para ello, se realizarán simulaciones con el máximo nivel de controlabilidad, es decir, conociendo la posición y el estado de cada uno de los agentes en cada instante de la simulación. Por último, se proporcionará una interfaz en ROS 2 [4] para registrar toda la información proporcionada por los sensores. Para la inclusión de esta interfaz se estudiará el uso del puente existente entre CARLA y ROS 2 y se investigarán qué paquetes de ROS son útiles para el propósito del trabajo.

1.4 Desarrollo del TFG

El primer paso del trabajo fue realizar toda la instalación de software y sistemas operativos necesarios para el desarrollo del proyecto. La comunicación entre el simulador

y ROS solo la realiza con la versión Foxy, por lo que el sistema operativo utilizado fue Ubuntu.

Después de una investigación a fondo de las características de cada uno de los sensores que se incluyen en la simulación se realizó el modelado de los mismos modificando los parámetros de los sensores definidos en *CARLA Simulator*. Se ha conseguido el modelo de una cámara de profundidad, la ZED2i, con un sensor IMU integrado; de una cámara térmica, la Seek CompactPRO, que nos permite conseguir información de la temperatura del entorno fácilmente; de un LiDAR 3D, el RS-Helios 5515, que proporciona una distribución asimétrica de haces; y, por último, de sensores de bajo coste, tanto GPS como IMU, el ZED F9P y MPU 9250, respectivamente, que proporcionan datos fiables a bajo coste. Después de esto se realizó una distribución a lo largo del techo del vehículo en posiciones detalladas.

Paralelamente, se aprendió a controlar las distintas características del simulador, esto es, los parámetros que se pueden modificar y el amplio número de mapas, agentes y modelos que nos proporciona. Para de este modo, poder realizar rutas personalizadas del vehículo y poder controlar el resto de los agentes, tanto vehículos como peatones. Con objeto de realizar diversas pruebas en entornos totalmente controlados.

Por último, se ha desarrollado una interfaz utilizando las herramientas proporcionadas por el simulador en ROS 2 en la que se puede obtener toda la información proporcionada por cada uno de los sensores incluidos en la simulación.

1.5 Estructura de la Memoria

El proyecto se ha estructurado del siguiente modo:

- El capítulo 1 muestra una visión general del TFG.
- En el capítulo 2 se presentan todas las herramientas software utilizadas para el desarrollo del proyecto, así como, la descripción de sus características más importantes.
- En el capítulo 3 se describe cómo se ha realizado el modelado de los sensores, desde la definición de parámetros en el simulador hasta la posición que ocupará en cada agente.
- En el capítulo 4 se describe la manera en la que se ha realizado la interfaz en ROS 2 y se describen los experimentos que se han realizado, así como los resultados obtenidos en éstos.
- En el capítulo 5 se muestran las conclusiones y los posibles trabajos futuros.
- Finalmente, en los anexos A, B y C se muestran las referencias empleadas en el TFG, el proceso de instalación del software y el manual de usuario para hacer uso del código, respectivamente.

2 Herramientas Software

2.1 *CARLA Simulator*

En la actualidad existen distintos simuladores de coches autónomos, pero en los últimos años uno de los más destacados es *CARLA Simulator* [3], de código abierto y desarrollado por investigadores de la Universidad Autónoma de Barcelona y el laboratorio Intel Computing Visual. Es un simulador basado en *Unreal Engine* [5], con lo que permite editar nuestro entorno usando esta herramienta. Este simulador proporciona un entorno virtual que es altamente personalizable en el cual se pueden probar algoritmos y sistemas de conducción autónoma. CARLA proporciona distintos activos creados específicamente para ese fin.

Las características más destacadas del simulador son:

- Escalabilidad. Cuenta con un servidor con que pueden conectarse múltiples clientes, controlando distintos actores cada uno de ellos.
- API (*Aplication Programming Interface*) flexible. Los clientes pueden controlar multitud de aspectos de la simulación, como el tiempo meteorológico, la hora del día y el tráfico, entre otros.
- Gran variedad de sensores. Se disponen de distintos sensores propioceptivos (inerciales, odometría) y sensores exteroceptivos (LiDAR, cámaras RGB, cámaras de profundidad, sonar, GNSS).
- Simulación para planificación y control. Se proporciona una serie de herramientas con las que se desactiva el renderizado para poder realizar ejecuciones rápidas.
- Generación de mapas. Se pueden crear mapas del entorno por medio de herramientas como *RoadRunner*.
- Biblioteca de escenarios. Se proporciona una herramienta con la que se pueden recrear situaciones de tráfico con comportamientos modulares.
- Integración de ROS. Cuenta con la integración tanto de ROS 1 como de ROS 2 por medio del paquete *ROS-bridge*.
- Es código abierto. Al ser accesible por todo el mundo, se fomenta la colaboración e intercambio de conocimiento entre investigadores.

En los últimos años este simulador ha sido varias veces actualizado, siendo la versión en la que se desarrolla este trabajo la última, CARLA 0.9.14.

Al estar basado *CARLA Simulator* en el programa *Unreal Engine* la base de todos los actores son los llamados *Blueprint*s, los cuales definen sus características por medio de sus *Attributes*. En la documentación de CARLA encontramos la biblioteca específica de *Blueprints* del simulador.

Otra de las características relevantes de CARLA es el *Traffic Manager*, que es el encargado gestionar un conjunto de vehículos que navegan por la simulación, creando obstáculos y desafíos para el vehículo de interés, es decir, el vehículo que estamos entrenando o controlando. Éste administra el comportamiento y los ciclos de vida de los *Non Player Character* (NPC) dentro del mapa.

Por último, CARLA Simulator cuenta con 2 versiones de instalación, la versión *source* y su versión *package*. El primer modo de instalación, mucha más amplia, permite llevar a cabo la personalización de agentes y de mapas, además de otros aspectos del entorno. Mientras que, la segunda versión, mucho más ligera, es un ejecutable que permite crear el servidor de CARLA con el cual nos podemos conectar y llevar a cabo las simulaciones (ver Figura 2.1).

Figura 2.1. Simulador de CARLA

2.2 ROS 2

El *Robot Operating System 2* (ROS 2) [4] es un conjunto de bibliotecas de software y herramientas para crear aplicaciones para robots. Con esta herramienta se pueden crear desde controladores hasta algoritmos de última generación. Se basa, principalmente, en un sistema de comunicación de publicación-subscripción, lo que hace accesible la información a todos los nodos de manera sencilla.

Desde que se inició ROS en 2007, muchas cosas han cambiado en la comunidad de robótica y del propio ROS. Por eso, ROS 2 toma lo mejor de su predecesor y lo mejora

para poder abordar un mayor número de problemas y desafíos. Entre las mejoras cabe destacar:

- Cambio en la API de ROS. ROS 2 está formado por más capas que ROS 1, incluyendo únicamente una biblioteca base, llamada RCL (*ROS 2 Client Library*). No es como en ROS 1 donde había que incluir una librería para C++ y otra para Python. Esto facilita de gran manera la implementación de nuevas funcionalidades.
- Cambio en la versión de C++. ROS 1 utilizaba las especificaciones C++98, mientras que ROS 2 puede usar las especificaciones C++11 y C++14, por defecto. Pudiendo usar las últimas innovaciones en este lenguaje.
- Cambios de convenciones en los nodos. En ROS 1 no había una estructura clara para crear un nodo e implementar todas sus funcionalidades. Esto provocaba problemas al compartir módulos. Por lo contrario, ROS 2 se ha establecido una convención clara.
- Introducción de nodos con ciclo de vida. Establece distintos estados de un nodo, permitiendo de este modo separar las fases de configuración y de ejecución de un nodo.
- Servicios. Se implementan los servicios asíncronos, lo que permite que el código pueda ejecutarse mientras que se espera la respuesta del servidor. Esto no era posible en ROS 1.
- Comunicación. En ROS 1 existe el máster de ROS, que actúa como un servidor DNS para los nodos. En ROS 2, esta figura desaparece y se crea un sistema en el que cada nodo es totalmente independiente, pudiendo descubrir otros nodos por sí mismo. Esto conlleva la desaparición de los parámetros globales: cada nodo gestiona sus propios parámetros.

Esta herramienta está diseñada para ser modular y escalable, lo que permite una gran flexibilidad y adaptabilidad en diferentes entornos y arquitecturas. Además, también ofrece una mayor compatibilidad con diferentes lenguajes de programación, entre ellos C++ y Python, lo que permite a los desarrolladores elegir el entorno que mejor se adapte a ellos. Además, cuenta con diversas herramientas, tanto de visualización (rviz, RQt) como de gestión de datos (*bag*), que permiten al usuario analizar y tratar los datos de una manera rápida y sencilla.

Por todo esto, desde el prototipado hasta el desarrollo y la producción, ROS 2 se ha convertido en una de las plataformas de referencia para la mayoría de los investigadores y desarrolladores en cualquier rama de la robótica.

2.2.1 RVIZ2

Herramienta de ROS2 que permite representar las imágenes generadas por distintos sensores, que están siendo publicadas en tiempo real [6]. Para este TFG es muy

útil, ya que se puede visualizar tanto las imágenes generadas por todos los tipos de cámaras, así como, la nube de puntos generada por el LiDAR (ver Figura 2.2).

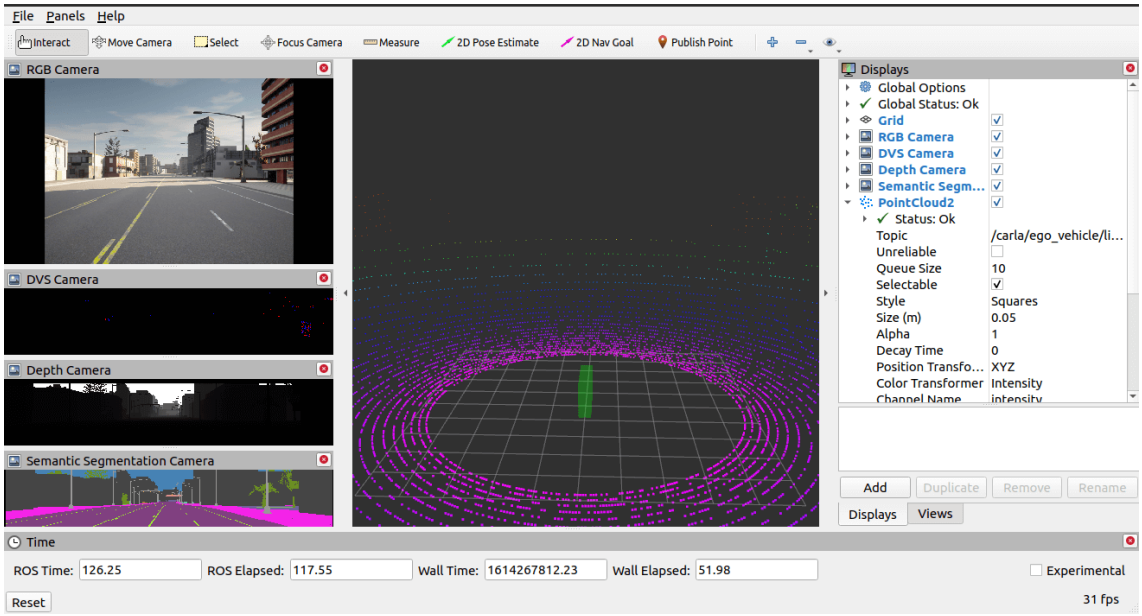


Figura 2.2. Ejemplo de uso de la librería RVIZ en CARLA [6]

2.2.2 RQt

RQt [7] es un conjunto de herramientas visuales que proporcionan una interfaz gráfica de usuario. Está diseñado para facilitar el uso de diferentes herramientas y funciones al agruparlas en forma de complementos. En lugar de tener varias ventanas separadas para cada herramienta, RQt permite ejecutar todas estas herramientas dentro de una sola ventana, lo que hace más sencillo organizar y gestionar todo en una única pantalla. Puedes verlo como una especie de "estación de trabajo" donde puedes acceder y utilizar todas las herramientas necesarias de manera más conveniente.

Esta herramienta se usará para realizar fácilmente las gráficas respecto al tiempo de los datos que serán necesarios, ya que, se puede subscribir a un *topic* y graficar en tiempo real los datos. También cuenta con la opción de reproducir las imágenes capturadas por una cámara (ver Figura 2.3).

2.2.3 ROS 2 *BAG*

ROS 2 *bag* [8] permite capturar y guardar los datos que se publican en los *topics* de tu sistema. Puedes usar estos datos más tarde para reproducir las mismas condiciones y resultados de tus pruebas, o incluso compartirlos con otros para que puedan recrear y analizar tu trabajo.

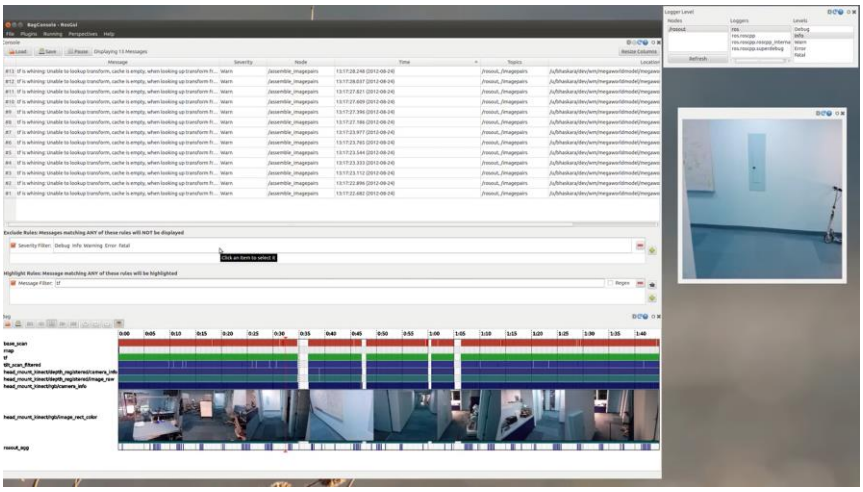


Figura 2.3. Ejemplo de uso de RQt [7]

Debido a la carga computacional y a las facilidades que aporta el uso de esta herramienta, además del posterior uso de los datos reproducidos en simulación se ha utilizado esta herramienta para el registro de datos y su posterior reproducción.

2.3 *CARLA-ROS Bridge*

El puente de ROS con CARLA *Simulator* [9] es un conjunto de paquetes que permite la comunicación bidireccional entre los dos programas. Es decir, la información del servidor de CARLA se traduce en publicaciones en *topics* de ROS y, los mensajes enviados entre nodos en ROS se traducen en comandos que se aplican sobre CARLA.

El puente tiene las siguientes características:

- Proporciona toda la información obtenida por los sensores (LiDAR, cámaras, GNSS, IMU, etc.) de CARLA a través de *topics*.
- Proporciona datos de objetos como transformaciones, estado de los semáforos, marcadores de visualización, colisiones e invasión de carril.
- Control de agentes de conducción autónoma a través del volante, el acelerador y los frenos.
- Control de aspectos de la simulación de CARLA, como el modo síncrono, la reproducción y pausa de la simulación y la configuración de parámetros de simulación.

Los paquetes que proporciona están en su mayoría escrito en Python y se ejecutan utilizando los comandos habituales de ROS. Los paquetes que se han usado mayoritariamente en ese proyecto son:

- *CARLA ROS Bridge*. Paquete principal que se tiene que ejecutar para iniciar el puente y para que funcionen el resto de los paquetes. En su interior vienen definidas los parámetros por defecto, como por ejemplo, el tiempo meteorológico y el mapa que se va a cargar.

- *ROS Compatibility Node*. Este paquete sólo tiene uso interno por el programa. Lo que hace es diferenciar la versión de ROS que se está usando y traducir los comandos correspondientes a los utilizados por cada una.
- *CARLA Spawn Objects*. Paquete que proporciona una forma genérica de generar actores como vehículos, peatones, sensores y pseudo-sensores.

2.4 TFM de Eloy Vergara

El TFM de Eloy Vergara [10] es el predecesor a este TFG. En este trabajo se describe de una manera detallada el modelado del mapa de la ampliación del campus universitario de la universidad de Málaga y su importación a *CARLA Simulator*. Este TFM es del año 2020 por lo que utiliza la versión 0.9.7 del simulador, varias versiones anteriores a la que se utiliza en este TFG y la versión 4.22 de *Unreal Engine*. Para el modelado del campus universitario se utilizó el programa *Road Runner* (ver Figura 2.4).

Además, también se realizaron varias pruebas en el mapa variando las condiciones meteorológicas (despejado, nublado, lluvioso, etc) y la hora del día (amanecer, mañana, atardecer y noche) registrando los datos de diferentes sensores (LiDAR, cámaras RGB, sensores inerciales, GNSS, sensor de colisión, etc). Con estos datos se podían estudiar las consecuencias en los sensores de cada uno de los cambios meteorológicos y de la hora del día.

La memoria de este TFM [10] ha sido una herramienta valiosa para aprender cómo utilizar CARLA como plataforma de simulación y cómo llevar a cabo tareas esenciales en el simulador. Por ejemplo, proporciona instrucciones claras sobre cómo crear y configurar escenarios de conducción, cómo interactuar con los agentes de conducción autónoma y cómo utilizar los diferentes sensores y sistemas de control disponibles en CARLA.

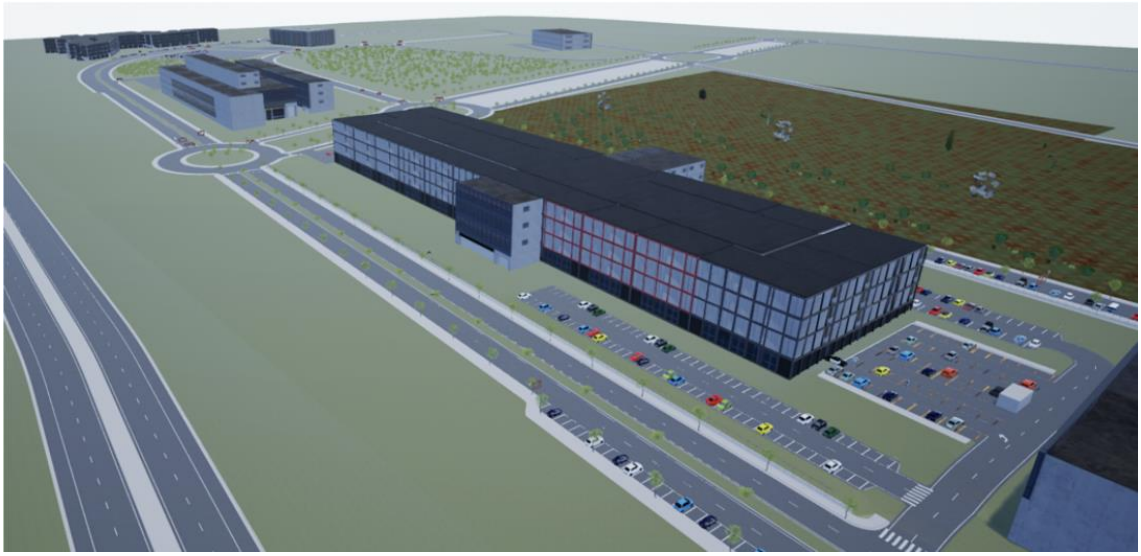


Figura 2.4. Entorno del Campus generado en el TFM [10]

3 Modelado de Sensores en *CARLA Simulator*

3.1 Incorporación de Sensores

CARLA proporciona una amplia biblioteca de sensores de manera que pueden ser configurados por medio de la modificación de distintos parámetros. Para el modelado de los distintos sensores se ha investigado las características de cada uno de los sensores, los parámetros configurables en el simulador de la documentación oficial de CARLA [11] y, por último, se han definido los parámetros a partir de las especificaciones técnicas.

Las características de cada uno de los sensores se definen por medio de un archivo JSON que utiliza el puente de ROS implementado en CARLA para generar los vehículos con todos los sensores y sus *topics* de publicación automáticamente. Este archivo está compuesto por objetos que pueden ser sensores, "pseudo-sensores" o agentes de la simulación, donde se pueden definir tanto su punto de aparición en la simulación como la posición de los sensores en el agente y sus características. El archivo utilizado para la definición de los sensores se llama "objects.json" o "objects_2.json", dependiendo de la ruta que se vaya a ejecutar, y se encuentra en el repositorio [12].

Para el proyecto PREMOVE se han escogido los siguientes modelos de sensores:

- LiDAR: RS-Helios 5515.
- Cámara RGB: ZED2i.
- Cámara térmica: Seek CompactPRO.
- Sensor GNSS: ZED F9P.
- Sensor IMU: MPU 9250.

3.1.1 LiDAR RS-Helios 5515

Este LiDAR 3D (ver Figura 3.1), desarrollado por la empresa Robosense [13] proporciona una distribución no uniforme de haces o canales que permite al usuario obtener información de un amplio campo de visión (FOV) vertical de 70º ([-15º, 55º]) teniendo una mayor precisión del entorno delante del vehículo y más dispersa en los extremos (ver Figura 3.2). De este modo con solo 32 canales de luz consigue reducir en gran medida los puntos ciegos del campo cercano a la vez que proporciona una percepción de largo alcance. Además, este instrumento cuenta con 2 frecuencias de medidas (10/20 Hz) proporcionando una nube de puntos densa (576.000/1.152.000 puntos/s).

Figura 3.1. RS-Helios 5515 [13]

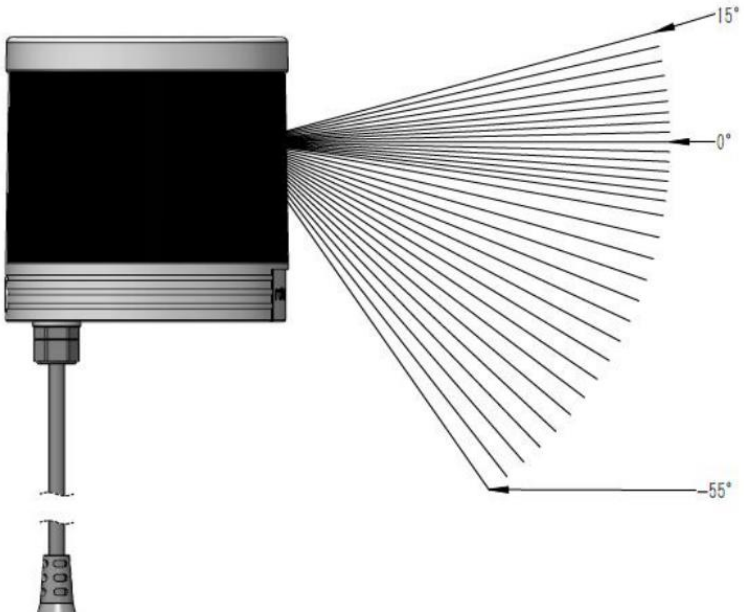


Figura 3.2. Distribución de canales láser [13]

La distribución no uniforme de canales se detalla en la Tabla 3.1, donde se indica el número de canal (*Channel No.*), su ángulo vertical (*Vertical Angle*), el rango (*Range*)

que alcanza y el 10% NIST, que indica el alcance para objetos con baja reflectividad. Analizando la tabla, vemos que efectivamente en los extremos la resolución es mayor que en el centro. Por ejemplo, entre los canales 17 y 32 la resolución es de 3º, mientras que entre los canales 7 y 16 es de 1.33º. Pero, además, observamos que el rango también varía. Siendo de 100 m o superior para los primeros 15 canales y decreciendo hasta 20 m para el último canal. Por esto se han utilizado 8 sensores virtuales (*sensor.lidar.ray_cast*) para modelar el RS-Helios 5515.

Tabla 3.1. Distribución de canales. Tabla extraída de [13]

Channel No.	*Vertical Angle(º)*	*10% NIST （m）*	*Range （m）*
1	15	90	100
2	13	90	100
3	11	90	100
4	9	90	100
5	7	90	100
6	5.5	90	100
7	4	90	150
8	2.67	90	150
9	1.33	90	150
10	0	90	150
11	-1.33	90	150
12	-2.67	90	100
13	-4	90	100
14	-5.33	90	100
15	-6.67	90	100
16	-8	90	50
17	-10	40	50
18	-16	40	50
19	-13	40	50
20	-19	40	50
21	-22	40	50
22	-28	40	50
23	-25	40	50
24	-31	40	50
25	-34	40	50

26	-37	40	50
27	-40	40	50
28	-43	40	50
29	-46	20	30
30	-49	20	30
31	-52	20	30
32	-55	10	20

Los parámetros que se pueden definir del sensor se pueden consultar en [11] y los valores que se le han otorgado se pueden observar en la Tabla 3.2. Para modelar cada uno se ha partido de la información de cada canal, se han agrupado los canales contiguos con características comunes y se ha establecido un LiDAR virtual para cada uno de los rangos. Todas las características se han obtenido de cada canal a excepción de la característica *points per second*, que se ha obtenido con la siguiente fórmula:

$$Points\ per\ second_i = 576.000 \cdot \frac{channels_i}{32} \tag{3.1}$$

donde *i* denota el número del LiDAR virtual.

Tabla 3.2. Parámetros en CARLA Simulator de los sensores LiDAR para el modelado del RS-Helios 5515

Características	LiDAR 0	LiDAR 1	LiDAR 2	LiDAR 3	LiDAR 4	LiDAR 5	LiDAR 6	LiDAR 7
Rotation frequency	10 Hz	10 Hz	10 Hz	10 Hz	10 Hz	10 Hz	10 Hz	10 Hz
Horizontal FOV	360º	360º	360º	360º	360º	360º	360º	360º
Upper FOV	15º	7º	4º	-2,67º	-8º	-13º	-46º	-55º
Lower FOV	9º	5,5º	-1,33º	-6,67º	-10º	-43º	-52º	-55º
Channels	4	2	5	4	2	11	3	1
Range	100 m	100 m	150 m	100 m	50 m	50 m	30 m	20 m
Point per second	72000	36000	90000	72000	36000	198000	54000	18000

De esta manera toda la información del LiDAR se reparte en 8 *topics* distintos. Por eso usamos un nodo de ROS, llamado “Fusion Lidar”, para unir esta información en un solo *topic*. El código de este nodo se encuentra en el repositorio [12]. Este nodo usa la librería de CARLA para obtener las nubes de puntos de cada uno de los sensores virtuales sincronizadamente en cada uno de los instantes de la simulación y lo publica en el *topic* correspondiente.

3.1.2 Cámara Estéreo ZED2i

La cámara estéreo ZED2i (ver Figura 3.3) es una cámara industrial fabricada por STEREOLABS [14] que proporciona un hardware resistente y un software potente que permite tener una visión 3D del entorno. Además de los 2 sensores RGB que permite que

el software obtenga una visión tridimensional del entorno, la cámara contiene un giroscopio, un acelerómetro y un magnetómetro.

Figura 3.3. Cámara ZED2i [14]

La resolución de la cámara tiene distintas configuraciones:

1. 2208x1242 a 15 FPS
2. 1920x1080 a 30 FPS
3. 1280x720 a 60 FPS
4. 662x376 a 100 FPS

Además de poder configurar la distancia focal entre 2.1 mm y 4 mm como observamos en la Figura 3.4. Para el TFG se utilizará la segunda configuración de la resolución y una distancia focal de 2.1 mm para una visión más amplia.

Figura 3.4. Diferencias entre distancias focales de la ZED2i [14]

En el proyecto PREMOVE se contarán con 3 cámaras de este modelo utilizando la información que nos proporciona todos sus sensores. Para el modelado en CARLA, se han utilizado 2 cámaras RGB con la misma configuración, una cámara de profundidad y un sensor IMU. Los parámetros configurables de cada uno de los actores se encuentran en [11]. El valor de los parámetros de la cámara RGB (*sensor.camera.rgb)*, la cámara de profundidad (*sensor.camera.depth)* y el sensor IMU (*sensor.other.imu)* se encuentran en las Tabla 3.3, 3.4 y 3.5, respectivamente.

Tabla 3.3. Parámetros en CARLA Simulator de la cámara RGB para el modelado de la ZED2i

Características	Cámara RGB
FOV	110º
Fstop	1,8
Image Size X	1280
Image Size Y	720
Sensor tick	0,0333
Shutter Speed	0,0333

Fstop es el parámetro que modela la apertura de la cámara, es directamente su inversa. El *sensor tick* y el *shutter speed* modelan los FPS de la imagen, siendo esta la inversa de este valor. Como se observa hay parámetros que no se modelan. En el caso, de los campos de visión horizontal y diagonal se establecen automáticamente al definir el horizontal y el tamaño de la imagen. En el caso de la distancia focal ocurre algo parecido, ya que esta se encuentra estrechamente relacionada con el campo de visión y el tamaño de la imagen. Por ejemplo, si la distancia focal aumenta, el campo de visión se reduce y se realizaría un *zoom* en la imagen como se observa en la Figura 3.4.

Tabla 3.4. Parámetros en CARLA Simulator de la cámara RGBD para el modelado de la ZED2i

Características	Cámara de Profundidad
FOV	110º
Image Size X	1280
Image Size Y	720
Shutter tick	0,0333

Los parámetros de la Tabla 3.5 se han obtenido directamente calibrando el sensor con el software que nos proporciona el fabricante [15]. Los parámetros de la calibración se pueden obtener usando ROS, ROS 2, C++ o Python, entre otros. La instalación no conlleva ninguna dificultad y se ha utilizado el código de ejemplo que nos proporciona el fabricante. El valor de las deviaciones típicas se ha obtenido directamente de la matriz de covarianza otorgada por el programa. Mientras que el valor medio del giroscopio (*bias*) de cada uno de los ejes se ha realizado tomando varias muestras de 10000 puntos y realizando la media de éstos. Se ha cogido el valor más desfavorable de las muestras tomadas.

Tabla 3.5. Parámetros en CARLA Simulator del sensor IMU para el modelado de la ZED2i

Características	IMU
Noise Accelerometer Standard Deviation X	0.327931
Noise Accelerometer Standard Deviation Y	0.277766
Noise Accelerometer Standard Deviation Z	0.300159
Noise Gyroscopy Bias X	-0.005999
Noise Gyroscopy Bias Y	0.005426

Noise Gyroscopy Bias Z	0.003003
Noise Gyroscopy Standard Deviation X	0.008774
Noise Gyroscopy Standard Deviation Y	0.002645
Noise Gyroscopy Standard Deviation Z	0.003605

3.1.3 Cámara Térmica Seek CompactPRO

La cámara Seek CompactPRO es una cámara térmica fabricada por Seek Thermal [16] que permite obtener información de la temperatura del entorno desde cualquier dispositivo móvil de manera muy sencilla (ver Figura 3.5). Aunque pertenece a la serie *Compact* del fabricante, destaca entre sus compañeras por tener un gran campo de visión (32º) y la mayor frecuencia de imagen (15 Hz).

Figura 3.5. Seek CompactPRO [16]

CARLA Simulator, por el momento, no cuenta con una implementación de temperaturas superficiales por lo que no contempla cámaras térmicas entre sus sensores. En este trabajo se ha decidido modelar por medio de cámaras RGB (*sensor.camera.rgb)*. Los valores que se le ha asignado a cada parámetro se encuentran en la Tabla 3.6.

Tabla 3.6. Parámetros en CARLA Simulator de la cámara RGB para el modelado de la Seek CompactPro

Características	Cámara RGB
FOV	32º
Image Size X	32
Image Size Y	240
Sensor tick	0,0666
Shutter Speed	0,0666

3.1.4 Sensor GNSS ZED F9P

Este sensor es un receptor multibanda (GPS, GLONASS, Galileo and BeiDou) de bajo coste fabricado por ublox [17]. Destaca por su precisión centimétrica, su forma compacta y rápida convergencia, además de su fácil integración RTK (ver Figura 3.6). En su hoja de especificaciones podemos encontrar las diferentes características dependiendo del modo en el que se use.

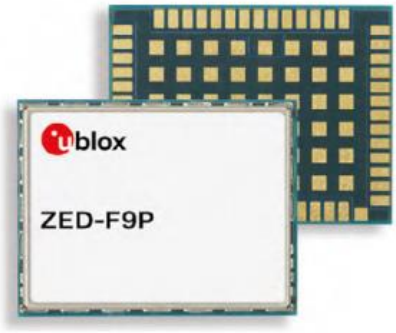


Figura 3.6. GNSS ZED F9P

Al modelar este sensor me encontré con el problema de que en los parámetros configurables solo se encuentran la media y la desviación típica de las magnitudes medidas [11], mientras que en la hoja de datos del producto no se hace referencia a ninguna de estas. Por lo que, se han tomado los valores de la Tabla 3.7 que se nos proporcionan en la Tabla 10 y la ecuación 5 de [18]. Para la definición de este sensor se utiliza el sensor GNSS de CARLA (*sensor.other.gnss).*

Tabla 3.7. Parámetros en CARLA Simulator del sensor GNSS para el modelado del GNSS F2P

Características	GNSS
Noise Altitud Standard Deviation	0.0098
Noise Latitud Standard Deviation	0.00001
Noise Longitud Standard Deviation	0.00001
Noise Altitud Bias	0.0
Noise Latitud Bias	0.0
Noise Longitud Bias	0.0

3.1.5 Sensor IMU MPU 9250

El sensor MPU 9250 (ver Figura 3.7) es una unidad inercial de bajo coste, compacto y de fácil integración que permite configurar el giroscopio y acelerómetro en 4 distintos rangos, desde ± 250 hasta ± 2000 º/s y desde $\pm 2g$ hasta $\pm 8g$, respectivamente [19].

Figura 3.7. Sensor IMU MPU 9250

De igual manera que ha ocurrido con el sensor GNSS, en la hoja de datos no encontramos los valores de los parámetros de CARLA. Así que, nos apoyamos en [20]. Los parámetros empleados se encuentran en la Tabla 3.8.

Tabla 3.8. Parámetros en CARLA Simulator del sensor IMU para el modelado del MPU 9250

Características	IMU
Noise Accelerometer Standard Deviation X	0.0129
Noise Accelerometer Standard Deviation Y	0.0134
Noise Accelerometer Standard Deviation Z	0.0201
Noise Gyroscopy Bias X	-0.0016
Noise Gyroscopy Bias Y	-0.0014
Noise Gyroscopy Bias Z	0.0020
Noise Gyroscopy Standard Deviation X	0.0007
Noise Gyroscopy Standard Deviation Y	0.0006
Noise Gyroscopy Standard Deviation Z	0.0006

3.2 Ubicación de los Sensores

La localización de los sensores dentro del cada agente se define también en el archivo JSON. Como se observa las coordenadas de posición que se indican son respecto al origen y los ejes del actor. Como se observa en la Figura 3.8 el origen del sistema de referencia de los vehículos se encuentra en su centro geométrico, a excepción de la posición vertical, que se encuentra en la base del actor. Mientras que en peatones (ver la Figura 3.9) se encuentra en su cadera. En ambos tipos de actores la dirección del eje X es hacia la cara anterior del actor y la dirección del eje Z hacia la cara superior. Además, de que observamos que CARLA no usa el sistema de la mano derecha. Esto hay que tenerlo en cuenta cuando definimos actores y sensores usando los comandos del simulador.

Figura 3.8. Ejemplos de vehículos de CARLA Simulator y de su sistema de referencia

La posición de los sensores que se van a usar va a depender del tipo de agente y de si se trata del vehículo principal o no. Esta calibración se describe en los siguientes apartados.

Figura 3.9. Ejemplos de peatones de CARLA Simulator y de su sistema de referencia

3.2.1 Colocación en el Vehículo Principal

El vehículo autónomo simulado va a contar con un sensor LiDAR, 3 cámaras estéreo, 3 cámaras térmicas y 2 sensores GNSS. Aunque como hemos comentado antes cada cámara estéreo incluye un sensor IMU. Todos los sensores irán montados sobre una baca instalada en el vehículo.

En PREMOVE se utilizará un Nissan Leaf [21]como coche autónomo, cuyas medidas podemos observar en la Figura 3.10. CARLA cuenta con un amplio catálogo [22] de vehículos entre los cuáles el que tiene las medidas más cercanas al Nissan es el Toyota Prius [23]que observamos en la Figura 3.11.

Figura 3.10. Medidas del Nissan Leaf. Medidas oficiales obtenidas del fabricante del vehículo [21]

La altura de la plataforma se ha establecido a 1.60 m desde el origen del vehículo. Además, se ha establecido el origen de ésta como la posición donde se colocará el LiDAR, que se encuentra en el centro de la anchura del vehículo y desplazado 0.2 m respecto al origen del vehículo como se observa en la Figura 3.12. Se ha planteado una plataforma que tenga aproximadamente el tamaño del techo (1.2x1.1 m), aunque también se puede utilizar una plataforma de 0.3x1.1m, de este modo el origen de la plataforma coincidirá con el centro de ésta. Además, su matriz de transformación, con las unidades en metros, vale:

$$ {}^{C}T_{P} = \begin{pmatrix} 1 & 0 & 0 & 0.2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1.6 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2} $$

Figura 3.11. Comparación visual entre el modelo real del Toyota Prius y el de CARLA [22] [23]

Figura 3.12. Sistema de referencia de la plataforma de sensores

Desde este punto se procede a establecer las posiciones de los sensores que irán incorporados en el vehículo. El esquema de la distribución de los sensores se encuentra en la Figura 3.13. En esta figura, como los siguientes esquemas de la plataforma, se observa que el eje Y definido tiene el sentido contrario al utilizado en CARLA. Esto es porque los sensores, así como las ubicaciones de los agentes se definen desde un archivo JSON, utilizando ROS 2, el cuál si usa el sistema de la mano derecha.

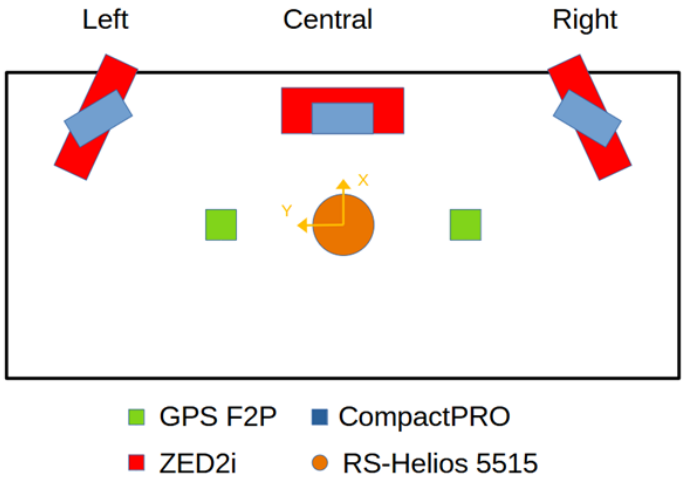


Figura 3.13. Distribución de sensores sobre la plataforma

En los siguientes subapartados se calcularán las matrices de transformación de cada uno de los sensores, calculando de este modo el vector de translación al origen de cada sensor desde el origen del agente. También se indicará el giro que se produce en torno al eje Z de cada uno de ellos.

3.2.1.1 Cámaras ZED2i

Como se ha visto en la Figura 3.13, se dispondrán 3 cámaras una central y una a cada lado del vehículo. Recordemos que por cada modelo de cámara ZED2i en CARLA se modelan 2 cámaras RGB, una cámara de profundidad y un sensor IMU. La cámara de profundidad y el sensor IMU se colocan en el centro geométrico del dispositivo. Mientras

que las 2 cámaras RGB tendrán un desplazamiento lateral respecto a este origen como se observa en la Figura 3.14. Sus matrices de transformación, en metros, son:

$$ {}^{CRGB}T_L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.06 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad {}^{CRGB}T_R = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.06 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3} $$

Las matrices de transformación del centro de la plataforma al origen de las cámaras de la izquierda, central y de la derecha se encuentran en son:

$$ {}^{P}T_{CRGB_L} = \begin{pmatrix} \cos(65º) & -sen(65º) & 0 & 0.1 \\ sen(65º) & \cos(65º) & 0 & 0.4 \\ 0 & 0 & 1 & 0.015 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4} $$

$$ {}^{P}T_{CRGB_C} = \begin{pmatrix} 1 & 0 & 0 & 0.1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.015 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.5} $$

$$ {}^{P}T_{CRGB_R} = \begin{pmatrix} \cos(-65º) & -sen(-65º) & 0 & 0.1 \\ sen(-65º) & \cos(-65º) & 0 & -0.4 \\ 0 & 0 & 1 & 0.015 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.6} $$

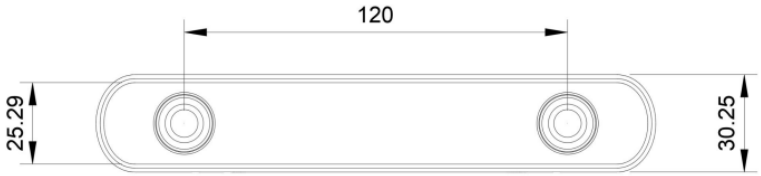


Figura 3.14. Geometría de la cámara ZED2i (dimensiones en mm)

Una vez definidas todas las matrices podemos calcular la posición del origen de cada uno de los sensores respecto al origen del vehículo. El origen de las cámaras ZED2i (izquierda, central y derecha), y, por tanto, la posición de las cámaras de profundidad y de los sensores IMU son:

$$ O = (0 \quad 0 \quad 0 \quad 1)^T \tag{3.7} $$

$$ {}^{C}O_{CRGB_L} = {}^{C}T_P \cdot {}^{P}T_{CRGB_L} \cdot O = \begin{pmatrix} 0.3 \\ 0.4 \\ 1.615 \\ 1 \end{pmatrix} \tag{3.8} $$

$$ {}^{C}O_{CRGB_C} = {}^{C}T_P \cdot {}^{P}T_{CRGB_C} \cdot O = \begin{pmatrix} 0.3 \\ 0 \\ 1.615 \\ 1 \end{pmatrix} \tag{3.9} $$

$$ {}^{C}O_{CRGB_R} = {}^{C}T_P \cdot {}^{P}T_{CRGB_R} \cdot O = \begin{pmatrix} 0.3 \\ -0.4 \\ 1.615 \\ 1 \end{pmatrix} \tag{3.10} $$

Por su parte, la posición de cada pareja de cámaras RGB valdrán:

$$ \begin{aligned} {}^{C}O_{CRGB_{LL}} &= {}^{C}T_P \cdot {}^{P}T_{CRGB_L} \cdot {}^{CRGB}T_L \cdot O = \begin{pmatrix} 0.2456 \\ 0.4254 \\ 1.615 \\ 1 \end{pmatrix} \\ {}^{C}O_{CRGB_{LR}} &= {}^{C}T_P \cdot {}^{P}T_{CRGB_L} \cdot {}^{CRGB}T_R \cdot O = \begin{pmatrix} 0.3544 \\ 0.3746 \\ 1.615 \\ 1 \end{pmatrix} \end{aligned} \tag{3.11} $$

$$ \begin{aligned} {}^{C}O_{CRGB_{CL}} &= {}^{C}T_P \cdot {}^{P}T_{CRGB_C} \cdot {}^{CRGB}T_L \cdot O = \begin{pmatrix} 0.3 \\ 0.06 \\ 1.615 \\ 1 \end{pmatrix} \\ {}^{C}O_{CRGB_{CR}} &= {}^{C}T_P \cdot {}^{P}T_{CRGB_C} \cdot {}^{CRGB}T_R \cdot O = \begin{pmatrix} 0.3 \\ -0.06 \\ 1.615 \\ 1 \end{pmatrix} \end{aligned} \tag{3.12} $$

$$ \begin{aligned} {}^{C}O_{CRGB_{RL}} &= {}^{C}T_P \cdot {}^{P}T_{CRGB_R} \cdot {}^{CRGB}T_L \cdot O = \begin{pmatrix} 0.3544 \\ -0.3746 \\ 1.615 \\ 1 \end{pmatrix} \\ {}^{C}O_{CRGB_{RR}} &= {}^{C}T_P \cdot {}^{P}T_{CRGB_R} \cdot {}^{CRGB}T_R \cdot O = \begin{pmatrix} 0.2456 \\ -0.4254 \\ 1.615 \\ 1 \end{pmatrix} \end{aligned} \tag{3.13} $$

La rotación respecto del eje Z que efectúa cada sensor modelado para cada cámara es de 65º, 0º y -65º, según si es la cámara de la izquierda, del centro o de la derecha. Por último, se dispone un esquema a escala en la Figura 3.15 con el campo de visión de cada cámara ZED2i.

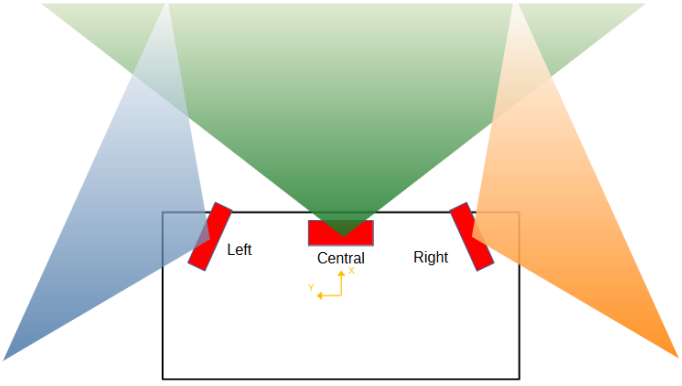


Figura 3.15. Campo de visión de las cámaras ZED2i en la baca

3.2.1.2 Cámaras CompactPRO

También se dispondrán de 3 cámaras térmicas con una distribución similar. En este caso, se han colocado las cámaras sobre el mismo punto que las anteriores, pero suponiendo que se encuentran por encima de estas sobre una cierta base. De este modo, no interfieren una con la otra. Se ha establecido que el centro de la lente se encuentre a 5 cm de altura sobre la plataforma. Por lo que las matrices de transformación de cada una de ellas valen:

$$ {}^{P}T_{CT_L} = \begin{pmatrix} \cos(31º) & -sen(31º) & 0 & 0.1 \\ sen(31º) & \cos(31º) & 0 & 0.4 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.14} $$

$$ {}^{P}T_{CT_C} = \begin{pmatrix} 1 & 0 & 0 & 0.1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.15} $$

$$ {}^{P}T_{CT_R} = \begin{pmatrix} \cos(-31º) & -sen(-31º) & 0 & 0.1 \\ sen(-31º) & \cos(-31º) & 0 & -0.4 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.16} $$

Sus respectivas posiciones, respecto al origen del vehículo son:

$$ {}^{C}O_{CT_L} = {}^{C}T_{P} \cdot {}^{P}T_{CT_L} \cdot O = \begin{pmatrix} 0.3 \\ 0.4 \\ 1.65 \\ 1 \end{pmatrix} \tag{3.17} $$

$$ {}^{C}O_{CT_C} = {}^{C}T_{P} \cdot {}^{P}T_{CT_C} \cdot O = \begin{pmatrix} 0.3 \\ 0 \\ 1.65 \\ 1 \end{pmatrix} \tag{3.18} $$

$$ {}^{C}O_{CT_R} = {}^{C}T_{P} \cdot {}^{P}T_{CT_R} \cdot O = \begin{pmatrix} 0.3 \\ -0.4 \\ 1.65 \\ 1 \end{pmatrix} \tag{3.19} $$

Las rotaciones de la cámara central, de la izquierda y de la derecha son 0º, 31º y -31º, respectivamente (ver Figura 3.16).

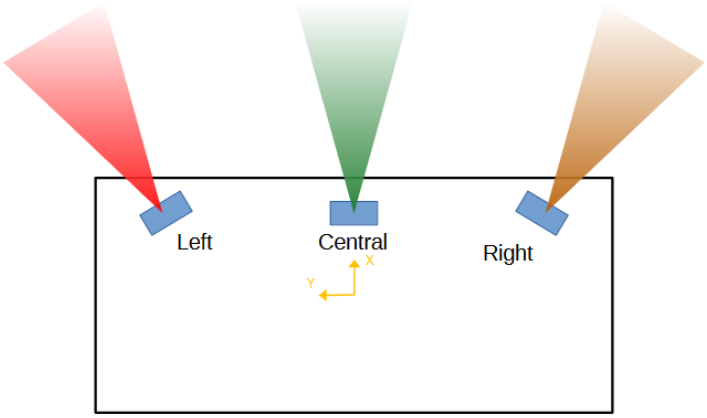


Figura 3.16. Campo de visión de las cámaras CompactPRO en la plataforma

3.2.1.3 LiDAR RS-Helios 5515

Como ya se ha comentado el LiDAR se establecerá en el origen de la plataforma, además de que no sufrirá ninguna rotación, por lo que solo se desplazará verticalmente. Este desplazamiento vertical tiene que ser lo suficientemente alto para que no interfieran el resto de los sensores en las mediciones o lo menos posible. Por ello, se ha escogido una altura de 0.2635 metros sobre la plataforma, 20 cm de margen más 6.35 cm, que es la altura donde se originan los rayos respecto a la base del sensor (ver Figura 3.17).

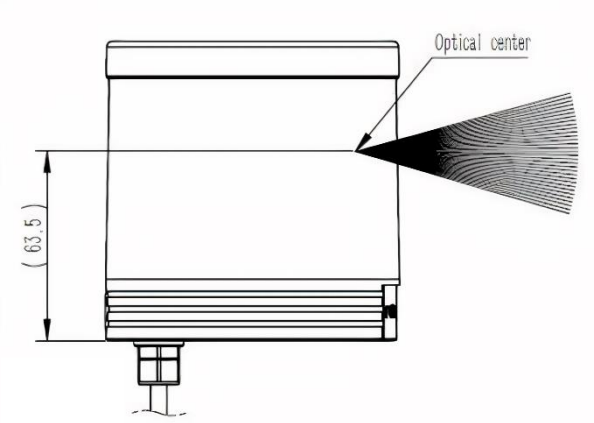


Figura 3.17. Centro óptico del RS-Helios 5515. Extraída del manual de usuario del fabricante [13]

Observamos su matriz de transformación y su posición a continuación:

$$ {}^{P}T_{LD} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.2635 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.20} $$

$$ {}^{C}O_{LD} = {}^{C}T_{P} \cdot {}^{P}T_{LD} \cdot O = \begin{pmatrix} 0.2 \\ 0 \\ 1.8635 \\ 1 \end{pmatrix} \tag{3.21} $$

3.2.1.4 Sensores GNSS F9P

En la Figura 3.13, observamos que los 2 sensores GNSS se encuentran en los puntos medios entre los limites laterales de la plataforma y el origen de esta. Estos sensores solo sufren una traslación. Las matrices de transformación y sus respectivas posiciones valen:

$$ {}^{P}T_{GPS_L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.22} $$

$$ {}^{P}T_{GPS_R} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.23} $$

$$ {}^{C}O_{GPS_L} = {}^{C}T_{P} \cdot {}^{P}T_{GPS_L} \cdot O = \begin{pmatrix} 0.2 \\ 0.2 \\ 1.6 \\ 1 \end{pmatrix} \tag{3.24} $$

$$ {}^{C}O_{GPS_R} = {}^{C}T_{P} \cdot {}^{P}T_{GPS_R} \cdot O = \begin{pmatrix} 0.2 \\ -0.2 \\ 1.6 \\ 1 \end{pmatrix} \tag{3.25} $$

3.2.2 Colocación en los Agentes de la Simulación

Cada agente de la simulación contará con un sensor IMU y un sensor GNSS los cuales son los mismos que se han utilizado en el vehículo principal. En el caso de peatones y ciclistas se le colocarán los sensores en un casco. En el caso de los vehículos se colocarán en el techo de estos. En el simulador, debido a la posición de los ejes de coordenadas solo se ha añadido un desplazamiento vertical, además de establecer una separación de 10 cm entre sensores. Las matrices de transformación y la posición respecto al origen de cada agente no sufren ninguna rotación:

$$ {}^{C}T_{GPS_{ag}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & alt_{ag} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad {}^{C}O_{GPS_{ag}} = \begin{pmatrix} 0 \\ 0 \\ alt_{ag} \\ 1 \end{pmatrix} \tag{3.26} $$

$$ {}^{c}T_{IMU_{ag}} = \begin{pmatrix} 1 & 0 & 0 & 0.1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & alt_{ag} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad {}^{c}O_{IMU_{ag}} = \begin{pmatrix} 0.1 \\ 0 \\ alt_{ag} \\ 1 \end{pmatrix} \tag{3.27} $$

En el caso, de los peatones $alt_{ag} = 0.93\ m$, ya que la altura de los mismos es de 1.86 m y su sistema de referencia está situado en su cadera (ver Figura 3.9). En el caso de los vehículos de 2 ruedas el valor debe ser medido en el simulador. Por último, en el resto de los vehículos, alt_{ag} equivale a la altura del vehículo que, en este caso, varía entre 1.6 y 1.7 m, a excepción del autobús que alcanza los 4.3 m.

4 Experimentos

4.1 Aspectos Generales

Los experimentos que se van a realizar son simulaciones controladas, es decir, se especificará la trayectoria exacta que debe de seguir el vehículo principal, así como, la trayectoria que seguirá cada uno de los agentes de la simulación. Cada una de las simulaciones contará con tres automóviles adicionales, un autobús, dos ciclistas y cinco peatones.

En total se realizarán cuatro simulaciones, las cuales se agruparán en parejas. Cada par tendrá un entorno de simulación idéntico, pero se cambiará la situación meteorológica entre ellas.

A continuación, se describe como se la información sensorial en ROS y cómo se generan las rutas de todos los participantes antes de comentar las pruebas realizadas.

4.2 Publicación de la Información en ROS 2

Como se ha comentado anteriormente para generar los agentes con los sensores incorporados se utilizará un archivo JSON. Esto se realiza de este modo porque por medio de este archivo el puente CARLA-ROS es capaz de generar cada uno de los *topics* de los sensores y publicar su información. Se puede observar un ejemplo de la definición de un agente en la Figura 4.1.

```
{
 "type": "walker.pedestrian.0003",
 "id": "walker_1",
 "spawn_point": {"x": -63.0, "y": -6.2, "z": 2.0, "roll": 0.0, "pitch": 0.0,
                 "yaw": 0.0},
  "sensors":
           [
            {
             "type": "sensor.other.gnss",
             "id": "GNSS",
             "spawn_point": {"x": 0.0, "y": 0.0, "z": 0.93, "roll": 0.0,
                             "pitch": 0.0, "yaw": 0.0},
              "noise_alt_stddev": 0.01, "noise_lat_stddev": 0.02,
              "noise_lon_stddev": 0.02, "noise_alt_bias": 0.0,
              "noise_lat_bias": 0.0, "noise_lon_bias": 0.0
            },
            {
             "type": "sensor.other.imu",
             "id": "IMU",
             "spawn_point": {"x": 0.1, "y": 0.0, "z": 0.93, "roll": 0.0,
                             "pitch": 0.0, "yaw": 0.0},
            "noise_accel_stddev_x": 0.0129, "noise_accel_stddev_y": 0.0132,
              "noise_accel_stddev_z": 0.0201, "noise_gyro_stddev_x": 0.0134,
              "noise_gyro_stddev_y": 0.0233, "noise_gyro_stddev_z": 0.0006,
             "noise_gyro_bias_x": -0.0006, "noise_gyro_bias_y": 0.0004,
              "noise_gyro_bias_z": 0.0020
            }
           ]
}
```

Figura 4.1. Ejemplo de definición de agente en archivo JSON

Los nombres de los *topics* de publicación de cada sensor en CARLA siguen el siguiente formato:

/carla/ [<NOMBRE DEL AGENTE>]/ [<NOMBRE DEL SENSOR>]

A excepción de las cámaras que disponen de dos *topics* uno en el que se publica la imagen (/carla/…/image) y otro en el que se publica la información de la cámara (/carla/…/camera_info). La información sobre los nombres de los *topics* dependiendo del sensor y del tipo del mensaje de cada uno los tenemos en [24].

En la simulación, se ha definido los siguientes nombres para cada vehículo:

- Vehículo principal: *ego_vehicle*.
- Coches: *car_1*, *car_2* y *car_3*.
- Autobús: *bus*
- Ciclistas: *bicycle_1* y *bicycle_2*.
- Peatones: *walker_1*, *walker_2*, *walker_3*, *walker_4* y *walker_5*.

Por ejemplo, los *topics* del primer peatón son:

/carla/walker_1/IMU

/carla/walker_1/GNSS

Recordemos que para el modelo del LiDAR se han definido 8 agentes del tipo *sensor.lidar.ray_cast*, lo que son 8 *topics* por lo que se distribuye la información del LiDAR. Así que se utiliza un nodo de ROS externo para unir la información de los 8 sensores y se publica en: /carla/ego_vehicle/LIDAR.

La manera en la que se inicializa el puente CARLA-ROS, se generan los agentes y se empieza a publicar información en los nodos es la siguiente, aunque se encuentra de una manera más detallada en el Anexo C.

1. Iniciar servidor de CARLA.
2. Iniciar el puente de ROS con el siguiente comando en un terminal:

```
ros2 launch carla_ros_bridge carla_ros_bridge.launch.py
passive:=True
```

3. Ejecutar el archivo que genera los objectos (todo en una misma línea) en otro terminal:

```
ros2 launch carla_spawn_objects carla_spawn_objects.launch.py
objects_definition_file:=path/to/objects.json
```

4. Ejecutar el nodo que fusiona el LiDAR en un solo topic, este nodo se encuentra en el *workspace TFG_ws*:

```
ros2 run my_python_TFG fusion_lidar
```

4.3 Generación de Rutas

La generación de rutas es una función que se ha añadido en las últimas versiones de CARLA. Esto provoca que haya que ser muy específicos y tener especial cuidado con que sean posibles los caminos generados. En el momento en que estos caminos no sean posibles de seguir, el vehículo seguirá un camino aleatorio.

La guía para generar rutas se encuentra en el tutorial que se encuentra en [25]. El modo de generar rutas para vehículos que tiene el simulador de CARLA consiste en darle el control del vehículo al *Traffic Manager* y pasarle los puntos que queremos que siga el vehículo.

En primer lugar, definimos el punto de generación y la ruta que queremos que siga el vehículo. Para ello, dentro de CARLA existen unos puntos de generación predefinidos en cada mapa que nos facilitan la generación de las rutas. Para obtener estos puntos y visualizarlos en el mapa se utilizan los siguientes comandos que vemos en la Figura 4.2. El resultado de ejecutarlos se encuentra en la Figura 4.3. Los *spawn points* son vectores que indican la posición (x, y, z) y la orientación (*roll*, *pitch*, *yaw*) que se quiere que tenga el vehículo en ese punto.

```
spawn_points = world.get_map().get_spawn_points()
# Draw the spawn point locations as numbers in the map
for i, spawn_point in enumerate(spawn_points):
    world.debug.draw_string(spawn_point.location, str(i), life_time=10)
```

Figura 4.2. Código para la aparicón de los "spawn points"

Figura 4.3. Puntos de generación en la Town10

Hay que tener cuidado con la versión de CARLA instalada, ya que los índices de estos puntos varían dependiendo de si se ha instalado la versión *package* o la versión *source*. El archivo utilizado para generar que vemos en la Figura 4.3 se ha generado ejecutando el archivo *spawn_points.py* que se encuentra en [12]. Este archivo hay que ejecutarlo con el siguiente comando dentro de la carpeta donde se encuentra y una vez que se ha iniciado el servidor:

```
python3 spawn_points.py
```

Una vez que visualizamos los puntos sobre el mapa podemos ir seleccionando los puntos de la ruta que queremos seguir y guardándolos en una lista. Posteriormente, a partir de ésta se genera otra lista donde se guardan las posiciones de los puntos y que se le pasa al *Traffic Manager* para genere la ruta (ver Figura 4.4).

```
spawn_point_1 =  spawn_points[32]
# Create route 1 from the chosen spawn points
route_1_indices = [129, 28, 124, 33, 97, 119, 58, 154, 147]
route_1 = []
for ind in route_1_indices:
    route_1.append(spawn_points[ind].location)
vehicle = world.try_spawn_actor(vehicle_bp, spawn_point_1)
if vehicle: # IF vehicle is succesfully spawned
            vehicle.set_autopilot(True) # Give TM control over vehicle
            # Set parameters of TM vehicle control, we don't want lane changes
            traffic_manager.update_vehicle_lights(vehicle, True)
            traffic_manager.random_left_lanechange_percentage(vehicle, 0)
            traffic_manager.random_right_lanechange_percentage(vehicle, 0)
            traffic_manager.auto_lane_change(vehicle, False)
            traffic_manager.set_path(vehicle, route_1)
```

Figura 4.4. Código para la generación de rutas de vehículos

El ejemplo que aparece en el tutorial se encuentra accesible y listo para ejecutar en *route_example.py* en el repositorio [12]. Hay que tener en cuenta que el comando *set_path()* es un comando que se ha incluido en las últimas versiones de CARLA, por lo que no está totalmente depurado. Además, de tener particular cuidado con los caminos

que generamos vamos a tener que combinar otros algoritmos de control como el comando *force_change_lane()*, que nos permite forzar el cambio de carril del vehículo.

Este es el método que se ha utilizado para el seguimiento de rutas en vehículos ya que se dispone de la facilidad de puntos de referencia. Pero todos los puntos de referencia otorgados por CARLA se encuentran en la calzada. Por lo que, para generar rutas para peatones se utiliza otro método. Además de que los peatones son actores distintos a los vehículos por lo que no se controlan como éstos.

El método utilizado para definir las rutas que siguen los peatones ha sido el siguiente:

1. Una vez iniciada la simulación de CARLA, llevar al espectador al punto que queremos incluir en la ruta.
2. Ejecutar el archivo *actors_location.py*. Este archivo devuelve la ubicación de todos los vehículos y peatones de la simulación. También devuelve la localización del espectador.
3. Guardar los valores del punto en una variable.
4. Crear el control del peatón e indicarle la ruta que tiene que seguir.

Los comandos necesarios para controlar un peatón los vemos en la Figura 4.5 y un ejemplo de un peatón siguiendo una ruta específica se obtiene al ejecutar el archivo “route_pedestrian_example.py”.

```
point1=carla.Location(x=-86.9,y=3.88,z=2.0)

walker_controller_bp = world.get_blueprint_library().find('controller.ai.walker')
walker_1_control=world.try_spawn_actor(walker_controller_bp, carla.Transform(),
walker_1)

walker_1_control.start()
walker_1_control.go_to_location(point1_pos)
walker_1_control.set_max_speed(0.5)
world.tick()
while True:
    world.tick()
```

Figura 4.5. Código para la generación de rutas de peatones

4.4 Pruebas

La primera pareja de pruebas se realizará en la ciudad *Town10* de CARLA. La ruta que seguirá el *ego vehicle* y el resto de los actores se observa en la Figura 4.6. El vehículo sigue una ruta sencilla en la empieza y termine casi en el mismo punto. A lo largo de la ruta se encuentra tanto vehículos en el sentido de circulación contrario como en el mismo sentido. Lo mismo ocurre con bicicletas. También se encuentra con peatones paseando, así como peatones quietos que simulan tener una conversación. Por último, también se encuentra con un peatón que está cruzando la calle.

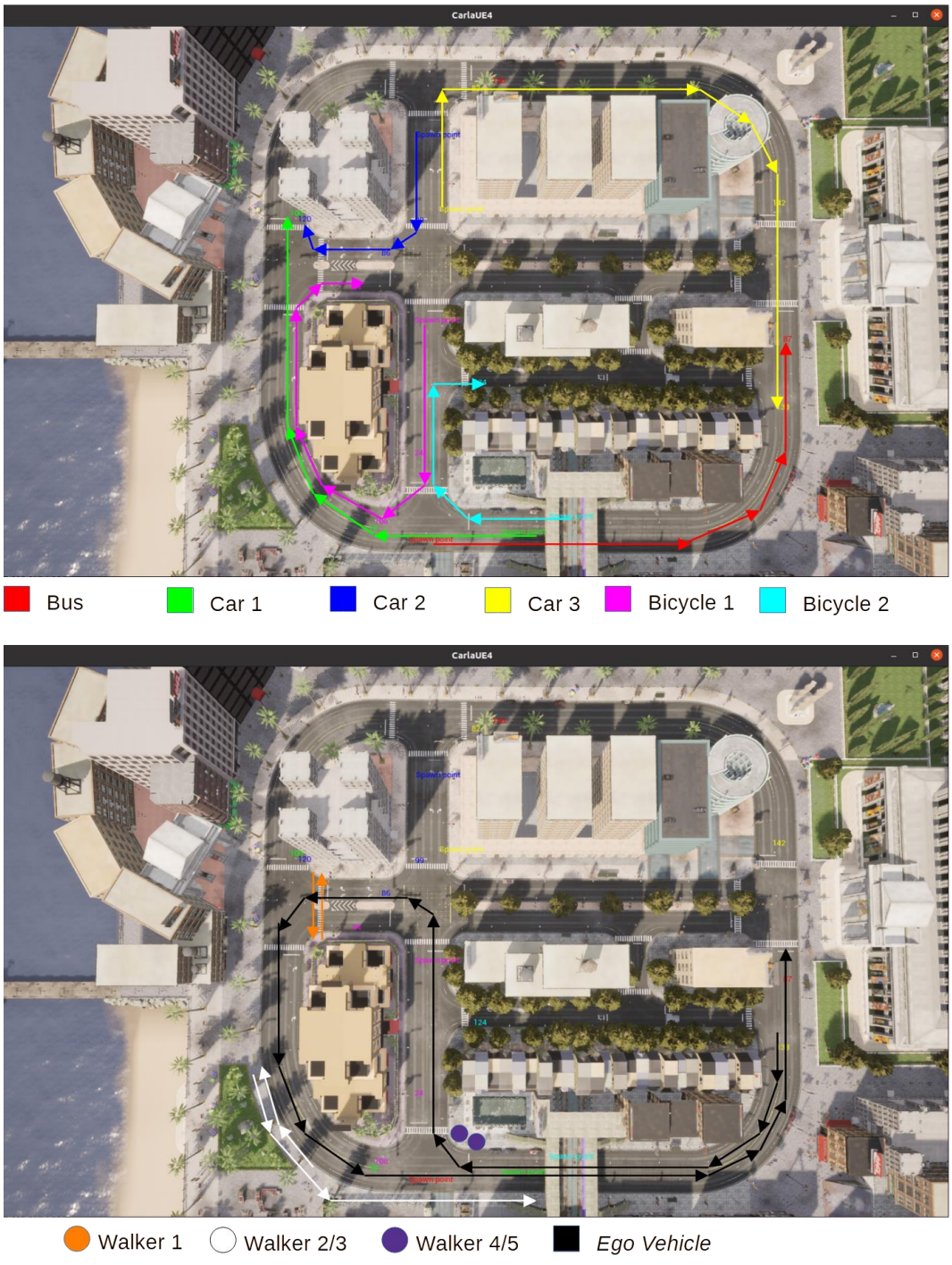


Figura 4.6. Trayectoria de los agentes en la primera pareja de pruebas

La segunda pareja de pruebas se realizará en la ciudad *Town03* de CARLA. La ruta que seguirá del *ego vehicle* y el resto de los actores se observa en la Figura 4.7. Esta ciudad es más grande y cuenta con zonas a distintas alturas, rotondas y cruces de mayor dificultad. Aunque en la simulación no se ha incluido, también cuenta con túneles y zonas cubiertas por un tranvía.

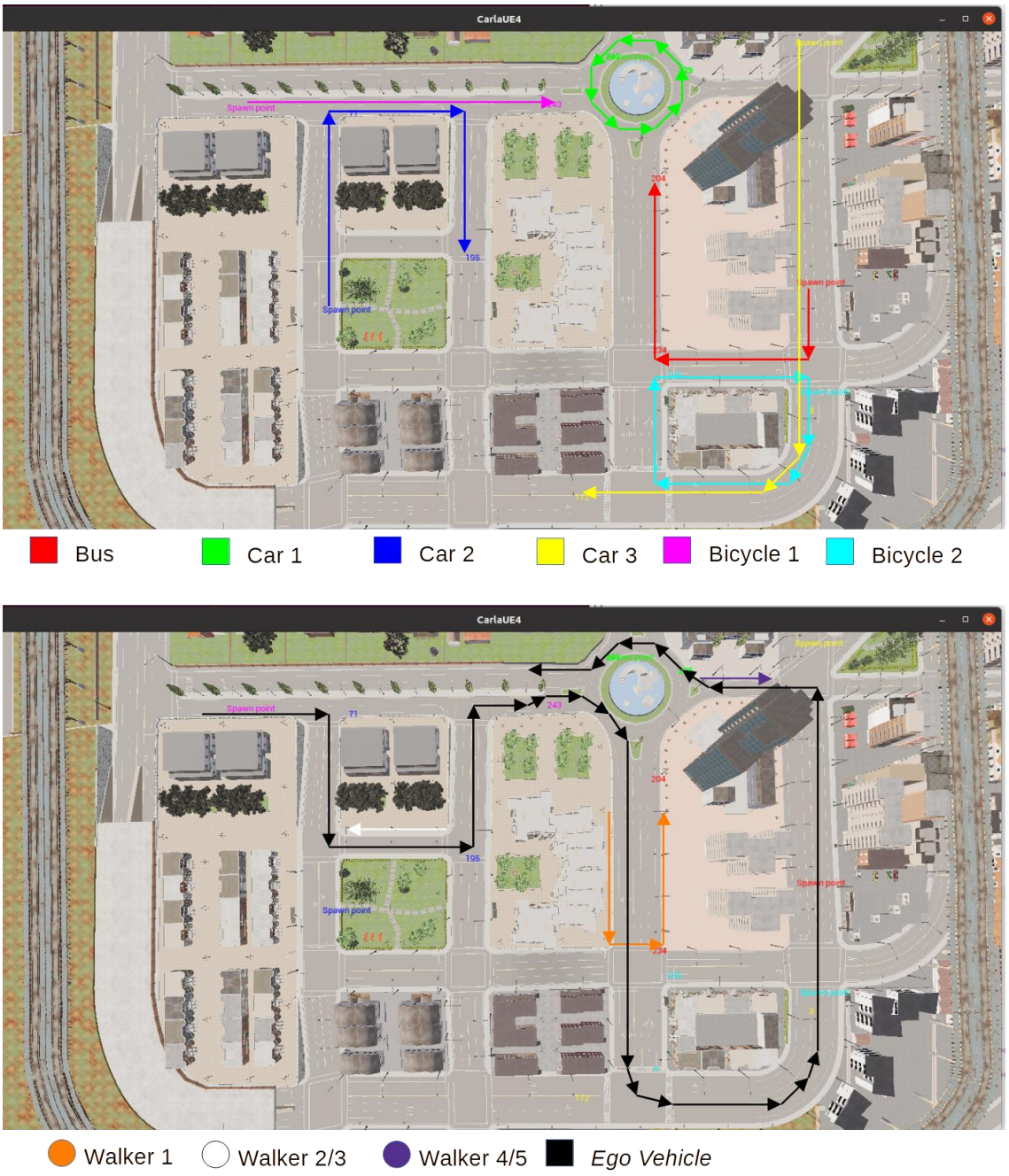


Figura 4.7. Trayectoria de los agentes en la segunda pareja de pruebas

Para ejecutar las rutas hay que seguir los siguientes pasos (Para una guía más detallada ver el Anexo C).

1. Iniciar el servidor del simulador en su versión *package*.
2. Iniciar el puente de ROS.
3. Ejecutar el *launch* “objects_route_1.json”.
4. Ejecutar el archivo “route_[<número de ruta>].py” correspondiente.

La diferencia entre los archivos de cada pareja de rutas es la inclusión del comando que vemos en la Figura 4.8. Este comando cambia el tiempo meteorológico y la hora del día, estableciendo las diferencias que observamos en la Figura 4.9. Debemos ser conscientes de que estos cambios solo afectan a las cámaras RGB.

```
atributos.weather=carla.WeatherParameters(precipitation=100.0,
sun_altitude_angle = 20.0, precipitation_deposits= 80.0)
world.set_weather(atributos.weather)
```

Figura 4.8. Código para la cambiar el tiempo

Los resultados en formato vídeo y de donde se han extraído las capturas que veremos en los próximos apartados se encuentran en [26].

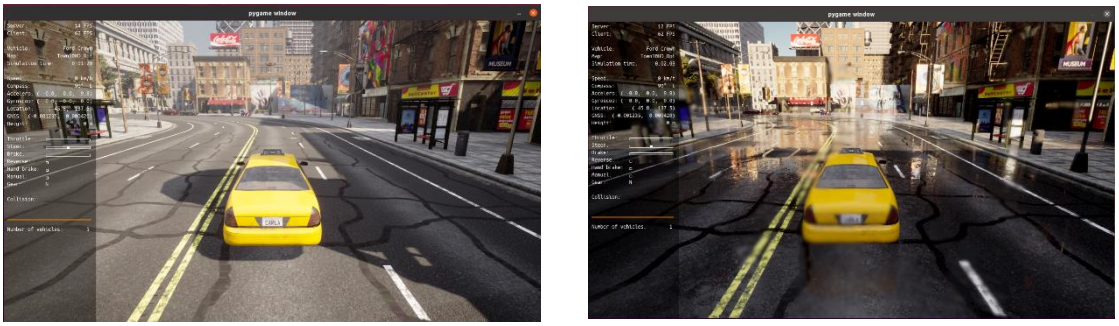


Figura 4.9. Comparación entre los tiempos meteorológicos.

Para el análisis de datos hay que tener en cuenta dos aspectos. Por una parte, el tiempo de la simulación empieza cuando se inicia el servidor de CARLA y no cuando se inicia cada una de las rutas, por eso en las gráficas que veremos en los resultados el tiempo no empieza en el instante cero, sino en el momento que se inicie la correspondiente ruta. Y, por otra parte, la ubicación de referencia geográfica del mapa se encuentra en el origen de éste.

4.4.1 Resultados de la Ruta 1.1

La Figura 4.10 muestra la vista de las 6 cámaras RGB implementadas en el vehículo. En esta figura podemos observar cómo con la disposición y configuración propuesta se tiene una vista de 180º, incluyendo el solapamiento de los campos de visión que se observa en la Figura 3.15. Así, en este ejemplo, el autobús aparece desde 4 puntos de vista distintos. Lo mismo ocurre con la moto sobre la acera de la derecha. También se observa la alta calidad de las imágenes que proporciona la cámara y su gran campo de visión.

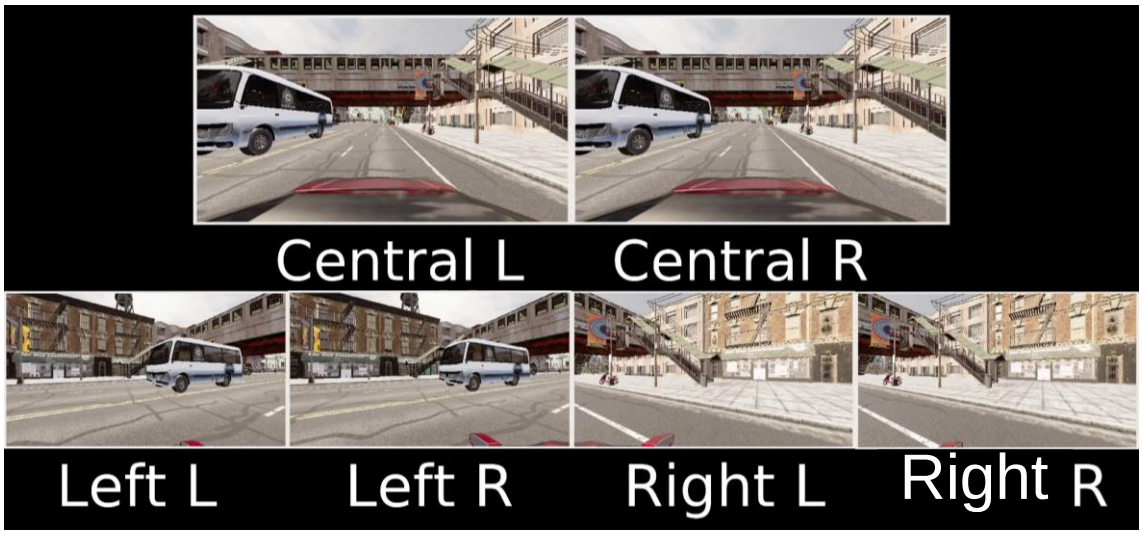


Figura 4.10. Vista del autobús con las cámaras ZED2i modeladas

En la Figura 4.11 se ve la vista de la cámara de profundidad complementaria a esos pares de cámaras RGB. Se observa nuevamente cómo se tiene una vista completa del entorno frontal del vehículo y como se diferencian las formas de los objetos más próximos.

Figura 4.11. Vista del autobús con las cámaras ZED2i modeladas (profundidad)

Podemos observar el mismo instante de la simulación, pero esta vez cómo se observaría con las cámaras térmicas (ver Figura 4.12). Comprobamos que estas cámaras tienen una menor resolución y que su campo de visión es más reducido, generando un *zoom* de la imagen. Por ejemplo, la parte delantera del vehículo queda fuera del rango de visión. Por otro lado, también se observa que la distribución de las cámaras proporciona una vista casi continua del entorno obteniendo la visión del mayor campo posible.

Figura 4.12. Vista del autobús con las cámaras Seek CompactPro modeladas

En la Figura 4.13, se observa que con la información que nos da el LiDAR tenemos una vista aproximada de todo lo que rodea al vehículo. Se observa la aparición de vehículo de gran tamaño por el carril izquierdo (círculo naranja) y también podemos diferenciar cerca de las escaleras un pequeño cúmulo de puntos que corresponden a la moto (círculo verde).

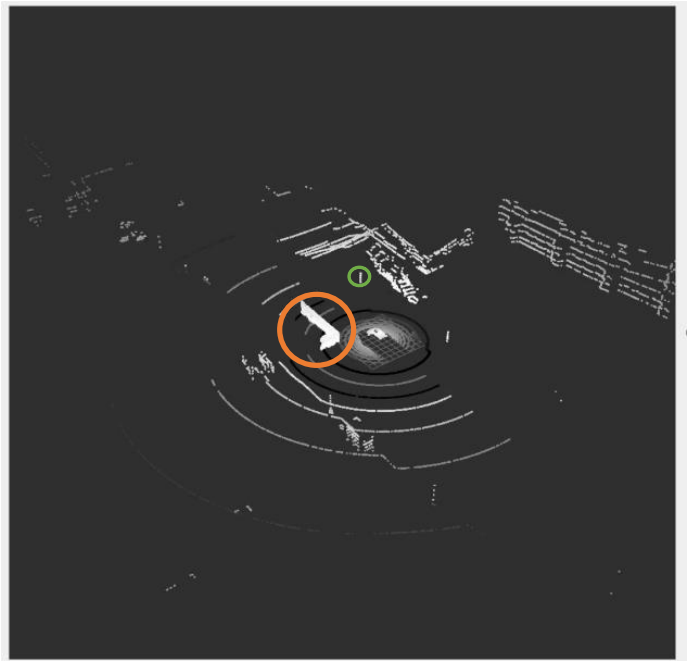
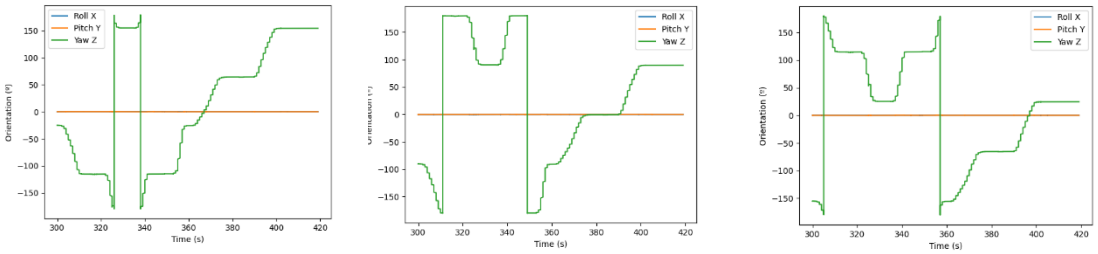


Figura 4.13. Vista del autobús con el LiDAR RS-Helios 5155 modelado

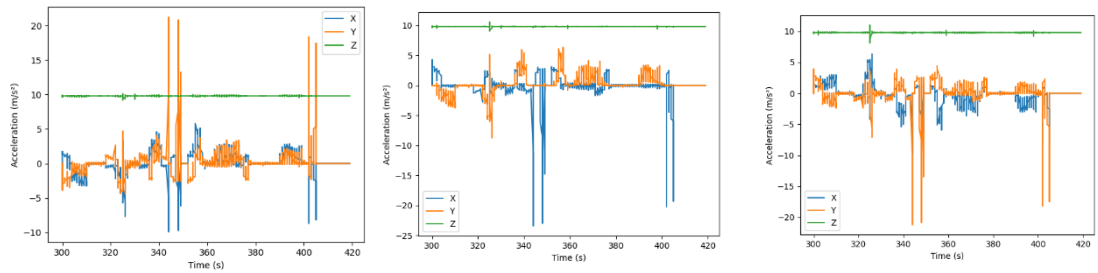
El sensor IMU de las cámaras ZED2i da a su salida la aceleración lineal y los ángulos de rotación en forma de cuaternio, por lo que se han procesado los datos para obtener los ángulos de Euler. Este procesamiento se encuentra en el script *plot_imu.py* del repositorio [12]. En las Figura 4.14 y 4.15, observamos los valores que toman los ángulos de orientación y la aceleración que toma el vehículo a lo largo de todo el recorrido, respectivamente.



a) *b)* *c)*

Figura 4.14. Resultados de los sensores IMU del ego vehicle (orientación) en la ruta 1. a) Left b) Central c) Right

En primer lugar, observamos que la orientación en X e Y no cambia ya que en este recorrido no tenemos ningún plano inclinado. Por otro lado, observando la Figura 4.6 podemos determinar que cada zona llana de las gráficas equivale a una de las orientaciones principales de las carreteras de este mapa y, por otro lado, cada cambio de ángulo equivale a un giro del vehículo. Además, si nos centramos, por ejemplo, en la zona comprendida entre los 370 y 400 segundos de la simulación veremos que el valor del sensor central es de 0º, el del sensor lateral izquierdo de 65º y el del lateral derecho de -65º. Esto se debe al giro incorporado a la colocación de la cámara.



a) *b)* *c)*

Figura 4.15. Resultados de los sensores IMU del ego vehicle (Aceleración) en la ruta 1. a) Left b) Central c) Right

Nuevamente como no hay ninguna subida ni bajada no hay ningún cambio significativo en la aceleración en Z, que se mantiene contante alrededor de 9.8 m/s^2, la aceleración de la gravedad. Comparando las dos figuras observamos que los mayores picos de aceleración se producen en los giros del vehículo. Por otra parte, se observa que en el sensor central del vehículo los picos de aceleración se producen en el eje X mientras que los de los extremos se producen en el eje Y, pero de mayor magnitud. Esto es debido a la rotación en la colocación de los sensores externos que realiza un reparto de la aceleración entre los 2 ejes del sensor tomando la mayor parte el eje Y (ángulo mayor a 45º).

En la Figura 4.16, observamos que, si representamos la latitud frente a la longitud dada por los sensores GNSS, obtenemos prácticamente la misma trayectoria de la Figura 4.6 independientemente de si es el de la izquierda o el de la derecha. Por su parte, la altura permanece constante.

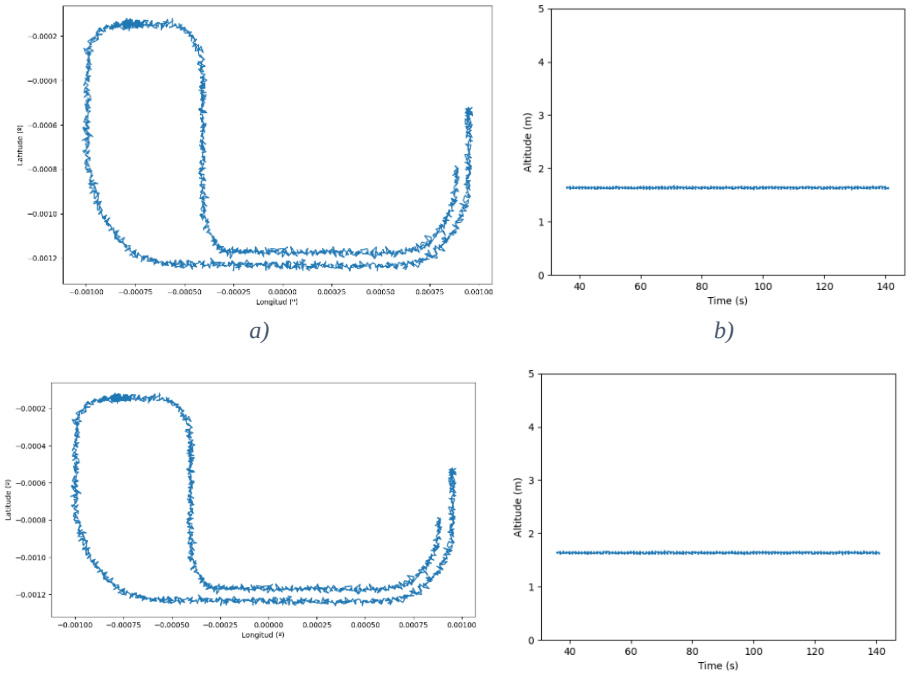


Figura 4.16. Resultados de los sensores GNSS del ego vehicle a) y b) Left c) y d) Right

En lo referente al resto de vehículos se muestran las medidas tomadas por el sensor IMU del autobús, el peatón 1 y el ciclista 1 en las Figura 4.17, 4.18 y 4.19, respectivamente.

El autobús sigue una ruta muy sencilla, solamente efectúa un giro a la izquierda, por lo que vemos en sus gráficas que solamente cambia la orientación de 90º. Se observa que este giro afecta a las aceleraciones lineales del vehículo.

El peatón 1 realiza una ruta de vaivén, por ello lo que hace es variar su orientación entre 2 puntos alternativamente. Los picos de aceleración se efectúan en estos giros.

El ciclista 1 efectúa 2 giros durante la simulación y es lo que vemos reflejado en sus gráficas.

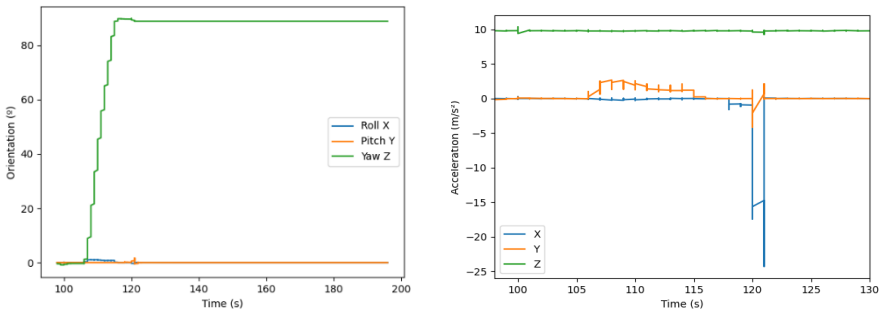


Figura 4.17. Resultados del IMU del autobús en la ruta 1

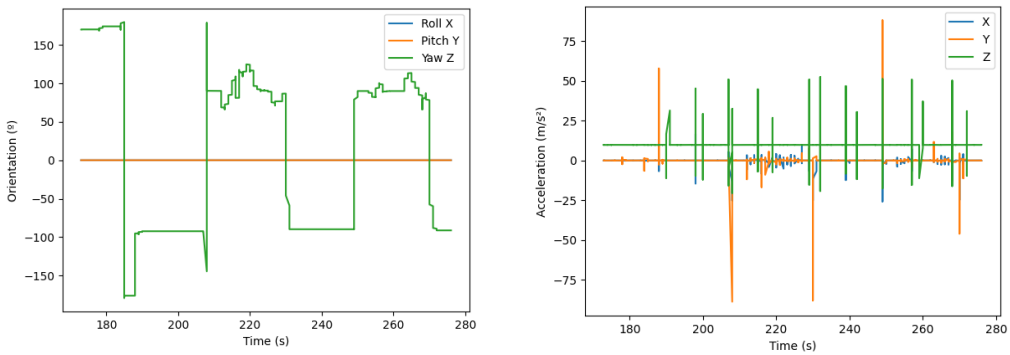


Figura 4.18. Resultados del IMU del peatón 1 en la ruta 1

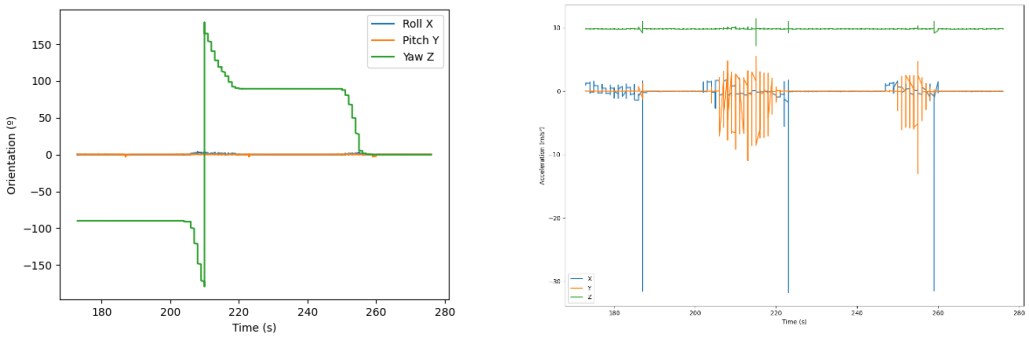


Figura 4.19. Resultados del IMU del ciclista 1 en la ruta 1

En las Figura 4.20, 4.21 y 4.22 observamos los datos obtenidos por los sensores GNSS de estos agentes. En lo referente a la latitud y longitud observamos que se siguen las rutas representadas en la Figura 4.6. Remarcar que en estas gráficas el ruido se hace más notorio que en la Figura 4.16 debido a la escala de la gráfica.

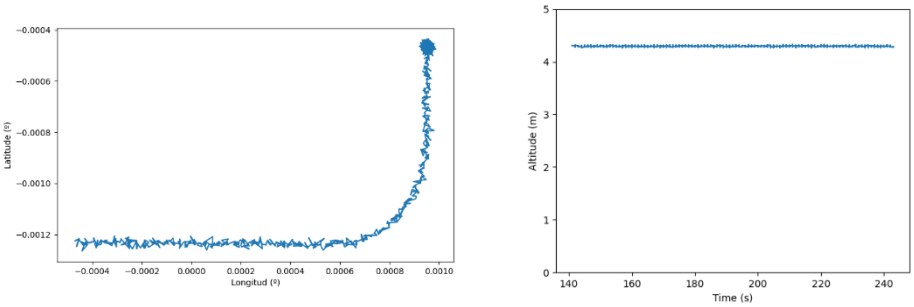


Figura 4.20. Resultados del GNSS del autobús en la ruta 1

En lo referente a la altura, en el caso del autobús observamos que se mantiene constante la altura del vehículo. En cambio, en el caso del peatón 1 observamos que hay

ligeras variaciones, esto se debe a la diferencia de altura de la acera. Por último, el ciclista también mantiene su altura constante.

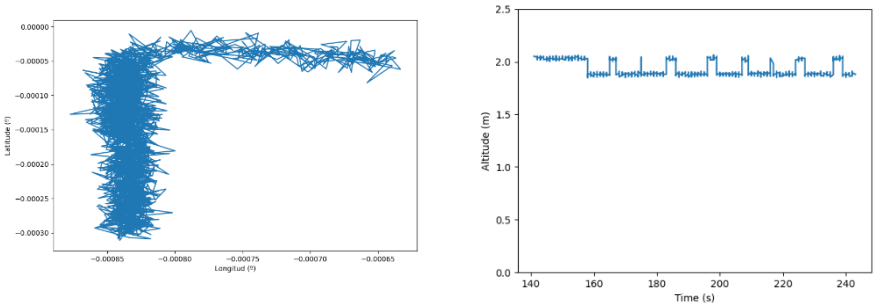


Figura 4.21. Resultados del GNSS del peatón 1 en la ruta 1

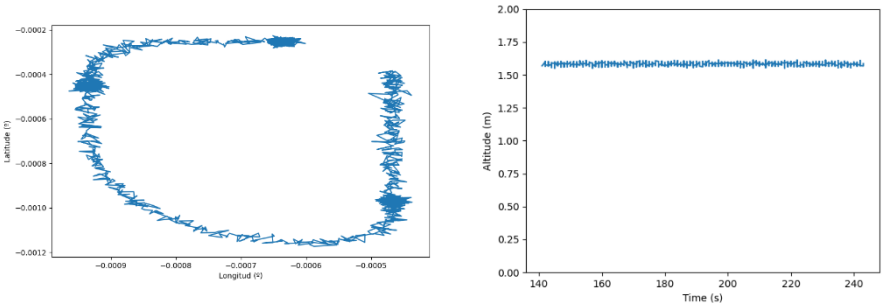


Figura 4.22. Resultados del GNSS del ciclista 1 en la ruta 1

4.4.2 Resultados de la Ruta 1.2

Como se ha dicho con anterioridad esta ruta es idéntica a su predecesora, pero con otras condiciones ambientales. Estas condiciones son lluvia y distinta inclinación solar. Los efectos de la lluvia solamente se ven reflejados en las cámaras por lo que no se añadirá información sobre el resto de los sensores. En cambio, en la Figura 4.23, se observa el cambio de iluminación del entorno y la superficie húmeda de la carretera en un punto donde nos encontramos con 2 peatones y 2 ciclistas. También podemos observar zonas donde se difumina un poco la imagen debido a las gotas de agua que hay sobre la lente de la cámara. En la Figura 4.24, donde observamos estos agentes con las cámaras de profundidad. También se observa que en este tipo de cámaras no hay diferencia apreciable entre las situaciones meteorológicas.

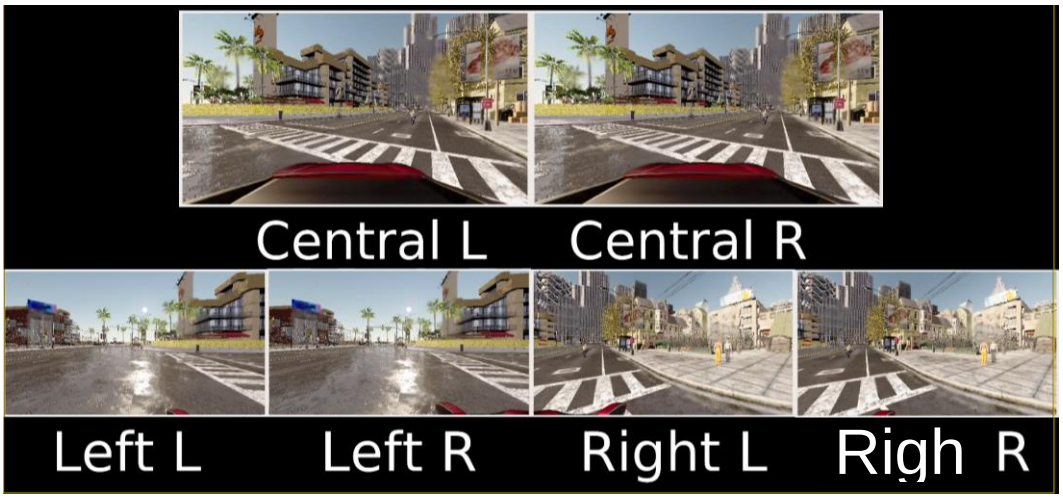


Figura 4.23. Vista de los peatones con las cámaras ZED2i modeladas

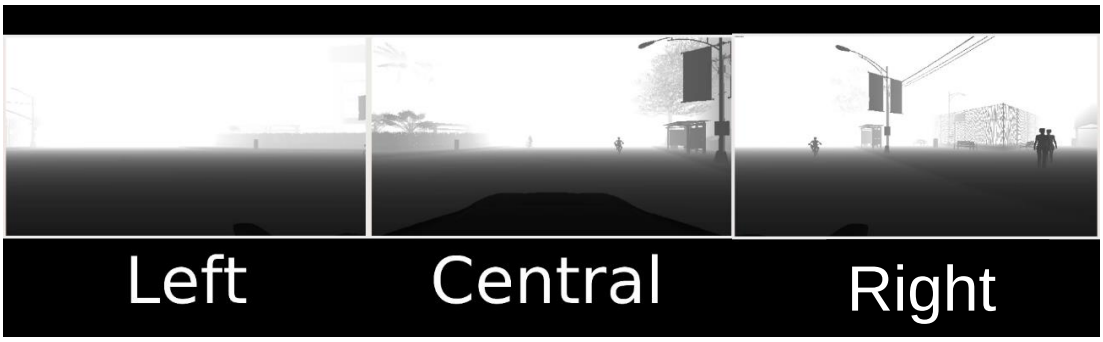


Figura 4.24. Vista de los peatones con las cámaras ZED2i modeladas (Profundidad)

El segundo ciclista se observa mejor en la Figura 4.25, debido al *zoom* realizado por la cámara, pero por lo mismo y debido a la reducción del campo de visión quedan fuera los peatones. También se hace más notable las gotas de agua en la lente de la cámara.

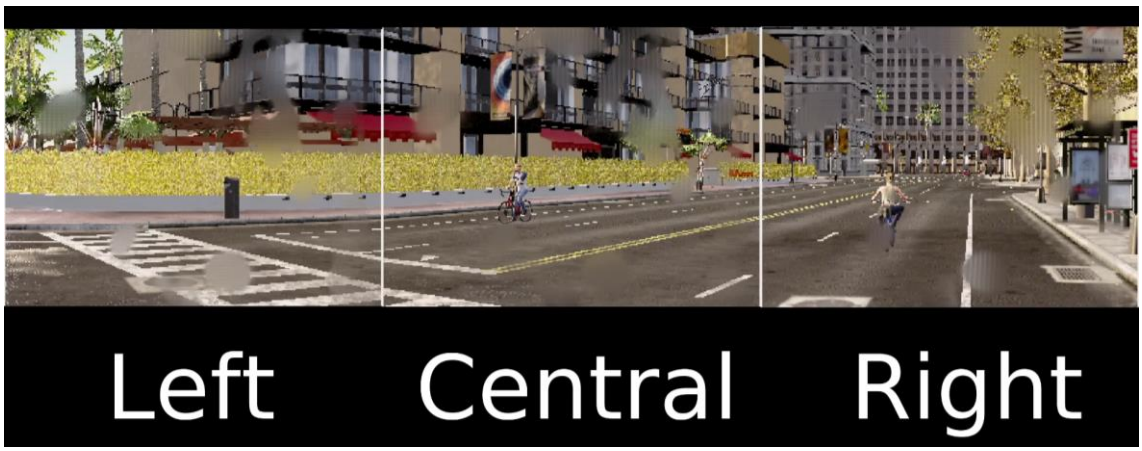


Figura 4.25. Vista de los ciclistas con las cámaras Seek CompactPRO modeladas

Y, por último, estos agentes aparecen como un pequeño cúmulo de puntos en la visión del LiDAR, como vemos en la Figura 4.26. El cúmulo de puntos del círculo naranja corresponde con uno de los ciclistas, el que vemos dentro del círculo verde corresponde con el otro y, finalmente, el que encontramos en el círculo rojo corresponde a los peatones que simulan tener una conversación en la acera derecha.

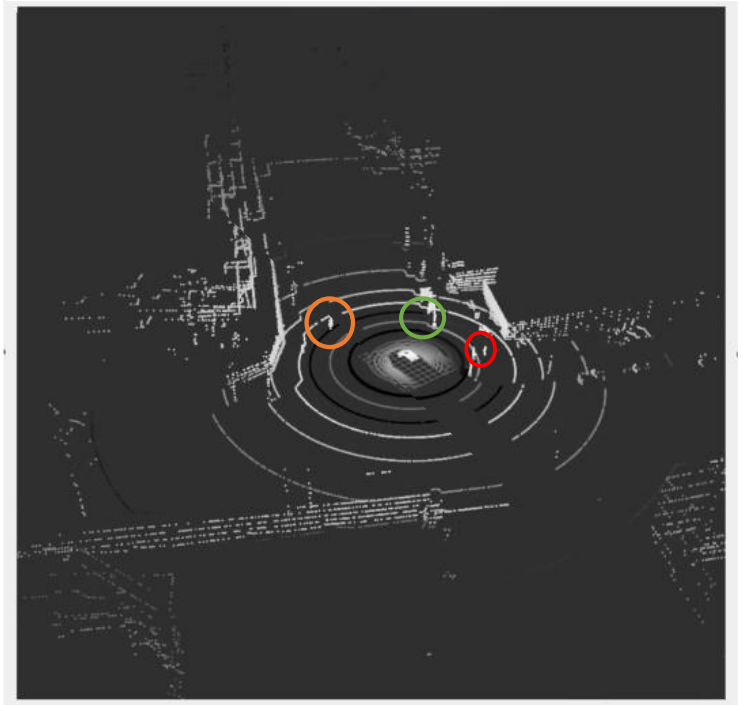


Figura 4.26. Vista de los peatones con el LiDAR RS-Helios 5155 modelado

4.4.3 Resultados de la Ruta 2.1

Esta ruta (ver Figura 4.7) cuenta con 2 situaciones interesantes: un desnivel en el recorrido y la interacción con el tráfico en una rotonda. Además, cuenta el mismo número de agentes que las rutas anteriores. Vemos con más detalle el resultado que nos otorgan las cámaras y el LiDAR en el momento de incorporación a la rotonda.

La Figura 4.27 se ha tomado justo en el momento que el coche frena al no poder incorporarse inmediatamente. Se destaca de nuevo el amplio campo de visión de estas cámaras. Aun cuando el vehículo de la rotonda se encuentra delante del principal las cámaras de la izquierda son capaces de captar su parte trasera.

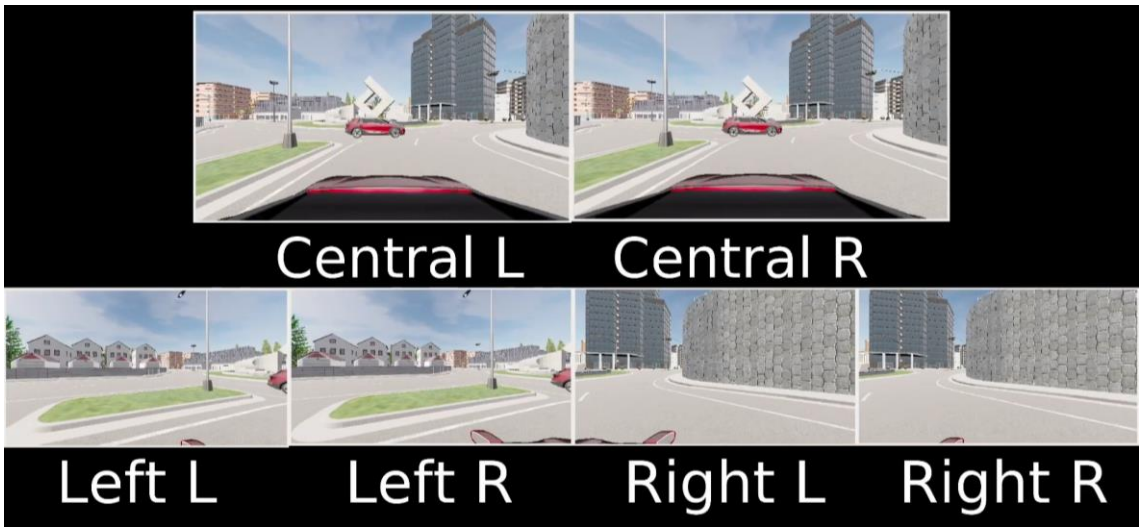


Figura 4.27. Vista durante la incorporación a la rotonda con las cámaras ZED2i modeladas

En la Figura 4.28 se observa como el vehículo es captado por las cámaras de profundidad modeladas. También observamos como los objetos más alejados salen del campo de visión de esta cámara.

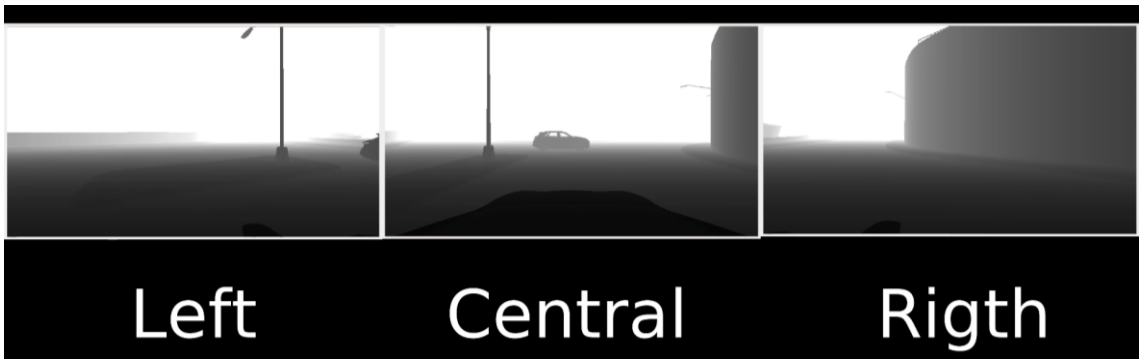


Figura 4.28.Vista durante la incorporación a la rotonda con las cámaras ZED2i modeladas (Profundidad)

Observando la vista proporcionada por las cámaras térmicas (ver Figura 4.29) nos damos cuenta la reducción del campo de visión, al ver que el vehículo en este instante solo es captado por la cámara central. Aún así se tiene una vista casi continua como ha ocurrido en las otras rutas.

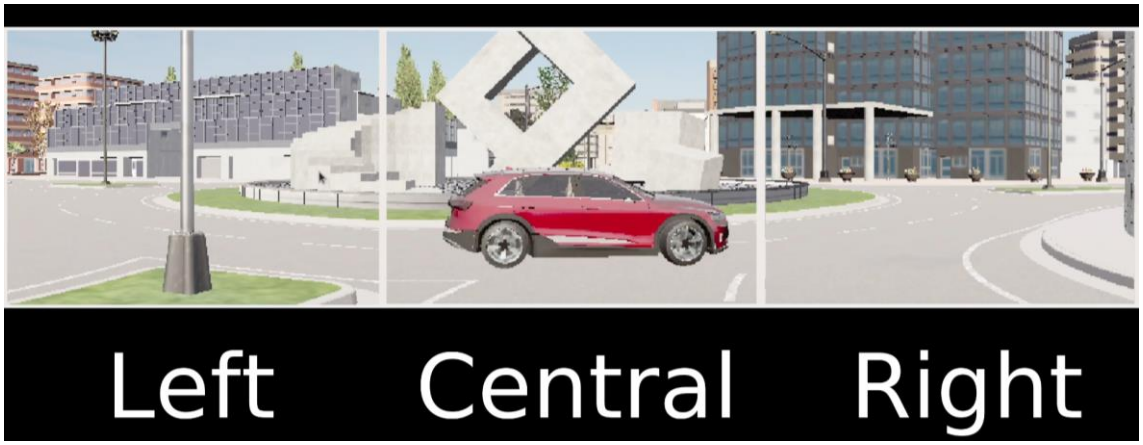


Figura 4.29. Vista durante la incorporación a la rotonda con las cámaras Seek CompactPRO modeladas

El último sensor que analizamos en este instante es el LiDAR (ver Figura 4.30), donde se observa que el cúmulo de puntos que concentrado en el círculo naranja es el vehículo de la rotonda. Este sensor nos da información de los 360º, así que, también observamos otro cúmulo (círculo verde) que marcan al ciclista que acaba de adelantar.

Ahora podemos analizar todo el recorrido con los datos que nos proporciona los IMU y uno de los GNSS. En esta ocasión solamente analizamos un GNSS, ya que como hemos visto con anterioridad los resultados son idénticos. En la Figura 4.31 se observa los resultados de la orientación. Las diferencias entre las 3 gráficas son equivalentes a las comentadas con anterioridad. De estás gráficas hay que destacar 2 períodos de tiempo:

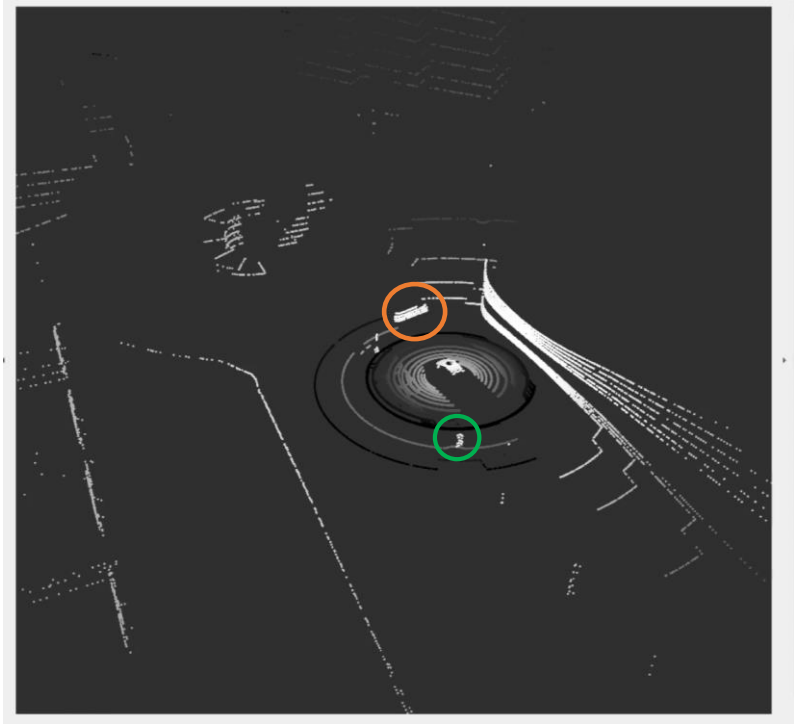
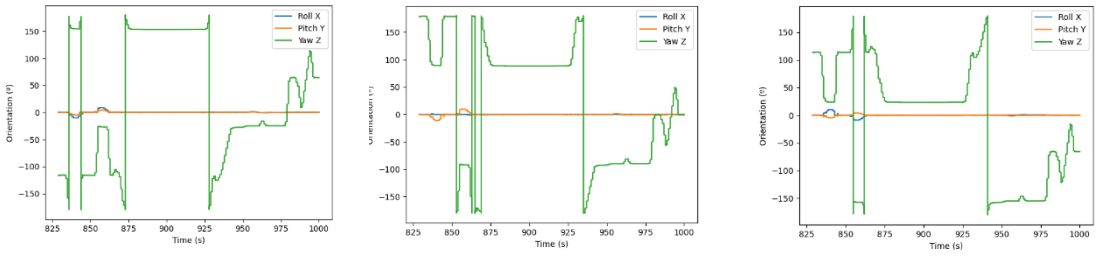


Figura 4.30. Vista durante la incorporación a la rotonda con el LiDAR RS Helios-5515 modelado

- [830 s,870 s]. Período donde se efectúa la subida y bajada de la zona elevada de la ciudad. Se observa el cambio de inclinación en el *roll* (ángulo de giro en el eje X) y el *pitch* (ángulo de giro en el eje Y) del sensor.
- [980 s, 1000 s]. Último tramo de la simulación donde se realiza media vuelta a la rotonda. Observamos que la rotación en Z decrece, crece 2 veces lo que ha decrecido y, por último, vuelve a decrecer hasta alcanzar la orientación original.

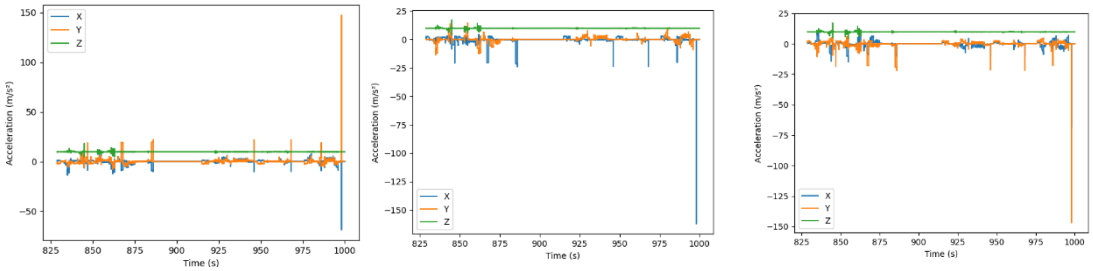


a) *b)* *c)*

Figura 4.31. Resultados de los sensores IMU del ego vehicle (orientación) en la ruta 2. a) Left b) Central c) Right

En la Figura 4.32 se observa que en el primer periodo la aceleración en Z sufre cambios debido a la subida y la bajada, mientras que en el segundo período se observa una mayor variación de aceleración debido a la circularidad del movimiento.

Por último, en la Figura 4.33, observamos los datos recogidos por el sensor GNSS. En lo referente a la latitud y longitud, podemos observar una reconstrucción en 2 dimensiones de la ruta seguida. En lo referente a la altura se observa la variación del desnivel de la ciudad del principio y una pequeña variación en la parte final del recorrido.



b) *b)* *c)*

Figura 4.32. Resultados de los sensores IMU del ego vehicle (aceleración) en la ruta 2. a) Left b) Central c) Right

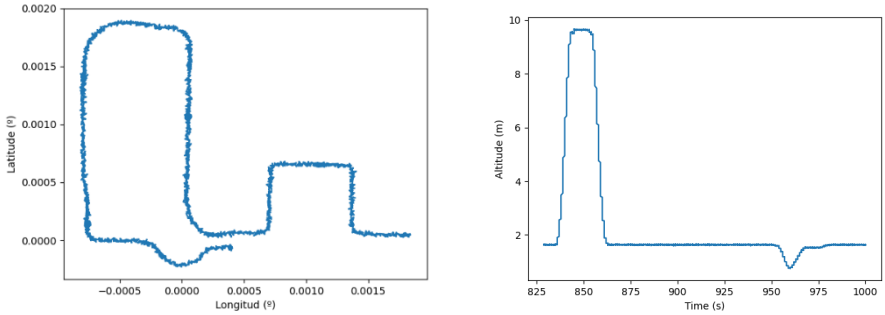


Figura 4.33. Resultados del GNSS del ego vehicle en la ruta 2

De manera, análoga a lo visto en la primera ruta ahora analizaremos el movimiento de algunos de los agentes que intervienen en la simulación. Analizaremos el movimiento del peatón 1, del coche 2 y del ciclista 2. En la Figura 4.34, 4.35 y 4.36 observamos el resultado del sensor IMU, respectivamente.

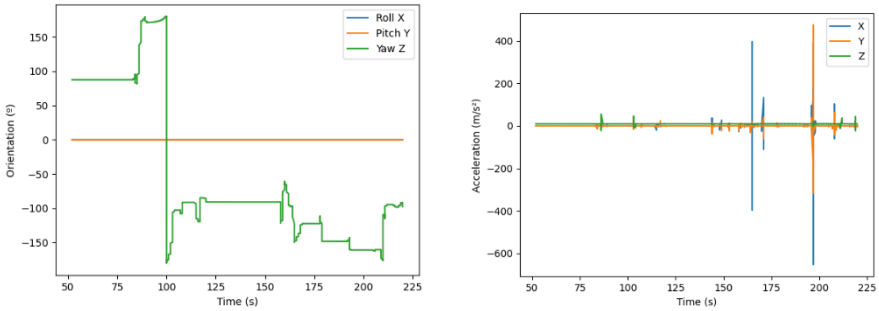


Figura 4.34. Resultados del IMU del peatón 1 en la ruta 2

Sin entrar en detalle, observamos coherencia en los resultados con la ruta programada. Los picos de aceleración se deben a fallos en la simulación debido al equipo informático utilizado. Como hemos observado no son muy comunes.

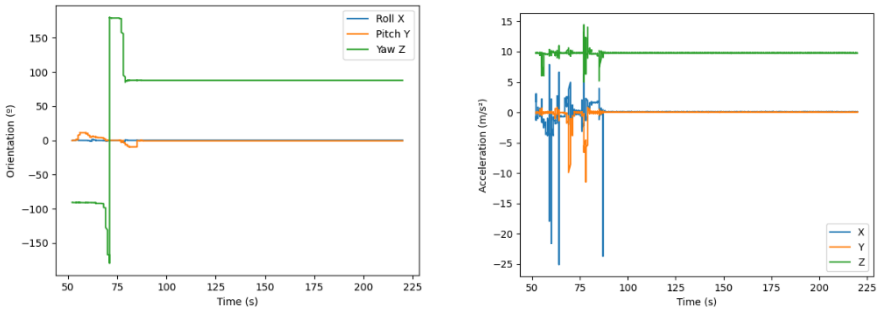


Figura 4.35. Resultados del IMU del coche 2 en la ruta 2

El recorrido de este coche es en el desnivel de la ciudad y lo realiza al principio de la simulación, por lo que el mayor tiempo de esta se queda en la posición final. Observamos los cambios de nivel en las variaciones de *roll* y *pitch* y de la aceleración en el eje Z, principalmente.

El ciclista 2 sigue una trayectoria cerrada, por lo que vemos cierto patrón en sus movimientos. Por otra parte, durante la simulación realiza distintas paradas bruscas, observamos que esto genera aceleraciones grandes.

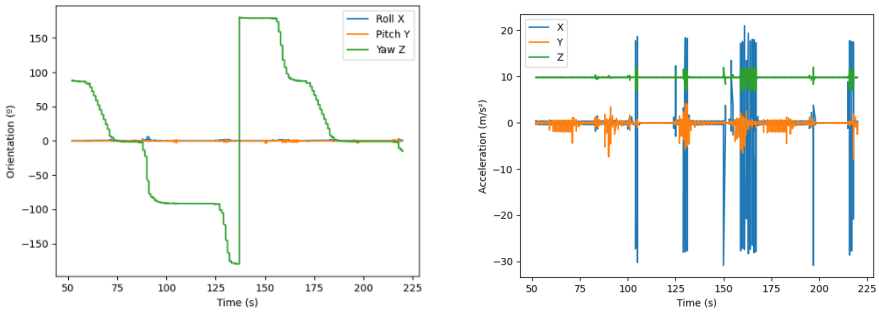


Figura 4.36. Resultados del IMU del ciclista 2 en la ruta 2

En la Figura 4.37, 4.38 y 4.39 observamos los datos obtenidos por los sensores GNSS de los mismos agentes. En lo referente a la latitud y longitud observamos que se siguen las rutas representadas en la Figura 4.7.

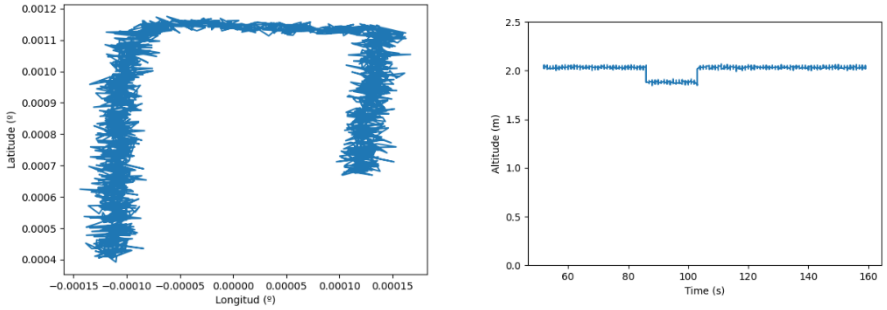


Figura 4.37. Resultados del GNSS del ego peatón 1 en la ruta 2

Sobre la altura del peatón 1, observamos el cambio de altura al cruzar la calzada. En el coche 2, se nota le cambio de altura notablemente. En cambio, en el ciclista 2 no hay ningún cambio en la altura.

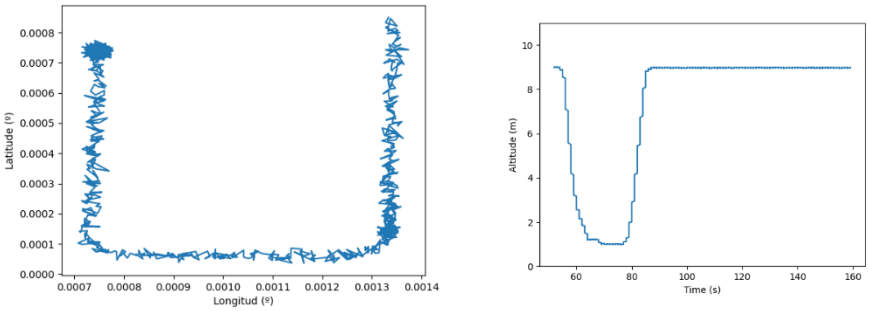


Figura 4.38. Resultados del GNSS del ego coche 2 en la ruta 2

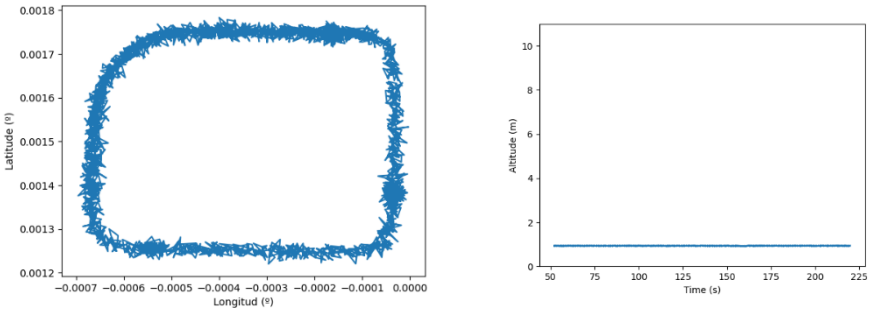


Figura 4.39. Resultados del GNSS del ego ciclista 2 en la ruta 2

4.4.4 Resultados de la Ruta 2.2

De nuevo, esta ruta es la misma que la anterior, pero cambiando las condiciones meteorológicas que se indican en Figura 4.8. Por lo que solo analizaremos los resultados de las cámaras y del LiDAR en un instante de la simulación. Este instante es cuando se cruza un coche en la rampa de bajada y se observa un ciclista en la calzada perpendicular.

En las cámaras RGB (ver Figura 4.40), se observa la carretera mojada y zonas del ambiente difusas, esto debido a las gotas de agua en las lentes. Debido a la lluvia en estas cámaras no se visualiza con claridad al ciclista.

En las cámaras de profundidad (ver Figura 4.41) el coche también es percibido a la perfección mientras que el ciclista se encuentra muy lejos para ser captado. Mientras que, aunque el efecto de la lluvia es mayor sobre las cámaras térmicas (ver Figura 4.42), observamos que se detectan perfectamente el ciclista y el coche.

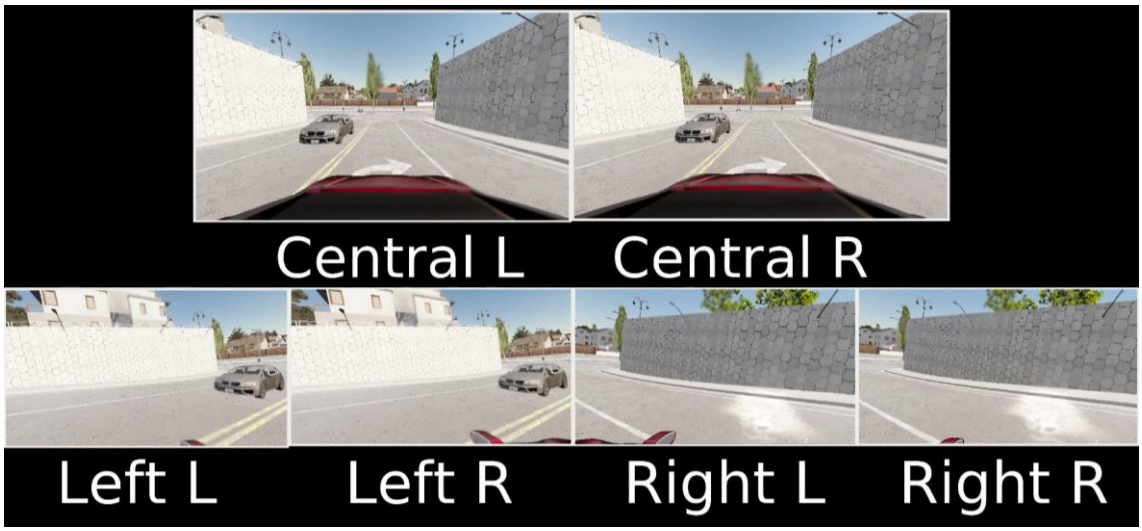


Figura 4.40. Vista de la rampa con las cámaras ZED2i modeladas

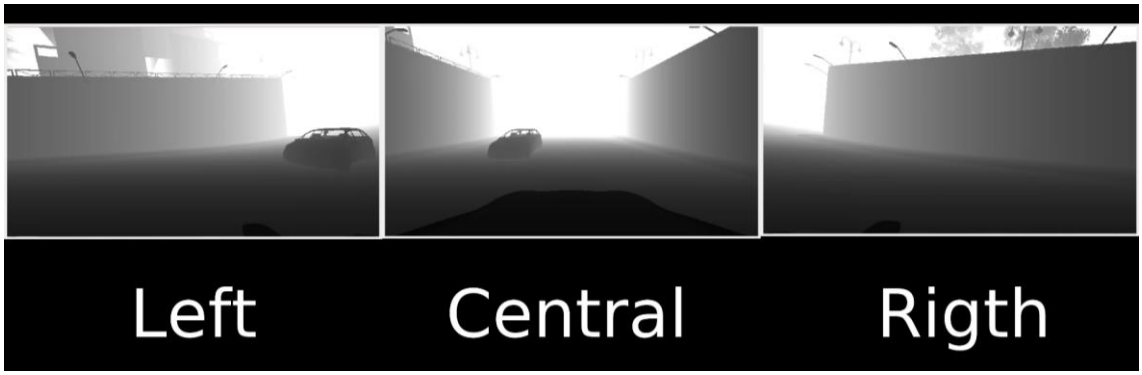


Figura 4.41. Vista de la rampa con las cámaras ZED2i modeladas (Profundidad)

Figura 4.42. Vista de la rampa con las cámaras Seek CompactPRO modeladas

5 Conclusiones

5.1 Recapitulación

En este trabajo se han estudiado a fondo los sensores que se van a utilizar en el proyecto PREMOVE para efectuar la predicción del movimiento de los agentes involucrados en el tráfico en áreas urbanas. Con esta investigación se ha logrado definir los valores concretos que deben tomar los parámetros de los sensores en *CARLA Simulator* para alcanzar un modelo realista en simulación. Además, con la combinación de estos sensores se ha conseguido simular un coche autónomo realista. Además, se ha propuesto una distribución de las cámaras y sensores con las que contará el vehículo para obtener los mejores resultados. Dentro de los sensores incluidos en la simulación, el vehículo cuenta con los necesarios para acceder, en todo momento, a la información de posición, orientación y velocidad del vehículo.

Por otra parte, se ha adquirido el conocimiento necesario para desarrollar simulaciones realistas en un entorno urbano y totalmente controlado. En estas simulaciones se ha generado una ruta que el vehículo ha seguido y donde ha podido interactuar con el tráfico que encontramos dentro de una ciudad (coches, autobuses, peatones y ciclistas). A todos los agentes que participan en el tráfico se le ha proporcionado sensores para conocer su localización y su orientación. Además, se han visto ejemplos específicos donde la información obtenida por un sensor no nos otorga información suficiente del entorno, que hay que completar añadiendo los datos de más sensores.

Este trabajo confirma lo necesario y efectivo que son los simuladores en el desarrollo de la ingeniería en la actualidad, apoyando el desarrollo de la Industria 4.0. Hemos visto que es posible desarrollar entornos realistas permitiendo obtener datos que podrán ser utilizados posteriormente. Además, contribuye en la reducción de los altos costes que tiene realizar un experimento de estas magnitudes, pudiendo realizar un gran número de simulaciones en poco tiempo y, de este modo, aumentar la cantidad de datos de los que se dispone.

5.2 Trabajos Futuros

Como ya se ha comentado anteriormente este trabajo forma parte del proyecto PREMOVE, por lo cual algunos de los posibles futuros trabajos que se pueden desarrollar son los trabajos que contribuyan al desarrollo de este proyecto como son los siguientes:

- Utilizar los datos obtenidos por los diversos sensores para entrenar redes neuronales para predecir el movimiento de los agentes que participan en el tráfico.
- Aumentar la variedad de agentes en la simulación como es personas con la movilidad reducida (uso de bastón, muletas o silla de ruedas) o vehículos que cada vez son más comunes en zonas urbanas como los patinetes eléctricos.

Además de estos, se pueden desarrollar otros trabajos:

- Realizar un desarrollo por medio de *Unreal Engine*, en el que se asocien distintas temperaturas a objetos o agentes dependiendo de su superficie y, posteriormente, incluir su variación dependiendo de la intensidad del sol. De este modo, se podrán simular cámaras térmicas.
- Con la utilización de los sensores realistas desarrollar un modelo de un coche autónomo que permita alcanzar alguno de los niveles superiores en la escala establecida por el SAE.

6 Anexos

A. Referencias

[1] «SAE International,» [En línea]. Available: https://www.sae.org/standards/content/j3016_202104/. [Último acceso: 02 06 2023].

[2] J. Morales y J. L. Martínez, «Universidad de Málaga,» [En línea]. Available: https://www.uma.es/robotics-and-mechatronics/info/138109/premove/. [Último acceso: 24 Mayo 2023].

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. López y V. Koltun, CARLA:An Open Urban Driving Simulator. In Proceedings of the 1st Conference on Robot Learning,, Mountain View, CA, USA: pp. 1–16., 2017.

[4] F. Martín Rico, A Concise Introduction to Robot Programming with ROS2., CRC Press, (2023).

[5] «Unreal Engine,» [En línea]. Available: https://www.unrealengine.com/en-US/?sessionInvalidated=true. [Último acceso: 08 06 2023].

[6] «CARLA Documentation RVIZ CARLA Plugin,» [En línea]. Available: https://carla.readthedocs.io/projects/ros-bridge/en/latest/rviz_plugin/. [Último acceso: 26 Mayo 2023].

[7] «ROS 2 Documentation: Foxy RQt,» [En línea]. Available: https://docs.ros.org/en/foxy/Concepts/About-RQt.html.

[8] «ROS 2 Documentation: Foxy ROS 2 bag,» [En línea]. Available: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Recording-And-Playing-Back-Data/Recording-And-Playing-Back-Data.html. [Último acceso: 26 Mayo 2023].

[9] «CARLA Documentation ROS bridge installation for ROS 2,» [En línea]. Available: https://carla.readthedocs.io/projects/ros-bridge/en/latest/ros_installation_ros2/. [Último acceso: 27 Mayo 2023].

[10] E. Vergara Gómez, *Desarrollo de un simulador para el diseño de sistemas de percepción y control de vehículos autónomos en el entorno de la ampliación del Campus de Teatinos,* Trabajo Fin de Máster en Ingeniería Mecatrónica. Universidad de Málaga., 2020.

[11] «CARLA Documentation Sensors Reference,» [En línea]. Available: https://carla.readthedocs.io/en/0.9.14/ref_sensors/. [Último acceso: 24 Mayo 2023].

[12] D. S. Gamba-Correa, «GitHub,» [En línea]. Available: https://github.com/danielgamba01/TFG.

[13] «Robosense,» [En línea]. Available: https://www.robosense.ai/resources-81. [Último acceso: 12 Mayo 2023].

[14] «STEREOLABS,» [En línea]. Available: https://cdn2.stereolabs.com/assets/datasheets/ZED%202i%20Datasheet%20Jan2023.pdf. [Último acceso: 24 Mayo 2023].

[15] «STEREOLABS DOCS,» [En línea]. Available: https://www.stereolabs.com/docs/. [Último acceso: 05 Junio 2023].

[16] «Seek Thermal,» [En línea]. Available: https://www.thermal.com/uploads/1/0/1/3/101388544/compactpro-sellsheet-usav1.pdf. [Último acceso: 25 Mayo 2023].

[17] «Sparkfun Start Something,» [En línea]. Available: https://cdn.sparkfun.com/assets/f/8/d/6/d/ZED-F9P-02B_DataSheet_UBX-21023276.pdf. [Último acceso: 25 Mayo 2023].

[18] G. Sanna, T. Pisanu y S. Garau, «Behavior of Low-Cost Receivers in Base-Rover Configuration,» *Sensors,* vol. 22, nº 2779, p. 17, 2022.

[19] «InvenSense,» [En línea]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf. [Último acceso: 25 Mayo 2023].

[20] R. Gonzalez y P. Dabove, «Performance Assessment of an Ultra Low-Cost Inertial Measurement Unit for Ground Vehicle Navigation,» *Sensors,* vol. 19, nº 3865, p. 14, 2019.

[21] «Nissan,» [En línea]. Available: https://www.nissan.es/vehiculos/nuevos-vehiculos/leaf/dimensiones-especificaciones.html. [Último acceso: 25 Mayo 2023].

[22] «CARLA Documentation CARLA Catalogue,» [En línea]. Available: https://carla.readthedocs.io/en/latest/catalogue/. [Último acceso: 25 Mayo 2023].

[23] «Medidas de coche,» [En línea]. Available: https://www.medidasdecoches.com/modelo/toyota/prius. [Último acceso: 25 Mayo 2023].

[24] «CARLA Documentation ROS Bridge Documentation,» [En línea]. Available: https://carla.readthedocs.io/projects/ros-bridge/en/stable/. [Último acceso: 26 Mayo 2023].

[25] «CARLA Documentation Traffic Manager,» [En línea]. Available: https://carla.readthedocs.io/en/0.9.14/tuto_G_traffic_manager/. [Último acceso: 27 Mayo 2023].

[26] D. S. Gamba-Correa, «YouTube,» 30 05 2023. [En línea]. Available: https://www.youtube.com/playlist?list=PL9v0AGYXNNUpr0JbA-fc2HgO-jpTOFPXB.

[27] «CARLA Documentation Linux Build,» [En línea]. Available: https://carla.readthedocs.io/en/0.9.14/build_linux/. [Último acceso: 2023 Mayo 27].

[28] «ROS 2 Docuemntation Ubuntu (Debian),» [En línea]. Available: https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html. [Último acceso: 27 Mayo 2023].

B. Instalación del Software

En este anexo se explica el proceso de instalación desde el sistema operativo hasta el último de los programas utilizados en este proyecto. Durante este proceso se detallan problemas con los que me he encontrado y la solución que he efectuado.

Aspectos generales de la instalación:

- Fecha de primera instalación: noviembre de 2022.
- Fecha de reinstalación: mayo de 2023.
- Modelo de equipo: ASUS TUF Gaming F15
- Sistema operativo: Ubuntu 20.04.
- Versión de *Unreal Engine*: 4.26.

El tiempo de instalación es aproximado de 5 horas, con una inutilización del ordenador de alrededor de 2 horas (período en el que se el ordenador se dedica a realizar instalaciones y comprobaciones).

Prerrequisitos

- Espacio de Disco: 250 GB. La instalación de Carla y *Unreal Engine* me ha ocupado alrededor de 151 GB, pero durante la instalación puede requerir más espacio.
- GPU: al menos de 6 GB (en mi caso he usado una de 16 GB).
- Buena conexión a internet.
- Versión de Ubuntu: 18.05 o superior.

Instalación de Ubuntu 20.04

Se ha realizado una partición de disco en el equipo utilizado para realizar la instalación de Ubuntu, aunque esta instalación se podría hacer directamente como el sistema operativo principal. Para realizar esta partición he seguido las indicaciones del siguiente:

https://www.youtube.com/watch?v=h9cPABYSJSI&t=298s&ab_channel=WalterRosero

Como se indica en el tutorial para la instalación de Ubuntu se ha requerido de un pendrive de al menos 8 GB de memoria e instalar una aplicación para crear una unidad flash USB de arranque. Se proveen los enlaces de descarga para la instalación de los programas necesarios:

- Rufus: https://rufus.ie/en/
- Ubuntu: https://releases.ubuntu.com/focal/

Instalación de *CARLA Simulator*

Para la instalación de CARLA se seguirán los pasos descritos en la página oficial del programa [27]. Así que para la instalación del simulador hay que seguir paralelamente este tutorial como el de la página oficial. La versión del simulador que se está instalando es la versión *source*. Para instalar la versión *package* seguir los pasos que se indican en la página oficial.

1. Primera Parte: Prerrequisitos
   1.1. Requerimientos del sistema.
   1.2. Requerimientos de Software. Estos son una serie de comandos que hay que ejecutar en la ventana de comandos del sistema que instalará programas que se usarán durante la instalación y Carla. En este punto se hace una distinción entre comandos que hay que usar dependiendo de la versión de Ubuntu. Aquí fue donde tuve el Problema nº1. Aunque en Ubuntu 20.04 ya no esté contemplado el uso de *clang-8* en los archivos ejecutables de Carla todavía se usa esta versión (no se usa *clang-10*). Para solucionarlo seguí los pasos del siguiente enlace: https://github.com/carlasimulator/carla/discussions/5509.
   1.3. *Unreal Engine*. Este paso consiste en crearse una cuenta en *Epic Games* y GitHub para tener acceso a todos los programas referidos a Carla y *Unreal Engine*. Los 2 últimos comandos tardan bastante tiempo en ejecutarse al completo.

2. Segunda Parte: Construir Carla
   2.1. Clonar el Repositorio de Carla. En este punto tuve mi Problema nº2, al clonar el repositorio me pedía la clave de mi cuenta de GitHub y al introducirla en aparecía un error que decía que esa forma de proceder se había eliminado hace un año. Por lo que, utilice el siguiente comando: `gh repo clone https://github.com/carla-simulator/carla`

   Para ejecutar este comando es necesario instalar ciertas librerías (seguir los pasos que indica la ventana de comandos al ejecutar el comando anterior).

   2.2. Obtener activos. Actualiza Carla para el uso de la última versión.
   2.3. Establecer la variable de entorno de Unreal. Seguir al tutorial.
   2.4. Construir Carla. Al ejecutar el primer comando me aparece el Problema nº3. Al ejecutarse el comando usa archivos que están en la red y uno de estos archivos fue actualizado el mes pasado. Para solucionarlo me dirigí al siguiente directorio: ~/carla/Util/BuildTools/SetUp.sh. Y reemplacé la dirección que había en la línea 432 por el siguiente “https://archive.apache.org/dist/xerces/c/3/sources/xerces-c-”.

   Después de esto elegí la opción B ya que es más sencilla y no ocupa mucho espacio. Por último, al compilar el servidor se me quedaba pillado el programa este fue mi Problema nº4, esto es debido a que el ordenador por defecto utiliza la gráfica de Intel y no la de NVIDIA. Para ello solo hay que instalar los controladores de la tarjeta gráfica (si no están ya) cambiar esta configuración.

Para ello, yo obtuve ayuda externa de otra persona, pero se puede seguir cualquier tutorial de internet.

Instalación de ROS2

El puente de CARLA con ROS solo soporta la versión de Foxy de ROS 2 por lo que procedí a realizar la instalación que se explica en la documentación oficial [28]. Este tutorial es muy sencillo y no me ha generado ningún problema la instalación. Resaltar que es importante instalar el paquete que incluye RViz, las demos y los tutoriales, ya que se hace uso de ello en CARLA y sirve para comprobar que se ha realizado la instalación correctamente.

Por último, es conveniente abrir el archivo ".bashrc" con el siguiente comando en un terminal:

```
gedit ~/.bashrc
```

Y añadir al final del archivo la siguiente línea:

```
source /opt/ros/foxy/setup.bash
```

Esto permitirá que cada vez que se abra un terminal nuevo se configure automáticamente el entorno de ROS.

Instalación del *CARLA-ROS Bridge*

He seguido el tutorial que proporciona la documentación oficial de CARLA [9]. Este tutorial es muy sencillo y no me ha generado ningún problema la instalación.

Por último, es conveniente abrir el archivo ".bashrc" con el siguiente comando en un terminal:

```
gedit ~/.bashrc
```

Y añadir al final del archivo la siguiente línea:

```
source ~/carla-ros-bridge/install/setup.bash
```

Esto permitirá que cada vez que se abra un terminal nuevo se configure automáticamente el entorno de ROS.

Instalación de los Resultados del TFG

Ir al repositorio [12] y descargar las 2 carpetas. Se pueden descargar ejecutando el siguiente comando en un terminal.

```
git clone https://github.com/danielgamba01/TFG.git
```

Las 2 carpetas que se descargarán son "TFG_ws" y "Code".

Una vez las carpetas estén descargadas hay que construir el espacio del trabajo del TFG. Esto se realiza ejecutando el siguiente comando en un terminal dentro de la carpeta "TFG_ws":

```
colcon build
```

Por último, es conveniente abrir el archivo ".bashrc" con el siguiente comando en un terminal:

```
gedit ~/.bashrc
```

Y añadir al final del archivo la siguiente línea:

```
source ~/TFG_ws/install/setup.bash
```

Esto permitirá que cada vez que se abra un terminal nuevo se configure automáticamente el entorno de ROS.

C. Manual de Usuario

En este Manual se indicarán los pasos necesarios para ejecutar las rutas que se han creado para este trabajo.

1. Instalar todo el software y los paquetes necesarios (ver Anexo B).
2. Iniciar el servidor de carla. Ejecutando en un terminal los siguientes comandos.

```
cd <ruta de carla>
make launch
```

   Si se ha instalado la versión *package* del simulador se iniciará con los siguientes comandos.

```
cd <ruta de carla>
./CarlaUE4.sh
```

3. Iniciar el puente de ROS. Para iniciar el puente de ROS se utiliza el siguiente comando.

```
ros2 launch carla_ros_bridge carla_ros_bridge.launch.py passive:=True
```

   Es posible que, al ejecutarse de un error, si ocurriese esto debe ejecutarse el paso número 4 antes que este.

   Al iniciar el puente de ROS se cambia el mapa por el predefinido en el archivo *carla_ros_bridge.launch.py* que se encuentra dentro de la carpeta *launch* del paquete "carla_ros_bridge". Esto se puede solucionar de distintas maneras:

   - Cambiar el mapa predefinido por el mapa que se quiera cargar directamente en el archivo nombrado anteriormente.
   - Cambiar el mapa que se quiera cargar en el archivo *change_map.py* y ejecutarlo con el siguiente comando dentro de la carpeta "Code".

```
python3 cahnge_map.py
```

   - Cambiar el mapa con el archivo predefinido de CARLA.

```
cd <ruta de carla>/PythonAPI/util
python3 config.py -map <nombre del mapa>
```

4. Ejecutar el nodo para generar los agentes de la simulación.

```
ros2 launch carla_spawn_objects carla_spawn_objects.launch.py
objects_definition_file:=path/to/objects.json
```

   En el caso de que la carpeta del directorio se encuentre en la raíz, el comando, para generar los agentes del primer par de rutas será:

```
ros2 launch carla_spawn_objects carla_spawn_objects.launch.py
objects_definition_file:=home/<nombre del equipo>/Code/objects.json
```

   Y para la segunda pareja de rutas:

```
ros2 launch carla_spawn_objects carla_spawn_objects.launch.py
objects_definition_file:=home/<nombre del equipo>/Code/objects_2.json
```

5. Ejecutar el nodo que “Fusion Lidar”.

```
ros2 run my_python_TFG fusion_lidar
```

6. Ejecutar las rutas definidas. Para ejecutar los siguientes comandos tiene que encontrarse en la carpeta “Code”. Dependiendo de si se quiere ejecutar la ruta 1.1, ruta 1.2, la ruta 2.1 o la ruta 2.2 se ejecutará los siguientes algoritmos, respectivamente.

```
python3 route_11.py
```

```
python3 route_12.py
```

```
python3 route_21.py
```

```
python3 route_22.py
```

Además de estos comandos, se han añadido diferentes archivos que facilitan la modificación de rutas y consulta de parámetros. Todos estos archivos se ejecutan desde la carpeta “Code” y usando “python3”.

- *Spawn_point.py*. Genera los números de los puntos de aparición que CARLA propone en cada uno de los mapas.
- *Spawn_route.py*. Genera los números de los puntos de aparición que se especifiquen.
- *Delete_actors.py*. Borra los actores de la simulación.
- *Restart_route1.py*. Borra los actores y los genera en su posición de aparición sin necesidad de utilizar el puente de ROS. Este archivo se puede utilizar si se quiere visualizar la ruta 1.1 o la ruta 1.2 sin tener ningún sensor ni utilizar el puente de ROS.
- *Restart_route2.py*. Borra los actores y los genera en su posición de aparición sin necesidad de utilizar el puente de ROS. Este archivo se puede utilizar si se quiere visualizar la ruta 2.1 o la ruta 2.2 sin tener ningún sensor ni utilizar el puente de ROS.
- *Draw_route.py*. Dibuja en una gráfica la ruta guardada en un archivo .bag. Para utilizar este archivo hay que hacer uso del nodo explicado posteriormente y guardar la información usando “ros2 bag”. Además, habrá que hacer la instalación de otro paquete con el siguiente comando.

```
pip3 install rosbags
```

Dentro del paquete “my_python_TFG” se encuentra el nodo “Publica ruta”, que publica la ruta seguida por el actor que se configure por un topic. De esta manera la información sobre la ruta seguida se puede guardar en un .bag.

Para ejecutar el paquete rviz2 basta con ejecutar en un terminal:

```
ros2 run rviz2 rviz2
```

Una vez abierto el programa y generado los agentes desde el puente de ROS se pueden añadir *displays* directamente buscando entre los *topic's* que están activos.

Por último, para aprender a usar el paquete de ros2 bag seguir el tutorial de la página oficial [8].