uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Electrónica Industrial y Automática
(2017-2018)

Trabajo Fin de Grado

"Implementación de funcionalidades para sistema de detección, clasificación y seguimiento de vehículos y peatones basado en información LiDAR"

Carlos Melero Vilches

Tutor

Jorge Beltrán de la Cita

Lugar y fecha de presentación prevista

Leganés, 5 de julio de 2018

RESUMEN

En el mundo actual, los accidentes relacionados con los automóviles y la conducción suponen una de las principales causas de muerte en el primer mundo. Por encima de otros factores externos, el factor humano es sin duda el más influyente como su causa; por ello, surge la necesidad de un progresivo desarrollo e incorporación de sistemas de automatización y control inteligentes en los llamados vehículos autónomos.

La presente memoria recoge la implementación de varias funciones relacionadas con la visión por ordenador aplicada a vehículos inteligentes, y están divididas en dos partes. La primera parte consiste en la obtención de imágenes a modo de vista de pájaro utilizando datos de nube de puntos obtenidos de sensores LiDAR (*Light Detection and Ranging*, es decir, visión y medición mediante luz), permitiendo un mejor estandarizado y procesado de la información entre distintos tipos de dispositivo. La segunda consiste en un sistema de *tracking*, que consiste en la asignación de un número de identificación a cada agente (peatón, ciclista, vehículo, etc.) detectado y mantenerlo en el tiempo.

Además, se presentan resultados para probar la utilidad y fiabilidad de las funciones desarrolladas, los cuales han sido obtenidos en entornos reales, cuya exigencia e impredecibilidad son superiores a situaciones llamadas "de laboratorio", donde se controlan diversas variables o elementos del sistema.

Palabras clave: Visión por ordenador, vehículos inteligentes, seguimiento de agentes, reconocimiento de objetos, sistemas en tiempo real.

ABSTRACT

In the current world, driving accidents are one of the main death causes in the first world. Above other external factors, the human factor is without a doubt the most influential as their cause; that's why it arises a necessity of a progressive development and incorporation of systems of automation and intelligent control in the so-called autonomous vehicles.

The following report comprises the implementation of several functions related with computer vision applied to autonomous vehicles, which are divided in two parts. The first one consists on the obtaining of images in birdview fashion using point cloud data obtained from LiDAR sensors (*Light Detection and Ranging*), which allows for a better standardizing and processing of the information between several types of device. The second one consists on a tracking system, which assigns an ID to every detected agent (pedestrian, cyclist, vehicle, etc.) and maintains it throughout time.

Furthermore, results are presented to prove the utility and reliability of the developed functions, which have been obtained in real environments, whose levels of exigence and randomness are higher than in "laboratory situations", where several variables or elements are controlled.

Keywords: Computer vision, intelligent vehicles, agent tracking, object recognition, real-time systems.

AGRADECIMIENTOS

En primer lugar, mi agradecimiento más importante a Jorge Beltrán, mi tutor del Trabajo, por confiar en mí, ayudarme en todo el proceso y desarrollo del proyecto y por su paciencia infinita conmigo. Como tutor ha sido excelente, y como mi profesor en el último curso puedo decir que es un ejemplo a seguir.

En segundo lugar, agradecimientos a mi familia y a mi pareja por aguantarme durante todo este tiempo y apoyarme lo mejor que han sabido para que terminase la carrera. Os quiero.

Por último, a todos mis compañeros, excompañeros, profesores y conocidos de la Universidad; especialmente a Daniel Florez por sus acertados comentarios para ayudarme a realizar esta memoria. Aunque no todo han sido buenos momentos durante estos cinco años, lo que queda al final es lo positivo, y en ese sentido me llevo de la Universidad muy buenas experiencias y relaciones, no solo a nivel académico sino también a nivel personal. Elegir la Universidad Carlos III fue un completo acierto.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación

Actualmente, la industria del automóvil representa un 4% del Producto Interior Bruto de la Unión Europea [1]. La cantidad de vehículos tiene una tendencia no solo creciente sino además exponencial, como puede observarse en la Figura 1, por lo que el sector automovilístico se asienta como uno de los más importantes del primer mundo actual y en el futuro cercano.

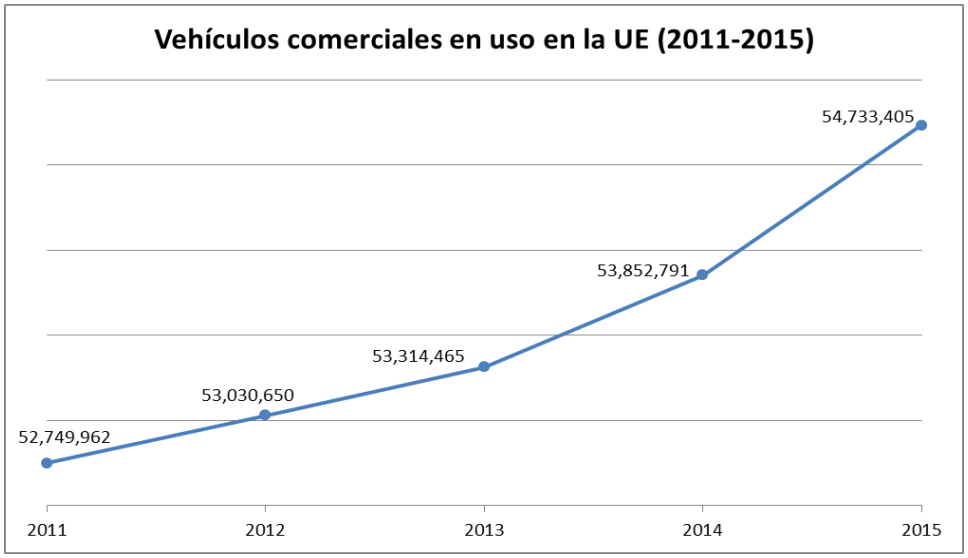

Fig. 1: Vehículos comerciales en uso en la Unión Europea (2011-2015).
Fuente: ACEA Vehicles in Use Report, 2017. [2]

Esta presencia generalizada de vehículos en las carreteras conlleva un gran factor de riesgo de accidentes. En la Figura 2 se puede ver que los accidentes de tráfico son una de las principales causas de muerte en el mundo, superando enfermedades como el sida o la tuberculosis, o las muertes por suicidio.

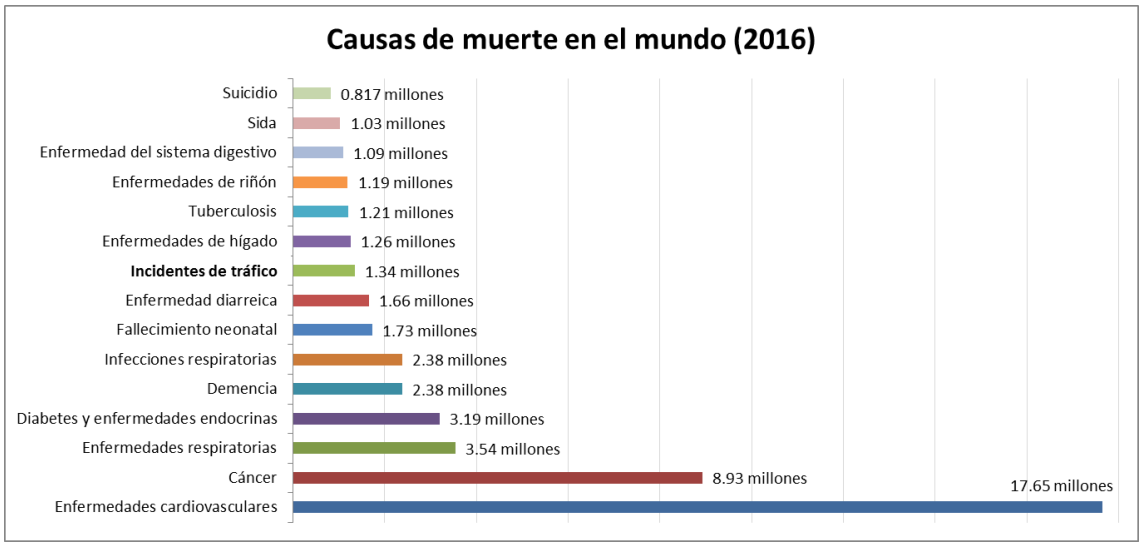

Fig. 2: Causas de muerte en el mundo (2016).
Fuente: Global Burden of Disease Study, 2016. [3]

La seguridad queda entonces como uno de los factores clave no solo para la consolidación del sector, sino para su mayor desarrollo. A lo largo de los años se han ideado diferentes sistemas para mejorar la seguridad de los vehículos. Por un lado, tenemos los sistemas de seguridad pasivos, los cuales tratan de minimizar los daños en caso de accidente (por ejemplo, cinturones de seguridad o airbags). Por otro lado, los sistemas de seguridad activos buscan evitar los accidentes o potenciales situaciones de peligro antes de que sea demasiado tarde (aquí entrarían elementos como mejores sistemas de frenado, suspensión, etc.).

Desde hace unos años, se están introduciendo en el sector nuevos sistemas de seguridad activa que combinen la mecánica del vehículo con la electrónica y la computación. De este modo, el vehículo puede realizar diversas funciones relacionadas con la visión y detección de elementos del entorno para facilitar el manejo del vehículo al conductor del mismo. Hoy en día, es ya frecuente encontrar coches con, por ejemplo, ayudas al aparcamiento que detectan obstáculos inminentes.

Aquí aparecen los vehículos inteligentes o autónomos, que tratan de ir más allá y proponer que el vehículo tenga independencia para sustituir completamente la labor del conductor. Esto supondría una mejora evidente en la seguridad del conductor y pasajeros del vehículo, pero también en la de peatones, ciclistas y motociclistas ajenos al vehículo y que suponen un 49% de las muertes totales en accidentes de tráfico [4].

Actualmente, los vehículos inteligentes son una idea en desarrollo incapaz de sustituir a un conductor humano. Como se muestra en la Figura 3, se planea que en los próximos años estos sistemas no solo se desarrollen completamente sino que se implementen y generalicen en la sociedad, de tal forma que el vehículo no solo detecte elementos "no autónomos" (peatones, ciclistas y otros elementos del entorno que no forman parte del conjunto de vehículos inteligentes), sino que también serán capaces de comunicarse con otros vehículos autónomos y actuar de forma autosuficiente en respuesta.

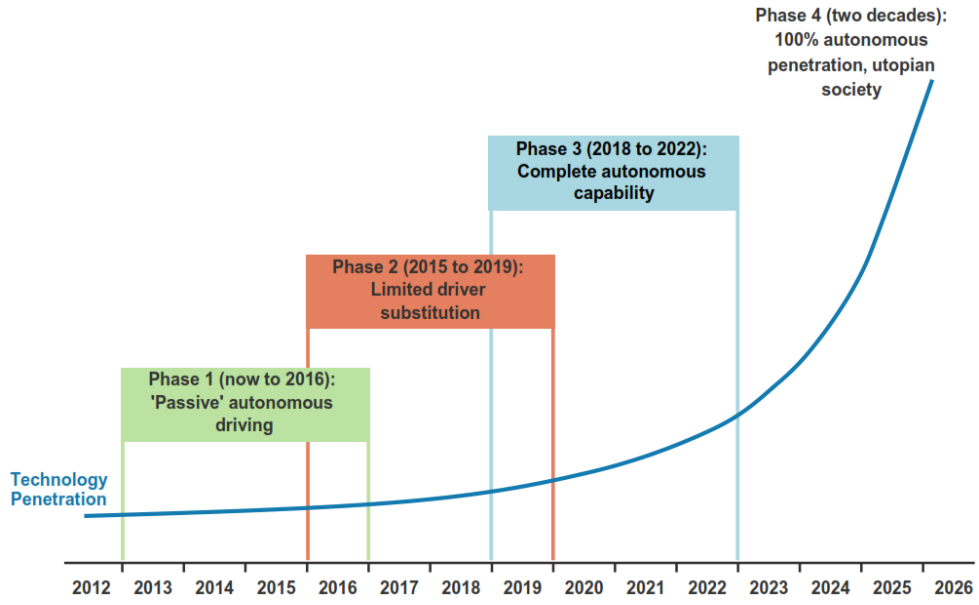

Fig. 3: Proyección de 2013 sobre el futuro de los vehículos autónomos.
Fuente: Morgan and Stanley, 2013. [5]

1.2. Descripción del proyecto y objetivos

Los sistemas de percepción de vehículos autónomos utilizan diferentes sensores y elementos para obtener la mayor cantidad de información posible del entorno. Algunos de estos elementos son: radares, dispositivos GPS, sensores LiDAR, Unidades de Medición Inercial (IMU), etc. Estos elementos aportan información de diversa índole, con la cual se trabaja posteriormente para convertirla en información útil para una navegación automatizada.

El presente proyecto se ha dividido en dos bloques principales. Para la primera parte, se ha utilizado información obtenida de sensores LiDAR, que se basan en el disparo de haces de luz (láseres) con forma de conos concéntricos que apuntan hacia el suelo y van progresivamente alejándose del vehículo. De esta forma, se puede hacer un mapa a modo de nube de puntos con los puntos de corte de cada láser con el entorno, en el que el vehículo se situaría en el centro. En la Figura 4 puede verse un ejemplo de nube de puntos obtenida mediante sensores LiDAR.

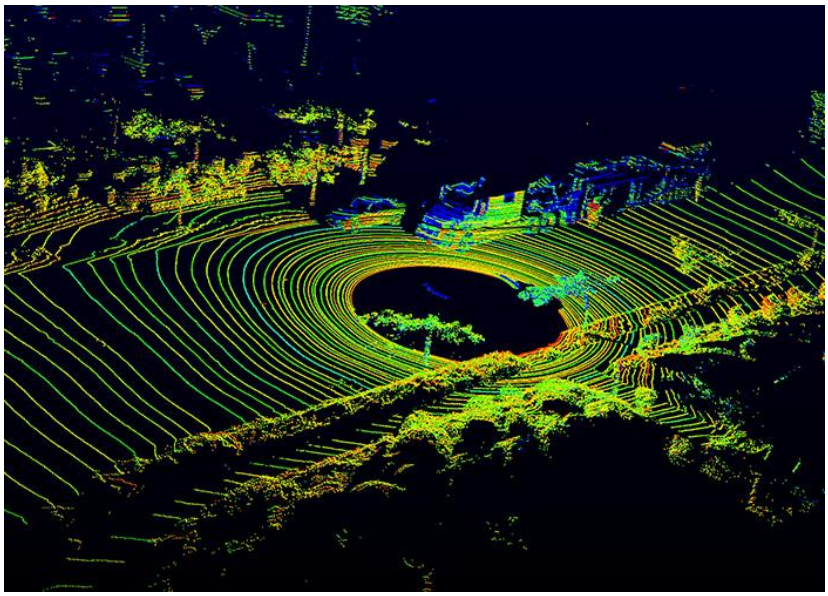

Fig. 4: Ejemplo de nube de puntos LiDAR.
Fuente: Ejemplos de datos de usuarios de la web de Velodyne. [6]

La cantidad de planos, o conos, viene determinada por el modelo del sensor. En esta primera parte, se ha partido de código funcional para el sensor Velodyne de modelo HDL-64E para, además de añadir pequeñas nuevas funcionalidades, poder utilizarlo también con los modelos HDL-32E y VLP-16. Esto nos permite poder utilizar indistintamente cualquiera de estos tres modelos sin tener que utilizar proyectos o código distintos ni realizar modificaciones a los mismos.

A la hora de representar e interpretar los datos LiDAR, se ha planteado una conversión de los datos de la nube de puntos a archivos de imagen, también llamados canales. En cada una de estas imágenes, cada pixel se puede interpretar como una celda de una matriz. Esta matriz representa el espacio alrededor del vehículo (el cual estaría en el centro de la imagen) a modo de vista de pájaro; es decir, visto desde arriba. Por ello,

cada imagen abarca cierta distancia alrededor del vehículo, con cada casilla representando una porción definida de este terreno, y cuyo valor en cada casilla (que se traduce en un color o tonalidad distintos en el pixel de la imagen) dependerá del tipo de información obtenida y de su valor en esa porción de terreno determinada.

Para garantizar el correcto funcionamiento de los tres modelos de sensor, es necesario que uno de estos canales represente la densidad de los puntos medida en cada celda. Este canal ha de tener en cuenta no solo la dispersión de los propios planos (que, como ya se ha dicho, provocaría que un mismo elemento situado cerca o lejos del vehículo generase distinta cantidad de puntos detectados) sino la cantidad de planos y haces de luz generados en cada modelo de sensor; cantidades que dependen de los parámetros internos de cada uno de éstos, y que se han de incluir de un modo u otro en el código.

La segunda parte del proyecto parte de datos de detecciones en 3D (LiDAR, detecciones de imagen, datos GPS, IMU, etc.) para asignar números de identificación, o ID, a diferentes agentes, a la vez que se visualiza un seguimiento mediante *bounding boxes* (también llamados ROI, o *Regions Of Interest* en inglés); cuadros delimitadores que nos indican la ubicación, tamaño e ID del agente, así como su tipo (peatón, vehículo, ciclista, etc.). Un ejemplo de cuadros delimitadores se puede ver en la Figura 5. A la hora de comprobar los resultados, se han usado imágenes pertenecientes al benchmark KITTI [14], que será explicado con más detalle más adelante.

Fig. 5: Ejemplo de bounding boxes con etiqueta.
Fuente: C. H. Lampert, M. B. Blaschko y T. Hofmann. [7]

La información de las detecciones 3D nos da una gran cantidad de parámetros tanto del vehículo (ubicación GPS, orientación, velocidad lineal, velocidad angular…) como de los agentes detectados, tanto a nivel relativo del vehículo (distancia) como puestos en relación a la imagen de la cámara frontal (posición en la imagen, dimensiones de la bounding box). A la hora de convertir esta información en un seguimiento con bounding boxes correcto, no basta únicamente con leer los datos de bounding box de los datos de imagen, sino que hay que asignar IDs de forma adecuada y mantenerlos con el paso del

tiempo. Sin un correcto seguimiento, en la imagen aparecerían las cajas dibujadas correctamente, pero ningún tipo de identificación en los agentes. Esto impediría realizar ningún tipo de predicciones futuras ya que se desconoce el comportamiento individual de cada agente; estas predicciones resultan de gran utilidad a la hora de tomar decisiones de forma anticipada y prevenir situaciones peligrosas.

Por lo tanto, el propósito de esta parte del proyecto es asignar una ID a cada agente y, mientras éste siga apareciendo en la imagen, que su ID se mantenga asignada a él. Una vez el agente desaparezca de la imagen, esta ID se descarta completamente. A la hora de asignar y mantener los ID, se ha utilizado el algoritmo de Munkres, o algoritmo húngaro, para poder relacionar los agentes detectados en el momento anterior con los del momento actual buscando el camino, o la combinación de asignaciones en este caso, de menor coste. Se han planteado tres formas diferentes de aplicar el algoritmo, las cuales serán explicadas con más detalle en el Capítulo 5.

Este proyecto da pie a posibles ampliaciones en las funcionalidades, como por ejemplo predicción de movimiento para un mejor seguimiento o división del PointCloud en varios sectores para una mejor identificación de elementos y entorno. Estos posibles trabajos futuros se explican con más detalle en esta memoria, en el capítulo con mismo nombre.

2. ESTADO DEL ARTE

Pese a los recientes avances en visión por computador, machine learning (aprendizaje autónomo) y vehículos inteligentes, aún estamos a décadas de conseguir coches capaces de una conducción completamente independiente. Esto se debe principalmente a dos motivos [8]:

- Los vehículos autónomos requieren de inteligencia artificial, la cual trata de solventar problemas de forma genérica y no está preparada, al menos por el momento, para enfrentarse a problemas complejos y dinámicos.
- Para tomar decisiones es necesario una percepción (visión autónoma) precisa del entorno. Actualmente, la mayoría de sistemas de visión por ordenador producen una cantidad de errores inaceptable para poder navegar con autonomía de forma segura.

2.1. Historia de la conducción autónoma

Los primeros proyectos de ITS (*Intelligent Transportation Systems*, Sistemas de Transporte Inteligente en español) aparecieron tan pronto como en 1986 con el proyecto PROMETHEUS, el cual se desarrolló en Europa e implicó a 13 fabricantes de vehículos y unidades de investigación de gobiernos y universidades de 19 países europeos [8]. Este proyecto desarrollaba un vehículo con un sistema VITA (*Vision Information Technology Application*, Aplicación de Tecnologías de Información de Visión en español) que, mediante pequeñas cámaras de vídeo, detectaba si el vehículo estaba en riesgo de colisión con otros objetos, y le permitía acelerar, frenar y manejar la dirección.

En 1995 se consiguió en Estados Unidos la primera conducción autónoma entre dos ciudades [9]. Durante esta época aparecen las primeras asociaciones, en EE.UU. y Japón, que promovían y apoyaban las ayudas a la conducción automática. Posteriormente, en el grupo VisLab se logró realizar un viaje semi-autónomo de Italia a China con el prototipo BRAiVE [10]. Este mismo prototipo, ligeramente modificado y mostrado en la Figura 6, fue utilizado en 2015 para una primera toma de contacto de conducción autónoma por ciudad en Parma, Italia, donde el vehículo pudo manejar situaciones como túneles, semáforos o cruces sin intervención humana.

Fig. 6: Prototipo BRAiVE modificado.

Pese a un incremento en la potencia y capacidad de los ordenadores de la época, los principales problemas para la conducción autónoma recaían en reflejos, carreteras mojadas, brillo solar, sombras y túneles; así como aspectos legales referidos al impacto y responsabilidad de la conducción autónoma con pasajeros humanos [11]. Así, surge la necesidad de un desarrollo gradual de infraestructuras específicas.

La conocida compañía Google también emprendió un proyecto de coche autónomo en 2009, el cual fue capaz de recorrer más de 2 millones de kilómetros desde ese año hasta 2016 [12], a la vez que se utilizaron elementos como cámaras, radares o LiDAR para detectar elementos en todas direcciones; este proyecto se relacionó con 14 accidentes de tráfico, donde se concluyó que 13 de éstos fueron provocados por causa ajena al vehículo [8, 12].

Tesla es otra conocida empresa que desarrolló un sistema de piloto automático en 2015, con la diferencia de que este sistema requería la atención del conductor para retomar el control si fuese necesario [13].

2.2. Percepción en vehículos autónomos

2.2.1. Detección basada en sensores LiDAR

Los sensores láser LiDAR, utilizados en este proyecto, proveen información en 3D de forma precisa; esto tiene gran utilidad a la hora de distinguir formas y figuras. Es por ello que estos sensores son los más utilizados para la reconstrucción urbana, tanto a nivel aéreo con sensores ubicados en aviones como a nivel de tierra con sensores en vehículos. Sin embargo, el principal inconveniente de detectores láser es que los datos suelen ser dispersos y con una resolución espacial limitada [8], lo que hace que la tarea de detección de agentes se complique, si bien aportan datos con muy buena resolución para la modelización del entorno, como se ve en la Figura 7.

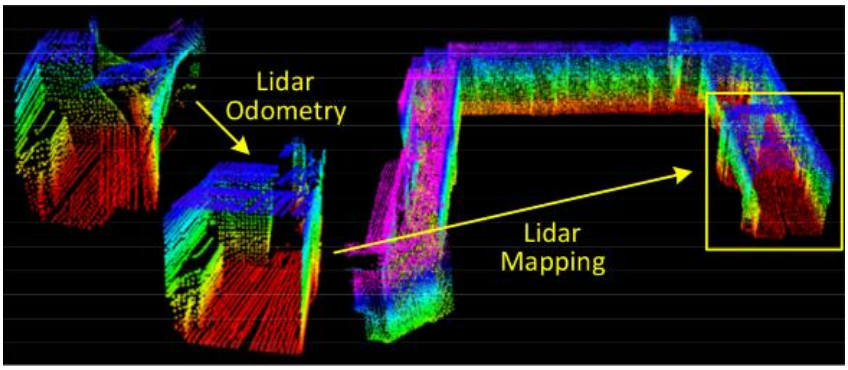

Fig. 7: Escaneo de un mapa utilizando odometría LiDAR.
Fuente: J. Zhang y S. Singh. [29]

Inicialmente, la estimación del *ego-motion* (es decir, la posición y orientación del vehículo) se realizaba previamente mediante encoders en las ruedas, pero este método era muy susceptible de condiciones adversas y deslizamiento de ruedas y, una vez los datos obtenidos resultaban erróneos, no se podían corregir. Con la aparición de la odometría LiDAR, los métodos más efectivos pasaron a estar basados en la utilización

de PointClouds en la estimación del ego-motion; esto fue desarrollándose hasta conseguir mapeos urbanos completos aplicando odometría y mapeado LiDAR. En concreto, la odometría LiDAR se realizaba a alta frecuencia con baja fidelidad, y relacionaba y comparaba dos escaneos consecutivos. El mapeado LiDAR era a baja frecuencia, y simplemente buscaba coincidencias y registraba los nuevos escaneos en un mapa.

Los sensores LiDAR tenían la competencia de las cámaras estéreo, que producían resultados competitivos a un menor precio. No obstante, hubo varios factores que hacían que los sensores LiDAR destacaran; uno de ellos era su gran rendimiento para la estimación del ego-motion. Otro factor fue su su gran contribución al SLAM (Localización y Mapeado Simultáneos), debido a que los sensores LiDAR, al estar basados en el rebote de los haces de luz, no dependen de la iluminación del entorno. Utilizando escáneres LiDAR se pudo crear un mapa que consistía en elementos del entorno que eran "muy probablemente" estáticos [30]; esto permitió un gran avance en uno de los mayores problemas del SLAM que eran los cambios de entorno. Este mapa binario fue posteriormente extendido a un mapa probabilístico donde cada celda del mapa representaba la probabilidad de que fuese un elemento no cambiante. En la Figura 8 se puede ver uno de los resultados experimentales, que corresponde con un mapeado de la ciudad de Burlingame, donde las zonas rojas respresentan la zona apta para conducción (la calzada), obtenidas superponiendo los resultados obtenidos tras diversas rutas y eliminar elementos "dinámicos" del entorno, como vehículos o peatones.

Fig. 8: Mapa aéreo de Burlingame (EE.UU), donde en rojo se muestra la calzada.
Fuente: J. Levinson, M. Montemerlo y S. Thrun. [30]

Hay que destacar también el trabajo de L. Wang et al. [46] que desarrollaron una localización y dibujado de mapa basado en detección de bordillos. Un ejemplo de dicho mapa obtenido puede verse en la Figura 9, que es el resultado de información LiDAR combinada con datos de GPS para la obtención de estimaciones de movimiento y de extracción de contornos, tras lo cual se realizaría un proceso de matching con el mapa digital y una aplicación de un filtro dual de Kalman.

Fig. 9: Mapa digital obtenido mediante detección de bordillos.
Fuente: L. Wang et al. [46]

En la Figura 10 se muestra un esquema que utiliza tanto datos de tipo LiDAR como datos de imagen obtenidos de la cámara frontal del vehículo en una red de tipo *deep fusion*, que permite combinar diferentes tipos de datos, como los mencionados, obteniendo así una mejora en los resultados obtenidos experimentalmente.

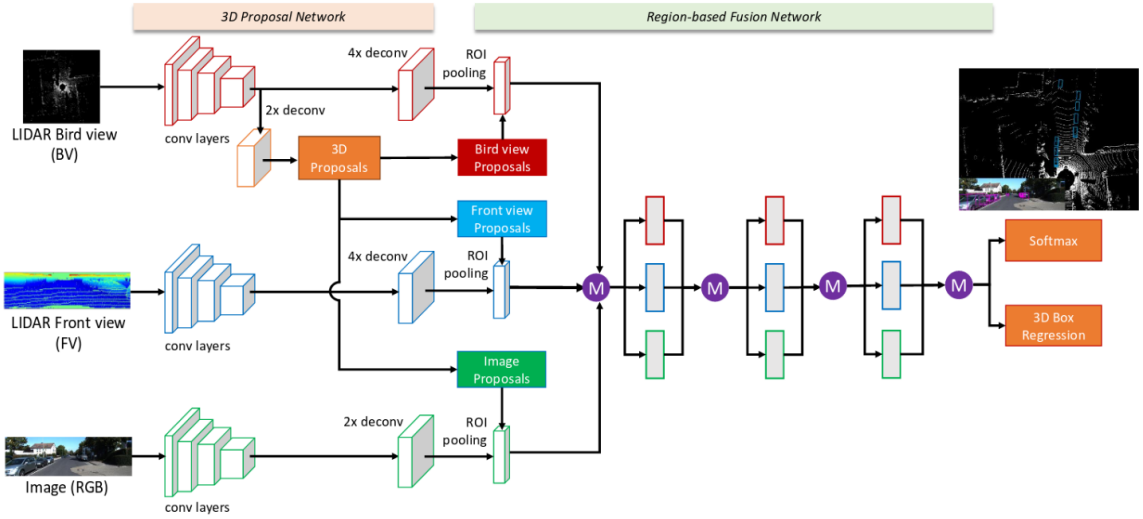

Fig. 10: Esquema planteado por Chen et al. [26], que utiliza la vista de pájaro, la vista frontal del PointCloud del LiDAR y la imagen RGB de la cámara como inputs de una red *deep fusion*.

Como puede verse, los datos LiDAR son muy útiles para funciones de mapeado y localización del entorno, y que permiten compatibilidad con datos de otro tipo, como GPS o información de imagen, para un análisis más completo.

2.2.2. Detección basada en imagen

En la conducción autónoma es de vital importancia la detección no solo del entorno, sino de peatones y otros agentes de forma individual, y suponen un paso más de dificultad debido a la impredecibilidad y variaciones de los agentes. Por ejemplo, un peatón puede adoptar una gran variedad de posiciones y diferentes vestimentas [8], así como moverse de forma inesperada.

Si bien existen vehículos con PPS (*Pedestrian Protection System*, sistemas que avisan al conductor de la presencia de un peatón cerca del vehículo), estos sistemas no son infalibles y pueden obtener falsos positivos (o falsos negativos, lo cual es aún peor ya que se pone en riesgo la salud de los peatones). En ambos casos, todavía se tiene el elemento del conductor para rectificar como fuese necesario; no obstante, si se quiere alcanzar una condición autónoma estos errores no pueden suceder.

Esto hizo que comenzasen una serie de mapeados y mediciones de datos relacionados con la obtención de características de peatones y vehículos a la hora de su correcta clasificación. Hay que destacar la importancia del Histograma de los Gradientes Orientados (HOG), un descriptor de la imagen que nos permite identificar y diferenciar diferentes elementos basándose en los cambios de color e intensidad de la imagen. Es uno de los descriptores más importantes debido a su sencillez de aplicación, a la vez que aporta resultados muy positivos para detección de objetos, tal como puede verse en la Figura 12.

De forma muy reciente, se ha llegado a la conclusión de que la mejor solución es la utilización de redes neuronales de *deep learning* para promover un aprendizaje autónomo por parte de la máquina. El deep learning es un sistema de aprendizaje que utiliza una gran cantidad de datos de entrada como referencia, a los cuales se les asignan ciertas definiciones. Estos datos son analizados y combinados entre sí de todas las formas posibles, como muestra la Figura 11, de tal modo que el sistema identifica internamente qué características corresponden a unas definiciones u otras. En el caso concreto de vehículos autónomos, las bases de datos de imágenes se dividen en imágenes clasificadas como peatones, vehículos, ciclistas, etc., que se usan para "entrenar" al sistema y permitirle que identifique las características relevantes.

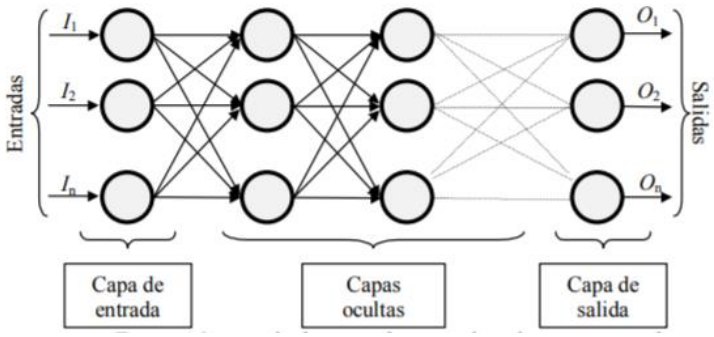

Fig. 11: Esquema de redes neuronales. Diferentes combinaciones entre elementos hasta llegar a la salida.

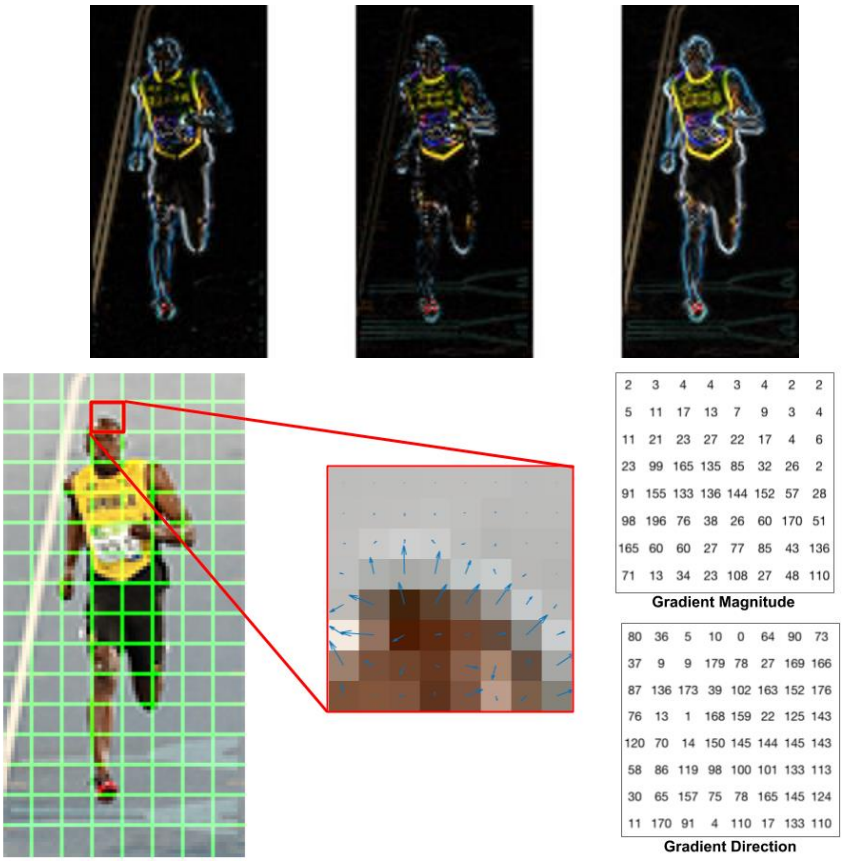

Fig. 12: Arriba, ejemplo de aplicación el HOG en el eje x (izquierda), en el eje y (centro), y en ambos (derecha). Abajo, subdivisión de la imagen en celdas 8x8 y representación de vectores de gradiente para una celda de ejemplo.
Fuente: Satya Mallick, 2016, en *Learn OpenCV*. [45]

Esto nos lleva a la importancia de una buena elección de imágenes y elementos que incorporar en el entrenamiento de estas redes; gran parte de éstos son altamente complejos, especialmente para la detección de personas, donde influyen el estilo de andar o la posición de las piernas en ciertas poses. Inicialmente, solo se utilizaban imágenes estáticas, pero el paso de los años ha permitido un gran avance en las investigaciones, que ya incorporan incluso elementos de vídeo (y, con ello, un análisis más completo de los elementos, incluyendo características dinámicas como el movimiento y la velocidad).

En la Figura 13 se muestra un ejemplo de imágenes utilizadas como parte de la base de datos para deep learning en detección de vehículos. Hay que destacar que estas imágenes son solo unos ejemplos, y en la realidad las bases de datos están formadas por miles, si no millones, de imágenes que analizar. Por ello, los entrenamientos de los sistemas de deep learning suelen llevar una gran cantidad de tiempo hasta que finalizan, si bien suelen ofrecer resultados de gran calidad y fiabilidad, especialmente con bases de datos lo suficientemente grandes y de información variada.

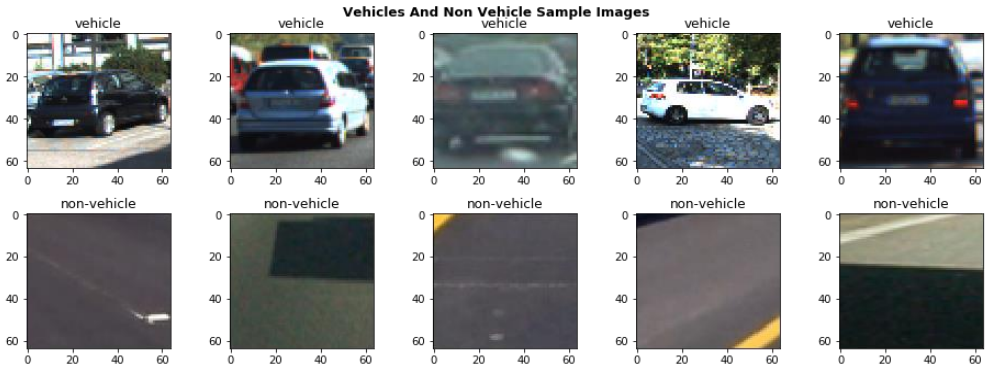

Fig. 13: Ejemplo de imágenes de vehículos (arriba) y no vehículos (abajo) a utilizar en procesos de deep learning.
Fuente: Towards Data Science, 2017. [44]

Una vez entrenado el sistema, éste se aplica en casos de imágenes reales para la búsqueda de agentes. El análisis de la imagen no se realiza como un único bloque, sino que es necesario analizar porciones más pequeñas de la misma. Al fin y al cabo, las imágenes de la base de datos corresponden a vehículos en situaciones reales, pero la imagen de los mismos no muestra el entorno completo; tiene sentido entonces que haya que analizar porciones más pequeñas de la imagen para la detección de vehículos tras el entrenamiento del sistema.

En el caso de detección de vehículos, se han desarrollado diferentes métodos para buscar la porción de imagen que interesa, la cual idealmente abarca el vehículo en cuestión en su totalidad y el mínimo posible del resto del entorno, para evitar errores. Uno de estos métodos, mostrado en la Figura 14, es el de las *Sliding Windows*, que divide la imagen en porciones con posición y dimensiones determinadas y se analizan las regiones positivas. Generalmente, se utilizan porciones más grandes en la zona inferior de la imagen (ya que los coches están más cerca debido a la perspectiva) y se deja de analizar la imagen cuando se excede de cierta altura.

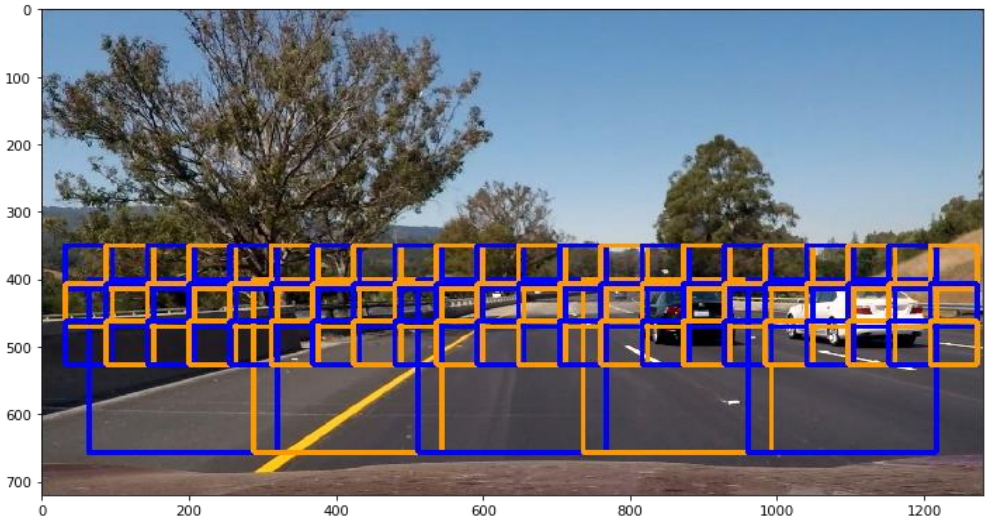

Fig. 14: Ejemplo de subdivisión de regiones en una imagen aplicando *Sliding Windows*.
Fuente: Towards Data Science, 2017. [44]

Otro método utilizado para la obtención de las regiones de los vehículos es mediante mapas de calor y definición de umbrales (*thresholding*) para los mismos. En la Figura 15, el mapa de calor de la izquierda muestra los datos originales, mientras que el de la derecha es el resultado de aplicar el thresholding, que modifican a 0 el resto de valores del mapa de calor por debajo de este umbral.

Fig. 15: Ejemplo de subdivisión de regiones en una imagen aplicando mapas de calor y thresholding.
Fuente: Towards Data Science, 2017. [44]

Para la detección de personas, los métodos anteriores también se aplican, pero hay que tener en cuenta los problemas específicos de la detección de peatones. El más importante de ellos es la detección de la pose, la cual es necesaria para estimar el comportamiento y la intención del individuo. Este es un problema muy complejo de resolver, y más teniendo en cuenta que el abanico de posiciones de una persona es muy amplio y las personas suelen aparecer en las imágenes en baja resolución y a una distancia relativamente alta. El primer gran acercamiento a este problema fue el modelo DeepCut de 2016 [28]; un ejemplo puede verse en la Figura 16. Éste se utilizaría como referencia para otros modelos posteriores que ofrecen buenos resultados al problema de la estimación de poses en peatones, que conllevaría en una mejor detección de éstos.

Fig. 16: Ejemplo de estimación de poses en personas usando DeepCut.
Fuente: L. Pishchulin et al. [28]

Otro factor muy importante es la velocidad de detección, ya que en caso de accidente es imperativo que la reacción del vehículo autónomo sea tan rápida como sea posible. En este sentido, cabe destacar la labor de Badino et al. [27], cuyo trabajo permitió la detección de peatones a 80 hercios, reduciendo notablemente el rango de búsqueda.

2.2.3. Bases de datos y benchmarks

Las bases de datos (*datasets* en inglés) son un factor clave en el progreso de los campos de investigación ya que proveen ejemplos reales ante los problemas presentados. De aquí proviene el término *ground truth*, que significa simplemente un conjunto de datos obtenidos de forma experimental; en el caso específico de visión por computador, ground truth se refiere a la precisión de la clasificación de los elementos de entrenamiento utilizando técnicas de aprendizaje supervisado.

Bajo el contexto de vehículos autónomos, los dos datasets más relevantes son KITTI [14] y Cityscapes [15], que se muestran en las Figuras 17 y 18 respectivamente. Estos datasets han servido de puente entre situaciones "de laboratorio" y situaciones reales de conducción, mucho más complicadas e impredecibles. Hasta hace pocos años, datasets con unos cientos de ejemplos eran suficientes para un aprendizaje autónomo correcto en gran parte de las situaciones estudiadas. Sin embargo, la introducción de datasets con una cantidad de datos mucho mayor ha permitido grandes progresos en la visión por computador a través del entrenamiento de modelos de alta capacidad de forma supervisada.

Por otro lado, los *benchmarks* se refieren a las diferentes técnicas que se crean para el cálculo y la comparación de diferentes algoritmos. En concreto, cabe destacar el benchmark estéreo de Middlebury, introducido por Scharstein y Szeliski en 2002 [18], que utilizaba iluminación estructurada y el cual se fue desarrollando durante los siguientes años hasta conseguir una gran precisión en la detección de la profundidad en imágenes, como se ve en la Figura 19.

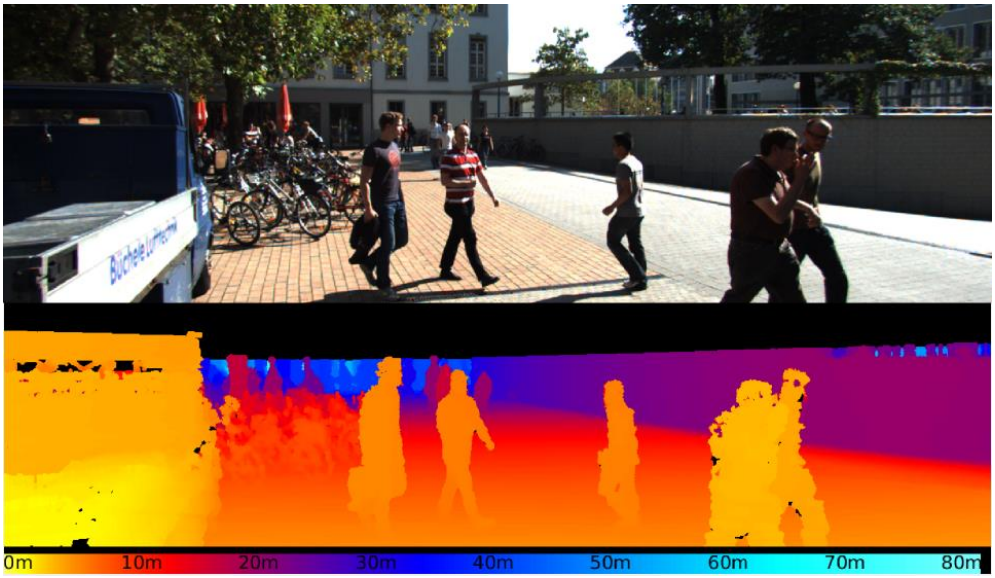

Fig. 17: Imagen real (arriba) e imagen KITTI de profundidad obtenida (abajo).
Fuente: Ghent University, Faculty of Engineering and Architecture. [16]

Fig. 18: Clases del dataset Cityscape superpuestas en la imagen original.
Fuente: Cityscapes Dataset. [17]

Fig. 19: Sistema de luz estructurado de Scharstein et al. [19]

Otro benchmark que cobró gran importancia en el reconocimiento y clasificación de objetos fue el PASCAL Visual Object Classes (VOC) [20], proyecto basado en una gran cantidad de imágenes con gran variabilidad en sus características obtenidas de Flickr, y que comenzó planteando únicamente 4 clases pero con el paso de los años se aumentó a 20. Para este mismo fin, apareció también el dataset Microsoft COCO [21], con más de 300 mil imágenes en total, entre las cuales se idenfiticaban 91 clases de objetos.

El siguiente paso tras la identificación de objetos era su seguimiento; ya no bastaba con identificar los objetos en un cierto momento, sino ser capaces de seguirlos en una imagen en movimiento. Este problema fue planteado en el MOTChallenge, un benchmark introducido por Leal-Taixé et al. [22] y Milan et al. [23] donde se distinguían tres clases: personas en posición vertical, personas en posición no vertical y otros agentes. Este benchmark incorpora elementos de otros benchmark, como por ejemplo el ya mencionado KITTI, que es el Benchmark en el que se basa este trabajo.

2.2.4. El benchmark KITTI en la conducción autónoma

El benchmark KITTI surgió como respuesta a una escasez de sistemas de reconocimiento visual aplicados a la robótica; para ello, se planteó una plataforma de grabación que permitiese la obtención de datos reales, debido a que los datasets en uso como, por ejemplo, Middlebury tenían un rendimiento bajo cuando se trataba de pasar del laboratorio a casos reales [14]. Inicialmente, se planteó una plataforma de grabación con tareas de estéreo, flujo óptico, odometría visual (*SLAM*: Localización y Mapeado Simultáneos) y detección de objetos en 3D, similares a la mostrada en la Figura 20. Las primeras grabaciones se extendieron durante casi 40 kilómetros, obteniendo más de 200 mil detecciones de objetos en 3D.

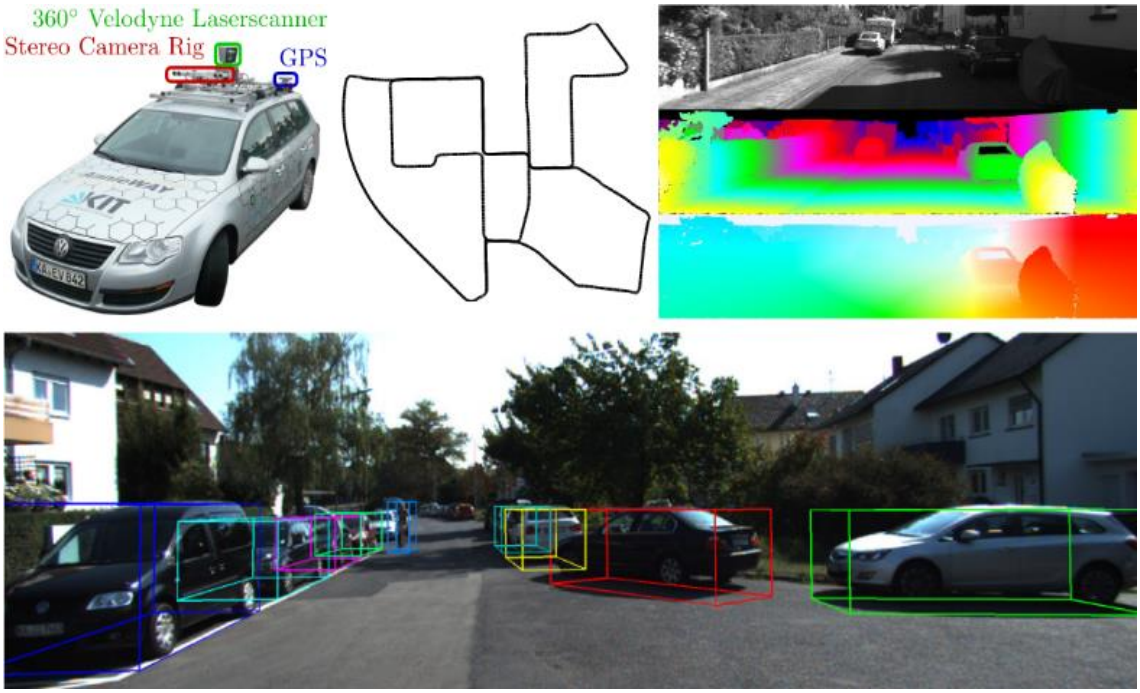

Fig. 20: Benchmark KITTI propuesto por Geiger et al. [14]. Arriba, la plataforma de grabación (izquierda), trayectoria del vehículo (centro) y disparidad y flujo óptico (derecha). Abajo, objetos 3D.

Tras su introducción en 2012, KITTI fue ampliado en 2013 [24] para abarcar la detección de carreteras y carriles. Desde entonces, KITTI se ha asentado como uno de los estándares en el campo de la conducción autónoma. No obstante, pese a estar centrado en conducción autónoma, todavía no se había abarcado el problema de autonomía a largo plazo, la cual implicaba cambios radicales de entorno.

Para tratar de resolver este problema, se presentó el Oxford robotcar dataset [25], que recogió imágenes y datos GPS y LiDAR durante 1 año y 1000 kilómetros en Reino Unido. Esto permitió investigaciones más avanzadas y arrojó luz sobre el problema de la identificación de objetos en diferentes épocas del año debido a cambios a largo plazo del entorno (cambios de clima, construcción de edificios, etc.).

Con esta información se puede llegar a la conclusión de que la detección de elementos ya tiene una efectividad muy elevada, pero solo bajo ciertas condiciones. Estas condiciones implicarían una alta resolución de imagen y, principalmente, la ausencia de

oclusiones en los agentes (es decir, elementos que tapan o bloquean a otros, tanto en la imagen como en los mapas de puntos).

El problema de las oclusiones es un problema típico en todos los sectores de visión por computador y, desafortunadamente, no tienen una solución sencilla. Actualmente, las oclusiones suponen una de las principales fuentes de errores en la detección y seguimiento de agentes en los vehículos autónomos; ejemplos típicos de esto serían grupos de personas o de ciclistas y largas líneas de coches. Ejemplos de estos casos se muestran en las Figuras 21 y 22 respectivamente. Para solventar este problema se necesita una gran precisión en los objetos y una capacidad de detección lejana; no obstante, esto no es sencillo de conseguir, ya que cuanto más alejado esté un agente del vehículo, menor información de calidad se podrá obtener de él.

Fig. 21: Ejemplo de detecciones KITTI erróneas de peatones y ciclistas.
Fuente: J. Janai, F. Güney, A. Behl y A. Geiger. [8]

Fig. 22: Ejemplo de detecciones KITTI erróneas de vehículos.
Fuente: J. Janai, F. Güney, A. Behl y A. Geiger. [8]

2.3. Seguimiento de agentes

El seguimiento de agentes es uno de los factores clave en la conducción autónoma, debido a que la distancia de frenado incrementa de forma cuadrática con la velocidad del vehículo, por lo que un vehículo autónomo ha de ser capaz de detectar, y en algunos casos predecir, la situación y movimiento de los agentes y frenar con el tiempo suficiente para evitar un accidente. Esto es especialmente complicado con peatones y ciclistas debido a su facilidad para producir movimientos erráticos o inesperados.

De este modo, el seguimiento de agentes se enfrenta a distintos y variados desafíos: Oclusiones de objetos, similitud de objetos de la misma clase (por ejemplo, coches del mismo modelo, o personas vestidas de forma similar), interacción de peatones con otros objetos (como una persona subiéndose a una bici o a un coche) o reflejos en espejos.

2.3.1. Métodos

En cuanto a los métodos de seguimientos, se van a mencionar los dos que se consideran más importantes, los modelos probabilísticos basados en Bayes y el método de coste mínimo FollowMe.

El método de Bayes está basado en la probabilidad y la estadística y, por tanto, contiene una gran base de matemática teórica que se ha preferido no recoger aquí, debido a que este método no se utiliza en el proyecto. No obstante, se puede consultar más información en la referencia [31]. Este método combina métodos de procesado de imagen con filtros Kalman y una ecuación de medición que proyecta del modelo 3 que proyecta del modelo 3D al espacio de imagen, y da como resultados muy buenas estimaciones de curvatura de la carretera, distancia y velocidad del vehículo detectado.

El método de coste mínimo [32] está basado en una asociación de un grupo de elementos en tiempos diferentes. Esta asociación está basada en ciertos parámetros que han de definirse de forma interna, como la posición de cada elemento, el color o su tamaño. Un ejemplo con casos de personas puede verse en la Figura 23, donde se comparan las imágenes de 3 personas distintas en varios momentos diferentes y se relacionan entre sí.

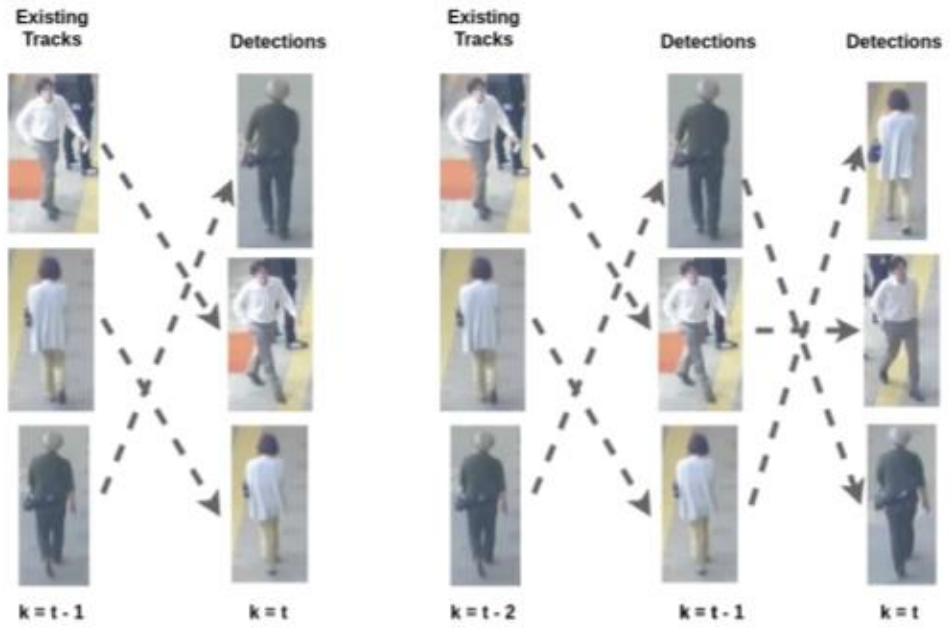

Fig. 23: Ejemplo de asociación de elementos por coste mínimo.
Fuente: P. Emami et al. [47]

Debido a que se comparan los estados de los agentes entre imágenes consecutivas, el seguimiento tendrá mayor fiabilidad cuanto mayor sea la frecuencia de las imágenes.

Este método es en el que se ha basado este proyecto para realizar el seguimiento de los agentes, concretamente utilizando el llamado método húngaro (explicado posteriormente), que nos permite resolver directamente una matriz de costes cuyos valores han sido previamente definidos y generados. Así, un dato clave en la obtención de dichos costes para poder identificar cada elemento en distintos momentos es cómo se define y se obtiene dicha matriz de costes.

Además, la utilización de este método del coste mínimo también se ha realizado en grupos de tres frames en vez de dos, como se muestra de forma esquemática en la Figura 24, donde cada línea que sale de cada elemento (en azul) representa cada posibilidad de asignación diferente a lo largo de los grupos de 3 frames. Este método aplica un nivel más de robustez al método; sin embargo, no se ha utilizado para este proyecto debido a que éste aumenta el tiempo de procesado del sistema y los resultados comparando en dos frames consecutivos son positivos.

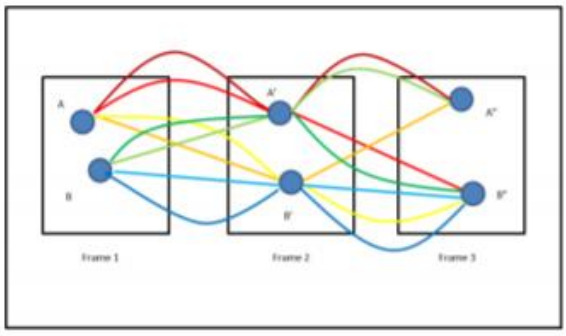

Fig. 24: Grupo de 3 frames y líneas de asignación de cada elemento.
Fuente: A. A. Butt y R. T. Collins. [48]

3. MATERIALES Y RECURSOS

En esta sección se va a hablar de los recursos utilizados durante el desarrollo del proyecto, tanto en lo referido a elementos físicos (hardware) como a programas, lenguajes de programación, etc. (software).

3.1. Hardware

El hardware utilizado tanto para el desarrollo del código como para la realización de distintas pruebas y obtención de resultados ha sido únicamente un ordenador portátil HP Pavilion G6 Notebook PC, al cual se le ha realizado una partición de disco de 92GB para la instalación del SO (Sistema Operativo) Ubuntu, ya que por defecto venía con Windows instalado. El ordenador y las principales características del sistema (a excepción de la Memoria RAM de 8 GB, que no aparece indicada) que se han utilizado se muestran en la Figura 25.

Fig. 25: A la izquierda, ordenador utilizado. A la derecha, SO y principales características.

3.2. Software

El proyecto está enfocado en el desarrollo de código y, por lo tanto, los elementos software utilizados son de gran importancia para éste. Concretamente, en esta sección se va a hablar de los elementos del software necesarios para el funcionamiento del sistema, pero no los elementos específicos requeridos para realizar partes concretas del código puesto que éstos serán explicados en su sección correspondiente más adelante.

3.2.1. Sistema operativo Ubuntu

El SO utilizado en este proyecto ha sido Ubuntu, en su versión 16.04. La utilización de Ubuntu frente a Windows, SO por defecto en el ordenador, tiene como motivo principal la mayor facilidad para descargar, utilizar y, en general, gestionar los repositorios y paquetes necesarios obtenidos desde internet. Aunque gran parte del código se ha desarrollado desde cero (sobre todo la segunda parte del trabajo), la utilización de librerías y paquetes adicionales ha sido necesaria, no solo para simplificar los

algoritmos del código creado sino para poder facilitar su comunicación y sincronización con otros elementos necesarios para el funcionamiento del sistema completo.

Como ya se ha explicado, el SO por defecto del ordenador utilizado es Windows, por lo que ha sido necesaria una partición de disco para instalar Ubuntu. Hay varias formas de realizar una partición; se ha elegido la opción de utilizar el programa de uso público Rufus, mostrado en la Figura 26, que nos ha permitido utilizar un dispositivo de almacenamiento USB para realizar la partición.

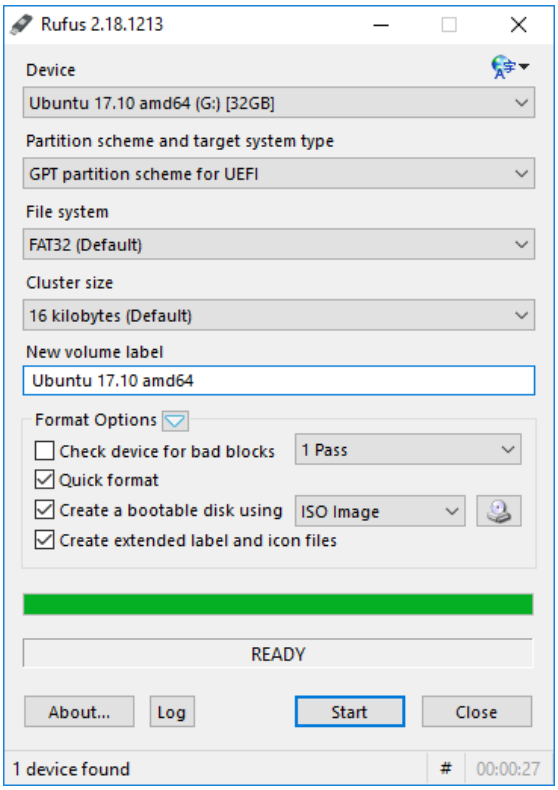

Fig. 26: Ejemplo de creación de un instalable de Ubuntu mediante USB, obtenido de la web de Rufus[1].

Una vez generado el instalable USB se puede realizar la partición en el disco, la cual se ha configurado para tener, tal como aparece en la Figura 25, 92GB de tamaño de disco; más que suficiente para la realización del proyecto y gestión de archivos necesarios para la obtención de resultados y realización de pruebas. Con Ubuntu funcionando, el siguiente paso ha sido instalar diversos programas y paquetes necesarios previo a cualquier desarrollo de código.

3.2.2. ROS Lunar

ROS (*Robot Operating System*) es un framework que proporciona herramientas relacionadas con el desarrollo software aplicado a la robótica, como drivers de dispositivos, librerías, gestión de paquetes, visualizadores, etc. ROS es una pieza indispensable para el funcionamiento tanto del código desarrollado como de las comunicaciones con el resto de elementos del sistema, y es comúnmente utilizado en

[1] https://rufus.akeo.ie/?locale

aplicaciones similares. Por ejemplo, la plataforma Udacity plantea un curso de conducción autónoma[2] donde los estudiantes desarrollan proyectos utilizando ROS como herramienta clave para ello. Algunos de estos proyectos incluyen detección de carriles en la calzada, lectura de señales de tráfico o incluso desarrollo de un simulador de conducción como el de la Figura 27.

Fig. 27: Simulación de conducción utilizando ROS.
Fuente: G. Sung, estudiante del curso de Udacity de conducción autónoma. [49]

La versión de ROS utilizada es ROS Lunar. Para su instalación se han seguido los pasos explicados en la web de ROS[3].

Para la utilización de ROS es imprescindible familiarizarse con varios términos que van a aparecer de ahora en adelante:

- **Nodo**: Son procesos que realizan operaciones; contienen funciones y código de alto nivel y pueden conectarse a otros nodos mediante envío (publicación) o recibimiento (suscripción) de datos.
- ***Topic:*** Es un conjunto de mensajes que pueden ser publicados o al que se puede suscribir cualquier nodo de ROS configurado para tal. Cada topic en el sistema en funcionamiento ha de tener un nombre único.
- **Mensaje**: Es una variable de un tipo determinado con un cierto valor.
- ***Publisher:*** Uno de los dos métodos de comunicación de nodos en ROS es mediante la publicación de topics. Al publicar un topic, se registra su nombre en el sistema en ejecución, así como la información (mensajes) contenida en él.
- ***Subscriber:*** Las suscripciones permiten a un nodo recibir información publicada previamente por otro nodo en forma de topics. Para ello, es necesario saber el nombre del topic concreto al que se quiere suscribir.

Un esquema general de funcionamiento de los nodos de ROS sería el de la siguiente Figura 28:

[2] https://eu.udacity.com/course/self-driving-car-engineer-nanodegree--nd013
[3] http://wiki.ros.org/lunar/Installation/Ubuntu

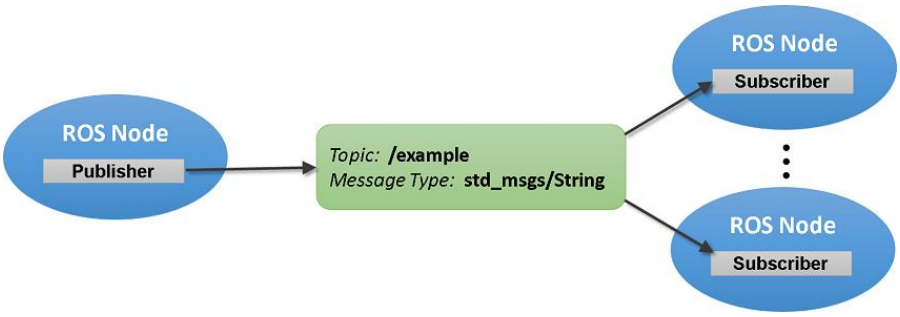

Fig. 28: Esquema general de funcionamiento de nodos en ROS.
Fuente: Matlab Documentation. [33]

Además, todos los nodos de los procesos en ejecución están supervisados por el ROS Master que, como su nombre indica, sirve de nodo maestro y permite la comunicación entre nodos registrándolos cuando se inicia su ejecución. Por ello, el primer paso a tomar cuando se utiliza ROS es ejecutar este nodo maestro, que en nuestro caso se hace introduciendo el comando *roscore* en la terminal de Ubuntu como aparece en la Figura 29.

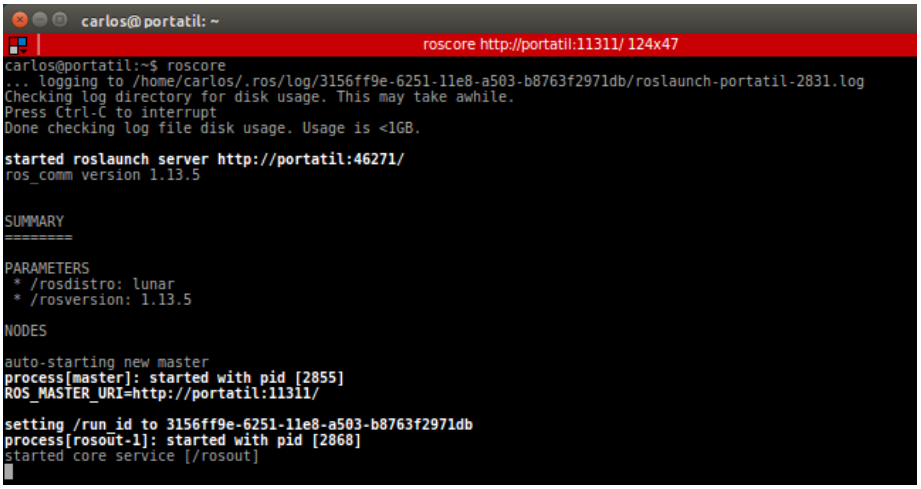

Fig. 29: Ejecución del nodo maestro por terminal a través del comando roscore.

Otro aspecto importante de ROS son los *workspaces*, o espacios de trabajo, que es una carpeta que contiene todos los proyectos con los que se va a trabajar en conjunto. Además, esta carpeta principal contiene una serie de subcarpetas y archivos concretos que gestionan las dependencias de los proyectos, nombres de los paquetes, ficheros para la generación de los ejecutables, etc. En la Figura 30 se puede ver un esquema básico de subcarpetas y ficheros de un workspace catkin (paquete que engloba toda la infraestructura y macros del sistema de compilación de ROS, y es instalado por defecto junto a éste).

Para la generación del workspace catkin se han seguido los pasos del tutorial correspondiente de la Wiki[4]. Además, siempre que se haga alguna modificación tanto a

[4] http://wiki.ros.org/catkin/Tutorials/create_a_workspace

la estructura de carpetas dentro del workspace como a cualquiera de los archivos contenidos en éstas es necesario compilar el workspace; esta acción se realizará mediante el comando *catkin make* en la terminal, estando ubicados en la carpeta base del workspace (en el caso del presente proyecto, dicha carpeta base tendrá de nombre *catkin_ws*).

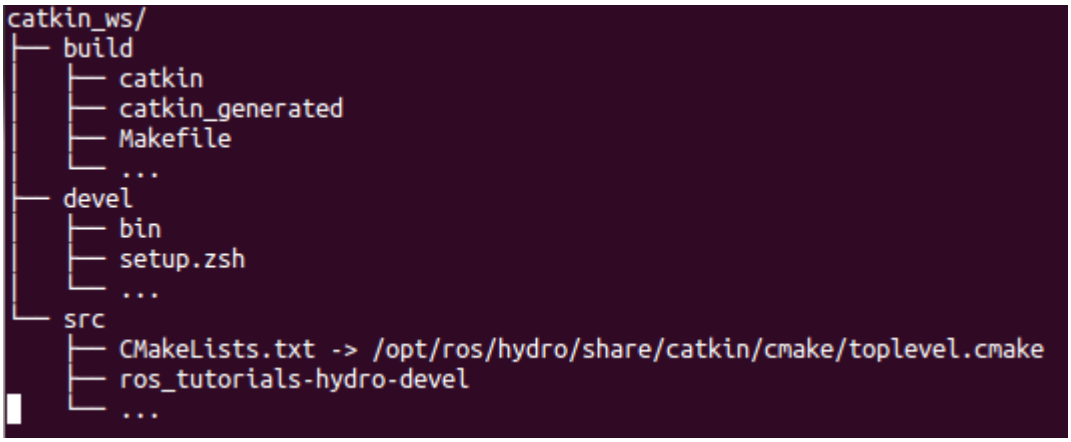

Fig. 30: Esquema básico de subcarpetas y ficheros de un workspace catkin.
Fuente: E. Fernández et al. [34]

Además, es necesario ajustarse al formato requerido por catkin a la hora de trabajar con paquetes. Dichos paquetes deberán estar ubicados en la subcarpeta *src* del workspace catkin, junto al archivo *CMakeLists.txt* del workspace; además, cada paquete deberá contener dos archivos clave para su correcto funcionamiento:

- ***CMakeLists.txt****:* Es un archivo de texto que proporciona información al sistema sobre el nombre del paquete, la versión catkin requerida, cómo generar el código y dónde y cómo instalarlo, otros paquetes necesarios para su instalación, etc. Es similar al archivo *CMakeLists.txt* ubicado en la carpeta base *src*, al que se suele referir como "Toplevel CMake file" y se genera al instalar el workspace catkin, pero cuyo alcance se reduce únicamente al paquete en el que está contenido.
- ***Package.xml****:* Es un archivo XML que especifica datos como el nombre del paquete, los números de versión, autores y dependencias a otros paquetes de catkin.

ROS incorpora además una herramienta de visualización de topics y nodos, llamada *rqt_graph*, que permite identificar los topics y nodos activos, así como las publicaciones y suscripciones. En la Figura 31 aparece un ejemplo de diagrama obtenido desde rqt_graph, donde un archivo de grabación .bag se está ejecutando (nodo con nombre "/play_...") y publicando el topic "/velodyne_points" al cual se suscribe el nodo "/velodyne_birdview", perteneciente al código del primer bloque de este proyecto. Esta herramienta ha resultado muy útil durante la realización del proyecto a la hora de comprobar las suscripciones y publicaciones de los nodos, así como para la gestión de los topics e información de los datos de grabación utilizados durante la segunda parte del proyecto, debido a la gran cantidad y variedad de información y topics utilizados en la misma.

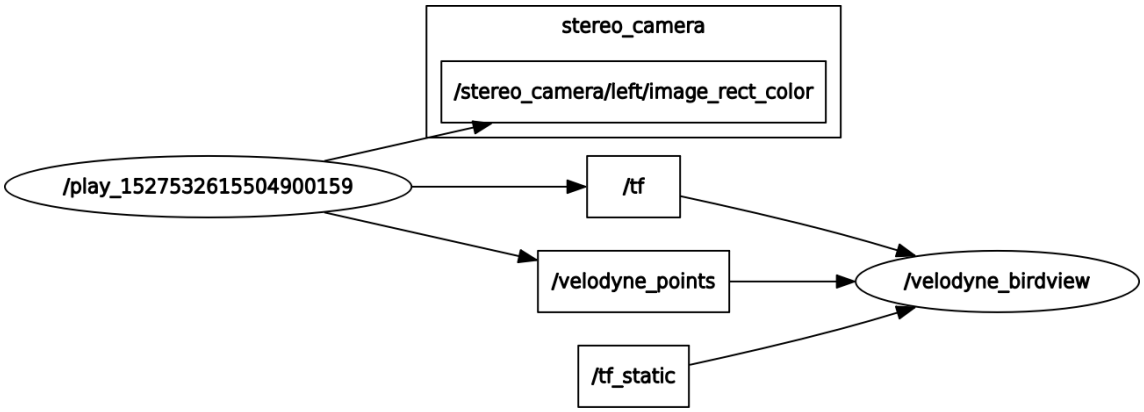

Fig. 31: Esquema de comunicaciones mostrado por rqt_graph.

Además, mediante el comando por terminal *rviz* se puede acceder a la herramienta con mismo nombre que nos permite, entre otras cosas, visualizar de forma sincronizada diferentes elementos de los archivos de grabación, como la nube de puntos y la imagen de la cámara frontal mostrados en la Figura 32.

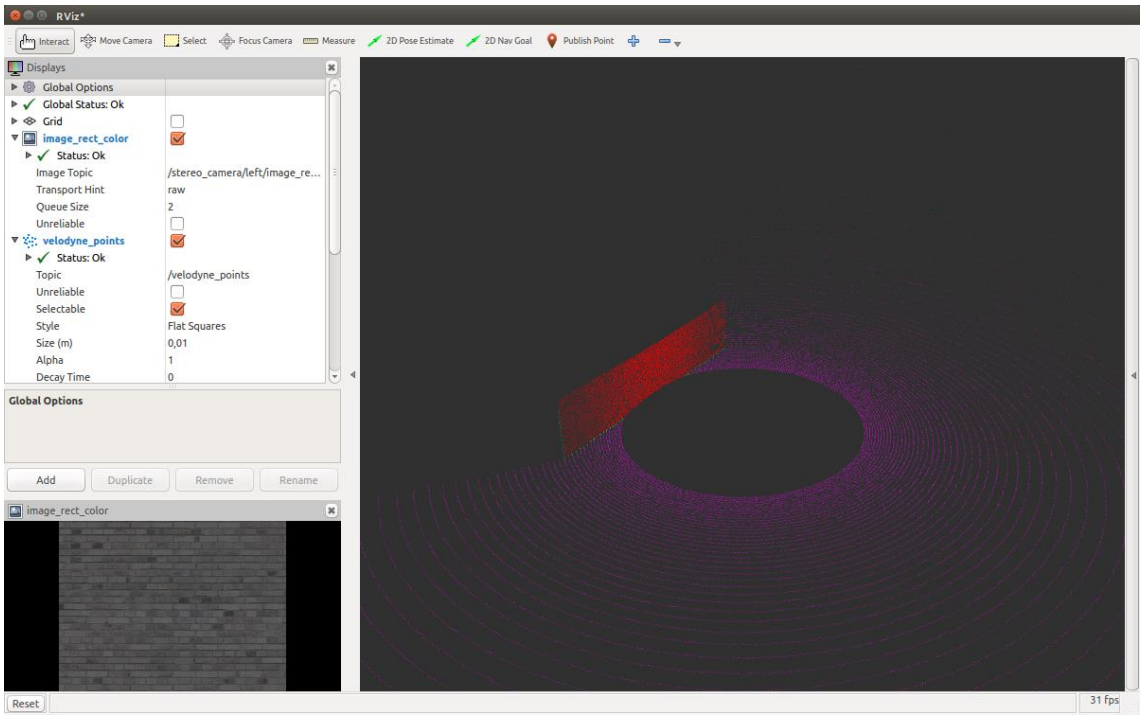

Fig. 32: Visualización sincronizada de imagen y nube de puntos a través de rviz.

3.2.3. OpenCV

OpenCV obtiene su nombre de *Open Source Computer Vision*, y es actualmente la librería más importante y relevante en visión por ordenador y también aprendizaje automático. Se distribuye con licencia BSD; es decir, de libre acceso de forma tanto académica como comercial.

Esta librería ofrece una gran cantidad de funciones y elementos utilizados en proyectos de visión por ordenador. En el caso específico de este proyecto, la utilización de esta

librería ha permitido usar elementos y funciones comunes en proyectos de visión por computador, como generación y refrescado de imágenes, definición de puntos o texto, o dibujado de bounding boxes con etiqueta, por ejemplo.

El lenguaje nativo de OpenCV es C++; mismo lenguaje nativo de otros paquetes, librerías y nodos que se han utilizado en el desarrollo de este proyecto. Por ello, salvo algún elemento puntual (explicado más adelante cuando sea conveniente), el lenguaje en el que se desarrolla este proyecto es el mismo, C++.

3.2.4. Qt Creator

Qt Creator es un entorno de desarrollo integrado (IDE) multiplataforma. Será el programa que utilizaremos para la gestión y desarrollo del código en C++ (para el resto de archivos de texto, se ha utilizado gedit, el editor de textos por defecto de Ubuntu). Para la instalación de Qt Creator 5 se han seguido los pasos de la Wiki[5].

Este IDE ha sido escogido debido que es de uso libre, compatible con C++, y permite una organización de ventanas comprensible e intuitiva para gestionar el código de cada fichero, los archivos del proyecto o proyectos en uso, etc. La distribución de ventanas utilizada puede verse en la Figura 33, donde a la izquierda aparecen (de arriba abajo) el explorador de ficheros del proyecto o proyectos en uso, ficheros de encabezado (headers) incluidos en el proyecto y listado de ficheros archivos recientemente (pertenezcan o no a los proyectos en uso). A la derecha de la imagen se muestra el código fuente del programa (arriba) y la terminal de mensajes (abajo).

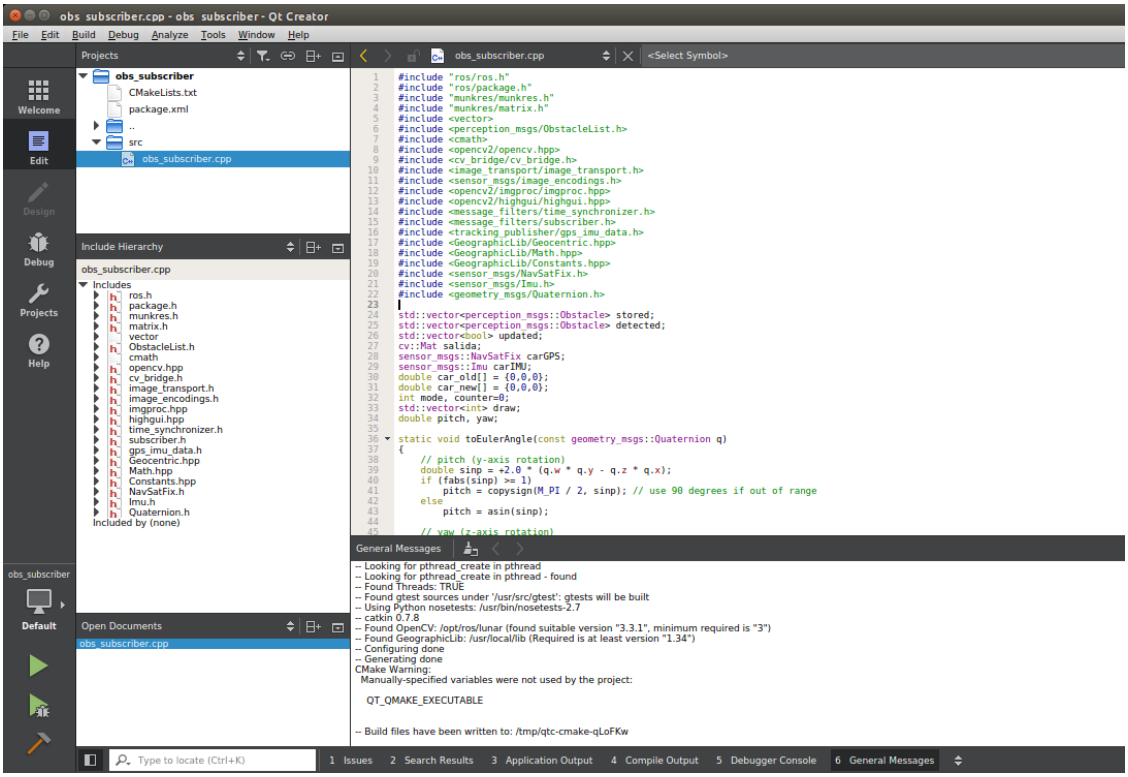

Fig. 33: Layout de Qt Creator.

[5] https://wiki.qt.io/Install_Qt_5_on_Ubuntu

3.2.5. Git

Git es un software libre de control de versiones que aporta gran utilidad al desarrollo de proyectos software de forma no lineal, debido a su capacidad de gestión de ramas, mezclado de diferentes versiones, gestión distribuida, copia local del historial del proyecto completo…

Así, Git ha permitido obtener código de otros repositorios relacionados con el recogido en esta memoria mediante la utilización de diversos comandos desde la terminal de Ubuntu, como los mostrados en la Figura 34, y sin los cuales el funcionamiento del mismo no habría sido posible.

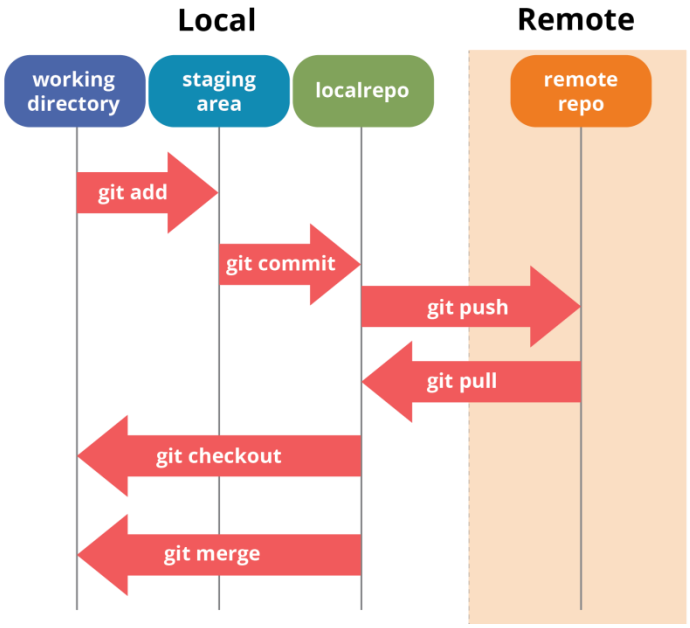

Fig. 34: Algunos comandos básicos de Git.
Fuente: Edureka. [35]

Además de los comandos de la figura, también es importante nombrar los comandos *fork* y *clone*. Con el comando fork se crea una copia del repositorio origen, de tal forma que se pueden hacer modificaciones al mismo sin ningún riesgo de alterar el código original, ya que ahora pertenece a otro repositorio distinto (a la copia que se acaba de crear). Por otro lado, el comando clone nos permite descargar un repositorio al ordenador o sistema local en uso.

Como el desarrollo del código de este proyecto requiere la utilización de otros proyectos del Laboratorio de Sistemas Inteligentes de la Universidad Carlos III de Madrid[6], se ha utilizado el comando fork para obtener una copia privada del mismo y poder trabajar con ese código sin riesgos de pérdida de información o modificaciones inesperadas en el repositorio en uso del LSI.

[6] Web del LSI: http://www.uc3m.es/islab

4. PROYECCIÓN DEL LIDAR A VISTA DE PÁJARO

La primera parte de este trabajo ha consistido en el desarrollo de una representación en forma de imágenes (también llamadas canales de ahora en adelante) de diversas características de los datos obtenidos, utilizando como datos de entrada la nube de puntos del sensor LiDAR. Dichas imágenes representan una vista de pájaro donde en el centro de la imagen se sitúa el sensor del vehículo, y cada pixel abarca una porción de terreno vista de forma vertical de tamaño 10x10 centímetros, situada alrededor del sensor según la posición del propio píxel. El propósito del desarrollo de estos canales es el de permitir un correcto funcionamiento del sistema de forma indistinta a la utilización de los modelos de sensor Velodyne VLP-16, HDL-32E y HDL-64E, así como de añadir funcionalidades adicionales que permitan una mejor detección y evaluación de los elementos del entorno.

El código no ha sido desarrollado de cero, sino que se ha utilizado uno de los paquetes disponibles en el Laboratorio de Sistemas Inteligentes de la universidad, obtenido a través de los comandos de Git *fork* y *clone*, de forma que se ha obtenido una copia privada del repositorio y se ha instalado en nuestro ordenador sin ningún riesgo de alteral el código original.

El paquete utilizado ya incorporaba el nodo en ROS creado y con funcionalidades aplicadas para el modelo Velodyne HDL-64E. Por ello, en esta primera sección se ha revisado el código y ampliado el funcionamiento para los distintos modelos planteados y, posteriormente, se han realizado varios cambios a los canales para mejorar su funcionamiento y permitir un mejor manejo y utilización de la información obtenida.

4.1. Funcionamiento inicial del paquete

El paquete *velodyne_points* detecta los datos LiDAR y los utiliza para generar datos en varias cuadrículas de 700x700 casillas a modo de vista de pájaro, similar a la Figura 35. Cada punto detectado por un láser se asigna a la celda en cuya región está contenido, de tal modo que tras contar todos los puntos y operar con esta información tras asignarlos a las celdas, el resultado serán varias matrices de números enteros con valores entre 0 y 255 (o, dicho de otra forma, una imagen en escala de grises) cuyas dimensiones serán iguales que la cantidad de celdas de la cuadrícula; es decir, de 700x700 píxeles. Los principales parámetros de la cuadrícula son:

- ***grid_dim***: La longitud total de la cuadrícula. En el caso de este proyecto, esta variable tendrá un valor de 70 (en metros).
- ***cell_size***: Variable obtenida a partir de la longitud total (*grid_dim*) dividida entre el número de celdas o casillas (en este caso, 700), por lo que se obtiene un valor de 0,1 (es decir, 10 centímetros por celda).
- x_1 e y_1: Ejes de coordenadas de los datos LiDAR obtenidos, que utilizan como origen el vehículo y tienen de dirección positiva la mostrada en la imagen.
- x_2 e y_2: Ejes reconvertidos y utilizados internamente en el paquete, para facilitar su comprensión y manejo a la hora de desarrollar los algoritmos necesarios para las funciones y operaciones internas.

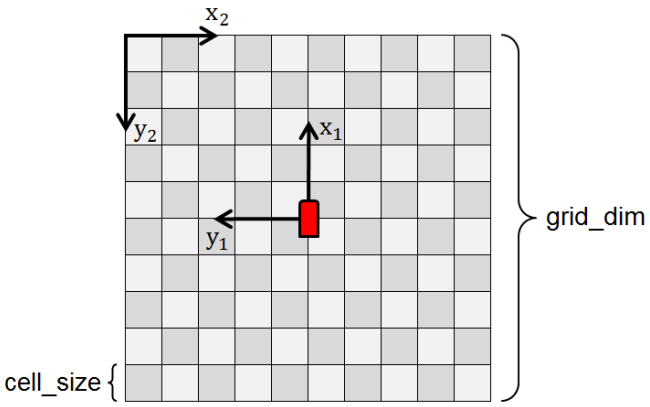

Fig. 35: Dibujo esquemático de la cuadrícula de la vista de pájaro y principales características.

La reconversión de los ejes nos ha permitido tener una numeración intuitiva para cada celda y que siga un orden similar al de las coordenadas de x_2 e y_2. La celda de la esquina superior izquierda es la (0,0), la de su derecha la (0,1), la de la derecha de ésta última la (0,2), y así sucesivamente. Siguiendo esta lógica, la última celda (es decir, la de la esquina inferior derecha) será la (699,699).

Inicialmente, este paquete generaba tres imágenes distintas, llamados "canales" de ahora en adelante:

- **Densidad**: Cantidad de puntos detectados en la celda en relación al número máximo de puntos totales detectables. Más adelante se explicará con más detalle el funcionamiento interno, ya que una de las principales modificaciones del paquete ha sido en este canal debido a un funcionamiento incorrecto.
- **Altura**: Altura máxima entre todos los puntos asignados a la celda. Limitada a valores de 0-3 metros. Otros de los principales cambios han sido en este canal.
- **Intensidad**: Intensidad del láser reflejado, lo cual se relaciona típicamente con el color del elemento detectado. Este canal no ha recibido modificaciones posteriores, a diferencia de los otros dos.

4.2. Modificación del archivo .launch

Como ya se ha dicho, el objetivo principal de esta parte del proyecto ha sido implementar código que permitiese incluir en el funcionamiento los modelos HDL-32E y VLP-16, además del HDL-64, modelo ya incluido en el paquete inicial.

El primer paso para garantizar la compatibilidad de versiones es entender el funcionamiento del archivo de ejecución (un fichero de texto con extensión .launch). El comando de terminal *roslaunch*, seguido del nombre del paquete y del archivo .launch, permite ejecutar dicho paquete en base a una cierta configuración, la cual está definida en este fichero de ejecución.

En este proyecto, el archivo *velodyne_birdview.launch* contiene, además de otros datos relacionados con la generación de la cuadrícula, información sobre los parámetros de los sensores; por defecto, estos parámetros eran los correspondientes del sensor HDL-64E. Por ello, se ha decidido que la mejor opción era introducir de nuevo estas mismas

líneas de código para el HDL-32E y el VLP-16, cambiando los valores de los parámetros por los correspondientes de cada sensor. Así, cuando se vaya a ejecutar el paquete, en el archivo .launch se deben dejar sin comentar las asignaciones del modelo utilizado, y comentar las pertenecientes a los otros dos.

Los parámetros de los sensores recogidos en el archivo .launch son:

- **Número de planos**: El número de planos generados por el sensor.
- **Resolución horizontal (*h_res*)**: ResoluciÁngulo horizontal en radianes (en el plano XY, en sentido antihorario) que separa dos láseres consecutivos pertenecientes al mismo plano.
- **Resolución vertical (*v_res*)**: Ángulo vertical en grados sexagesimales que separa dos planos consecutivos.
- **Ángulo inicial (*low_opening*)**: Ángulo (en grados sexagesimales) que forma con la horizontal el primer plano comenzando por arriba.
- **Intensidad normalizada**: Variable booleana que indica si los datos obtenidos están normalizados. Al obtener los datos directamente del sensor, esta variable debe valer siempre cero (*false*) independientemente del sensor utilizado.

En la Figura 36 se pueden ver dos dibujos sencillos que ayudan a entender cómo funcionan algunos parámetros. Además, en la Tabla 1 se muestran los valores de los parámetros para cada modelo de sensor.

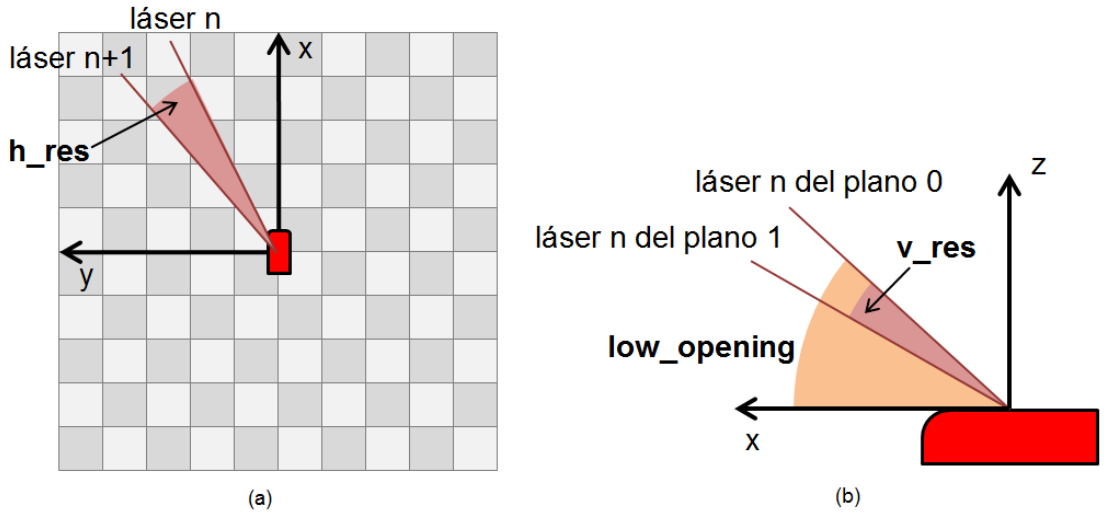

Fig. 36: Esquemas con los parámetros importantes de los sensores. En (a), vista de pájaro; en (b), vista desde el lateral del vehículo mirando hacia la izquierda.

	HDL-64E	HDL-32E	VLP-16
Planos	64	32	16
Resolución horizontal	0.00523598775	0.00287296996	0.00335103216
Resolución vertical	0.4	1.33	2
Ángulo inicial	24.9	10.67	15
Intensidad normalizada	false	false	false

Tabla 1: Tabla de valores de los parámetros de cada modelo de sensor.

Esta modificación del archivo .launch permite que los parámetros registrados por el nodo de ROS relacionados con el sensor activo sean los correctos. Sin embargo, han hecho falta más modificaciones en el código fuente del paquete para garantizar una correcta funcionalidad del mismo.

4.3. Modificación del código fuente

El código fuente del paquete realiza diversas operaciones, como conectarse al nodo maestro de ROS, suscribirse a los topics correspondientes, etc. Para que el paquete funcione correctamente, éste ha de ejecutarse junto a un archivo de tipo .bag; estos archivos contienen topics e información en ROS a modo de grabación. La ejecución de los archivos .bag se realiza mediante el comando por terminal *rosbag play*, mientras que el del paquete se hace con *roslaunch*, como ya se ha explicado.

Al ejecutarse a la vez, el paquete se suscribe al topic publicado por el archivo .bag (siempre y cuando los nombres coincidan, por lo que se ha asegurado de que tengan el mismo nombre para evitar que el paquete no encontrase la información). Para ello, mediante el comando *rostopic list* se pueden ver los nombres de todos los topics que están siendo publicados. Mediante el comando *rostopic info* seguido del nombre del topic se puede ver información adicional sobre el tipo de mensaje publicado, así como su publicador y si hay otros nodos suscritos al topic. Esto se muestra como ejemplo en la Figura 37, sin ninguna suscripción a los topics del archivo .bag en ejecución.

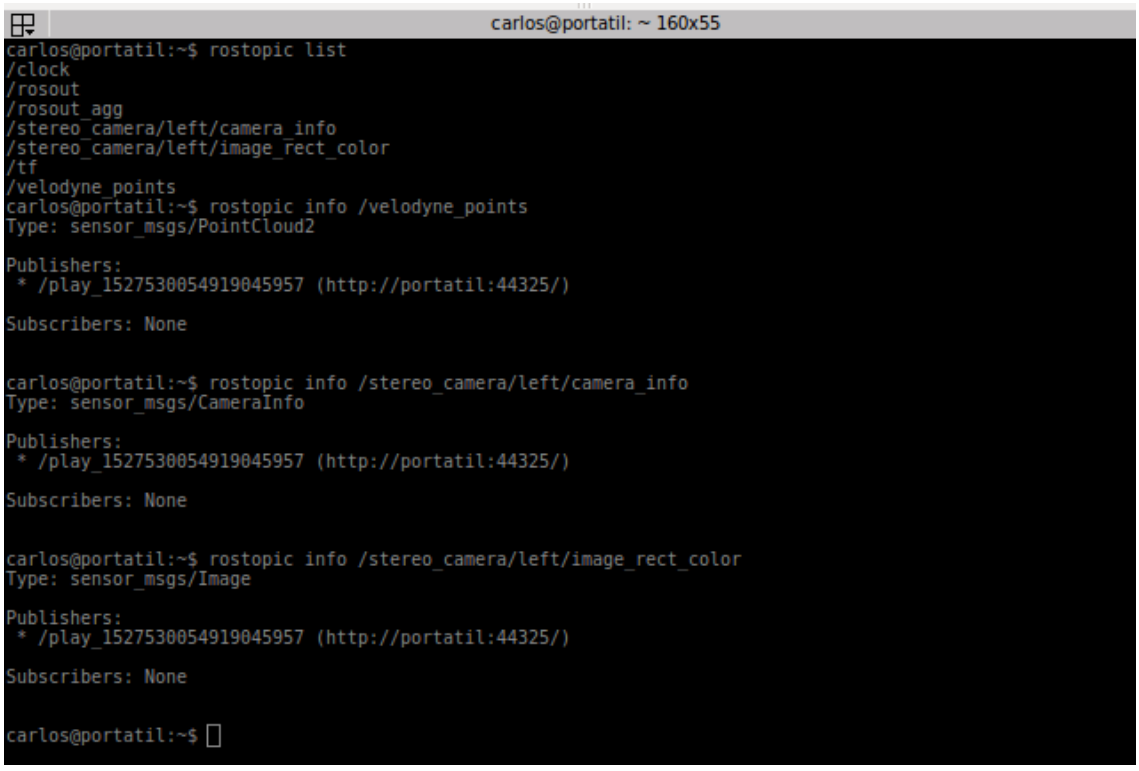

Fig. 37: Ejecución de los comandos rostopic list y rostopic info en 3 de los topics pulicados por el archivo .bag.

El topic *velodyne_points* contiene datos de tipo PointCloud2, que representan una nube de puntos y pertenecen a la librería *sensor_msgs*, la cual está incluida en el paquete

mediante la línea #include al comienzo del código, al igual que otras librerías requeridas como la librería OpenCV, la librería ROS, etc. Éste será el topic al que se suscribirá el paquete velodyne_birdview; con esta nube de puntos se obtendrán las imágenes finales para los canales de densidad, altura e intensidad, con modificaciones explicadas debidamente más adelante.

Una vez explicado el funcionamiento del paquete y las herramientas principales para la visualización y comprobación de su funcionamiento, se van a explicar los cambios realizados cambios en el canal de densidad, cuya funcionalidad se ha expandido (cambios que corresponden a la primera versión del paquete modificado), y en el de altura, que se ha dividido en tres canales, cada uno afectando a rangos de altura distintos (cambios que corresponden a la segunda versión del paquete); estos cambios se resumen en la Figura 38.

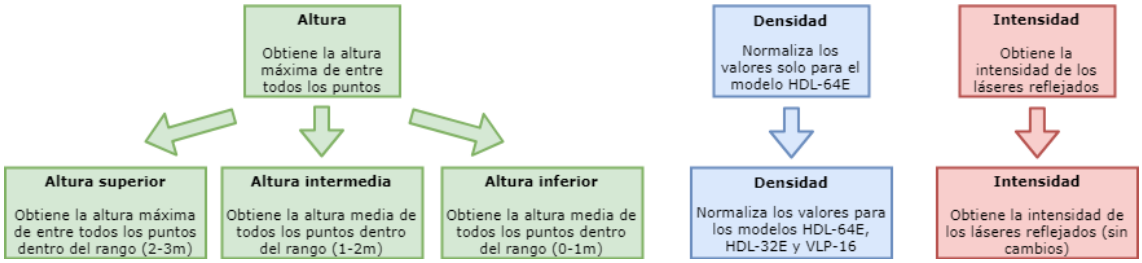

Fig. 38: Resumen de los cambios realizados en los canales.

4.3.1. Canal de densidad

En primer lugar, ha sido necesario hacer un reajuste al cálculo del canal de la densidad. Este canal muestra como salida la cantidad de puntos máxima detectables por el sensor en cada celda para así compararlo con el número de detecciones reales y obtener un dato relativo.

Para comprender mejor este concepto, se plantea el siguiente ejemplo: Un elemento situado muy cerca del sensor abarcará un porcentaje alto de los láseres ya que, al estar situado tan cerca, los láseres no tendrán distancia para dispersarse lo suficiente como para no chocar con este elemento. Sin embargo, si este mismo elemento se situase a mayor distancia del sensor, el número de puntos de corte será menor, simplemente debido a que la dispersión de los láseres hace que no sea cortado por tantos haces de luz (este efecto se demuestra en la Figura 39). Por tanto, sin ningún tipo de normalización o relativización de los puntos de corte detectados, el canal de densidad tendría una evidente tendencia a mostrar valores más altos según nos acerquemos al centro de la imagen, donde el sensor se sitúa.

Para realizar esta normalización de valores, se ha planteado un algoritmo directo que generase los láseres matemáticamente y calculase los puntos de corte celda a celda. El primer paso será plantear el problema, de tal forma que cada haz de luz se exprese como una recta. En la Figura 40 se enseñan las ecuaciones planteadas desde las que se han realizado los cálculos necesarios.

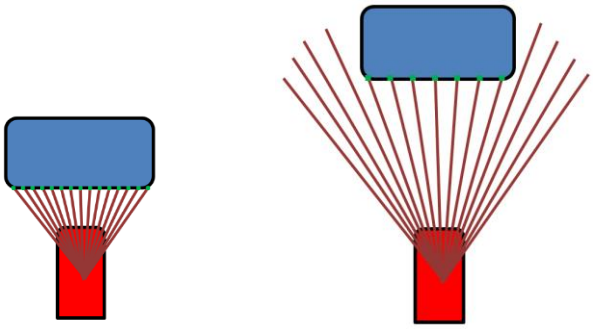

Fig. 39: Comparativa de puntos de corte con un mismo agente situado cerca y lejos del vehículo.

$$x = \lambda \cdot \cos\alpha \cdot \cos\beta$$
$$y = \lambda \cdot \sin\alpha \cdot \cos\beta$$
$$z = \lambda \cdot \sin\beta + h_0$$
$$\alpha = n_l \cdot h\_res$$
$$\beta = (low\_opening - n_p \cdot v\_res) \cdot \frac{\pi}{180}$$

Fig. 40 : Ecuaciones de la recta.

En estas ecuaciones, las variables α y β son los ángulos (ambos en radianes) que forma la recta con el plano XY (ángulo horizontal) y el ángulo vertical, respectivamente. h_0 es la altura del sensor en relación con el nivel del suelo. n_l y n_p son, respectivamente, el número de línea y de plano de cada haz de luz. λ representa la longitud de la recta y es una variable cuyo valor ha de ser obtenido tras plantear los puntos de corte.

A la vista de estas ecuaciones se ha decicido que los dos primeros bucles anidados son los que generan la ecuación para cada haz de luz (el primer bucle para cambiar de plano y el segundo bucle para cambiar de línea dentro del mismo plano). Habiendo generado la recta de cada haz de luz, se ha planteado el cálculo de los puntos de corte de la siguiente forma: Al pasar por cada casilla, en dicha casilla se tiene un bloque de base cuadrada y altura de 0 a 3 metros (a nivel del suelo, no del sensor), y el resto de la cuadrícula estará vacía.

De ahora en adelante, el resto de ecuaciones y explicaciones han sido basadas en el segundo eje de coordenadas planteado anteriormente; el ubicado en la esquina de la cuadrícula, y no el situado en el sensor del coche.

Este problema de puntos de corte en tres dimensiones se simplifica si, en vez de utilizar un bloque, utilizamos dos planos verticales que coincidan con los límites de cada casilla en su extremo más cercano al sensor (ver Figura 41 como ejemplo). Sabiendo que los planos son verticales y que una de sus coordenadas es constante (coordenada y en el caso del plano morado de la imagen, o coordenada x en el caso del plano verde), es fácil deducir el valor que ha de tener λ, si existe, para calcular el punto de corte con uno de los planos.

En concreto, el proceso que se ha seguido para determinar si existe punto de corte queda explicado con en el flujograma de la Figura 42.

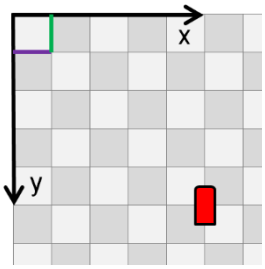

Fig. 41: Ejemplo de planos verticales generados para la casilla (0,0), en morado y verde.

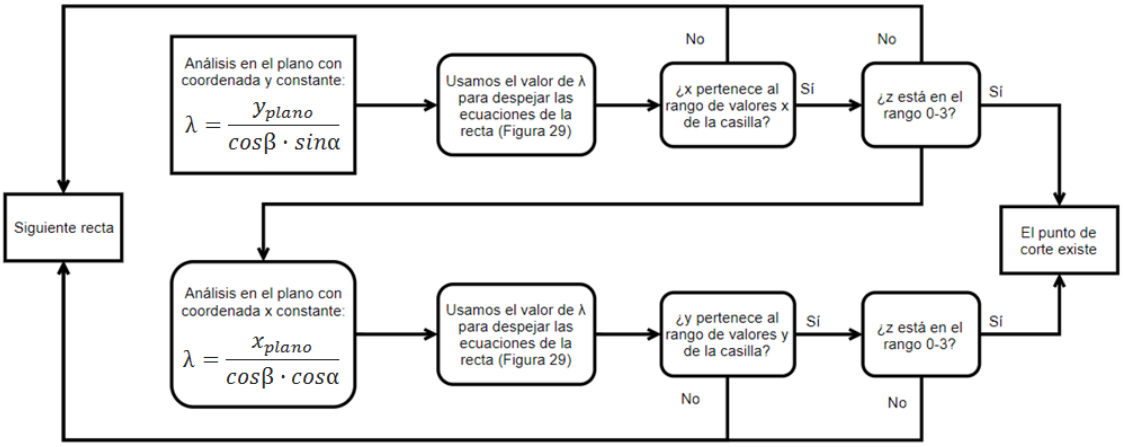

Fig. 42: Diagrama explicativo de la lógica del algoritmo de cálculo de los puntos de corte.

A fin de reducir el tiempo de ejecución, solo se ha realizado el algoritmo en la mitad del primer cuadrante de la cuadrícula, de tal forma que, debido a la simetría del problema, cualquier punto detectado en esa región se puede extrapolar a las otras 7 regiones simétricas de forma sencilla. Esto se muestra en la Figura 43, donde en rosa aparece la zona escaneada. En morado se muestra un punto de ejemplo detectado dentro de esta zona, a partir del cual se deducen los otros 7 puntos simétricos, en color verde. Esto también ha permitido simplificar la generación de planos, ya que en los otros tres cuadrantes se tendrían posiciones distintas relativas al origen (extremo superior izquierdo) de cada casilla.

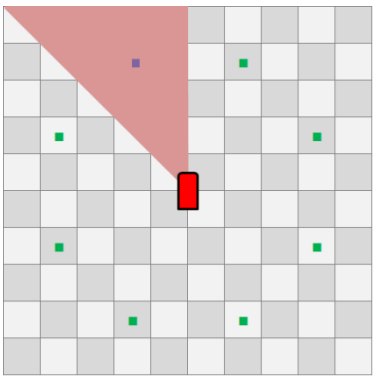

Fig. 43: Zona escaneada y puntos simétricos, en verde, al detectado dentro de la zona, en morado.

Este procedimiento se realiza al comienzo de la ejecución del paquete y, pese a las simplificaciones, el cálculo del mapa de valores es relativamente lento debido a la alta cantidad de iteraciones de los cuatro bucles anidados; dos de ellos tienen 350 iteraciones y los otros dos dependen del modelo del sensor. Por ejemplo, para el modelo VLP-16, estos dos últimos bucles serían de 16 y 1875 iteraciones, referidos al número de planos y de rectas por plano (el número de rectas se deduce a partir de la resolución horizontal, h_res), lo cual resultaría en casi 4 millones de procesos a ejecutar. Si bien esto requiere tan solo uno o dos minutos para procesarse en la mayoría de computadoras, una opción alternativa es realizar este procedimiento solo una vez y guardar los valores del array como un archivo externo de tipo .txt, al cual se accedería cada vez que se volviera a acceder al paquete, agilizando así el proceso de carga del mapa.

A modo ilustrativo, se ha normalizado uno de estos arrays entre los valores 0 y 255 para poder mostrarlo como imagen en escala de grises en la Figura 44. Si bien intuitivamente se podría pensar que el mapa debería tener una forma circular, esta forma semejante a un trébol tiene sentido, ya que los planos tienen una dispersión que provoca tanto secciones de corte cónicas con los planos como una gran cantidad de puntos perdidos debido a que exceden los límites de altura planteados (0-3 metros).

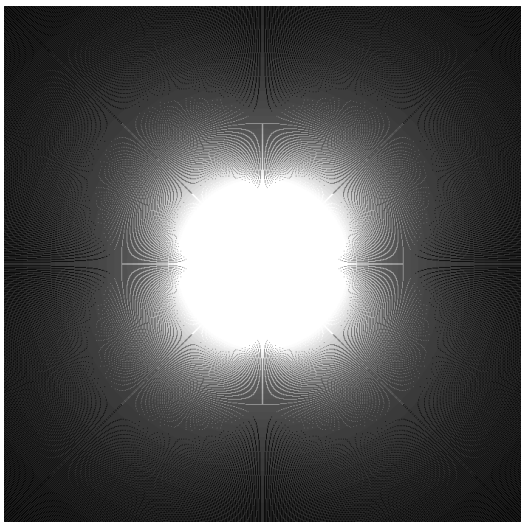

Fig. 44: Representación normalizada de los máximos puntos detectables en cada celda.

Con el array de puntos máximos detectables generado, el canal de densidad normalizado se calcula simplemente sumando el total de puntos detectados en esa celda y dividiendo este número entre el máximo detectable en esa misma celda. Por ejemplo, si en una celda se detectan 25 puntos y previamente se ha deducido que el máximo detectable eran 100, tenemos una densidad relativa del 25%. Tras la normalización a 0-255 para su exportación como imagen, tendríamos un valor para esa casilla de 63 (un cuarto de 255, comenzando por el valor 0 en vez de 1). Además, en las celdas en las que no se registre ningún punto, el valor asignado a dicha celda será 0. Este valor por defecto será el mismo que en el resto de canales.

4.3.2. Canales de altura

El que previamente era un solo canal de altura se ha dividido en tres canales, de tal forma que se separan tres rangos: Inferior, intermedio y superior.

Previo a este cambio, el canal de altura estaba siempre limitado entre 0 y 3 metros (teniendo en cuenta la altura del sensor; es decir, una altura de 0 metros es a nivel del suelo y no a nivel de sensor). Con este cambio, se ha dividido la altura en tres capas: 0-1 metros, 1-2 metros y 2-3 metros.

Además, la información que se obtiene de cada capa es diferente. Previamente, el canal de altura nos devolvía el valor máximo de altura registrado entre todos los puntos pertenecientes a cada casilla. Esto se ha mantenido para el canal de altura superior, pero para el inferior y el intermedio se ha modificado el código para obtener la media de las alturas de todos los puntos de cada casilla. Por ejemplo, si en la capa intermedia se detectasen dos puntos, uno de altura 1.2 y otro de altura 1.8, el valor de salida sería la media; es decir, 1.5.

Por último, hay que resaltar que, al igual que sucede en el resto de canales, la ausencia de puntos detectados en una casilla se corresponde con un valor de 0. A fin de evitar confundir un 0 obtenido por la no detección de puntos con un 0 obtenido a través de la normalización (es decir, un punto obtenido justo al comienzo del rango, que se normaliza al valor mínimo que es 0), es conveniente utilizar y analizar los tres canales de altura de forma conjunta y ponerlos en relación.

Con esto se concluye la primera parte del proyecto. En el siguiente capítulo se va a explicar la segunda parte, que corresponde al dibujado de bounding boxes y seguimiento de agentes utilizando información en 3D, incluyendo datos LiDAR, coordenadas GPS, IMU, etc.

5. IDENTIFICACIÓN Y SEGUIMIENTO DE AGENTES

El propósito de esta segunda parte del proyecto ha sido crear desde cero un nodo en ROS que se suscriba a los diferentes topics publicados por los archivos de grabación de datos de los sensores del vehículo (archivos .bag). Una vez suscrito, utiliza estos datos y los transforma en información utilizada en el seguimiento de agentes, tal como se explica de forma simplificada en la Figura 45.

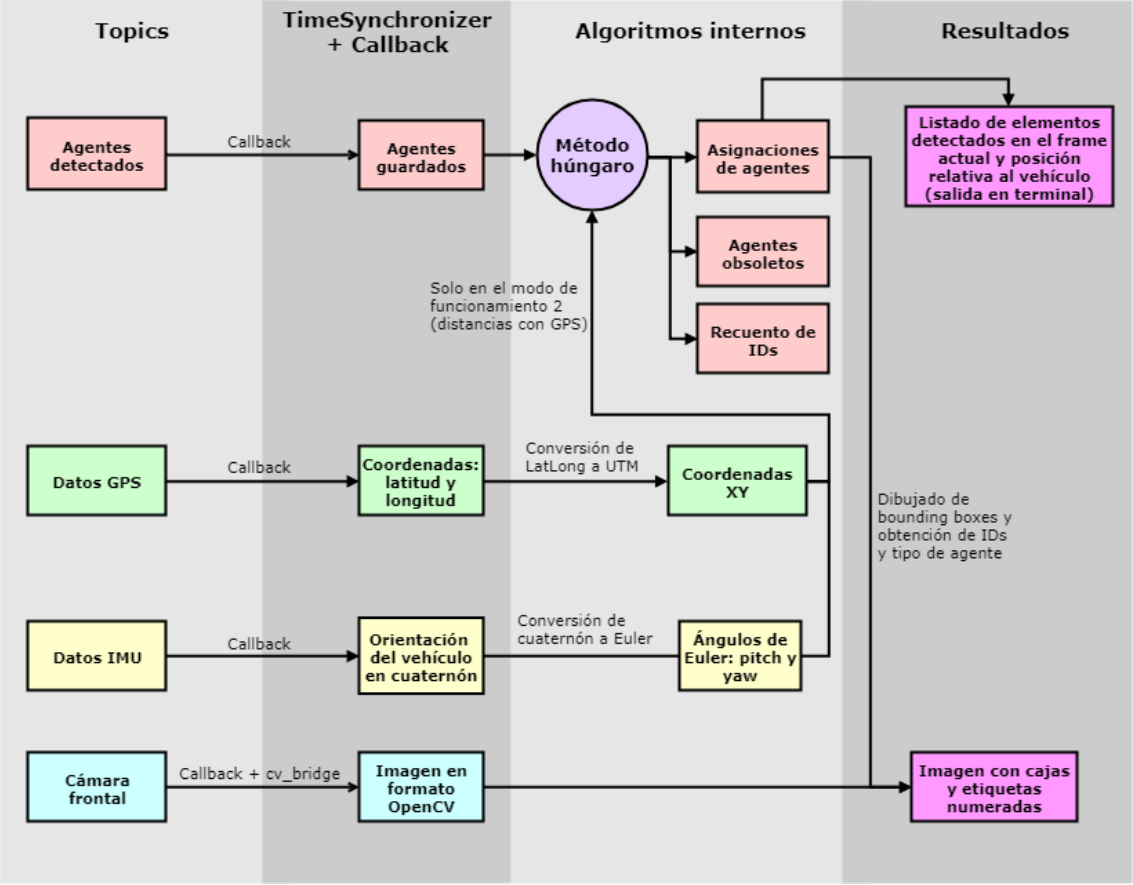

Fig. 45: Esquema de funcionamiento general del código del paquete.

5.1. Desarrollo del nodo en ROS

Si bien en la primera parte del proyecto se ha utilizado un paquete ya existente, el desarrollo de código de esta parte ha sido realizado desde cero. El primer paso a la hora de crear un paquete es la configuración de los archivos *CMakeLists.txt* y *Package.xml*. Al estar nuestro paquete vacío, estos archivos se han creado desde cero, utilizando como referencia otros paquetes del LSI de la Universidad para obtener información sobre cómo estructurarlos para su correcta instalación y funcionamiento en ROS.

5.1.1. CMakeLists.txt y Package.xml

En este archivo se define información clave relacionada con la instalación del paquete. El paquete creado ha sido llamado *obs_subscriber*, mismo nombre que el ejecutable del mismo, y su respectivo archivo fuente en C++ *obs_subscriber.cpp*. El paquete necesita de otros paquetes que se buscan en el workspace catkin a través del comando

find_package: cv_bridge, image_transport, message_filters, roscpp, sensor_msgs, std_msg, perception_msgs y GeographicLib. También se ha utilizado el paquete munkres-cpp, instalado de forma manual siguiendo los pasos del link de GitHub desde el que se ha obtenido dicho paquete[7].

Además, mediante el comando *target_link_libraries* se conectan varias librerías al paquete, concretamente las librerías de catkin, de OpenCV y del paquete GeographicLib.

En cuanto al archivo Package.xml, contiene información similar. Se utilizan los comandos *build_depend* y *run_depend* para, como su nombre indica, condicionar la generación y ejecución del paquete a otros paquetes (los mismos mencionados más arriba). Este archivo también puede contener información adicional, como el encargado o encargados de mantener y actualizar el código, enlaces externos relacionados con el mismo, la versión y descripción del paquete, etc.

5.1.2. Suscripciones y llamada al callback

En cualquier archivo C++ es imperativa la inclusión del método main. Este método se ejecuta por defecto al iniciar el programa; por lo tanto, es aquí donde se han introducido las líneas de código para iniciar el funcionamiento del nodo.

Para crear un nodo en ROS, se ha llamado a la función ros::init, la cual inicializa el nodo con el nombre especificado. Después, mediante la creación de una variable de tipo ros::NodeHandle, la ejecución del nodo comienza.

El siguiente paso es configurar las suscripciones a los topics correspondientes. Para obtener tanto los nombres de los topic como los tipos de variable recibidos de éstos, es preciso ejecutar uno de los archivos .bag que se vayan a utilizar y, mediante los comandos *rostopic list* y *rostopic info*, se han obtenido tanto los nombres como el tipo de dato publicado en los topics. Al estar los datos estandarizados, una vez configurados los nombres y tipo de los topics a los que suscribirse, este nodo debería funcionar con múltiples archivos de grabación de varias fuentes distintas, siempre y cuando se respete dicha estandarización.

Para gestionar las suscripciones, es necesario llamar a la función message_filters::Subscriber, una vez por cada topic suscrito. Éstos topics han sido:

- **"/obstacle_list"**: Es el topic de tipo perception_msgs::ObstacleList, que contiene un vector de elementos de tipo Obstacle, los cuales contienen a su vez información para cada agente detectado por el sensor.
- **"/kitti/image_left_rect_color"**: Este topic, de tipo sensor_msgs::Image, contiene una imagen, como su nombre indica. Estas imágenes han sido obtenidas con la cámara frontal del vehículo.
- **"/kitti/GPS"**: Información GPS del vehículo, de tipo sensor_msgs::NavSatFix.
- **"/kitti/IMU"**: Información adicional del vehículo, como su orientación, de tipo sensor_msgs::Imu.

[7] https://github.com/saebyn/munkres-cpp

Además, al estar suscrito a cuatro topics que deben funcionar de forma simultánea, se ha utilizado la función message_filters::TimeSynchronizer para asegurarse de que toda la información de los topics recibida pertenece al mismo frame de tiempo.

Una vez suscritos a los topics, el siguiente paso es llamar al callback. El callback es un método ajeno al main que obtiene los elementos que contiene cada topic como parámetros de entrada; de este modo, se ha podido trabajar con ellos.

Además, en esta función main también se ha añadido un pequeño bloque de código para que el usuario elija el modo de funcionamiento deseado mediante una entrada de teclado (se explicará más adelante), así como la función ros::spin() que permite el refresco de información obtenida a través del archivo de grabación.

5.1.3. Conversión de datos

Para facilitar la comprensión del código y evitar estar trabajando continuamente con punteros, se han creado variables globales que permiten guardar los elementos necesarios.

Todos los elementos detectados (de tipo Obstacle) se guardan en dos vectores, tanto los del time frame actual como del anterior. Estos elementos se actualizan continuamente, de tal forma que nada más comenzar el nodo no habrá elementos "antiguos" guardados y, además, según se dejen de detectar se eliminarán también. Los agentes recientes guardados pasarán al vector de agentes antiguos al comienzo del siguiente frame. De este modo, solo se trabaja con los datos que se detectan, evitando mantener guardados agentes obsoletos que han dejado de detectarse. Además, también se mantiene de forma interna tanto un recuento del número de agentes totales detectados (de tal forma que cada agente tiene una ID única e irrepetible, y dicho número lógicamente irá incrementando a medida que el tiempo avanza y se detectan nuevos agentes) como una pequeña lógica interna que nos indica si un agente ha dejado de ser detectado en la imagen.

Además, se obtienen datos adicionales del vehículo, de los topics GPS e IMU. De estos datos se guardan aquellos que se utilizan en el código, es decir, las coordenadas de latitud y longitud, y la orientación del vehículo (expresada en cuaternión). Estos datos se modificarán posteriormente a coordenadas XY, que permiten obtener una posición global tanto del vehículo como de los elementos.

5.1.4. Asignación y seguimiento de agentes

El siguiente paso es realizar la asignación de IDs en la primera iteración de la ejecución del programa. Desde un punto de vista lógico, el programa realiza una asignación directa, sin ningún proceso de seguimiento o relacionado de agentes, cuando el vector de agentes actuales está vacío; esto significaría que, o bien el programa acaba de comenzar (luego todavía no se han podido guardar elementos), o no se registra ningún agente cercano al coche que se guarde en el vector. En ambos casos, la situación es la misma: No se puede realizar ningún seguimiento de agentes si no tenemos agentes que comparar.

Cabe destacar que los agentes detectados vienen con su ID como parámetro detectado, pero ésta no se ha tenido en cuenta. En vez de eso, simplemente se asignan IDs comenzando con 0 para el primer elemento y de forma ascendente para los siguientes, según se vayan leyendo. El problema de la asignación aparece en las siguientes iteraciones; cuando en dos time frames consecutivos se han detectado elementos. En esta situación, desde el callback se llama al método de aplicación del método húngaro.

El primer paso dentro del método húngaro es la generación de la matriz de costes. En este problema, se ha definido "coste" como "distancia"; sin embargo, se han planteado tres modos de funcionamiento distintos en función de qué distancias se comparan:

- En el **modo 0**, se comparan las distancias en el plano XY relativas al vehículo. Es decir, se utiliza el vehículo como referencia y se comparan las posiciones en el momento anterior respecto a esta referencia con las posiciones en el momento actual.
- En el **modo 1**, se comparan las distancias de los centros de las bounding boxes entre los dos tiempos. Este modo, pues, utiliza los datos de la imagen de la cámara frontal.
- En el **modo 2**, se comparan las distancias en el plano XY también, pero en este caso no se hace de forma relativa al vehículo, sino de forma global. Para ello, ha sido necesario ubicar el vehículo con una posición global, en coordenadas UTM que, lógicamente, van cambiando en cada instante. Este modo es, a priori, el más fiable, ya que estos cambios en la posición global no se tienen en cuenta en ninguno de los otros modos; si el coche se mantuviese estático y el resto de elementos se moviesen alrededor de él, estos dos primeros modos tendrían exactamente el mismo funcionamiento.

Tras generar la matriz de distancias, donde las filas representan los elementos en el time frame anterior y las columnas los elementos detectados en el instante actual, se aplica el algoritmo húngaro, el cual modifica los valores de esta matriz y hasta obtener ceros en las celdas de coste mínimo.

El siguiente paso es buscar los ceros dentro de la matriz. Con una simple búsqueda elemento a elemento se han podido detectar las celdas con valor cero, que relacionan los agentes antiguos con los recientes. Nuevamente, la ID de los elementos recientes se descarta y se sustituye por la ID del agente antiguo al que se le relaciona, y finalmente se actualizan los datos del agente correspondiente para su posterior utilización en el siguiente time frame. En la Figura 46 se explica gráficamente la asignación de agentes donde los cuadrados naranjas representarían las asignaciones entre agentes guardados (filas) y agentes detectados en el time frame actual (columnas). Esta asignación se explica con tres casos distintos: mismo número de detecciones que agentes guardados (es decir, los agentes de la imagen se mantienen), más agentes detectados que guardados (es decir, aparece algún agente nuevo en la imagen en ese time frame concreto) y más agentes guardados que detectados (es decir, desaparece de la imagen alguno de los agentes detectados hasta el momento). Cabe destacar que los números de fila y columna son las posiciones en la matriz, y no la ID de los elementos.

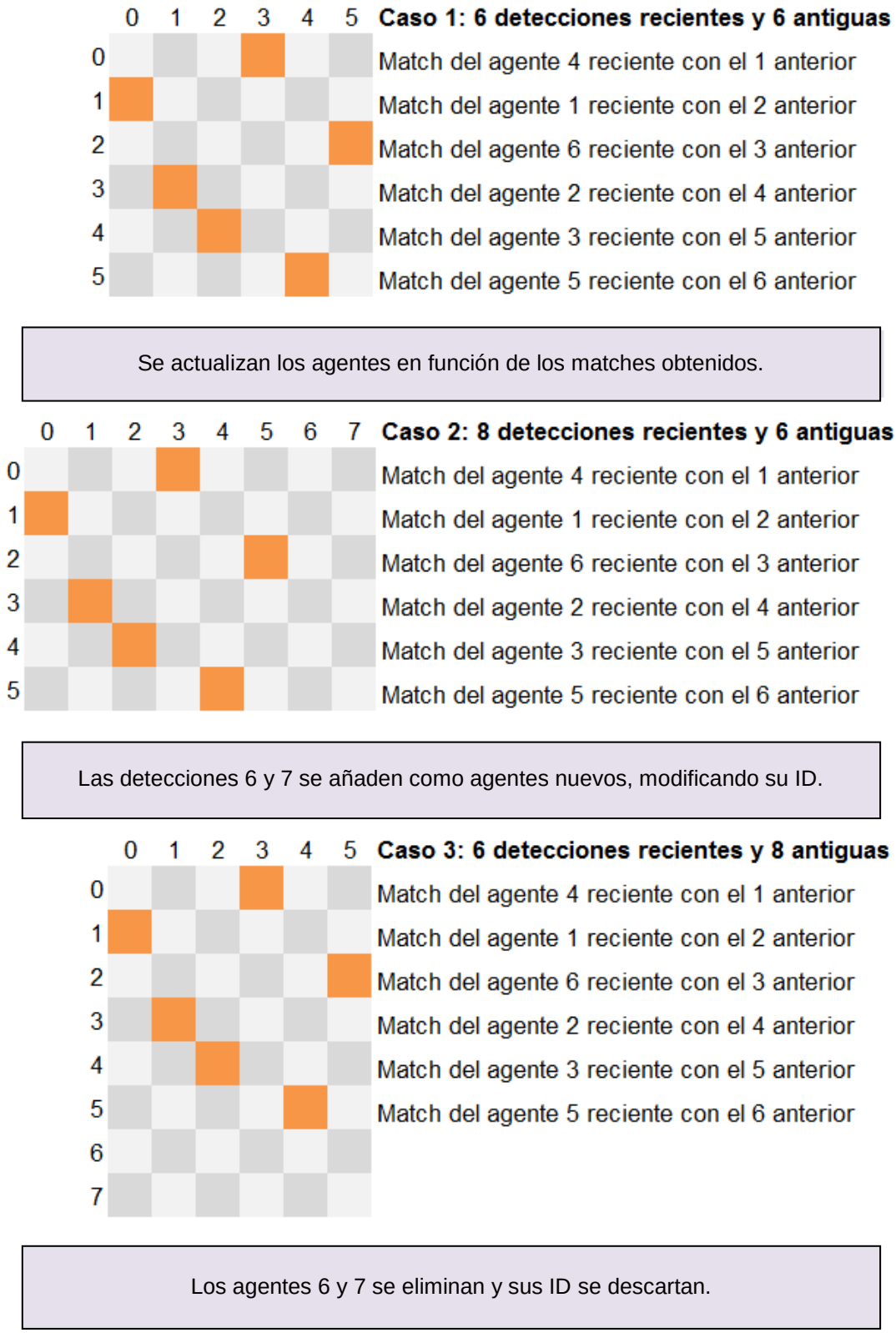

Fig. 46: Ejemplo de funcionamiento del algoritmo.

5.2. Explicación de los métodos auxiliares

En este apartado se van a explicar los métodos auxiliares que realizan cálculos intermedios necesarios para el correcto funcionamiento del nodo de ROS.

5.2.1. Conversión de cuaternión a ángulos de Euler

En el método *toEulerAngle* se parte de un parámetro de entrada de tipo Quaternion, obtenido del topic "/Kitti/Imu", para obtener los llamados ángulos de Euler o de navegación, expresados en la Figura 47. En este caso, el sistema de coordenadas estaría alineado con el vehículo, de tal forma que el coche quedaría alineado con el eje x, mirando hacia donde apunta la flecha roja de dicho eje.

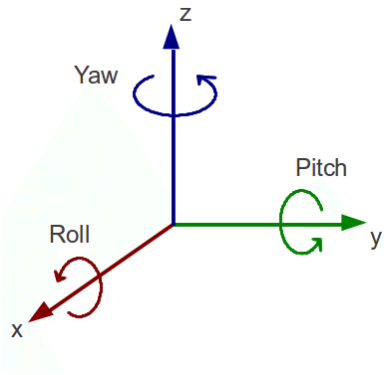

Fig. 47: Representación de los ángulos de Euler en el eje XYZ.
Fuente: I. Dotlic et al. [37].

El elemento de tipo Quaternion corresponde a un cuaternión, que es un sistema de notación matemático para rotaciones de ángulos en tres dimensiones. Al ser una matriz de tamaño 4x4, esta notación resulta conveniente en cierto tipo de problemas al evitar el problema del bloqueo del cardán, que ocurre cuando dos de los tres ejes se colocan en paralelo y bloquean así uno de los tres grados de libertad del sistema.

Sin embargo, este suceso no está presente en nuestro problema, por lo que ha resultado más conveniente e intuitivo realizar una conversión a los ángulos de Euler según la ecuación matricial de la Figura 48 (donde se expresan, de arriba abajo, los ángulos roll, pitch y yaw) para un mejor control y gestión de los ángulos de movimiento del vehículo.

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \operatorname{asin}(2(q_0 q_2 - q_3 q_1)) \\ \operatorname{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

Fig. 48: Ecuaciones de conversión de cuaternión a ángulos de Euler.
Fuente: J. L. Blanco [38].

Además, se ha descartado la utilización del *roll* en el código, debido a su escasa utilidad en casos reales. En algunos casos de proyectos de percepción en vehículos es también

común descartar el ángulo de *pitch* por el mismo motivo; no obstante, se ha preferido mantenerlo durante el desarrollo del proyecto y la obtención de los resultados.

Una vez obtenidos los ángulos de *yaw* y *pitch*, se ha realizado una conversión entre las coordenadas relativas al vehículo de los agentes detectados y las coordenadas UTM absolutas que se han utilizado.

5.2.2. Conversión de coordenadas GPS a UTM

En el método *toUTM* se parte del elemento NavSatFix, obtenido del topic "/Kitti/GPS". Estas coordenadas contienen los elementos de latitud, longitud y altura, que se han convertido a coordenadas UTM (*Universal Transverse Mercator*); un sistema de coordenadas basado en una proyección cartográfica secante a un meridiano. Esto significa que las coordenadas UTM tienen, además de sus coordenadas (x,y,z), un código de zona, como se muestra en la Figura 49. Sin embargo, para este proyecto no ha sido necesario dicho código de zona debido al funcionamiento de la librería GeographicLib, que incluye la función necesaria para realizar dicha conversión de forma directa.

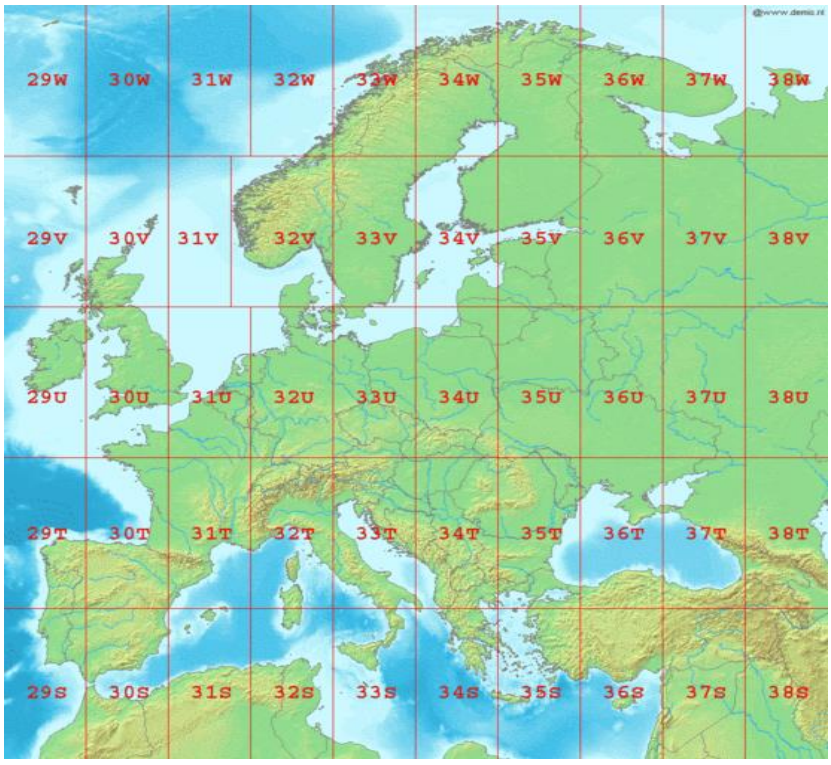

Fig. 49: Códigos de zona en Europa.

Matemáticamente, esta conversión de coordenadas de latitud y longitud a UTM tiene una gran complejidad, no solo de planteamiento del problema sino también de ejecución en código (y su consiguiente depuración); por ello, ha sido conveniente buscar código de uso abierto que tuviese dicho proceso ya implementado.

Con la inclusión de la librería GeographicLib, esta operación se realiza en un único método, tras previamente haber declarado una variable intermedia con el sistema de

coordenadas geográficas mundial WGS84, que define la Tierra como un elipsoide de revolución con radio 6.378,137 km y un achatamiento inverso de 298,257223563.

Este método guarda las coordenadas (x,y,z) en variables aparte, que se gestionan de forma interna en el código del paquete. La disposición de los ejes de referencia globales es la siguiente: el eje x apuntaría hacia la derecha, el eje y hacia arriba, y el eje z apuntaría hacia fuera, dirigido al lector u observador de la imagen.

5.2.3. Cálculo de la matriz de distancias

Como ya se ha explicado, este nodo tiene tres modos de funcionamiento, cuyas diferencias residen en cómo se ejecuta éste método, *obs_dist*, el cual devuelve el valor de distancia utilizando la clásica fórmula de distancias en el plano, mostrada en la Figura 50. La elección del modo de funcionamiento del nodo modifica qué entiende este método como y_1, y_2, x_1 y x_2.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Fig. 50: Ecuación para el cálculo de distancias a partir de las coordenadas en 2D.

Como ya se ha explicado, los modos de funcionamiento 0 y 1 utilizan las distancias entre las posiciones relativas al vehículo y entre los centros de las bounding boxes, respectivamente. En estos casos, tan solo se ha referenciado a dichos elementos a la hora de calcular la distancia.

En el modo 2, con posiciones en el plano utilizando las coordenadas UTM, ha sido necesario realizar varias operaciones intermedias. Lógicamente, el primer paso ha sido llamar a las funciones *toEulerAngle* y *toUTM* para obtener tanto la posición "global" del vehículo como para obtener su orientación en ángulos de Euler. Los ángulos se han utilizado para poder ajustar el sistema de coordenadas del vehículo a un sistema de referencia global; una vez convertidas las posiciones relativas al sistema de coordenadas externo, dichas coordenadas se suman con las coordenadas del vehículo en el instante actual o en el instante anterior según corresponda.

Debido a que se ha descartado la aplicación del ángulo *roll*, esta conversión entre orientación y posición global se simplifica muchísimo, matemáticamente hablando. Típicamente, este tipo de transformaciones requieren operaciones matriciales en 3 dimensiones[8]. Al eliminar una de las rotaciones, esta operación se simplifica y se puede resolver con trigonometría básica, teniendo también en cuenta que los ejes de coordenadas del vehículo son distintos a los del plano UTM. La visualización de este problema puede ser algo compleja, pero se simplifica bastante si tenemos en cuenta que los ejes x e y globales son los mismos que los locales del vehículo pero cambiados entre

[8] Este problema de rotaciones matriciales requiere una carga teórica importante para poder ponerse sobre contexto. Debido a que la resolución final del problema no implica este tipo de planteamientos, se ha preferido omitirlo. Más información sobre matrices de rotación en 3 dimensiones se puede leer aquí: http://planning.cs.uiuc.edu/node102.html

sí; por tanto, una vez obtenidas las coordenadas globales sin cambiar los ejes de referencia, tan solo se ha tenido que intercambiar las coordenadas x con las coordenadas y globales, y de igual forma con las coordenadas y con las x globales.

5.2.4. Dibujado de las bounding boxes con etiqueta

El dibujado de las cajas es el último proceso a realizar por el nodo (previo al vaciado de los vectores correspondientes al final de cada iteración), y obtiene la mayoría de la información de los elementos de tipo Obstacle obtenidos a través del topic "/obstacle_list".

Los parámetros de la imagen obtenidos para dibujar cada bounding box se obtienen directamente del topic (posición de inicio de la caja y posición final, la cual se deduce usando la posición de inicio y las dimensiones de la misma). Además, el color de cada caja cambia en función del tipo de agente detectado, de forma que la caja se vuelve roja si el elemento es un vehículo ("Car"), amarillo si es un ciclista ("Cyclist") y verde si es un peatón ("pedestrian"). Para cualquier otro tipo de agente, la caja dibujada tiene color negro. Esto permite una diferenciación gráfica sencilla entre elementos de distinto tipo.

En la etiqueta, el texto consiste simplemente en: "ID xx", donde *xx* representa el número de ID del elemento utilizando dos cifras. Una vez generado el elemento de texto, para el dibujado del rectángulo que abarca la misma, se ha creado otro rectángulo relleno que comienza en el origen de la bounding box, con el mismo color, y se ha usado la función cv::getTextSize para obtener el tamaño del texto de la etiqueta y que así el rectángulo tenga unas dimensiones apropiadas, que abarquen el texto pero no sobrepasen demasiado para abarcar parte innecesaria de la imagen.

Este proceso se realiza una vez para cada agente guardado, y continúa dibujando las cajas en el orden en el que se leen en el bucle. Así, se ha podido realizar un seguimiento de los agentes al mismo tiempo que se muestran por terminal los ID y coordenadas de posición relativas al vehículo de todos los agentes presentes en la imagen, como se puede ver en la Figura 51. Hay que destacar que la posición relativa se expresa en las coordenadas locales del vehículo, y no en las coordenadas UTM globales utilizadas internamente por el nodo.

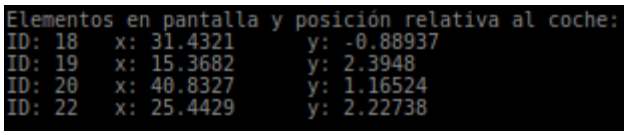

Fig. 51: Ejemplo de salida por terminal de los agentes detectados.

6. RESULTADOS OBTENIDOS

En esta sección se exponen los resultados intermedios y finales más representativos obtenidos durante cada una de las partes del proyecto, así como, en lo referido a la primera parte, varias comparaciones entre resultados iniciales del funcionamiento del paquete y resultados finales, válidos para su aplicación entre los tres modelos de sensores y con las nuevas funcionalidades.

6.1. Birdview y compatibilidad de sensores

6.1.1. Canal de densidad

Uno de los aspectos más importantes para la compatibilidad de los sensores ha sido el correcto funcionamiento del canal de densidad. Para poder evaluar las diferencias, se han utilizado varias secuencias grabadas en simulación, situando diferentes modelos LiDAR en la misma posición dentro del mismo entorno para poder comparar de forma justa y efectiva el efecto de la normalización en el canal de la densidad de la birdview.

Además, a las imágenes en escala de grises se les ha aplicado un filtro colormap, que convierte los valores 0-255 de la escala de negro a blanco a una escala distinta de colores. En este caso, se ha utilizado el colormap HSV, debido a que sus cambios de color más pronunciados han permitido una mejor visualización de las diferencias en las imágenes.

Fig. 52: Espectro del colormap HSV utilizado.

En la Figura 53 se muestran los mapas de puntos iniciales, previo a la realización del ajuste en la normalización del canal de densidad. Como se puede ver en la imagen, las mayores diferencias están en el modelo de 64 planos (modelo para el cual el código estaba programado inicialmente). Si bien el de 16 y 32 tienen cierta diferencia, los valores obtenidos son bastante cercanos; sin embargo, todavía no se tiene información para juzgar qué mapa es el correcto.

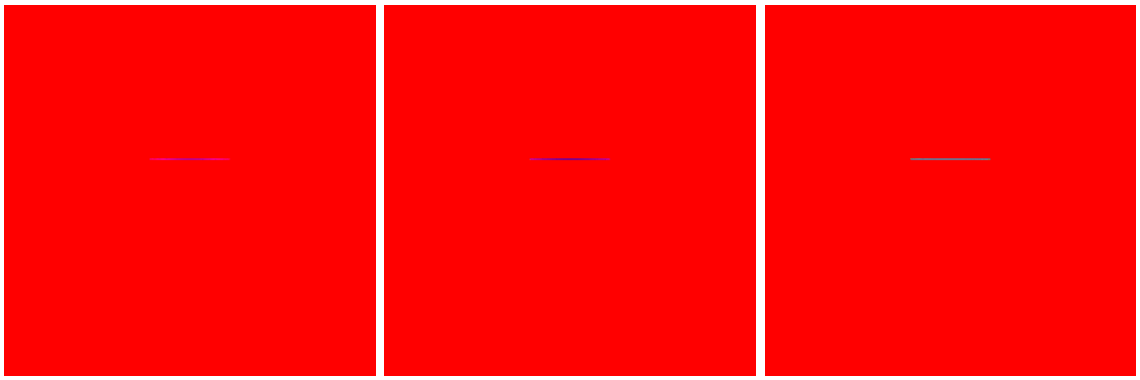

Fig. 53: De izquierda a derecha, visualización y comparativa de los mapas de puntos, previa normalización del canal de densidad, usando los modelos VLP-16, HDL-32E y HDL-64E respectivamente.

Como se puede observar, las diferencias en la representación entre los diferentes modelos son muy notables, haciendo imposible obtener representaciones similares del entorno que permitiese la detección de objetos independientemente del sensor utilizado. Es por ello que la normalización de este canal es necesaria. Como se puede ver en las imágenes, la tendencia que tienen los valores a obtener valores más bajos según nos alejamos del centro de la imagen (lo cual indica una mala normalización, ya que no está comparando el total de puntos detectado con el máximo detectable de forma correcta). Este efecto se puede ver con más detalle en la Figura 54; un ampliado de la imagen izquierda de la Figura 53, correspondiente al modelo VLP-16, que muestra los puntos relacionados con el muro detectado y dicho decremento de los valores al alejarnos del centro de la imagen. En esta imagen puede observarse una clara tendencia de los puntos a reducir su valor según se alejan del centro de la imagen, lo cual indica que la normalización no funciona correctamente y hay que revisarla.

Fig. 54: Ampliación del mapa de puntos obtenido para el VLP-16 previo a la modificación de la normalización del canal de densidad.

Una vez realizados los ajustes en el canal de densidad, se espera una mayor similitud entre las imágenes, las cuales están mostradas en la Figura 55, tanto en el rango de valores obtenido como en el decremento de valores según nos alejamos del centro de la imagen. Estas imágenes, como se esperaba, nos muestran resultados muy similares entre sí, que además son muy cercanos a los mostrados originalmente para el modelo HDL-64E antes de realizar el ajuste en el canal de densidad. Puesto que el código original estaba pensado para la utilización de este modelo y no de los otros dos, tiene sentido que tras ajustar y corregir la normalización los datos sean similares a los obtenidos en este caso.

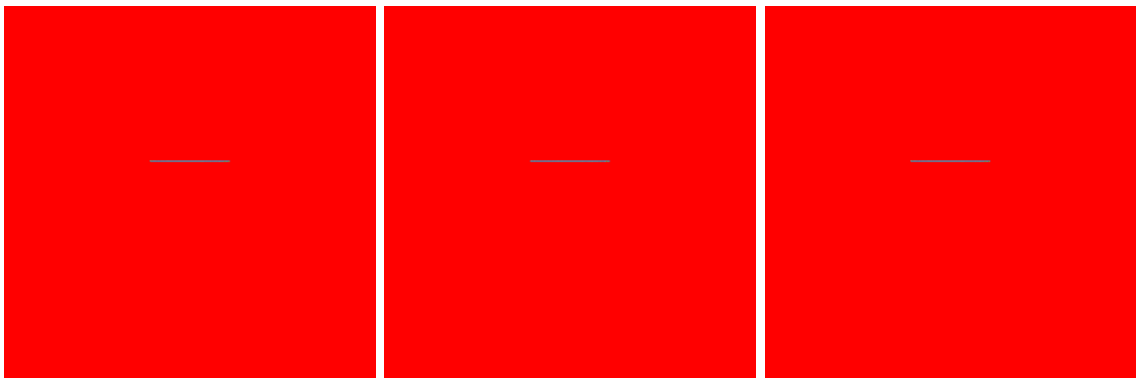

Fig. 55: De izquierda a derecha, visualización y comparativa de los mapas de puntos, tras la normalización del canal de densidad, usando los modelos VLP-16, HDL-32E y HDL-64E respectivamente.

A fin de comparar los estados antes y después de realizar la normalización del canal de densidad, en la Figura 56 se muestra la misma ampliación en el mapa de puntos del

modelo de 16 planos. Como se puede ver, en este caso los valores son mucho más estables y ha desaparecido ese efecto de degradado hacia el exterior presente previo a la normalización de los valores del canal. Además, los valores obtenidos para los 3 modelos son entre sí muy semejantes, por lo que se ha concluido que la normalización funciona correctamente y proporciona resultados muy positivos.

Fig. 56: Ampliación del mapa de puntos del VLP-16, tras la normalización del canal de densidad.

6.1.2. Canales de altura

Como ya se ha comentado, inicialmente el canal de altura mostrado era tan solo un canal que exportaba el valor máximo de altura obtenido en los rangos 0-3 metros. Los primeros resultados obtenidos, previo a realizar ningún tipo de modificación en el paquete, se muestran en la Figura 57 (recortados para mostrar con más detalle la zona de interés: el muro frente al vehículo).

Fig. 57: De izquierda a derecha, ampliación en el canal de altura (resultados del paquete inicial) para los modelos VLP-16, HDL-32E y HDL-64E, respectivamente.

A simple vista, se ve que los resultados no son del todo válidos. Las posibles diferencias en las imágenes pueden estar derivadas de la dispersión de los planos, efecto que se acentúa todavía más al estar el muro situado tan cerca del vehículo. El corte de los planos con el muro no es una línea recta sino una curva, y los rayos pueden dispersarse más arriba y debajo de los límites del muro.

No obstante, el canal de altura ha sido el último elemento en recibir modificaciones y corregirse. Si bien la normalización del canal de densidad no debería afectar a los resultados en este canal, sí que tuvo un efecto positivo, y es que durante el desarrollo de dicha normalización se identificó que los valores de los parámetros de los sensores estaban mal introducidos en el código, lo cual pudo ser uno de los factores que influyesen en estos resultados. No obstante, los parámetros que se han mostrado y utilizado a lo largo de esta memoria son los correctos; se ha preferido dejar los parámetros incorrectos como una referencia a un error puntual que no supuso mayores complicaciones una vez identificado dicho problema.

Así pues, tras el cambio en el canal de densidad se ha realizado la mencionada subdivisión en el canal de la altura, dividiéndolo en 3. En la Figura 58 se muestran los resultados obtenidos para el modelo de 16 planos.

Fig. 58: Ampliación en los canales de altura para el modelo VLP-16. De izquierda a derecha, capa inferior, intermedia y superior, respectivamente.

Estos resultados son más similares a los esperados. Por un lado, el canal superior (2-3 metros) aparece vacío, lo cual es correcto ya que el muro tan solo medía 2 metros, por lo que no debería aparecer en la imagen de la capa superior. Por otro lado, el canal intermedio (1-2 metros) debería mostrar una línea cuyo valor de color estuviese alrededor de a la mitad del espectro HSV (Figura 52). El color obtenido es azul verdoso, no exactamente el esperado pero muy cercano a éste.

Por último, se observa que el canal inferior (0-1 metros) presenta irregularidades en el color, apareciendo de nuevo este efecto de dispersión. No obstante, hay que tener en cuenta la ya mencionada dispersión de los planos, que produce que los puntos de corte del plano con el muro no formen una línea recta, sino una forma cónica. Tiene sentido que este efecto se acentúe en el canal inferior, ya que es donde los planos tienen mayor ángulo al estar situados a mayor diferencia de altura respecto a la altura del sensor, el cual está a una altura aproximada de 1,7 metros en este caso. Por ello, este degradado tiene sentido en este caso particular, y el rango de colores (azul turquesa en el exterior y azul marino en el interior) se corresponde con la mitad del espectro HSV.

En la Figura 59 se muestra un ejemplo de resultados en los canales de altura en una situación real (sin aplicación de colormap). En estas imágenes se observan valores relativamente bajos para los canales inferior e intermedio porque, al calcularse la media de los valores de altura, siempre habrá valores inferiores que reduzcan este valor, mientras que los valores del canal superior son mucho más pronunciados y tienen formas mucho más definidas, debido a estar utilizando el valor de altura máximo detectado en cada casilla. Esto nos indica un funcionamiento correcto del canal de altura, que supone un enfoque más avanzado en las funcionalidades iniciales del paquete y deben dar paso a un desarrollo aún más complejo de los canales, para una mayor especificidad y capacidad de interpretación de los datos obtenidos en cada canal.

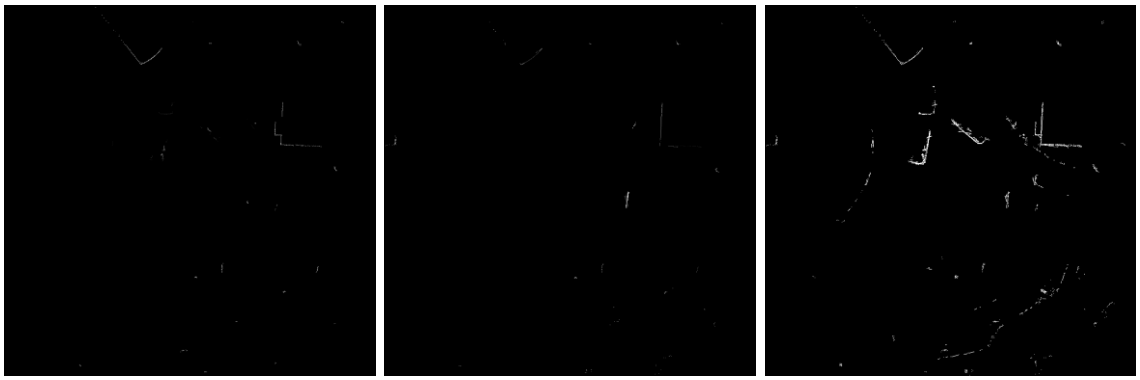

Fig. 59: Ejemplo de imágenes obtenidas en los canales de altura, sin colormap. De izquierda a derecha: capas inferior, intermedia y superior, respectivamente.

Una vez finalizado el análisis de los resultados de la primera mitad del trabajo, se va a proceder a hablar de los obtenidos en la segunda mitad, relacionada con la creación del nodo de ROS de identificación y seguimiento de agentes y dibujado de bounding boxes con etiqueta en la imagen de salida.

6.2. Identificación y seguimiento de agentes

En esta parte del proyecto, el objetivo es asignar una ID única a cada agente detectado y mantenerla en tanto que dicho elemento siguiese apareciendo en la imagen. Para ello, se ha partido de información con formato estandarizado; es decir, se sigue una estructura definida para los datos obtenidos, el nombre de los topics y la información contenida en éstos, de tal forma que se pueden leer diferentes archivos de diferentes fuentes sin tener que modificar el código del programa.

A la hora de comparar los resultados de los tres modos de funcionamiento, ha sido preciso analizar las imágenes obtenidas en forma de vídeo y compararlas. Se ha centrado el foco de las comparaciones en dos situaciones especialmente problemáticas en el seguimiento de agentes: el cruce de agentes y los grupos grandes.

6.2.1. Cruces entre agentes

El cruce de agentes se refiere a, como su nombre indica, situaciones en las que un agente se mueve muy cerca por delante de otro, de tal forma que se inducen errores a la hora de obtener los parámetros de dichos agentes. Esto abarca no solo cruces de posición, que a priori son los más sencillos a resolver, sino también otro tipo de cruces como pueden ser colisiones directas o interacciones físicas entre personas (abrazos, agarre de manos, etc.).

A continuación se muestran comparativas de uno de estos cruces en las imágenes analizadas. En la Figura 60 se analiza el funcionamiento del modo 0 (comparativa de distancias en coordenadas XY relativas al vehículo), en la Figura 61 el del modo 1 (comparativa de distancias del centro de las bounding boxes de cada agente) y en la Figura 62 el del modo 2 (comparativa de distancias en coordenadas XY globales, teniendo en cuenta la posición GPS y orientación del vehículo y la posición relativa de los agentes respecto al mismo). En este ejemplo se representa el comportamiento del sistema en un cruce de peatones sencillo, con una mujer apoyada en una mesa y un hombre pasando por delante de ella.

Como nota adicional, hay que recordar que independientemente del modo de funcionamiento, las cajas y las etiquetas se dibujan exactamente de la misma forma y los datos obtenidos de cada agente (a excepción de la asignación de ID) son los mismos.

Fig. 60: Ejemplo de cruce entre 2 personas en distintos momentos en el modo 0.

Fig. 61: Ejemplo de cruce entre dos personas en distintos momentos en el modo 1.

Fig. 62: Ejemplo de cruce entre dos personas en distintos momentos en el modo 2.

Estos resultados siguen la misma línea general durante la duración de los archivos estudiados. El modo de funcionamiento 0 funciona de forma similar al 1, con errores puntuales de intercambios de ID o aparentes modificaciones en las asignaciones sin ninguna explicación aparente; sin embargo, estos fallos entran dentro de los márgenes de error esperables teniendo en cuenta las técnicas utilizadas. El modo de funcionamiento 2 es un procedimiento más complejo que utiliza un mayor tipo de datos y, por ello, los resultados obtenidos son mucho más positivos ante situaciones de cruce de agentes, donde no se han observado errores en el seguimiento de agentes y sus ID asociadas.

Además, se muestra otro ejemplo en las siguientes imágenes, en los que se va a mostrar el comportamiento ante un cruce múltiple (una señora que cruza delante de varias personas). De nuevo, las Figuras 63, 64 y 65 reflejan el funcionamiento del paquete en los modos 0, 1 y 2, respectivamente.

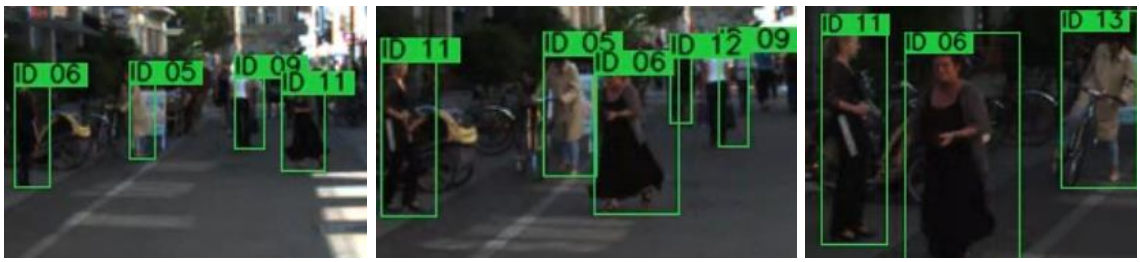

Fig. 63: Ejemplo de cruce sobre varios agentes en el modo 0.

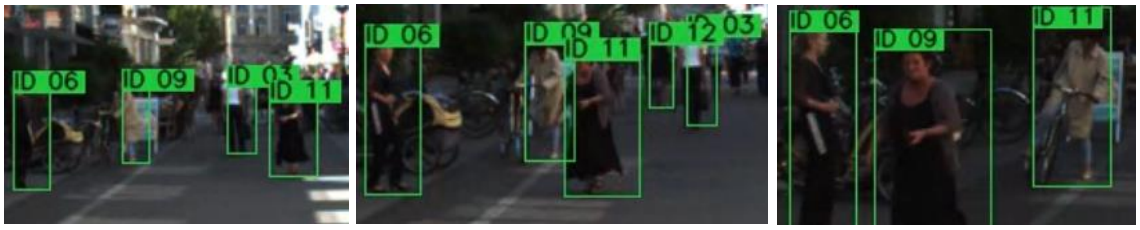

Fig. 64: Ejemplo de cruce sobre varios agentes en el modo 1.

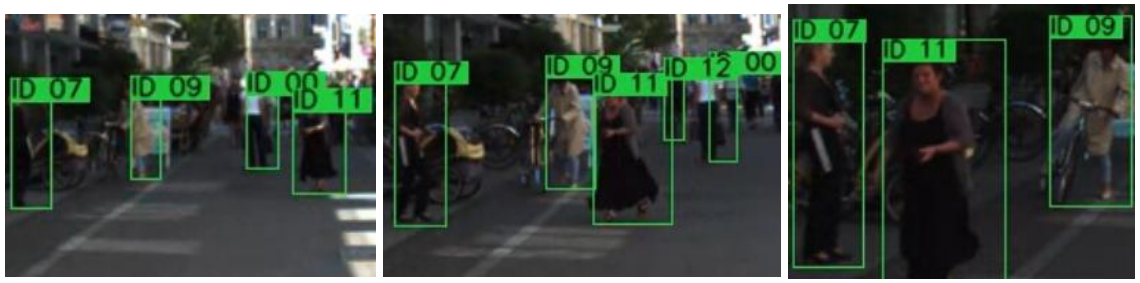

Fig. 65: Ejemplo de cruce sobre varios agentes en el modo 2.

En estos ejemplos se acentúan aún más los errores de los modos 0 y 1; mientras que el modo 2 resulta muy superior en este sentido. La obtención de resultados mucho mejores en este modo tiene sentido, ya que se utilizan posiciones globales de los agentes, por lo que los cruces de agentes no suponen un mayor desafío al estar en todo momento

registradas sus posiciones de forma ajena a la posición y orientación del vehículo y a la imagen detectada por la cámara frontal del mismo.

Además, la fiabilidad de la asignación de ID no se reduce únicamente a entornos de ciudad, sino que también se pueden ejecutar correctamente en situaciones de carretera a mayores velocidades, como el de las siguientes Figuras 66 y 67, donde no se aprecian cambios inesperados de ID y el funcionamiento es correcto.

Fig. 66: Ejemplo de cruce de dos vehículos en una carretera utilizando el modo 2.

Fig. 67: Ejemplo en carretera utilizando el modo 2.

6.2.2. Grupos de agentes

El funcionamiento durante los grupos de agentes es muy complicado de analizar, no solo por el mayor número de elementos e ID que visualizar para comprobar su validez, sino también por el propio solapamiento entre cajas situadas muy cerca entre sí. Además, las cajas de los elementos más cercanos no se dibujan encima de los lejanos,

por lo que en algunas circunstancias puede no quedar claro a qué agente hace referencia dicha caja.

No obstante, con los archivos de grabación analizados se observan numerosos cambios puntuales en las ID de los elementos; este tipo de cambios son erráticos, y algunas veces se revierten al finalizar la escena del grupo de agentes, pero en otros casos se cambian estos valores varias veces, de tal forma que la asignación de ID al principio de la escena es totalmente diferente al del final.

En la Figura 68 para el modo 0 se muestra un error muy común en los grupos de gente: el intercambio de ID. En las imágenes analizadas, estos intercambios suelen durar tan solo un frame, por lo que a veces es complicado localizarlos (especialmente si tras ese frame el intercambio "de vuelta" no se realiza entre los dos mismos agentes). Si bien esta situación no ha aparecido de forma frecuente, en momentos concretos sí que se observa un incorrecto seguimiento de los agentes, que inician y terminan la escena con diferentes ID. Como ya se ha comentado, este tipo de situaciones no son la norma, y entran dentro del margen de error esperable dadas las técnicas utilizadas en el proyecto.

Fig. 68: Ejemplo de cambio puntual de ID durante una escena con un grupo de gente para el modo 0.

Si bien el funcionamiento en el modo 1 es prácticamente igual que el del modo 0 en lo que a grupos de agentes se refiere, el modo 2 ha mostrado mucha mayor resiliencia y fiabilidad en el seguimiento de agentes. Los intercambios puntuales de ID no aparecen en ningún momento y la única fuente de errores son frames concretos en los que la información no queda del todo clara (por ejemplo, varias personas que durante un momento concreto tapan a una que se encuentra detrás del todo, de tal forma que este agente no se detecta durante un frame concreto, pero luego se vuelve a detectar). No obstante, este tipo de errores aparecen independientemente del modo de funcionamiento utilizado ya que dependen de los datos del archivo de grabación y no del funcionamiento del nodo como tal. Por todo esto, se puede concluir que el modo 2, utilizando datos GPS e IMU, presenta resultados con un muy bajo margen de error.

Al margen de las comparaciones entre los tres modos, el funcionamiento del sistema en líneas generales es muy positivo, y para ello un factor muy importante es la calidad de los datos obtenidos, de tipo LiDAR, no solo por la fiabilidad de la información de los agentes recibida (por ejemplo, en la Figura 69 se muestra el cambio instantáneo del tipo de agente cuando una persona se sube a una bici y una comparación con una persona

caminando junto a una bici pero no subida a ella, la cual es detectada correctamente como peatón y no como ciclista), sino también por la cantidad de información obtenida en lo relacionado a variables con las que poder trabajar y que dan posibilidad a un mayor desarrollo de las técnicas empleadas en este trabajo.

(a) (b)

Fig. 69: El agente cambia de verde (peatón) a amarillo (ciclista) cuando la persona se monta en la bici (a). En (b), una persona es correctamente detectada como peatón, pese a caminar con una bici.

7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1. Conclusiones

La primera parte de este proyecto tenía como objetivo principal garantizar el funcionamiento del paquete para los tres modelos de sensor Velodyne, VLP-16, HDL-32E y HDL-64E, y una posterior ampliación de las funcionalidades del mismo.

Mediante los resultados de los canales de densidad, se ha podido concluir que el objetivo de la compatibilidad de modelos se ha cumplido. Además, la división en tres canales de altura (y la corrección en el cálculo de la matriz de máximos puntos detectables) han permitido dicha expansión de las funciones del paquete, por lo que se puede concluir que se han cumplido los objetivos propuestos.

En cuanto a la segunda parte, el objetivo principal era crear y desarrollar el código y archivos necesarios para poner en funcionamiento un nodo en ROS que permitiese suscribirse a topics estandarizados de información de detecciones 3D de agentes, así como utilizar la información de dichos topics para realizar una asignación de ID, seguimiento de los agentes y dibujado de sus respectivas cajas delimitadoras.

Estos objetivos también se han cumplido, obteniendo además resultados muy positivos mediante la utilización de datos GPS e IMU para poner los agentes detectados en una referencia global en vez de en relación al sensor del vehículo. El nodo funciona sin aparentes errores y la asignación y el seguimiento de los agentes son muy positivos, funcionando sin errores en los archivos de grabación analizados.

Además de los objetivos específicos de cada parte del trabajo, también hay que mencionar como objetivo general la familiarización con entornos y procedimientos de visión por computador y vehículos inteligentes, así como el manejo de archivos y paquetes online. El aprendizaje del funcionamiento de ROS y sus herramientas relacionadas ha sido el objetivo más complicado de cumplir debido a la escasa información al respecto de forma gratuita (especialmente a la hora de resolver errores o dudas puntuales, que no se cubren en los contenidos de la wiki de ROS); sin embargo, tras la finalización de ambos bloques del proyecto se ha concluido que este objetivo general también ha sido cumplido.

7.2. Trabajos futuros

En este apartado se mencionan algunas de las posibles mejoras o ampliaciones que se pueden aplicar en el presente proyecto, por ejemplo, como parte del futuro TFG de alumnos de la universidad.

7.2.1. Ajustes en los canales

La división del canal de altura en tres canales nos permite una mayor diferenciación entre elementos del entorno, agentes, etc. Es por esto por lo que una división similar en el canal de densidad tendría mucho sentido. De este modo, se podrían agrupar los canales de altura y densidad en tres grupos de dos, cada uno relacionado con su "capa" correspondiente, para realizar un análisis más completo que permitiese una mejor

identificación de los elementos del entorno, sobre todo aquellos por debajo del umbral de la capa superior, como pueden ser buzones, pivotes, etc.

7.2.2. Predicciones para la asignación de ID

Si bien el sistema de asignación de ID proporciona resultados muy positivos cuando se tienen en cuenta la geolocalización y orientación del vehículo, es posible optimizarlo todavía más. Una de las posibilidades que se plantean es la utilización de un filtro de Kalman para realizar una predicción en la posición de los agentes en el siguiente time frame. Esta posición entonces se compararía con la posición real del agente en dicho time frame, y se obtendría la función de coste en función de la distancia de la posición predicha de los elementos con las posiciones reales.

Este filtro se podría aplicar en conjunción con el sistema de GPS y orientación empleado en el modo 2 del proyecto para una mejor funcionalidad, de tal forma que se tendría en cuenta la posición estimada de forma global, en coordenadas UTM, con la real, expresada de la misma forma. Esto evitaría un gran número de errores de asignación ID en situaciones estándar que suceden debido al propio movimiento del coche cuando se ponen las características de los agentes en relación a éste en vez de utilizar una referencia externa. Es especialmente interesante en casos de agentes que se cruzan o que forman grandes grupos, sobre todo en entornos a alta velocidad donde la obtención de las coordenadas de cada agente, aun expresadas de forma global, no proporciona información suficiente para una asignación de ID sin errores.

Otra posibilidad sería trabajar con las velocidades de cada agente. Las velocidades no son un dato de entrada, y por ello su obtención puede ser complicada; sin embargo, esto evitaría errores en las situaciones de mayor complicación, como son los cruces y los grupos de agentes, especialmente en las situaciones donde los agentes tengan velocidades muy diferentes tanto en módulo como en dirección; por ejemplo, dos personas que se cruzan pero cada una camina hacia una dirección serían relativamente sencillas de diferenciar. Para obtener dichas velocidades hay diferentes opciones; por ejemplo, se pueden realizar aproximaciones basadas en datos pasados asumiendo que cada agente va a comportarse de manera similar a como se ha comportado hasta el momento. Esto supondría almacenar información de los agentes durante más tiempo y trabajar con, por ejemplo, sus posiciones globales, y derivar su velocidad en función de las posiciones registradas en distintos momentos. Esto supondría una complejidad matemática relativamente alta pero mejoraría aún más la fiabilidad del paquete, salvo situaciones muy puntuales de comportamiento impredecible, como puede ser un choque entre agentes que provoque un cambio brusco de velocidad o posición de los mismos.

La introducción de las velocidades supondría también realizar modificaciones en la función de coste del filtro de Munkres. Como posibilidad, se podrían generar dos matrices, una de distancias y la otra de diferencia de velocidades, y aplicar el método húngaro a ambas, y después analizar las dos, elemento a elemento, asignándoles pesos.

Las posibilidades son muy variadas y aquí solo se han mencionado las consideradas más importantes. No obstante, hay una gran cantidad de información disponible que se puede utilizar para el desarrollo de nuevos modos de funcionamiento.

7.2.3. Diferenciación de dibujado de cajas

Actualmente el dibujado de las cajas es completamente funcional, pero con grandes grupos de agentes o con agentes que se cruzan los resultados resultan confusos de entender en algunas situaciones debido a la superposición de las cajas. Una de las posibles mejoras podría ser dibujar primero las cajas de los elementos más alejados del vehículo, de tal modo que las cajas de los agentes cercanos se dibujarían por encima de éstas y quedarían visibles, al igual que el agente al que corresponde. Esto permitiría que, en situaciones con elementos ocluidos por otros agentes, asegurarnos de que las cajas visibles pertenecen a los agentes visibles, y lo mismo para los elementos ocluidos (ya sea parcial o totalmente, mientras los siga detectando el sensor). Esto supondría o bien tener que reevaluar continuamente los agentes mientras se realiza el dibujado o bien reordenarlos directamente en el vector según la distancia al coche, pero en cualquier caso podría ralentizar ligeramente el proceso de dibujado (que no debería afectar al funcionamiento del sistema en tanto que se siga utilizando el TimeSynchronizer).

Además, para los grupos grandes de gente, aun mostrando con prioridad las cajas de los agentes más cercanos al vehículo, debido a la gran cantidad de rectángulos dibujados puede no quedar claro cuál es el límite de la caja visible ya que se superpone con otros rectángulos. La solución a priori más sencilla para este problema podría ser modificar los colores de las cajas para que en vez de un único color, cada tipo de agente tenga asignado un rango de colores (por ejemplo, los peatones podrían tener asignado todo el espectro de verde a azul). De este modo, dibujando la caja en un color aleatorio dentro de ese rango, se reducen las posibilidades de que dos cajas del mismo color se superpongan y no se sepa a qué hace referencia cada una.

Esta modificación en el dibujado de las cajas no supone una modificación del funcionamiento de la asignación de ID como tal, pero sí que supondría una mejora desde el punto de vista de la persona encargada de comprobar que el paquete funciona correctamente, ya que facilitaría la identificación de errores (o la ausencia de éstos).

7.2.4. Visualización en vista de pájaro

Si se posee información de detecciones en 3D, se podría generar una imagen que represente una vista de pájaro que muestre los distintos agentes y sus posiciones (en cierto modo, esta imagen sería similar a los canales de la primera parte del proyecto). Esto permitiría una visualización en 3D de los agentes detectados, o bien de forma relativa al vehículo o incluso de forma global, de tal forma que se puede realizar un seguimiento más objetivo, sin la presencia de oclusiones debido a la imagen de la cámara frontal, y supondría una forma más visual de realizar el seguimiento de los agentes y calcular sus posiciones, velocidades, etc.

8. ENTORNO SOCIOECONÓMICO

8.1. Impacto socio-económico

La integración de los vehículos autónomos en el sector supondría un cambio radical no solo de la industria, sino también del día a día en las carreteras y en la ciudad. En esta sección se mencionan brevemente algunos de estos factores que contribuirían a un cambio de estas dimensiones, con carácter global debido a la gran expansión y alcance de la industria del automóvil.

Por un lado, pese a la preocupación ante la ocasional noticia puntual de un accidente de vehículo autónomo, la realidad es que una menor implicación del conductor conlleva una reducción en el número de accidentes; esto puede comprobarse en la gráfica de la Figura 70. Esto no solo conllevaría una mayor seguridad en las carreteras, sino también cambios asociados en las políticas de seguros, que deberían adaptarse a un nuevo modelo donde los accidentes y, por lo tanto, sus intervenciones, son mucho menos frecuentes. Además, según el estudio de J. R. Treat et al. [50], que analizaba cientos de casos de accidentes de vehículos, se concluyó que en más de un 90% de los accidentes el factor humano fue influyente o causa directa del mismo; estos errores humanos se concentran en situaciones donde el conductor se encuentra en condiciones poco familiares; esta falta de familiaridad con el entorno no sería un problema con la incorporación de la conducción automática.

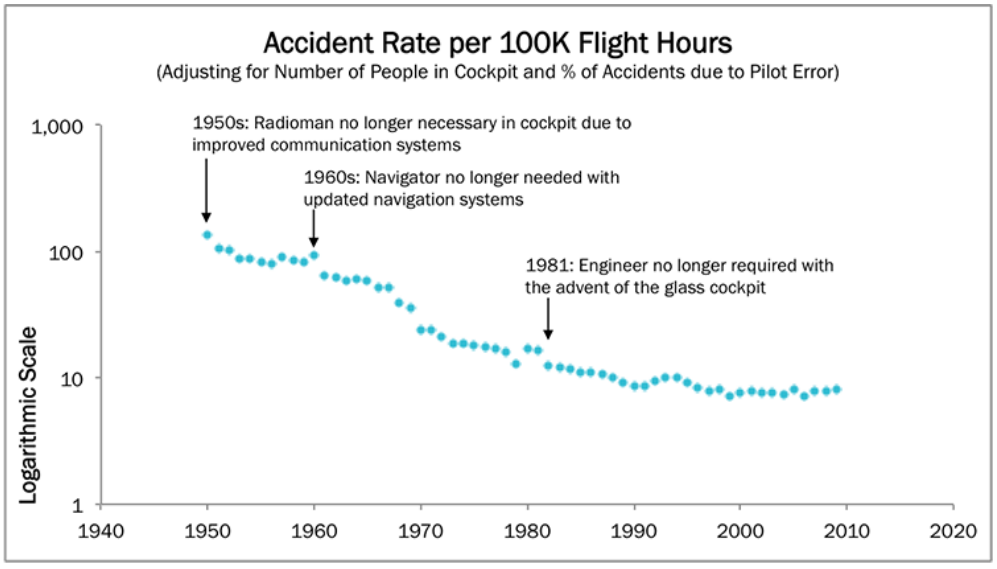

Fig. 70: Reducción en la tasa de accidentes cada 100 mil horas de vuelo.
Fuente: A. T. Keeney [39].

Además de estas situaciones desconocidas, las principales causes de los accidentes de tráfico son el alcohol, exceso de velocidad, pasar semáforos en rojo, fatiga, distracciones y uso del teléfono móvil [51]. Una conducción automatizada, incluso de forma parcial, sería un elemento clave en gran parte de estos factores (si bien hay otros que deben ser regulados de forma externa, como la conducción tras consumir alcohol).

Además, debido a la autonomía de los vehículos, los costes de logística y transporte serían reducidos considerablemente.

Al reducirse el efecto del conductor sobre el volante, aumentaría también el número de conductores, debido por ejemplo a personas mayores no capacitadas para una conducción completa o usuarios con poca experiencia, que causaría una mayor demanda de vehículos y por tanto un mayor número de ventas y de beneficio a las empresas del sector y a sectores relacionados, como el de la gasolina (si bien hay que tener en cuenta el auge de los vehículos eléctricos también).

Por último, al estar dotados de sistemas digitales y electrónicos, este tipo de tecnologías comenzarían a generalizarse y desarrollarse todavía más, creando un impulso adicional a un sector que de por sí está ya en constante crecimiento.

8.2. Presupuesto del proyecto

En este apartado se muestra el presupuesto planteado para el proyecto.

Para el cálculo de los costes materiales, se incluye únicamente el uso del ordenador portátil HP mencionado anteriormente. El resto de elementos de software son completamente libres tanto de acceso como de utilización, por lo que no han conllevado ningún tipo de gasto adicional.

Así pues, la fórmula utilizada para los costes materiales es la proporcionada por el Ministerio de Hacienda, mostrada en la Figura 71, la cual asigna un 26% de coeficiente de amortización para equipos y sistemas informáticos.

$$Coste = precio \cdot \frac{coef.de\ amortización}{100} \cdot \frac{tiempo\ (meses)}{12}$$

Fig. 71: Ecuación de cálculo de costes materiales.
Fuente: Agencia Tributaria [40]

Para el coste asociado al personal, se ha calculado utilizando el sueldo medio bruto anual como programador junior en España a fecha de mayo de 2018 [41], el cual corresponde a 17.772€ anuales.

En la Tabla 2 se recoge el desglose del coste total.

Costes de material				
Elemento	Precio	Coeficiente de amortización	Tiempo utilizado	Coste para el proyecto
HP Notebook G6 PC	500 €	26%	7 meses	75,83 €

Costes de personal		
Sueldo bruto anual	Duración del proyecto	Coste
17.772 €	7 meses	10.367 €

COSTE TOTAL	**10.442,83 €**

Tabla 2: Presupuesto total del proyecto.

9. MARCO REGULADOR

Actualmente, la Legislación española no está preparada para la regulación de coches autónomos, ya que la irrupción de los mismos en el mercado del automóvil plantea retos de diferentes dimensiones. Uno de los retos más importantes está en el plano de la ética y la moral, donde hay un gran debate sobre cuál debe ser el comportamiento de los vehículos autónomos ante accidentes inminentes en los que se ha de elegir entre la vida del conductor o la de personas ajenas al vehículo, por ejemplo. Esto también plantea el problema de, en caso de accidente, determinar la responsabilidad del mismo.

Otro aspecto a tener en cuenta es en relación a la cberseguridad, ya que cualquier tipo de intrusión indeseada en el sistema podría resultar fatal para la vida de los pasajeros del vehículo.

Si bien los vehículos autónomos no están regulados en España ni autorizados legalmente, actualmente sí que se permite la realización de pruebas de los mismos gracias a una instrucción de la DGT [42].

En esta instrucción, se da permiso de conducción de pruebas a vehículos con niveles de automatización 3, 4 y 5 definidos por la SAE (Sociedad de Ingenieros de Automoción). Estos tres niveles son los tres más altos del ranking y están resumidos en la Tabla 3 junto al resto de niveles.

Nivel	Nombre	Descripción
0	Sin automatización	El conductor realiza la totalidad de acciones en el vehículo.
1	Conducción asistida	El vehículo puede tener cierto control en algunos momentos en la ejecución de ciertas acciones tales como el acelerado o frenado o el manejo del volante utilizando información sobre el entorno de conducción.
2	Automatización parcial	El vehículo tiene control en varios sistemas de asistencia a la conducción, tomando el control de acelerado y frenado así como de manejo del volante utilizando información del entorno de conducción.
3	Automatización condicional	El vehículo toma el control con la expectativa de que el conductor humano va a responder de forma apropiada a cualquier solicitud específica de intervención humana.
4	Alta automatización	El vehículo toma el control y es capaz de responder de forma automatizada incluso ante una respuesta no apropiada de solicitud de intervención humana.
5	Automatización completa	El vehículo está controlado de forma autónoma en todos los aspectos de la conducción, bajo todas las condiciones posibles que podrían ser controladas por un conductor humano.

Tabla 3: Niveles de automatización definidos por la SAE.

Entonces, los niveles 3-5 abarcan el conjunto de vehículos capaces de realizar conducción general, tanto con ayuda del conductor como sin ella. Este conjunto de niveles de automatización supondrían que la influencia del conductor sobre el manejo del vehículo tendría un papel secundario o directamente inexistente.

Este tipo de pruebas pueden ser tanto en circuitos cerrados como en tráfico abierto, e implican una responsabilidad total del conductor registrado del vehículo autónomo en caso de incidente o daños derivados de dichas pruebas.

Así pues, para garantizar una correcta inclusión del vehículo autónomo en la Legislación española es necesario resolver varios interrogantes éticos y jurídicos, los cuales quedan también planteados por la jurista Pilar Álvarez Olalla [43]:

- En situaciones de riesgo o accidente inminente, ¿cuál debe ser la reacción del vehículo? ¿Ha de priorizarse la vida del conductor y los pasajeros a toda costa, o ha de buscar siempre minimizar los daños y afectar al menor número posible de personas (incluso si ello supone lesiones graves o incluso el fallecimiento de los pasajeros del vehículo)?
- En caso de accidente, ¿sobre quién recae la responsabilidad del mismo? ¿Se ha de responsabilizar al conductor del mismo, a los pasajeros, al creador del sistema de automatización, o a nadie?
- ¿Qué consecuencias tendrán los vehículos autónomos sobre la privacidad y ciberseguridad de los usuarios? ¿Cómo se gestiona la información utilizada en los sistemas de automatización y, más a largo plazo, la comunicación entre diferentes vehículos autónomos? ¿Qué sucede ante casos de robo de información a través de los sistemas del vehículo?

Durante los próximos años se tendrá que ir dando respuesta a estas preguntas para la inclusión del vehículo autónomo de forma funcional en las carreteras españolas, ya que actualmente estos interrogantes suponen vacíos legales muy complicados de afrontar para los desarrolladores de este tipo de vehículos fuera de entornos de prueba.

10. BIBLIOGRAFÍA

[1] Croissance, "Automotive Industry", *Comisión Europea*. [En línea] Disponible en: https://ec.europa.eu/growth/sectors/automotive_fr [Último acceso: 21 mayo 2018].

[2] ACEA Vehicles in Use Report, 2017.

[3] Global Burden of Disease Study 2016 (GBD 2016) Results. Seattle, United States: Institute for Health Metrics and Evaluation (IHME), 2017.

[4] OMS, "Informe de 2015 sobre la seguridad vial". [En línea] Disponible en: http://www.who.int/violence_injury_prevention/road_safety_status/2015/en/ [Último acceso: 21 mayo 2018].

[5] Morgan & Stanley, "Autonomous Cars: Self-Driving the New Auto Industry Paradigm", 6 de noviembre de 2013.

[6] Ejemplos de datos de usuario del Velodyne LiDAR HDL-64E. [En línea] Disponible en: http://velodynelidar.com/hdl-64e.html [Último acceso: 21 mayo 2018].

[7] C. H. Lampert, M. B. Blaschko y T. Hofmann, "Beyond Sliding Windows: Object Localization by Efficient Subwindow Search". [En línea] Disponible en: http://technodocbox.com/3D_Graphics/71863259-Beyond-sliding-windows-object-localization-by-efficient-subwindow-search.html [Último acceso: 22 mayo 2018]

[8] J. Janai, F. Güney, A. Behl y A. Geiger, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art", 2017.

[9] Carnegie Mellon University, "Look, Ma, No Hands", 2015. [En línea] Disponible en: https://www.cmu.edu/news/stories/archives/2015/july/look-ma-no-hands.html [Último acceso: 23 mayo 2018].

[10] M. Bertozzi et al., "VIAC: an out-of-ordinary experiment", en *Proc. IEEE Intelligent Vehicles Symposium (IV)*, pp.175-180, 2000.

[11] M. Bertozzi, A. Broggi, y A. Fascioli, "Vision-based intelligent vehicles: State of the art and perspectives", en *Robotics and Autonomous Systems*, vol. 32, pp.1-16, 2011.

[12] Google , "Self-Driving Car Project Monthly Report", Marzo de 2016.

[13] Tesla, "Autopilot Full Self-Driving Hardware", *Vimeo*, 2017. [Vídeo en línea] Disponible en: https://vimeo.com/192179726.

[14] A. Geiger, P. Lenz y R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite", en *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012b.

[15] M. Cordts et al., "The cityscapes dataset for semantic urban-scene understanding", en *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[16] Ghent University, Faculty of Engineering and Architecture, "Depth reconstruction". [En línea] Disponible en: https://telin.ugent.be/~mdimitri/depth.html [Último acceso: 23 mayo 2018].

[17] Cityscapes Dataset, "Coarse annotations examples". [En línea] Disponible en: https://www.cityscapes-dataset.com/examples/ [Último acceso: 23 mayo 2018].

[18] D. Scharstein y R. Szeliski, "A taxonomy and evaluation of dense two frame stereo correspondence algorithms", en *International Journal of Computer Vision (IJCV)*, vol. 47, pp.7-42, 2002.

[19] D. Scharstein et al., "High-resolution stereo datasets with subpixelaccurate ground truth", en *Proc. of the German Conference on Pattern Recognition (GCPR)*, 2014.

[20] "The PASCAL Visual Object Classes Homepage". [En línea] Disponible en: http://host.robots.ox.ac.uk/pascal/VOC/ [Último acceso: 24 mayo 2018]

[21] "COCO dataset". [En línea] Disponible en: http://cocodataset.org/#home [Último acceso: 24 mayo 2018]

[22] L. Leal-Taixé et al., "Motchallenge 2015: Towards a benchmark for multi-target tracking", en *arXiv.org (ID 1504.01942)*, 2015.

[23] A. Milan et al., "MOT16: A benchmark for multi-object tracking", en *arXiv.org (ID 1603.00831)*, 2016.

[24] J. Fritsch, T. Kuehnl y A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms", en *Proc. IEEE Conf. on Intelligent Transportation Systems (ITSC)*, 2013.

[25] W. Maddern et al., "1 year, 1000km: The Oxford robotcar dataset", *International Journal of Robotics Research (IJRR)*, 2016.

[26] X. Chen et al., "3d object proposals using stereo imagery for accurate object class detection", en *arXiv.org (ID 1608.07711)*, 2016b.

[27] H. Badino, U. Franke y D. Pfeiffer, "The stixel world – a compact medium level representation of the 3d-world.", en *Proc. of the DAGM Symptosium on Pattern Recognition (DAGM)*, 2009.

[28] L. Pishchulin et al., "Deepcut: Joint subset partition and labelling for multi person pose estimation", en *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[29] J. Zhang y S. Singh, "LOAM: Lidar Odometry And Mapping in real-time", en *Proc. Robotics: Science and Systems (RSS)*, 2014.

[30] J. Levinson, M. Montemerlo y S. Thrun, "Map-based precision vehicle localization in urban environments", en *Proc. Robotics: Science and Systems (RSS)*, 2007.

[31] F. Dellaert y C. Thorpe, "Robust car tracking using Kalman fltering and Bayesian templates", Department of Computer Science and The Robotics Institute, Carnegie Mellon University, USA, 1997.

[32] P. Lenz, A. Geiger y R. Urtasun, "FollowMe: Efficient Online Min-Cost Flow Tracking with Bounded Memory and Computation", en *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[33] Matlab Documentation, “Data with ROS Publishers and Subscribers”, MathWorks, [En línea] Disponible en: https://es.mathworks.com/help/robotics/examples/exchange-data-with-ros-publishers-and-subscribers.html [Último acceso: 28 mayo 2018].

[34] E. Fernández, L. S. Crespo, A. Mahtani y A. Martínez, “The ROS Filesystem levels”, Hub Packtpub, [En línea] Disponible en: https://hub.packtpub.com/ros-filesystem-levels/ [Último acceso: 28 mayo 2018].

[35] Edureka, “Git Tutorial – Commands And Operations In Git” [En línea] Disponible en: https://www.edureka.co/blog/git-tutorial/ [Último acceso: 28 mayo 2018].

[36] R. A. Pilgrim, “Munkres’ Assignment Algorithm. Modified for Rectangular Matrices”, Course notes, Murray State University.

[37] I. Dotlic et al., “Angle of arrival estimation using decawave DW1000 integrated circuits”, en *Conference: 2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, 2017.

[38] J. L. Blanco, “A tutorial on SE(3) transformation parameterizations and on-manifold optimization”, Dpto. de Ingeniería de Sistemas y Automática, Universidad de Málaga, España, 9 de mayo de 2013.

[39] A. T. Keeney, “Autonomous Vehicle Safety: Reduce Auto Accidents by 83%”. [En línea] Disponible en: https://ark-invest.com/research/autonomous-vehicle-safety#fn-7907-2. ARK Investment Management, 2018. [Último acceso: 05 junio 2018].

[40] Agencia Tributaria, "Tabla de amortización simplificada", [En línea]. Disponible en: http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresarios_individuales_y_profesionales/Rendimientos_de_actividades_economicas_en_el_IRPF/Regimenes_para_determinar_el_rendimiento_de_las_actividades_economicas/Estimacion_Directa_Simplificada.shtml [Último acceso: 05 junio 2018].

[41] Indeed, “Sueldos en Programador/a junior en España”. [En línea] Disponible en: https://www.indeed.es/salaries/Programador/a-junior-Salaries?period=yearly. [Último acceso: 05 junio 2018].

[42] Instrucción 15/V-113, “Autorización de pruebas o ensayos de investigación realizados con vehículos de conducción automatizada en vías abiertas al tráfico en general”, Dirección General de Tráfico. [En línea] Disponible en: http://www.dgt.es/Galerias/seguridad-vial/normativa-legislacion/otras-normas/modificaciones/15.V-113-Vehiculos-Conduccion-automatizada.pdf [Último acceso: 05 junio 2018].

[43] Pilar Álvarez Olalla, "Desafíos legales ante la circulación de los coches autónomos: Implicaciones éticas, responsabilidad por accidente y ciberseguridad", en *Revista Doctrinal Aranzadi*, Civil-Mercantil num.2/2017 parte Legislación. Comentarios, 2017.

[44] Towards Data Science (2017). “Teaching Cars To See — Vehicle Detection Using Machine Learning And Computer Vision”. [En línea] Disponible en:

https://towardsdatascience.com/teaching-cars-to-see-vehicle-detection-using-machine-learning-and-computer-vision-54628888079a [Último acceso: 10 junio 2018].

[45] S. Mallick, "Histogram of Oriented Gradients", en *Learn OpenCV*. [En línea] Disponible en: https://www.learnopencv.com/histogram-of-oriented-gradients/ [Último acceso: 10 junio 2018].

[46] L.Wang, Y.Zhang y J.Wang, "Map-Based Localization Method for Autonomous Vehicles Using 3D-LIDAR", en *IFAC-PapersOnLine 50-1 (2017) 276–281*.

[47] P. Emami et al., "Machine Learning Methods for Solving Assignment Problems in Multi-Target Tracking", en *arXiv.org (ID 1802.06897)*, 2018.

[48] A. A. Butt y R. T. Collins, "Multiple Target Tracking Using Frame Triplets", Dept. of Computer Science and Engineering, The Pennsylvania State University, EE.UU., 2018.

[49] G. Sung, "Behavioral Cloning - Udacity Self-Driving Car Nanodegree", *Youtube*, 13 de diciembre de 2016. [Vídeo en línea] Disponible en: https://www.youtube.com/watch?v=eHJ6yHTEdBM.

[50] J. R. Treat et al., "Tri-level study of the causes of traffic accidents: final report. Executive summary".

[51] RMIIA, "Cost of Auto Crashes & Statistics". [En línea] Disponible en: http://www.rmiia.org/auto/traffic_safety/Cost_of_crashes.asp [Último acceso: 11 junio 2018].