eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Máster en Ingeniería Computacional y Sistemas Inteligentes
Vicomtech

Trabajo de Fin de Máster

Reproyección multi-vista de objetos para vehículos autónomos

Autor
Mikel García Fonseca

Directores
Ignacio Arganda, Gorka Azkune y Marcos Nieto

2019

Resumen

La detección y reproyección de objetos 3D en imágenes monoculares ha sido poco usada en el ámbito de la conducción autónoma debido a la gran eficacia del uso de LiDARs (*Laser Imaging Detection and Ranging*). El problema con este método es el precio de estos sistemas láser. Con el aumento de la capacidad de cómputo de tanto tarjetas gráficas como procesadores, la tarea de reproyección usando solo las detecciones de redes neuronales y cámaras calibradas está volviéndose más viable. Es por esto que se ha propuesto un método de reproyección de detecciones 2D, realizando la calibración de un sistema de 4 cámaras en el vehículo de testeo del centro de investigación Vicomtech. El método se ha integrado en un diagrama RTMaps (*Real Time Multisensor applications*) en tiempo real, y utilizado en el vehículo de test en diferentes demostraciones. Adicionalmente, se ha realizado un estudio del error de reproyección utilizando una base de datos anotada y accesible públicamente.

Índice general

Resumen I

Índice general III

Índice de figuras V

Índice de tablas XI

1. Introducción 1

1.1. Estructura de la memoria 2

1.2. Objetivos 2

2. Conceptos Fundamentales 5

2.1. Conducción autónoma 5

2.2. Visión por computador 9

2.2.1. Detección de objetos 10

2.2.2. Cámaras Pinhole 13

2.2.3. Calibración de cámaras 18

2.3. Reproyección 2D-3D 20

3. Desarrollo del proyecto 23

3.1. CarLOTA 24

3.2. Proceso de calibración 25

3.3. Detección de objetos 32

3.4. Método de reproyección 2D-3D 34
3.5. Tracking 37
3.6. Diagramas RTMaps 38
3.6.1. Diagrama Offline 39
3.6.2. Diagrama Offline con mapas y LiDAR 41
3.6.3. Diagrama a tiempo real 45

4. Análisis de los resultados **49**
4.1. Dataset KITTI 49
4.2. Análisis de reproyección para coches 50
4.3. Análisis de reproyección para peatones 57

5. Conclusiones y trabajo futuro **63**

Anexos

A. Verificación de la calibración. **67**

B. RTMaps **69**

C. Grabaciones realizadas **73**
C.1. Miramón 73
C.2. Simulación demostración Eindhoven 73
C.3. Demo ITS Eindhoven 75

Bibliografía **79**

Índice de figuras

2.1. Distintos niveles de autonomía según la SAE. 6

2.2. Componentes principales de un coche autónomo. 8

2.3. Datos del estudio realizado en 2015 por la NHTSA, mostrando el principal motivo de accidentes de tráfico en más de 2 millones de accidentes en EE.UU. 9

2.4. Distintos ejemplos de aplicaciones reales en visión por computador. . . . 10

2.5. Detección de objetos a nivel de reconocimiento de imagen, detección con *bounding boxes* o segmentación a nivel de píxel. 11

2.6. Variaciones que pueden sufrir los objetos o la misma imagen, complicando la tarea de detección [1]. 12

2.7. Representación gráfica básica del funcionamiento de una CNN usando una capa de convolución y otra de *pooling*. 13

2.8. Ejemplo de proyección de una cámara tipo *pinhole* proyectando una imagen invertida de un árbol. 13

2.9. Modelo de cámara *pinhole* con un plano de imagen paralelo a (X_C, Y_C) [2]. 14

2.10. Relación entre los sistemas de coordenadas frente a la distancia focal [2]. 15

2.11. Origen del sistema de coordenadas del plano de imagen (x, y) y de la cámara$(\mathrm{x}_{cam}, \mathrm{y}_{cam})$ [2]. 16

2.12. Cambio de coordenadas entre los sistemas de referencia del mundo y la cámara [2]. 17

2.13. Tipos de distorsión radial que puede generar una lente en una imagen [3]. 18

2.14. Ejemplo de una lente paralela al plano de imagen (izquierda), y una lente mal alineada causando distorsión tangencial (derecha) [3]. 19

2.15. Varios patrones de calibración para emplear en calibración de cámaras. . . 20

2.16. Proyección de una nube de puntos generada por un LiDAR en un entorno virtual. 21

3.1. Foto de CarLOTA en el garaje de Vicomtech, con paneles de calibración a su alrededor. 23

3.2. Nvidia Drive PX 2 AutoChauffeur con sus distintos conectores. 24

3.3. Dos de las cámaras Sekonix instaladas en el coche. 26

3.4. Dos marcadores de Nvidia para su herramienta de calibración con los identificadores 89 y 183. 27

3.5. Dos fotos de ejemplo en el que se muestran dos marcadores para la calibración de extrínsecos de las cámaras. 27

3.6. Estructura que tiene que tener el directorio para la calibración de Nvidia. . 28

3.7. Imagen de validación de la herramienta de Nvidia, cuanto mejor esté alineado el plano verde con los marcadores, más precisa habrá sido la calibración. 30

3.8. Imagen de validación para los parámetros intrínsecos de la herramienta de Nvidia, los puntos azules son los puntos detectados del marcador y los rojos los reproyectados. Las líneas amarillas unen estos puntos, por lo que, si hay una línea amarilla muy grande, el error de reproyección ha sido alto, mientras que si la línea no se percibe, la reproyección ha sido precisa. 30

3.9. Posición de las cámaras obtenida por la herramienta de calibración de Nvidia frente a un modelo de Toyota Prius simulado en Prescan. 32

3.10. Posición de las 4 cámaras Sekonix en CarLOTA. 33

3.11. Detección de objetos a tiempo real en RTMaps usando YOLO v3 en CarLOTA. 34

3.12. Explicación gráfica del proceso de reproyección a realizar para estimar la distancia de una detección junto a los ejes de coordenadas del mundo. . . 35

3.13. Visualizador 3D de RTMaps, mostrando la posición de las reproyecciones hechas como cubos rectangulares. El color azul es para las detecciones de la cámara frontal, amarillo para las izquierda, rojo para la derecha y verde para las trasera. 36

3.14. Campo de visión de cada cámara Sekonix de CarLOTA, mostrando las regiones de solapamiento entre cámaras a 30 metros. 37

3.15. Array $Objects$ de tamaño $N+1$ representando los objetos que obtenemos tras la reproyección y el filtro del tracking 2D. 38

3.16. Vista de pájaro mostrando un ejemplo de solapamiento entre dos reproyecciones con identificadores distintos entre la cámara frontal y la izquierda. En el caso de las reproyecciones que vemos en la zona inferior izquierda, entre la cámara derecha y la trasera, las reproyecciones no están lo suficientemente cerca del *threshold* de 4.5 metros como para considerar un solapamiento. 39

3.17. Diagrama de RTMaps para la visualización offline del entorno 3D y las cámaras de CarLOTA. 40

3.18. Output del diagrama, en el que podemos ver el visualizador 3D, los *streams* de vídeo de las 4 cámaras con sus detecciones de YOLO y las distancias de las detecciones. 41

3.19. En la esfera izquierda vemos el ángulo que representa la latitud, mientras que en la esfera derecha vemos el ángulo que representa la longitud. Con estos dos ángulos, podemos obtener un punto en la superficie de la tierra representando la posición en la que nos encontramos. 42

3.20. Representación de un mapa usando el sistema UTM, en el que podemos ver cómo está separado por 60 husos de 6º de longitud verticalmente y 20 bandas de 8º de longitud horizontalmente representadas por letras de la C a la X excluyendo a la I y la O. 43

3.21. Diagrama de RTMaps para la visualización *offline* del entorno 3D junto al mapa y el LiDAR con las cámaras de CarLOTA. 44

3.22. Output del diagrama de RTMaps para la visualización *offline* del entorno 3D, junto al mapa y el LiDAR con las reproyecciones de CarLOTA. . . . 45

3.23. Diagrama de RTMaps para el funcionamiento a tiempo real de todos los componentes de CarLOTA. 46

4.1. *Heatmap* con la posición de los coches del *dataset* KITTI, limitado a un rango de 40 metros y sin truncamiento. 51

4.2. *Heatmap* con la reproyección de las posiciones de los coches del *dataset* KITTI, limitado a un rango de 40 metros y sin truncamiento. 51

4.3. Gráfico 3D con el error de reproyección total para cada coche etiquetado en KITTI tras el filtro de datos. 52

4.4. Gráfico 3D con el error de reproyección en X para cada coche etiquetado en KITTI tras el filtro de detecciones. 53

4.5. Diagrama de dispersión mostrando el valor reproyectado frente al real. Los puntos están coloreados según su error de reproyección en X, usando la paleta de colores superior derecha. En la parte inferior podemos ver un *heatmap* con la densidad de los puntos del diagrama de dispersión. 54

4.6. Imagen de validación del *dataset* KITTI, en el que comprobamos el error cometido en la reproyección frente a los valores reales de posición. 54

4.7. Fotos mostrando la diferencia en el proceso de reproyección en el caso de resolver la asunción del plano horizontal del suelo y poder calcularlo. . . 55

4.8. Gráfico 3D con el error de reproyección en Y para cada coche etiquetado en KITTI tras el filtro de detecciones. 56

4.9. Diagrama de dispersión mostrando el valor reproyectado frente al real. Los puntos están coloreados según su error de reproyección en Y, usando la paleta de colores superior derecha. En la parte inferior podemos ver un *heatmap* con la densidad de los puntos del diagrama de dispersión. 57

4.10. Imagen de validación del *dataset* KITTI, en el que comprobamos el error cometido en la reproyección frente a los valores reales de posición en un *bounding box* cercano a la línea del horizonte. 58

4.11. Gráfico 3D con el error de reproyección total para cada persona etiquetada en KITTI tras el filtro de detecciones. 58

4.12. Gráfico 3D con el error de reproyección en X para cada persona etiquetada en KITTI tras el filtro de detecciones. 59

4.13. Diagrama de dispersión mostrando el valor reproyectado frente al real para peatones. Los puntos están coloreados según su error de reproyección en X, usando la paleta de colores superior derecha. En la parte inferior podemos ver un *heatmap* con la densidad de los puntos del diagrama de dispersión. 60

4.14. Gráfico 3D con el error de reproyección en Y para cada persona etiquetada en KITTI tras el filtro de detecciones. 61

4.15. Diagrama de dispersión mostrando el valor reproyectado frente al real para peatones. Los puntos están coloreados según su error de reproyección en Y, usando la paleta de colores superior derecha. En la parte inferior podemos ver un *heatmap* con la densidad de los puntos del diagrama de dispersión. 62

A.1. Verificación de las 4 cámaras para las calibraciones, usando la lona con forma de tablero de ajedrez. 68

B.1. Ejemplo de un diagrama de RTMaps con distintos componentes interconectados entre sí. 69

B.2. Dos componentes leyendo la salida desde el *buffer* del componente 1. . . 70

C.1. Ruta marcada en Google Maps para la grabación simulando la incorporación al carril del ITS en Eindhoven. 74

C.2. Ruta marcada en Google Maps para la grabación simulando la incorporación al carril en la AP-8 para el uso de HD Maps. 75

C.3. Maniobra a realizar en uno de los casos de uso del ITS, cambio de carril a la izquierda entre dos vehículos. 76

C.4. Todas las ventanas de los componentes del diagrama en tiempo real usado en la demo del ITS. 76

C.5. Visualización de las ventanas generadas por el diagrama en tiempo real de RTMaps junto a imágenes de CarLOTA incorporándose al carril izquierdo en el ITS. 77

Índice de tablas

3.1. Resultados de los parámetros intrínsecos y extrínsecos de la herramienta de calibración de Nvidia 31

4.1. Error medio de reproyección para los distintos tipos de detecciones, analizados usando el *dataset* de KITTI. 62

1. CAPÍTULO

Introducción

En el contexto de aplicaciones de ayuda a la conducción **ADAS** (*Advanced Driver Assistance Systems*) y de conducción autónoma **AD** *(Autonomous Driving)*, la percepción del exterior del vehículo es el primer componente sobre el que se construyen sistemas de frenado de emergencia, ayuda en aparcamiento, control de velocidad o mantenimiento en el carril.

La percepción se basa en la utilización de sensores de medición del entorno, para determinar su posición, velocidad, tamaño o clase. Típicamente se han utilizado sensores de rango como Láseres (LiDAR) o radares, capaces de muestrear físicamente el entorno. Las cámaras se utilizan para generar proyecciones (imágenes) del entorno sobre las cuales realizar detección y clasificación de los elementos de interés. El análisis de imagen se ha vuelto cada vez más popular y una solución aceptada en el sector gracias a la reducción de costes y tamaños de las propias cámaras, así como en los grandes avances en visión artificial y aprendizaje automático. Desde la aparición reciente de las técnicas modernas de Aprendizaje Profundo (*Deep Learning*), junto con equipamiento HW de cómputo de altas capacidades (e.g. NVIDIA Drive PX 2), es posible disponer de SW de detección y clasificación de imágenes embarcado en vehículos de testeo, con gran robustez frente a las condiciones reales de funcionamiento, incluyendo condiciones meteorológicas adversas, variabilidad de entornos urbanos y peri-urbanos, etc.

Uno de los mayores retos de las tecnologías de visión artificial en este contexto es proporcionar capacidades de medición espacial y volumétrica. De forma inherente, una imagen es una proyección 2D del mundo 3D, donde una dimensión se pierde en el proceso proyectivo. Por tanto, cualquier elemento 2D definido en la imagen no se puede reproyectar de forma unívoca al espacio 3D, sino que existe siempre una ambigüedad de tamaño-posición (e.g. un mismo objeto visual 2D puede ser la proyección de un objeto 3D muy grande y lejano, o un objeto 3D muy pequeño y cercano).

En el entorno de conducción, se tiende a asumir que el vehículo circula sobre una su-

perficie aproximadamente plana en su entorno cercano (i.e. menos de 15 metros). Esta asunción permite resolver la ambigüedad proyectiva de la reproyección de objetos 2D de imagen, ya que se puede intersectar su reproyección en el plano de suelo y determinar su posición y tamaño. Esta técnica es solo válida para objetos que de verdad pertenezcan al plano del suelo como, por ejemplo, los carriles, o para aquellas partes de los objetos que estén en contacto con el suelo, como las ruedas de los coches o la parte inferior de los peatones.

La reproyección con múltiples cámaras, puede virtualmente proporcionar mediciones espaciales 3D en un rango 360º alrededor del vehículo. Los retos a resolver son:

1. Disponer de un conjunto de cámaras calibradas, es decir, sus posiciones y rotaciones relativas son conocidas entre sí y con respecto del suelo.
2. Utilizar técnicas de detección (e.g. Redes neuronales convolucionales), que proporcionen detecciones 2D precisas.
3. Desarrollar la reproyección resolviendo duplicidades o ruido proyectivo, aplicando técnicas de seguimiento temporal.
4. Implementar un diagrama de proceso en equipamiento HW instalable en un vehículo de manera que opere en tiempo real.

1.1. Estructura de la memoria

La memoria de este proyecto está estructurada de la siguiente manera. Primero, tenemos un capítulo de *Conceptos Fundamentales* en el que tratamos los temas de *Conducción autónoma*, *Visión por computador* y *Reproyección 2D-3D*. Después, analizamos todo el trabajo hecho en el proyecto en el capítulo *Desarrollo del Proyecto*. Tras esto, usaremos un *dataset* público para analizar los resultados obtenidos en el capítulo *Análisis de los resultados*. Por último, tenemos el capítulo de *Conclusiones y trabajo futuro*, seguido del *Anexo* en el que aportamos información adicional sobre el proyecto.

1.2. Objetivos

Este proyecto tiene como finalidad crear un método de reproyección y seguimiento de detecciones 2D a 3D en un vehículo de testeo. Para ello, se busca completar los siguientes objetivos:

- Obtener los parámetros intrínsecos y extrínsecos mediante la calibración del sistema de cámaras.

- Diseñar e implementar un componente software para la reproyección 2D-3D, usando la calibración de las cámaras
- Implementar un componente software para el filtrado y seguimiento de las proyecciones.
- Crear un diagrama completo que conecte detector, re-proyector y seguimiento en el vehículo.
- Evaluar el error de reproyección cometido.

2. CAPÍTULO

Conceptos Fundamentales

En este capítulo revisaremos los conceptos fundamentales en los que se enfoca el proyecto. Primero, hablaremos sobre el estado actual de la **conducción autónoma**. En segundo lugar, hablaremos de los campos estudiados en **visión por computador** en este proyecto, y mencionaremos también brevemente temas como redes neuronales o *Deep Learning*. Por último, analizaremos el estado del arte de la **reproyección 2D-3D**, tema principal de este proyecto.

2.1. Conducción autónoma

Un coche totalmente autónomo es un vehículo capaz de conducir libremente sin la necesidad de un conductor. Esta idea ha estado presente desde poco después de la creación de los automóviles, siendo limitada por la tecnología de la época. Después de los grandes avances científicos, tanto en hardware como en áreas como la visión por computador, inteligencia artificial, *Deep Learning*, etc. esta idea ha ido materializándose poco a poco.

En la última década, y gracias a los avances en investigación en el área, ya se han creado y probado en escenarios reales prototipos de vehículos autónomos. Las primeras pruebas con estos prototipos se han realizado en zonas específicas y protegidas, demostrando que la conducción autónoma está cada vez más cerca de llegar a un uso cotidiano.

La mayoría de pioneros en la industria empezaron a hacer pruebas con estos prototipos sin llevar a un conductor en 2013, incluyendo empresas como General Motors, BMW, Volkswagen, Mercedes Benz, Nissan, Audi, Toyota, Volvo y Ford. Sin embargo, se llevan haciendo experimentos relacionados con la conducción autónoma desde 1926 [4], donde la compañía *Houdina Radio Control* en Nueva York emitía señales de radio desde un coche que circulaba detrás de otro, el cual controlaban por medio de estas señales. En los

años 80, se llevó a cabo uno de los proyectos en torno a la conducción autónoma con más financiación hasta la época: el proyecto *PROMETHEUS* [5] (*PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety*), llevado a cabo por *EUREKA* (Organización intergubernamental de ciencia e investigación), donde se invirtieron más de mil millones de dólares. No fue hasta 1987 que se realizó la primera demostración que usaba visión por computador, LiDAR y control autónomo para dirigir un vehículo robotizado que alcanzaba hasta 31 km/h [6]. Gracias a estos avances, en 1994 los coches autónomos VaMP y VITA II consiguieron viajar más de 1000 km en autopistas cercanas a París, llegando a alcanzar velocidades de hasta 130 km/h en autopistas de 3 carriles con tráfico [7]. Otro evento importante fue el *DARPA Grand Challenge* de 2004, en el que había que recorrer una distancia de 240 km fuera de carreteras de California a Nevada, con un premio de 1 millón de dólares. Pese a que en 2004 ningún participante logro acabar la carrera con éxito, en la siguiente competición realizada en 2005 el *Stanford Racing Team* logró realizar la ruta en seis horas y cincuenta y cuatro minutos. Desde entonces, se han realizado varias ediciones en distintos entornos.

La *NHTSA* (*National Highway Traffic Safety Administration*) creó una jerarquía de varios niveles en 2013 para clasificar la autonomía de estos vehículos [8], pero se actualizó a una más moderna de 6 niveles creada por la *SAE* (*Society of Automotive Engineers*) como vemos en la Figura 2.1.

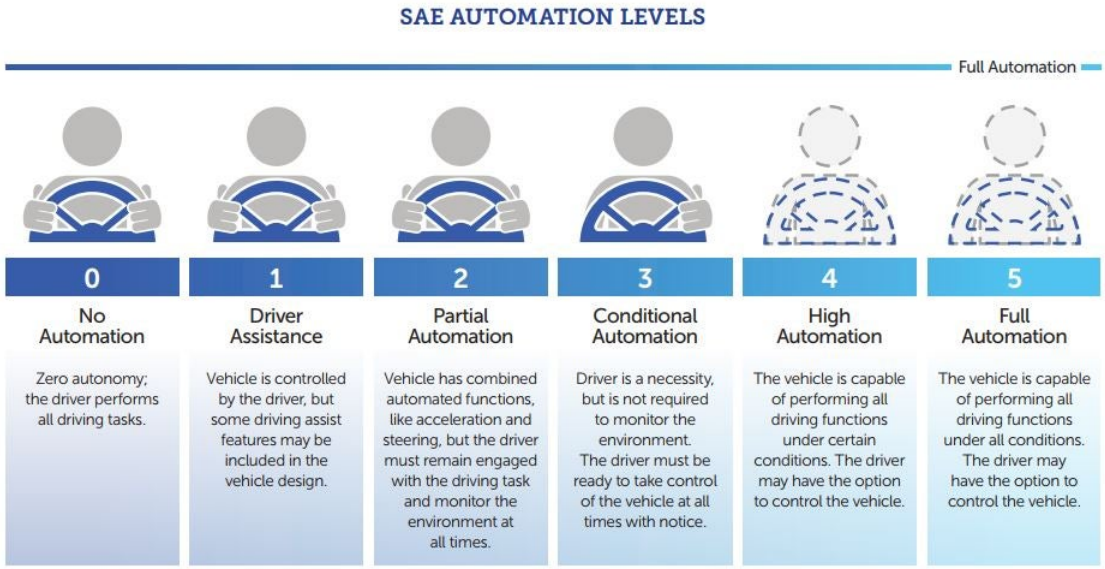


Figura 2.1: Distintos niveles de autonomía según la SAE.

- **Nivel 0 (No hay autonomía):** El conductor está totalmente a cargo de todas las funciones del vehículo (frenar, girar, acelerar...) en todo momento y es el único responsable de que el vehículo circule adecuadamente.

- **Nivel 1 (Asistencia al conductor):** El conductor debe mantener las manos en el volante en todo momento y el coche puede ayudarle en ciertas tareas, como control de crucero o asistencia en el aparcado con cámaras o sensores.

- **Nivel 2 (Autonomía parcial):** El vehículo es capaz de acelerar, frenar o girar el volante pero, aun así, el conductor tiene que estar alerta en todo momento para retomar el control del vehículo, ya que en algunos casos el vehículo no sabe cómo actuar correctamente. El conductor sigue teniendo la responsabilidad y tiene que estar atento en todo momento.

- **Nivel 3 (Autonomía condicionada):** El vehículo es capaz de conducir sin intervención humana bajo ciertas condiciones, ya que es capaz incluso de reaccionar ante situaciones imprevistas. Si estas condiciones se cumplen, el conductor puede realizar tareas que no tengan que ver con la conducción, como usar el móvil o ver una película. Sin embargo, el conductor ha de estar listo para intervenir si el coche le avisa de que no es capaz de seguir autónomamente por algún motivo.

- **Nivel 4 (Autonomía elevada):** Similar al nivel 3 de autonomía, pero no es necesaria la atención del conductor, por lo que este podría dormir o cambiarse de asiento. Esta autonomía todavía está limitada a ciertos factores, como la geolocalización del vehículo. En caso de que esto no se cumpla, el vehículo ha de ser capaz de aparcar o ir a un sitio seguro si el conductor no retoma la tarea de conducción.

- **Nivel 5 (Autonomía total):** El vehículo no necesita siquiera un volante, ya que el conductor no es necesario. Un ejemplo de esto sería un taxi robótico capaz de llevar a gente a su destino sin ningún conductor.

Los componentes principales de un vehículo autónomo los podemos ver en la Figura 2.2, aunque hay vehículos autónomos que constan de más sensores de los que se ven en la imagen, dependiendo de su autonomía y características.

- **GPS** (*Global Positioning System*)**:** Es un sistema capaz de determinar la posición en la que se encuentra este. Normalmente tiene una precisión de unos pocos metros, pero ya existen mejoras como el GPS diferencial o *DGPS* capaces de proporcionar una precisión exacta de unos pocos centímetros de error.

- **IMU** (*Inertial Measurement Unit*)**:** Es un dispositivo electrónico que mide la velocidad, orientación y fuerzas gravitacionales, usando para ello acelerómetros y giroscopios.

- **LiDAR** (*Laser Imaging Detection and Ranging*)**:** Es un dispositivo capaz de medir la distancia frente a otros objetos, emitiendo haces láser pulsados y calculando el tiempo que tarda en rebotar contra la superficie del objeto en cuestión. Estos dispositivos ya tienen ángulos de visión de 360º horizontalmente, por lo que pueden calcular distancias contra obstáculos alrededor de todo el vehículo. Este aparato

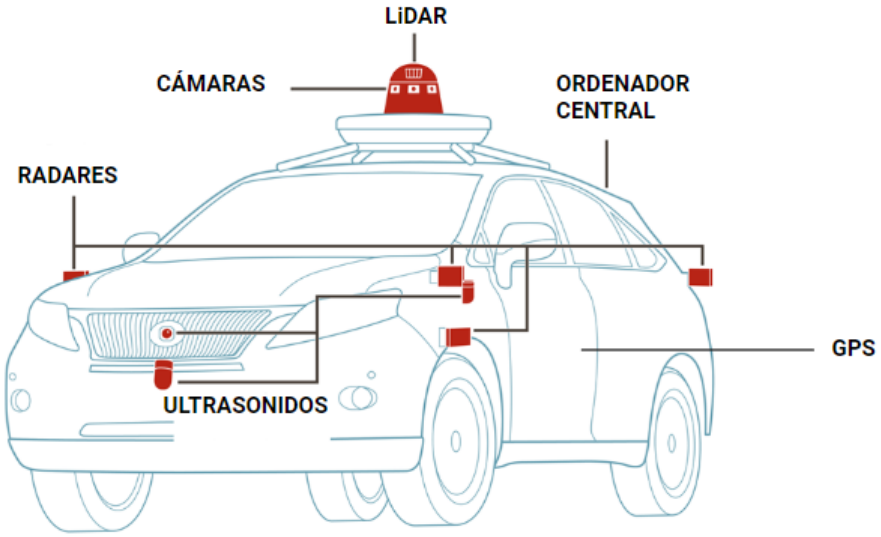


Figura 2.2: Componentes principales de un coche autónomo.

tiene una precisión de ± 2 cm, pero no funciona a la perfección en condiciones meteorológicas como lluvia, nieve o niebla. Los LiDAR usados en automóviles tienen un alcance de unos 100 metros, aunque algunas versiones más recientes de estos dispositivos ya alcanzan los 300 metros.

- **Radar** (*Radio Detection And Ranging*)**:** Es un sistema que usa ondas de radio para determinar la distancia, ángulo o velocidad de objetos. Entre sus ámbitos de aplicación se encuentran la meteorología, tráfico y usos militares. Funciona bien en cualquier condición climatológica y, dependiendo del modelo usado tiene mayor alcance que el LiDAR, en cambio, tiene una menor precisión.

- **Ultrasonidos:** A semejanza de algunos animales como los murciélagos, emite ondas sonoras y capta el eco que producen al impactar con los objetos para determinar la distancia a estos. Su eficacia es máxima a bajas velocidades y con obstáculos cercanos, por eso se suele utilizar principalmente para ayuda en el aparcado.

- **Cámaras:** Dependiendo de cuántas cámaras tenga el vehículo es capaz de proporcionar visión de 360º alrededor del mismo. Junto a hardware de cómputo adecuado, también se podrán usar las imágenes obtenidas de las cámaras para la detección y reconocimiento de objetos o personas.

- **Ordenador central:** Podríamos considerarlo el cerebro del vehículo. Toda la información obtenida por los sensores será tratada aquí para hacer uso de esos datos y realizar la comunicación entre distintos componentes. Existe hardware específicamente creado para vehículos autónomos, como la Nvidia Drive PX que mencionábamos en la introducción.

Otro de los aspectos más importantes a tratar de la conducción autónoma es la seguridad de estos vehículos. Si bien aún no están en una fase final, se prevé que van a ser mucho más seguros que los vehículos conducidos por personas. En un estudio [9] realizado por la NHTSA en el que se estudiaron más de 2 millones de accidentes de tráfico en EE.UU. se atribuyen casi un 95 % de estos a errores del conductor, como vemos en la Figura 2.3. Conseguir vehículos autónomos que conduzcan adecuadamente podría reducir la tasa de mortalidad en gran medida. Hasta el día de hoy, solo se han registrado 5 accidentes mortales con vehículos autónomos[1].

Causa principal del accidente	Estimado	
	Número de accidentes	Porcentaje + Intervalo de Confianza
Conductor	2,046,000	94% ± 2.2%
Vehículo	44,000	2% ± 0.7%
Entorno	52,000	2% ± 1.3%
Razones desconocidas	47,000	2% ± 1.4%
Total	2,189,000	100%

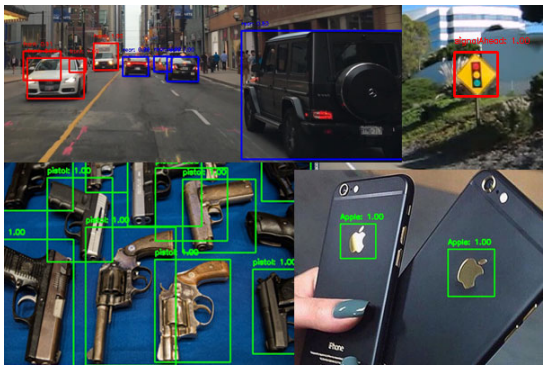
Figura 2.3: Datos del estudio realizado en 2015 por la NHTSA, mostrando el principal motivo de accidentes de tráfico en más de 2 millones de accidentes en EE.UU.

2.2. Visión por computador

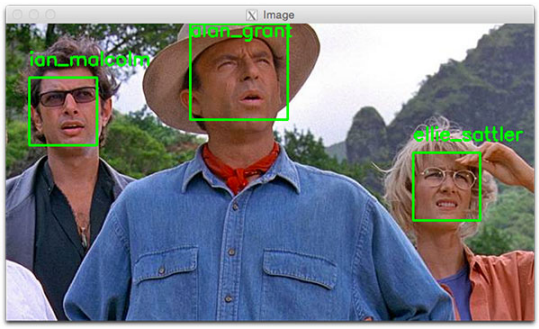
Como humanos, somos capaces de percibir las tres dimensiones del mundo que nos rodea con facilidad. Simplemente mirando a nuestro alrededor podemos distinguir perfectamente objetos, sombras, colores, cómo afectan las distintas fuentes de luz a los objetos que nos rodean, e incluso distinguir personas que conocemos en un grupo muy grande de gente. Estas tareas que nos resultan tan triviales y cotidianas, no lo son tanto para un ordenador.

La misión de la *Visión por computador* o *Visión Artificial* es hacer posible que un ordenador interprete imágenes de la misma forma que lo hacemos los seres humanos. Gracias a las investigaciones llevadas a cabo en este campo, hoy en día somos capaces de realizar tareas como detección de objetos, reconocimiento facial, seguimiento del movimiento corporal o reconstrucción 3D, como vemos en la Figura 2.4.

[1] https://en.wikipedia.org/wiki/List_of_self-driving_car_fatalities

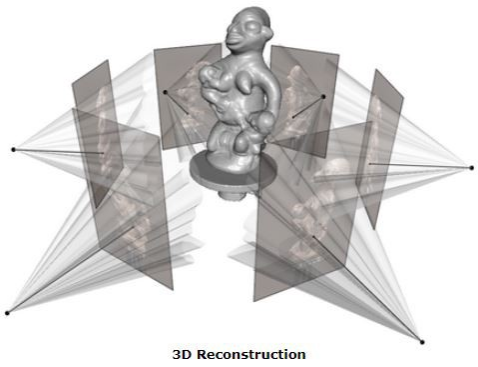


(a) Detección de objetos mediante la red neuronal YOLO.



(b) Reconocimiento facial de distintos actores de cine en la película Jurassic Park.

(c) Detección de la pose de distintas personas en una clase de baile.



(d) Reconstrucción 3D de un objeto mediante varias imágenes tomadas desde distintos puntos de vista.

Figura 2.4: Distintos ejemplos de aplicaciones reales en visión por computador.

2.2.1. Detección de objetos

Para la detección de objetos, es indispensable mencionar temas tan de actualidad como las **redes neuronales** o el aprendizaje profundo, más conocido como ***Deep Learning***, que no son más que un subcampo del aprendizaje máquina, o *machine learning*, que a su vez, es un subcampo de la inteligencia artificial. Aunque parezca mentira, estos temas tan actuales existen desde 1940 [10], aunque bajo nombres distintos como cibernética o conexionismo. Las primeras redes neuronales se inspiraron en el cerebro humano y cómo las neuronas interactúan entre sí. Hay que entender que estos modelos no pretenden emular el funcionamiento de un cerebro realisticamente, sino usarlo como inspiración para imitar su funcionamiento.

El primer modelo de red neuronal apareció en 1943 [11], esta red neuronal era un clasificador binario capaz de diferenciar dos categorías distintas basado en un input. El problema de esta red era que los *pesos* entre las neuronas usadas para clasificar debían ser asignados por un humano, lo que hacía que no fuese un método fácilmente escalable para modelos

más grandes. No fue hasta 1958 en que se inventó el algoritmo del *perceptron* [12], que se sigue usando hoy en día para hacer que estos pesos se asignasen automáticamente.

Debido a la poca capacidad de cómputo de la época, y que las redes neuronales aún estaban en un estado inicial, en la década de los 60 no se realizaron muchas investigaciones sobre el tema, al teorizarse que usando perceptrones no se podrían resolver problemas no lineares [13]. No se equivocaban al decir que faltaba capacidad de cómputo, pero gracias al algoritmo de propagación hacia atrás, o *Backpropagation* [14], creado por Werbos, Rumelhart y LeCun consiguieron volver a centrar la atención en este tema al permitir la creación de las redes neuronales prealimentadas (*feed-forward*).

Gracias a estos avances, la detección de objetos es un tema en auge, el cual se define de la siguiente manera: Dada una imagen, determinar si existen o no instancias de objetos de categorías predefinidas y si existen determinar su posición en la imagen. En algunas ocasiones, como, por ejemplo, en la competición del *dataset* ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) [15], pueden ser hasta 200 categorías sobre las que detectar.

Aunque en nuestro día a día veamos miles de objetos distintos, la comunidad científica se centra principalmente en la localización de ciertos objetos o patrones, como por ejemplo, coches, caras, bicicletas, personas, aviones o animales, y no tanto en elementos como el cielo, nubes o hierba. Para determinar su posición en la imagen, generalmente se usa un *bounding box* alineado con los ejes horizontal y vertical o segmentación a nivel de píxel, siendo la primera de estas la más popular en el campo y la técnica que usamos en el proyecto. Podemos ver ejemplos de esto en la Figura 2.5.

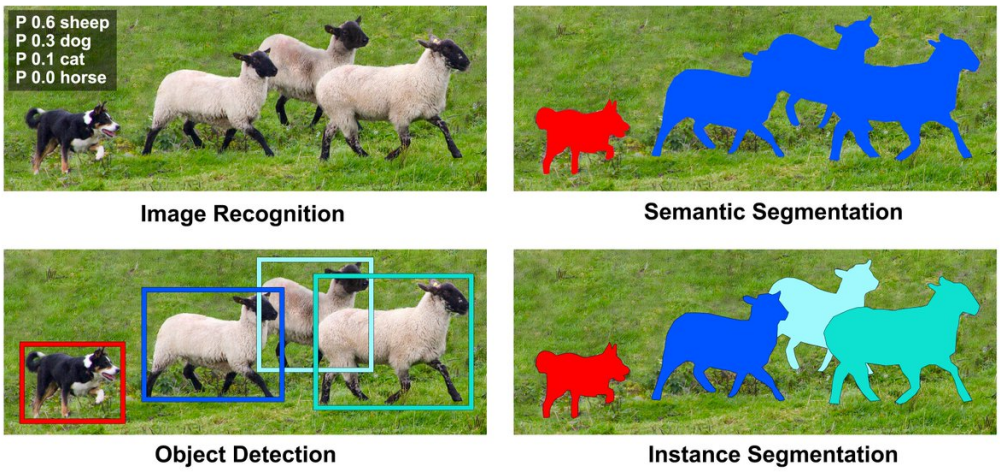


Figura 2.5: Detección de objetos a nivel de reconocimiento de imagen, detección con *bounding boxes* o segmentación a nivel de píxel.

La meta principal de la detección de objetos es obtener un algoritmo que sea capaz de detectar objetos cumpliendo con dos requisitos: gran precisión y gran eficiencia. El problema con la precisión es que una imagen y sus objetos pueden estar afectados por muchos

parámetros distintos, como iluminación, oclusión, pose, etc., y, como vemos en la Figura 2.6, debemos ser capaces de detectar los objetos en todo momento. En cuanto a la eficiencia, debemos de ser capaces de ejecutarlo a tiempo real para detectar estos objetos en *streamings* de vídeo.

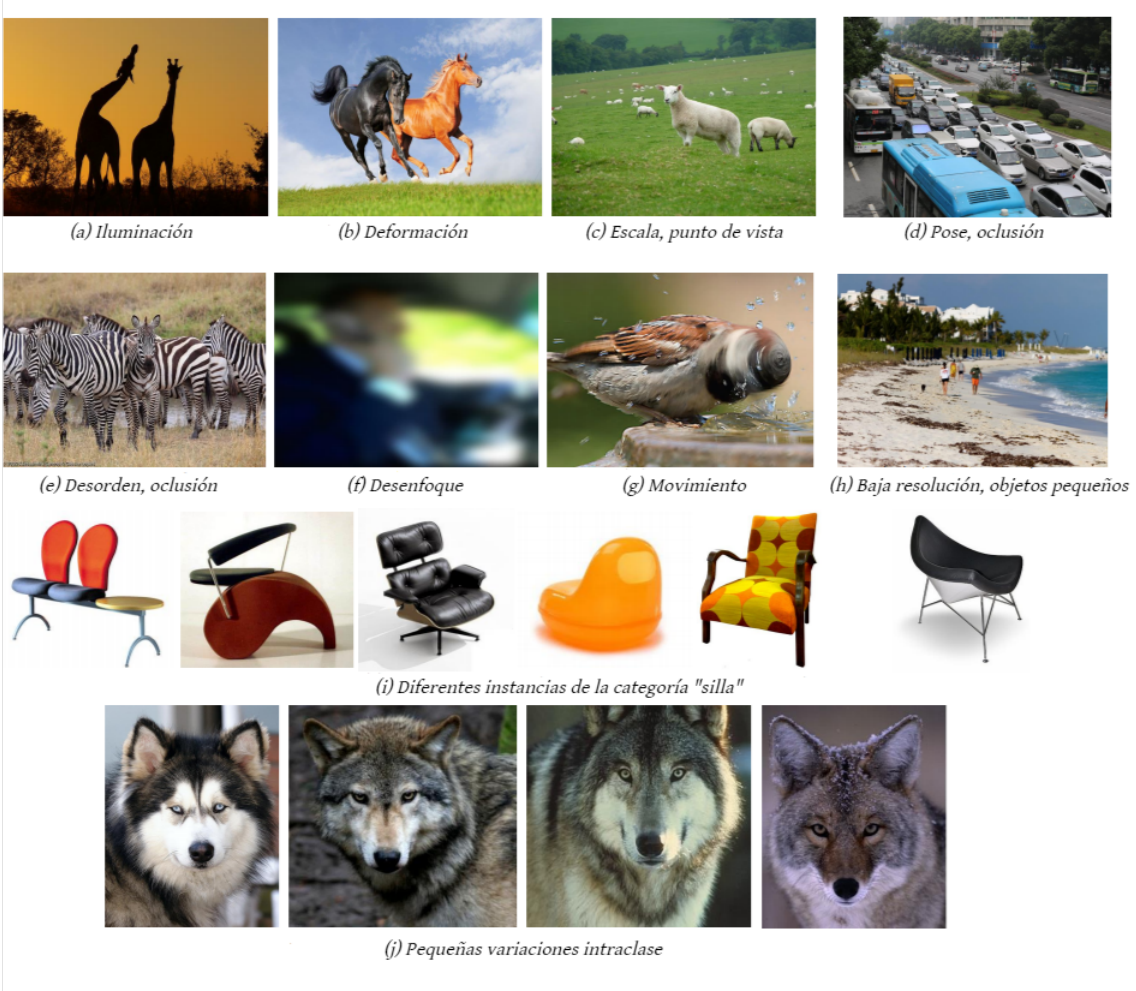


Figura 2.6: Variaciones que pueden sufrir los objetos o la misma imagen, complicando la tarea de detección [1].

Uno de los modelos más usados dentro del *Deep Learning* son las redes neuronales convolucionales, o *Convolutional Neural Networks* (CNNs), centradas sobre todo en procesar imágenes. Son un tipo especializado de red neuronal para procesar datos en forma de cuadrícula, toman este nombre al realizar una operación matemática lineal conocida como convolución [16]. De forma resumida, la convolución aplica filtros a la imagen y obtiene como salida una imagen transformada de dimensión reducida. La otra operación fundamental de las CNNs es el *pooling*, el cual extrae los valores más significativos de la imagen reducida que se obtiene de la convolución. Podemos ver una representación gráfica básica

de su funcionamiento en la Figura 2.7. Estas operaciones pueden usarse decenas o cientos de veces en una misma imagen, generando varias capas de convolución.

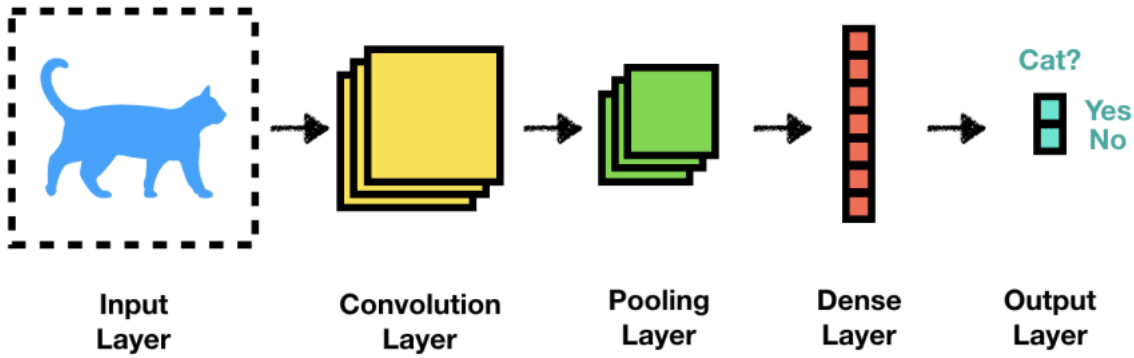


Figura 2.7: Representación gráfica básica del funcionamiento de una CNN usando una capa de convolución y otra de *pooling*.

2.2.2. Cámaras Pinhole

Las cámaras tipo ***pinhole*** o **estenopeica** son cámaras sin lente, pero con una pequeña apertura o *pinhole* y con una caja aislada de la luz al otro lado de la apertura. La luz pasa por la apertura y proyecta una imagen invertida en el lado contrario de la caja, como vemos en la Figura 2.8, este efecto es conocido como **camera obscura** o **cámara oscura**.

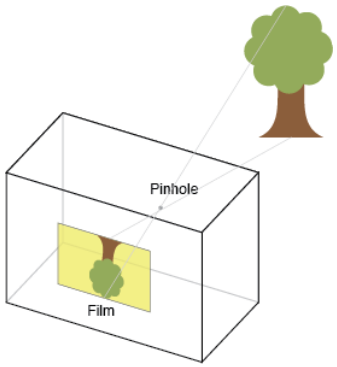


Figura 2.8: Ejemplo de proyección de una cámara tipo *pinhole* proyectando una imagen invertida de un árbol.

Existen referencias a la cámara oscura desde hace varios siglos, las primeras referencias escritas encontradas fueron en el siglo V a. C., donde Aristóteles y Euclides ya hablaban de este fenómeno. Estas cámaras constan de los siguientes elementos:

- **Plano de imagen**: Es el plano donde se genera la imagen. Se coloca detrás del centro de proyección, por lo que la imagen aparecerá invertida.
- **Centro de proyección** (*Pinhole*): Toda la luz de la escena a captar por la cámara atraviesa el centro de proyección, formando el plano de imagen.
- **Distancia Focal**: Es la distancia desde el centro de proyección de la cámara al plano de imagen.
- **Campo de Visión** (*Field of View*): Es la extensión del mundo real que la cámara es capaz de observar, también conocido como *ángulo de visión*.

Este tipo de cámaras es de las más sencillas de modelar matemáticamente, como vamos a ver a continuación.

Una cámara proyecta puntos 3D del mundo en un plano bidimensional. Como vemos en la Figura 2.9, el punto **C** es donde convergen todos los rayos de luz que entran por la apertura. Este punto es el origen de coordenadas de este sistema Euclídeo, y es conocido como el **centro de proyección**. El **plano de imagen** se encuentra delante del centro de proyección para evitar las imágenes invertidas. El plano $Z_C = f$ es el plano de imagen. El eje que atraviesa perpendicularmente el centro de proyección y el plano de imagen es conocido como el **punto principal**. Un punto 3D $\tilde{\mathbf{X}} = (X_c, Y_c, Z_c)^\top$ en el espacio Euclídeo (X_c, Y_c, Z_c) centrado en **C** se mapea a un punto $\tilde{\mathbf{x}} = (x, y)^\top$ en el plano de imagen, donde la línea que va de $\tilde{\mathbf{X}}$ a **C** intersecta con este plano [2].

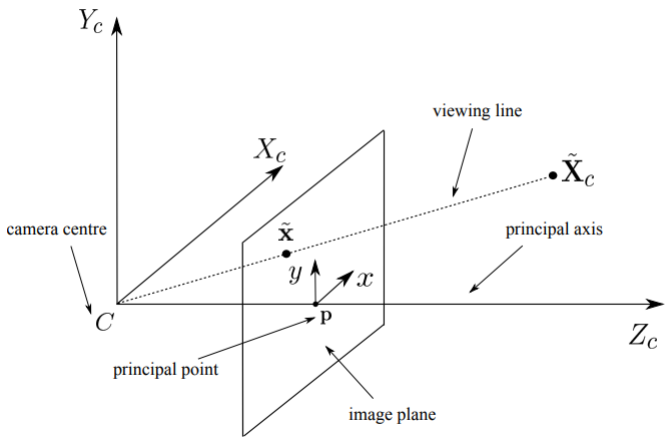


Figura 2.9: Modelo de cámara *pinhole* con un plano de imagen paralelo a (X_C, Y_C) [2].

Considerando el plano (Y_c, Z_c) que vemos en la Figura 2.10, para poder mapear los puntos 3D al plano sabemos que $y = \dfrac{fY_c}{Z_c}$ y, a su vez, en el plano (X_c, Z_c) tenemos que $x = \dfrac{fX_c}{Z_c}$.

Por tanto, el punto $(X_c, Y_c, Z_c)^\top$ es $(fX_c, fY_c, f)^\top$ para que esté en el plano de imagen. Ignorando la tercera coordenada, siendo esta la distancia focal, podemos deducir que

$$\tilde{\mathbf{X}} = (X_c, Y_c, Z_c)\top \longrightarrow \tilde{\mathbf{x}} = (x, y)^\top = \left(\frac{fX_c}{Z_c}, \frac{fY_c}{Z_c}\right)^\top \tag{2.1}$$

De esta forma mapeamos un espacio Euclídeo $\mathbb{R}^3$ a un espacio Euclídeo $\mathbb{R}^2$, realizando un proceso conocido como proyección.

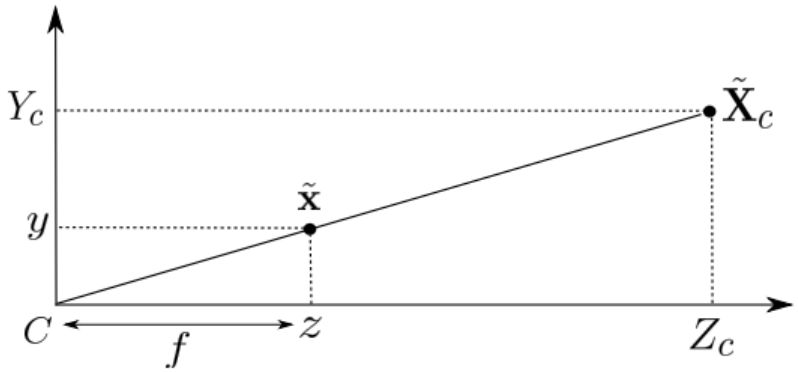


Figura 2.10: Relación entre los sistemas de coordenadas frente a la distancia focal [2].

Si los puntos 3D en el plano de imagen se representan mediante vectores homogéneos, facilitamos el cálculo de la proyección mediante un mapeado lineal de sus coordenadas homogéneas. Podemos describirlo como una multiplicación de matrices como vemos a continuación:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \longrightarrow \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.2}$$

La matriz se puede definir también como $\text{diag}(f, f, 1)[\ \text{I} \mid 0\ ]$, donde $\text{diag}(f, f, 1)$ es una matriz diagonal y $[\ \text{I} \mid 0\ ]$ representa una matriz dividida en un bloque de 3×3 (matriz identidad), y añadimos una columna del vector $\mathbf{0}$. Ahora podemos decir que $\mathbf{X}$ representa un punto en el mundo expresado mediante un vector homogéneo $(X, Y, Z, 1)^\top$, $\mathbf{x}$ para el punto en el plano de imagen mediante un vector homogéneo de 3 dimensiones y P para la **matriz de proyección de la cámara** 3×4. De esta forma, tenemos que la Ecuación (2.2) se puede escribir compactamente como

$$\mathbf{x} = P\mathbf{X}$$

siendo

$$P = diag(f, f, 1)[\text{I}|\mathbf{0}]$$

La Ecuación (2.1) asume que el origen de coordenadas del plano de imagen está en el punto principal. En teoría es así, pero en las cámaras reales existe un *offset* conocido como ***principal point offset*** que se puede mapear como

$$(X,Y,Z)^\top \longrightarrow \left(\frac{fX}{Z}+p_x,\frac{fY}{Z}+p_y\right)^\top$$

donde $(p_x,p_y)^\top$ son las coordenadas del punto principal, como vemos en la Figura 2.11. La ecuación anterior se puede expresar en coordenadas homogéneas de la siguiente forma

$$\begin{pmatrix}X\\Y\\Z\\1\end{pmatrix} \longrightarrow \begin{pmatrix}fX+Zp_x\\fY+Zp_y\\Z\end{pmatrix} = \begin{bmatrix}f & & p_x & 0\\ & f & p_y & 0\\ & & 1 & 0\end{bmatrix}\begin{pmatrix}X\\Y\\Z\\1\end{pmatrix} \tag{2.3}$$

y teniendo que

$$K = \begin{bmatrix}f & & p_x\\ & f & p_y\\ & & 1\end{bmatrix} \tag{2.4}$$

Entonces podemos decir que la Ecuación (2.3) es igual a

$$\mathbf{x} = K[\,\mathrm{I}\mid \mathbf{0}\,]\mathbf{X}_{cam} \tag{2.5}$$

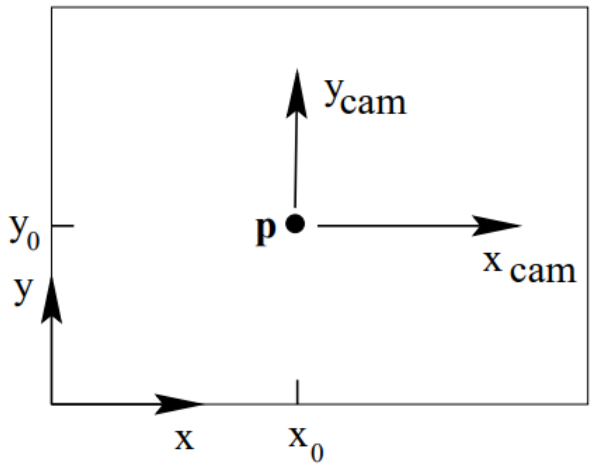


Figura 2.11: Origen del sistema de coordenadas del plano de imagen (x,y) y de la cámara$(\mathrm{x}_{cam},\mathrm{y}_{cam})$ [2].

La matriz K es conocida como *matriz de calibración* de la cámara, o **matriz de intrínsecos**. En la Ecuación (2.5) hemos escrito $(X,Y,Z,1)^\top$ como $\mathbf{X}_{cam}$ para enfatizar que la

cámara está situada en el origen de coordenadas con el punto principal de la cámara apuntando hacia el eje Z y expresando $\mathbf{X}_{cam}$ en este sistema de coordenadas, el **sistema de coordenadas de la cámara**.

Por lo general, los puntos 3D se expresarán en un sistema de coordenadas distinto al de la cámara: el **sistema de coordenadas del mundo**. Estos dos sistemas de coordenadas estarán relacionados mediante una **rotación y traslación**, como vemos en la Figura 2.12.

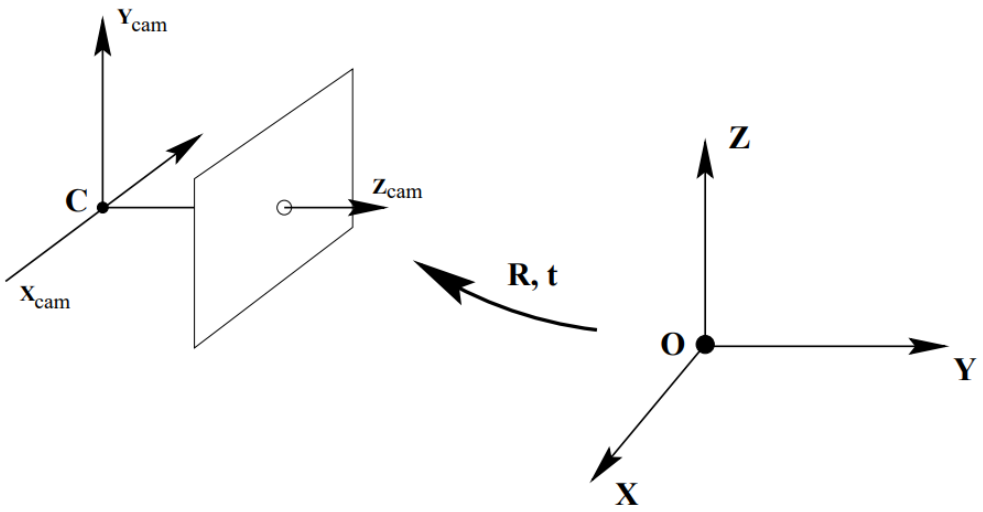


Figura 2.12: Cambio de coordenadas entre los sistemas de referencia del mundo y la cámara [2].

Teniendo un vector no homogéneo de 3 coordenadas $\tilde{\mathbf{X}}$ representando un punto en el sistema de coordenadas del mundo, $\tilde{\mathbf{X}}_{cam}$ representa el mismo punto, pero en el sistema de coordenadas de la cámara. Por ello, podemos decir que $\tilde{\mathbf{X}}_{cam} = R(\tilde{\mathbf{X}} - \tilde{\mathbf{C}})$, donde $\tilde{\mathbf{C}}$ representa las coordenadas del centro de la cámara en el sistema de coordenadas del mundo y R es una matriz de rotación 3×3 representando la orientación del sistema de coordenadas de la cámara. Esta ecuación la podemos escribir de la siguiente forma usando coordenadas homogéneas

$$\mathbf{X}_{cam} = \begin{bmatrix} R & -R\tilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} R & -R\tilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{2.6}$$

Que junto a la Ecuación (2.5) obtenemos la siguiente fórmula:

$$\mathbf{x} = KR[\,\mathrm{I}\,|\,-\tilde{\mathbf{C}}]\mathbf{X} \tag{2.7}$$

estando el punto ahora en el sistema de referencia del mundo. Como habíamos mencionado previamente, la matriz K con los parámetros relacionados a la propia cámara es la

matriz de intrínsecos. Los parámetros de R y $\tilde{C}$ son los parámetros extrínsecos. Se suele representar la transformación de cambio de coordenadas del mundo a imagen como $\tilde{\mathbf{X}}_{cam} = R\tilde{\mathbf{X}} + \mathbf{t}$. En este caso, la matriz de la cámara sería la siguiente

$$P = K[\,\mathrm{R}\,|\mathbf{t}] \tag{2.8}$$

que de la Ecuación (2.7) obtenemos que $\mathbf{t} = -\mathrm{R}\tilde{\mathbf{C}}$, siendo la matriz $[\,\mathrm{R} \mid \mathbf{t}]$ la **matriz de extrínsecos** de la cámara.

2.2.3. Calibración de cámaras

Las cámaras que utilizamos en nuestro día a día tienen cierta distorsión en las imágenes que generan, todo dependiendo tanto de la calidad de la cámara como del tipo de lente que utilice. Gracias a este tipo de calibración, podemos obtener los valores de distancia focal, centro de proyección y **coeficientes de distorsión**. La matriz de la cámara del apartado anterior no tiene en cuenta la distorsión, debido a que una cámara *pinhole* ideal no tiene lente y, por tanto, no causa distorsión. Existen dos tipos de distorsión a tener en cuenta, la distorsión **radial** y la **tangencial**.

La distorsión radial ocurre cuando los rayos de luz se curvan más en las esquinas que en el centro óptico de la lente. Cuanto más pequeña es la lente que usamos, mayor será la distorsión radial. Podemos ver cómo afectaría esta distorsión en la Figura 2.13.

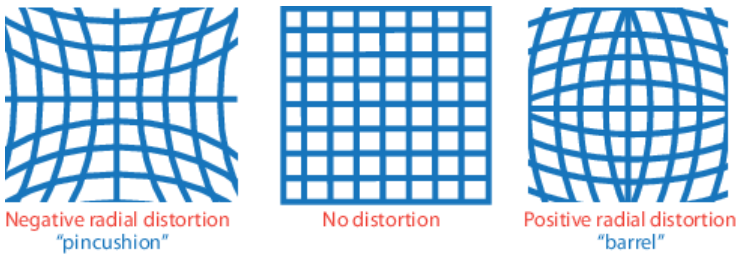


Figura 2.13: Tipos de distorsión radial que puede generar una lente en una imagen [3].

Los coeficientes de distorsión radial modelan esta distorsión y, realizando un proceso conocido como *undistortion*, podemos eliminar esta distorsión. Teniendo en cuenta que los puntos con distorsión corregida de una imagen corresponden a $(x_{\text{corrected}}, y_{\text{corrected}})$ [17].

$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

- x, y – Posición de los píxeles de la imagen con distorsión, x e y están en coordenadas normalizadas de la imagen. Las coordenadas normalizadas de la imagen se

calculan trasladando la posición de los píxeles al centro óptico, y dividiéndolo por la distancia focal.

- k_1, k_2, k_3 – Coeficientes de distorsión radial de la lente.
- $r^2 = x^2 + y^2$ [3]

Normalmente, con dos coeficientes de distorsión radial se pueden corregir las imágenes obteniendo buenos resultados, pero para algún tipo de lente, como los gran angulares es mejor calcular los 3 coeficientes.

La distorsión tangencial ocurre cuando la lente y el plano de imagen no están alineados correctamente y, por tanto, no son paralelos como podemos ver en la Figura 2.14.

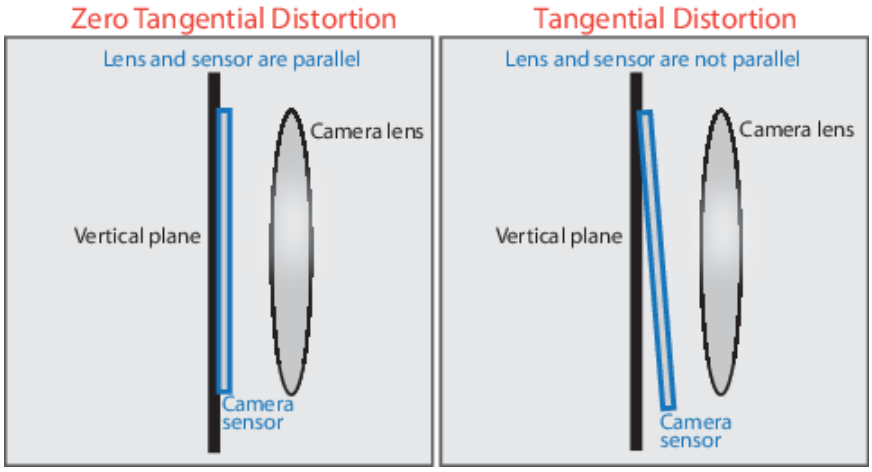


Figura 2.14: Ejemplo de una lente paralela al plano de imagen (izquierda), y una lente mal alineada causando distorsión tangencial (derecha) [3].

Al igual que en la distorsión radial, los puntos $(x_{\text{corrected}}, y_{\text{corrected}})$ corresponden a los puntos tras la corrección en la imagen

$$x_{\text{corrected}} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$
$$y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

- p_1 y p_2 – Coeficientes de distorsión tangencial de la lente.

Por lo que tenemos 5 coeficientes de distorsión distintos

$$\text{Coeficientes de distorsión} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Hoy en día, la técnica de calibración más popular para obtener estos coeficientes se basa en el uso de marcadores con formas, como la de un tablero de ajedrez, o patrones similares

en blanco y negro, como vemos en la Figura 2.15. Podemos ver una explicación más detallada de cómo se obtienen los coeficientes de distorsión en [18]. Como explicación breve, sabiendo la posición en el mundo real de algunos de los puntos de los marcadores como, por ejemplo, las esquinas interiores de los cuadrados, y conociendo su posición en el plano de imagen, podemos obtener matemáticamente estos coeficientes de distorsión. Normalmente hay que sacar varias fotos o una secuencia de vídeo en el que vayamos rotando y moviendo el marcador, a la vez que lo acercamos y alejamos de la cámara.

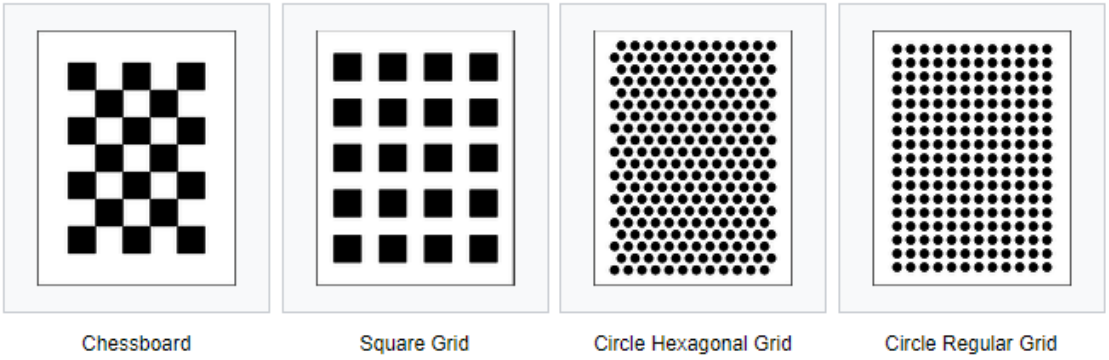


Figura 2.15: Varios patrones de calibración para emplear en calibración de cámaras.

2.3. Reproyección 2D-3D

En esta sección vamos a mencionar métodos y técnicas del estado del arte relacionados con la reproyección de objetos.

Primero, **detección de objetos 2D**, la detección de *bounding boxes* 2D en imágenes ha sido profundamente estudiada, y hay propuestas que destacan hasta en los *datasets* más modernos [19, 20, 21, 22]. Los métodos de detección existentes se pueden separar en dos categorías: detectores *single stage*, como YOLO [23], SSD [24] o RetinaNet, [25] o *two stage*, como Faster R-CNN [26] o FPN [27].

El método más usado en detección autónoma para analizar los objetos del entorno es la **detección de objetos 3D con LiDAR**. Este método es el más utilizado en la actualidad, debido a sus buenos resultados y la precisión de estos dispositivos. La mayoría de métodos difiere en cómo trabaja con las nubes de puntos (Figura 2.16) generadas por el LiDAR.

El Frustum-PointNet [28] o el trabajo de [29], opera directamente en la nube de puntos. Otras propuestas como [30, 31] proyectan la nube de puntos en la imagen 2D y aplican arquitecturas como las de Faster R-CNN. Por último, también se ha estudiado el discretizar la nube de puntos a vista de pájaro y trabajar con la intensidad de los puntos en ese plano como en [32, 33].

Hay métodos que usan cámaras estéreo para trabajar con la **profundidad** en las imágenes

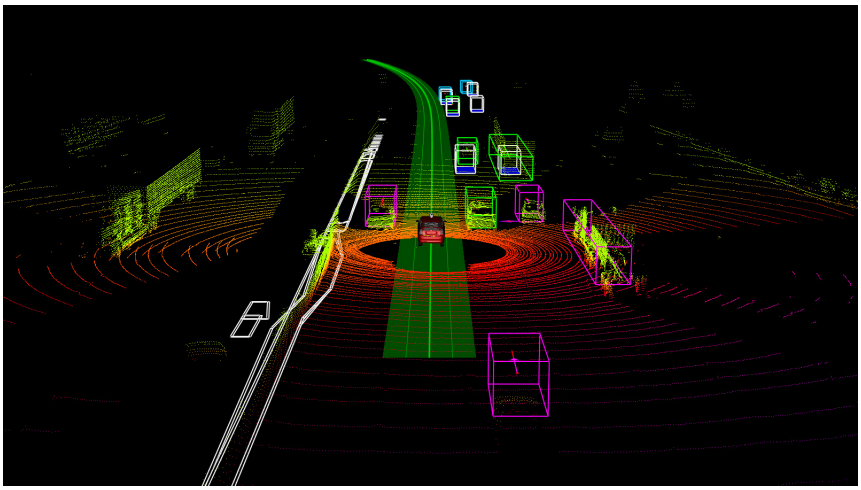


Figura 2.16: Proyección de una nube de puntos generada por un LiDAR en un entorno virtual.

obtenidas por las cámaras [34, 35, 36]. Las cámaras que van a ser usadas en este proyecto son monoculares y, por tanto, no se puede determinar con exactitud la profundidad de la imagen. Aun así, ya están siendo estudiados métodos **MDE** *(Monocular Depth Estimation)*, como podemos ver en estas investigaciones. [37, 38, 39] .

3. CAPÍTULO

Desarrollo del proyecto

En este apartado vamos a explicar los componentes del vehículo de testeo de Vicomtech, de aquí en adelante **CarLOTA** (*Car Learn On-road, Think Autonomous*), que vemos en la Figura 3.1, los pasos seguidos para llegar al estado final del proyecto y los diagramas creados para el funcionamiento de los componentes del CarLOTA.

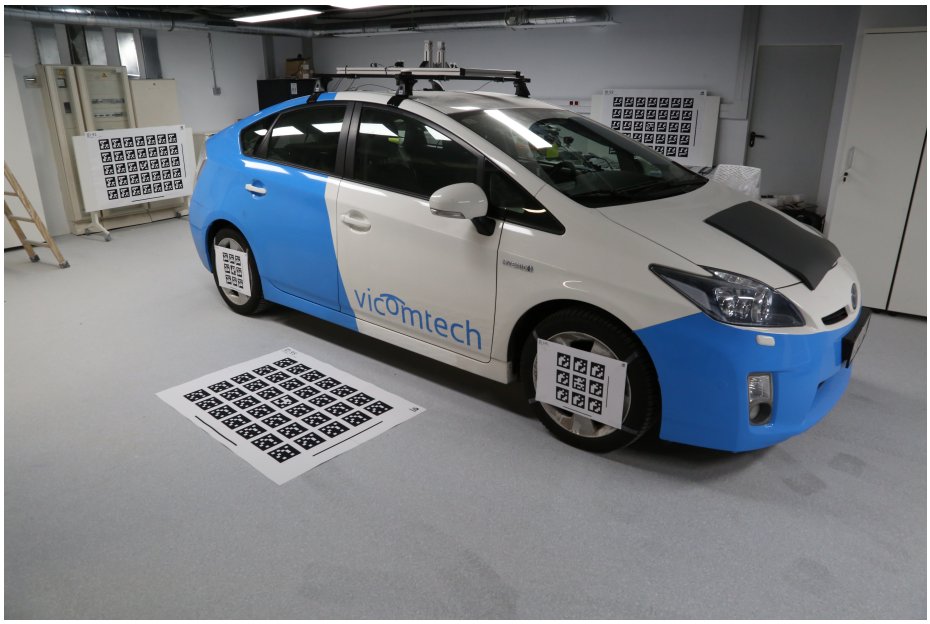


Figura 3.1: Foto de CarLOTA en el garaje de Vicomtech, con paneles de calibración a su alrededor.

3.1. CarLOTA

CarLOTA es el vehículo de testeo de Vicomtech, es un Toyota Prius híbrido de 3º generación (2009-2015). El vehículo está modificado internamente con el fin de tener espacio para los componentes que se listan a continuación:

- **Nvidia Drive PX2 AutoChauffeur**: Es una plataforma de computación diseñada específicamente para automóviles, con el fin de manejar procesos de *Deep Learning*, inteligencia artificial o lectura de varios *streamings* de vídeo y datos. Dispone de dos Tegra X2 SoC (Tegra A y Tegra B) con 2 núcleos Denver, 4 núcleos ARM A57 y una GPU de la generación Pascal en cada Tegra. Estas Tegras funcionan como ordenadores independientes para llevar a cabo tareas distintas, y no con el fin de paralelizar procesos. Consta de conectores como GMSL, Ethernet, CAN, USB o FlexRay para usarlos en sensores como GPS, LiDAR, señales de bus CAN, etc. El sistema operativo se basa en la distribución Ubuntu Linux, podemos ver su estructura en la Figura 3.2.

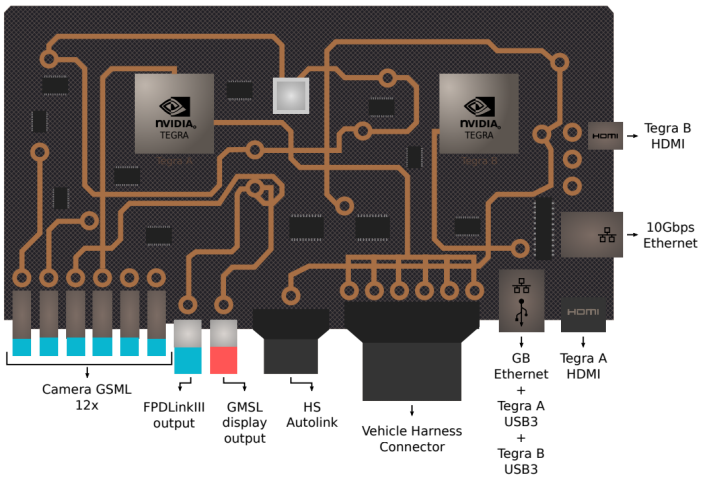


Figura 3.2: Nvidia Drive PX 2 AutoChauffeur con sus distintos conectores.

- **SIMULADOR-PROC** : Este ordenador consta de un procesador I9-7940X, 64GB de RAM y 2 Nvidia 2080 Ti de 11GB. Es el ordenador donde se realizarán la mayoría de procesos necesarios para el funcionamiento de los distintos componentes del vehículo. Usando los *streaming* de vídeos que se obtienen de la NVIDIA Drive PX2, se realizará la detección de objetos, seguimiento, reproyección, localización, DMS (*Driving Monitoring System*), etc.

- **Portátil para HMI**: CarLOTA dispone de un portátil Xiaomi Mi Air para controlar el HMI (*Human Machine Interface*). Este hace uso de unas tiras LED que indican al usuario cuándo puede cambiar de carril con seguridad.

- **4 cámaras Sekonix.**: Las 4 cámaras utilizadas son de la gama Sekonix, recomendada por Nvidia para su uso junto a la Drive Px 2. En nuestro caso, son el modelo SF3322 AR0231 con 100º FoV y una resolución de 1920×1208 píxeles. Son cámaras pequeñas de 30 x 30 x 20.2 mm y pesan 45 gramos. Han sido posicionadas de forma que cada cámara observa una zona del vehículo: parte frontal, trasera, izquierda y derecha.
- **1 U-Blox EVK-M8T**: Es un GNSS (*Global Navigation Satellite System*) pensado para automoción. RTMaps (Anexo B) consta de componentes para tratar los datos de dispositivos de la marca U-Blox. Tiene una precisión de unos pocos metros, dependiendo de la situación en la que se encuentre.
- **1 OXTS XNAV 550 para DGPS**: Un DGPS (*Differential Global Positioning System*) es un sistema de localización con correcciones a tiempo real, capaz de dar la precisión del dispositivo con un error de ± 1-3 cm.
- **1 Intel RealSense D415**: Es una cámara capaz de grabar RGB, infrarrojos y profundidad de la escena. Es usada para el DMS (*Driver Monitoring System*) y para comprobar que el conductor no se queda dormido o no presta atención a la carretera. Es capaz de grabar a 30 fps con una resolución de 1920×1080.

3.2. Proceso de calibración

Un proceso muy importante para el correcto funcionamiento de la reproyección ha sido la labor de calibración de las 4 cámaras Sekonix de CarLOTA. Existen varias herramientas para la calibración de cámaras pero, debido a que pretendemos calibrar varias cámaras al mismo tiempo y que trabajamos con una Nvidia Drive PX 2, usaremos la propia herramienta de calibración de Nvidia.

La versión de Nvidia Driveworks que se usa en nuestra Nvidia Drive PX 2 es la 0.6, esta versión tiene una herramienta de calibración que ha sido mejorada en versiones posteriores. Como hemos mencionado previamente, todo lo relacionado a CarLOTA usando la Nvidia Drive PX 2 está en la versión 0.6, por lo que hemos usado la herramienta de calibración de la versión 1.2 de Driveworks en un ordenador distinto para obtener tanto las matrices de intrínsecos y extrínsecos como los coeficientes de distorsión.

Para usar esta herramienta de calibración necesitamos N cámaras que vayan en el coche y una cámara externa con una lente con distancia focal fija con, por lo menos, 12 MP. En nuestro caso hemos usado las 4 cámaras Sekonix que hemos mencionado previamente, y las cuales podemos ver en la Figura 3.3, además de una cámara reflex Canon EOS 5D con un objetivo 24-105 mm fijo en 24 mm.

Figura 3.3: Dos de las cámaras Sekonix instaladas en el coche.

La herramienta de calibración de Nvidia consta de más de 200 marcadores con patrones e identificadores distintos, podemos ver dos ejemplos en la Figura 3.4. Estos marcadores debemos colocarlos cumpliendo las siguientes reglas:

- Cada cámara que va a ser calibrada tiene que ver, por lo menos, un marcador de tamaño mínimo de 1×1 metros.
- El marcador que ve cada cámara ha de cubrir el máximo ángulo de visión posible.
- Hay que colocar un marcador de tamaño A3, centrado lo mejor posible en cada rueda del vehículo.
- Se pueden colocar marcadores de 1×1 metros o de mayor tamaño en el suelo en las zonas laterales del coche, pero es opcional.
- No se pueden usar dos marcadores con el mismo identificador en una misma calibración.
- Es necesario conocer la medida de las líneas horizontales y verticales de los marcadores que se ven en la Figura 3.4.

Los marcadores con identificadores que van del 180-183 son específicos para ser colocados en las ruedas. También es útil iluminar la escena lo mejor posible, y que los marcadores estén totalmente planos.

Para nuestro proceso de calibración usamos 6 marcadores de 1×1 metros, uno para cada cámara más dos marcadores en el suelo, en los laterales del coche. También usamos 4 marcadores de tamaño A3 en las ruedas, como podemos ver en la Figura 3.1.

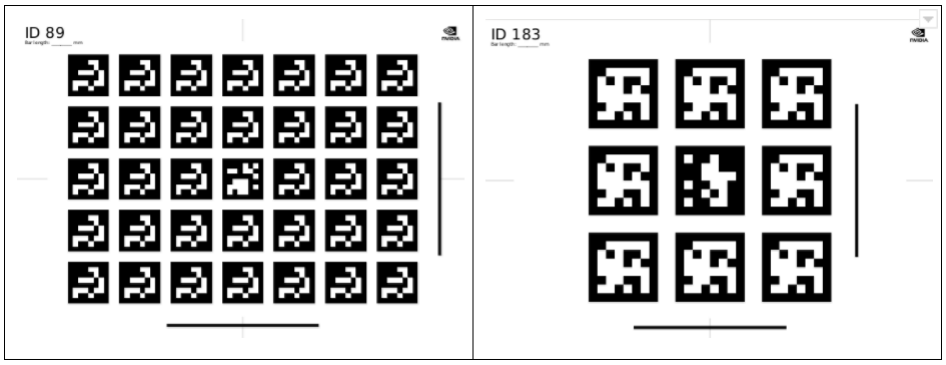


Figura 3.4: Dos marcadores de Nvidia para su herramienta de calibración con los identificadores 89 y 183.

Una vez tenemos todos los marcadores preparados, deberemos sacar al menos 20 fotos en las que se vea más de un marcador, para que la herramienta pueda calcular la matriz de extrínsecos a usar. Estas fotos serán como las de la Figura 3.5.

Figura 3.5: Dos fotos de ejemplo en el que se muestran dos marcadores para la calibración de extrínsecos de las cámaras.

Además de esto, deberemos realizar una foto con cada cámara Sekonix del vehículo a un marcador y un vídeo por cada cámara en el que movamos un marcador. Este puede ser uno de los de Nvidia o también del tipo tablero de ajedrez, e ir rotándolo mientras lo movemos por el campo de visión de la cámara.

Una vez tengamos tanto los vídeos como las fotos preparadas, deberemos crear un directorio con la estructura que vemos en la Figura 3.6. Explicaremos sus elementos a continuación:

- **targets.json**: Un archivo que nos aporta Nvidia para la calibración en el que deberemos indicar las medidas de los marcadores que usamos.
- **special-targets.json**: En este archivo indicaremos qué identificadores tienen los marcadores que usamos para el suelo y para las ruedas. Debemos especificar tam-

```
<calib-data-path>
|-- targets.json                    The target database with the measured bar lengths
|-- special-targets.json            List all special targets in this file
|-- intrinsics                      Constraints generated using intrinsic tool
|   |-- <camera-0>.json
|   |-- ...
|   |-- <camera-N>.json
|
|-- extrinsics                      Images captured by car cameras
|   |-- <camera-0>.[png/jpg]        NOTE: names of the files must match the .json
|   |-- ...                         file names from the intrinsic/ folder
|   |-- <camera-N>.[png/jpg]
|
|-- external                        Images captured by external camera
|   |-- <image-0>.[png/jpg/JPG]     NOTE: names of the images are irrelevant
|   |-- ...
|   |-- <image-M>.[png/jpg/JPG]
```

Figura 3.6: Estructura que tiene que tener el directorio para la calibración de Nvidia.

bién en qué rueda está cada marcador, para que la herramienta pueda obtener los resultados correctamente.

- **intrinsics**: Utilizando los vídeos que hemos grabado previamente para cada cámara instalada en el coche, Nvidia calcula unos ficheros JSON que deberemos insertar en esta carpeta.
- **extrinsics**: Con cada cámara del coche deberemos sacar una foto al marcador que observa, e introducir la foto en esta carpeta.
- **external**: Todas las fotos realizadas con la cámara externa en la que se muestran varios marcadores deberán estar en esta carpeta.

Una vez tengamos todo preparado y ejecutemos la herramienta de calibración, obtendremos imágenes de validación como las que vemos en la Figura 3.7 y 3.8.

Además de esto, obtendremos un fichero JSON con los parámetros intrínsecos, extrínsecos y los coeficientes de distorsión con el formato que vemos a continuación:

```
{
"rig": {
        "sensors": [
            {
                "correction_rig_T": [
                    0.0,
                    0.0,
                    0.0
                ],
                "correction_sensor_R_FLU": {
                    "roll-pitch-yaw": [
                        0.0,
                        -8.19172569777038e-08,
                        0.0
```

```
                    ]
                },
                "name": "camera-1",
                "nominalSensor2Rig_FLU": {
                    "roll-pitch-yaw": [
                        0.731416583061218,
                        3.13823938369751,
                        90.2723922729492
                    ],
                    "t": [
                        1.02196669578552,
                        0.170759424567223,
                        1.61348462104797
                    ]
                },
                "parameter": "",
                "properties": {
                    "Model": "pinhole",
                    "cx": "960.00000",
                    "cy": "604.00000",
                    "distortion": "-4.14278566837311e-1 2.37353309988976e-1 0.000000000000000 ",
                    "fx": "1220.7068",
                    "fy": "1216.7816",
                    "height": "1208",
                    "width": "1920"
                },
                "protocol": "camera"
            },
            {
                "correction_rig_T": [
                    0.0,
                    0.0,
                    0.0
                ],
                "correction_sensor_R_FLU": {
                    "roll-pitch-yaw": [
                        1.66752667229986e-09,
                        -2.13443414054382e-07,
                        0.0
                    ]
                }
            }
        }
    }
}
```

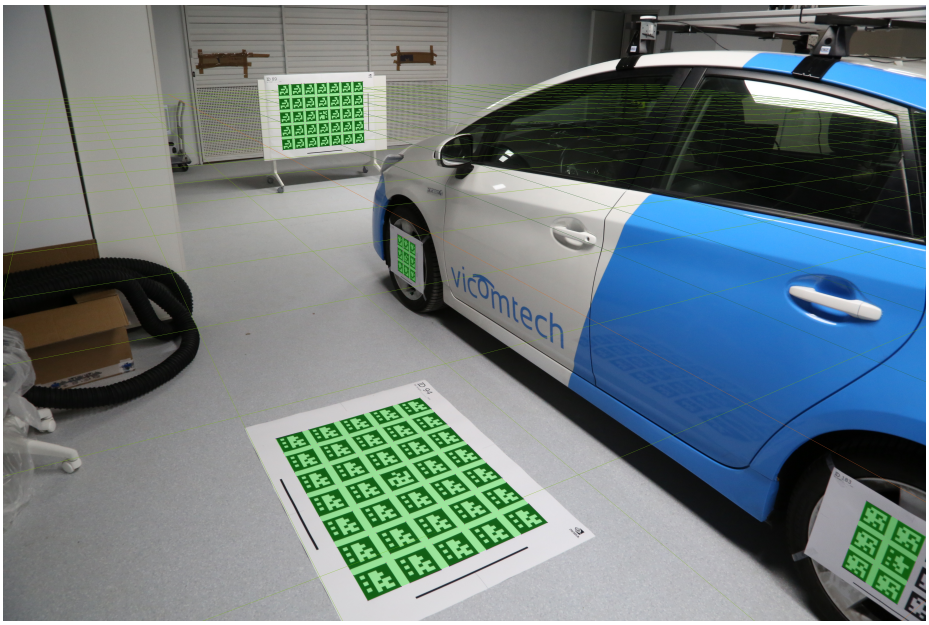


Figura 3.7: Imagen de validación de la herramienta de Nvidia, cuanto mejor esté alineado el plano verde con los marcadores, más precisa habrá sido la calibración.

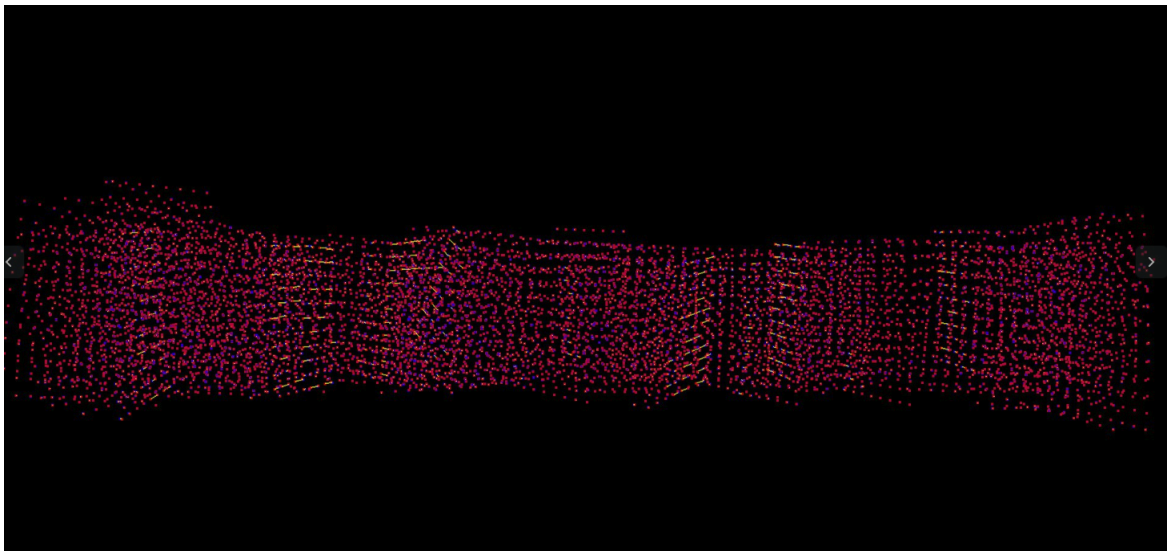


Figura 3.8: Imagen de validación para los parámetros intrínsecos de la herramienta de Nvidia, los puntos azules son los puntos detectados del marcador y los rojos los reproyectados. Las líneas amarillas unen estos puntos, por lo que, si hay una línea amarilla muy grande, el error de reproyección ha sido alto, mientras que si la línea no se percibe, la reproyección ha sido precisa.

Los parámetros en los que nos debemos fijar del fichero .json anterior son los siguientes:

- **cx/cy**: Coordenadas del centro de proyección del plano de imagen.
- **distortion**: Coeficientes de distorsión de la lente.
- **fx/fy**: Distancia focal en el eje x y en el eje y de la cámara.
- **height/width**: Tamaño del plano de imagen que captura la cámara.
- **roll-pitch-yaw** (*Nominal Sensor*): Vector de rotación indicando hacia dónde apunta la cámara con respecto al eje de coordenadas del mundo.
- **t**: Vector de translación de la cámara representado en metros con respecto al eje de coordenadas del mundo.

Debemos de tener en cuenta que Nvidia usa como origen del centro de coordenadas del mundo el punto central del eje de las ruedas traseras del vehículo. El eje X apuntando hacia la parte delantera del vehículo, el eje Y hacia la izquierda y el eje Z hacia arriba. Tras varias calibraciones y correcciones, los mejores parámetros que hemos obtenido son los que vemos en la Tabla 3.1.

	Front	Rear	Left	Right
Fx	1202.82 px	1222.20 px	1220.70 px	1222.09 px
Fy	1215.39 px	1217.89 px	1216.78 px	1218.55 px
Cx	960 px	960 px	960 px	960 px
Cy	604 px	604 px	604 px	604 px
Distortion	-0.412, 0.248, 0	-0.425, 0.28, 0	-0.414, 0.237, 0	-0.431, 0.292, 0
Roll (X)	0.00º	2.22º	0.73º	-0.41º
Pitch (Y)	1.00º	3.50º	4.13º	5.03º
Yaw (Z)	0.00º	-179.74º	90.27º	-90.54º
Tx	2.014 m	0.45 m	1.02 m	1.08 m
Ty	-0.04 m	0.00 m	0.34 m	-0.34 m
Tz	1.29 m	1.69 m	1.61 m	1.63 m

Tabla 3.1: Resultados de los parámetros intrínsecos y extrínsecos de la herramienta de calibración de Nvidia

Con los resultados obtenidos, y teniendo en cuenta que el origen de coordenadas del mundo está en el punto central del eje trasero del coche, podemos comprobar que los vectores de translación para las posiciones de las cámaras son correctos simulando su posición en Prescan[1] como vemos en la Figura 3.9.

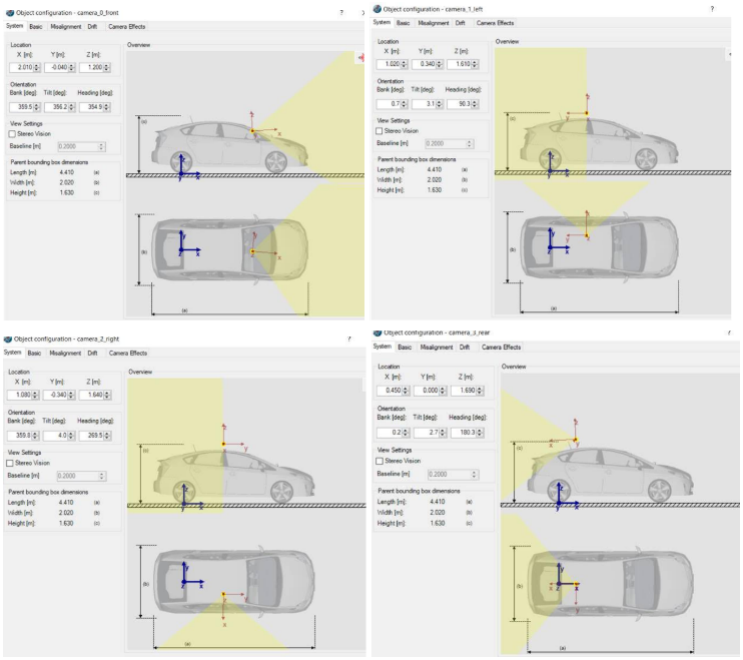


Figura 3.9: Posición de las cámaras obtenida por la herramienta de calibración de Nvidia frente a un modelo de Toyota Prius simulado en Prescan.

Podemos comparar la simulación de Prescan con una imagen real de las posiciones de las cámaras, como la de la Figura 3.10, y ver que es correcto.

3.3. Detección de objetos

Para la detección de objetos, se han estudiado distintas alternativas del estado del arte de métodos de *Deep Learning*, incluyendo DetectNet, YOLO V3, TensorRT, OpenVINO SSD, etc. para encontrar un modelo que cumpla con la precisión, detección y rendimiento necesarios para funcionar en tiempo real con 4 cámaras. De las alternativas investigadas, las dos mejores opciones han sido las siguientes:

- El modelo de red neuronal DetectNet ejecutándose en la Nvidia Drive Px 2.
- La red neuronal YOLO v3 ejecutándose en el SIMULADOR-PROC.

[1] https://tass.plm.automation.siemens.com/prescan

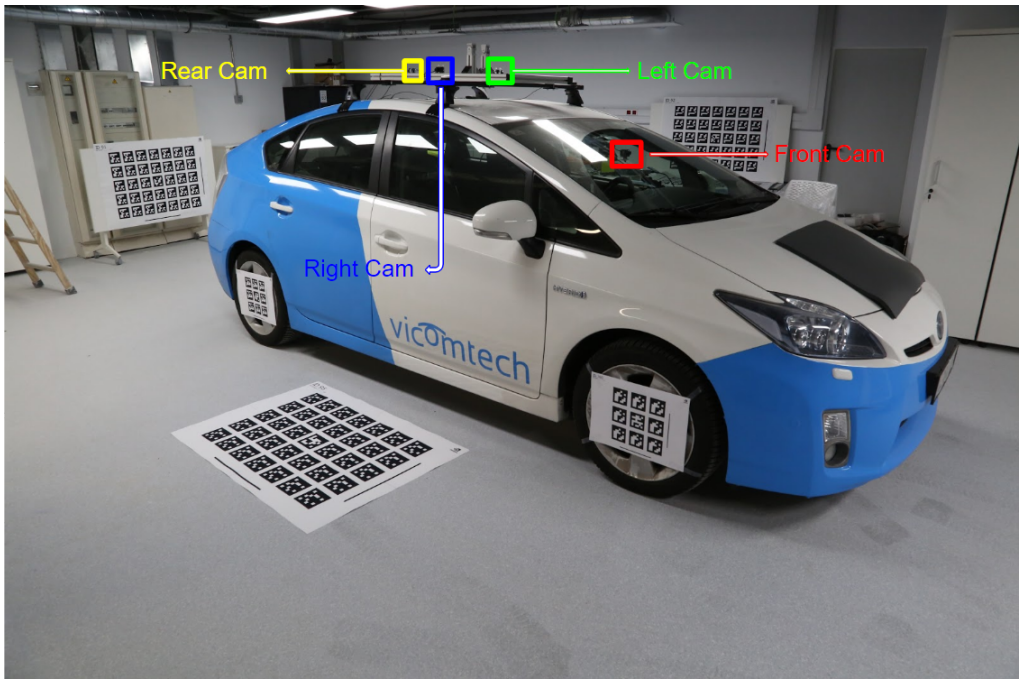


Figura 3.10: Posición de las 4 cámaras Sekonix en CarLOTA.

El modelo DetectNet se ejecuta nativamente en la Nvidia Drive Px 2 usando la librería de Nvidia Driveworks, ya que es el modelo que viene por defecto con Nvidia procesando las imágenes RCCB obtenidas por las cámaras Sekonix. De esta forma, la Nvidia Drive Px 2 realiza tanto la tarea de captura de imágenes como detección, enviando los resultados de las detecciones vía UDP al SIMULADOR-PROC, donde se ejecutan los demás componentes (reproyección, tracking, etc.). De esta forma, somos capaces de ejecutar las 4 cámaras a 30 fotogramas por segundo, obteniendo las detecciones. El problema con el modelo de DetectNet es que, bajo mala iluminación o lluvia, su precisión se ve reducida.

La segunda opción, YOLO v3 tiene una precisión mayor a la de DetectNet y es más veloz que otras opciones del estado del arte, incluso llegando a cuadruplicar en velocidad a otros modelos [40]. Este modelo ha sido publicado recientemente y es capaz de funcionar incluso en condiciones climatológicas adversas. Sin embargo, uno de sus problemas es el optimizar la inferencia sobre los 4 *streams* de vídeo bajo RTMaps. Por ello, se han realizado las siguientes acciones:

- Preprocesar los *streams* usando la librería de CUDA.
- Preparar un modelo optimizado de inferencia, usando TensorRT sobre los 4 *streams* de vídeo simultáneamente y paralelizando el proceso en la tarjeta gráfica.
- Post-procesado de los *streams* en RTMaps.

De esta forma, somos capaces de ejecutar el modelo a tiempo real a 40 fotogramas por segundo. Podemos ver los 4 *streams* de vídeo ejecutándose a tiempo real en la Figura 3.11.

Figura 3.11: Detección de objetos a tiempo real en RTMaps usando YOLO v3 en CarLOTA.

3.4. Método de reproyección 2D-3D

Ahora que tenemos las detecciones de los vehículos y los parámetros de calibración, podemos realizar la reproyección de los puntos 2D de la imagen a un entorno tridimensional. Para ello, teniendo un modelo de cámara tipo *pinhole* a la que hemos suprimido la distorsión de la lente, gracias a sus coeficientes de distorsión, podemos interpretar un punto en el plano de imagen como un rayo que pasa por el centro óptico de la cámara. Este rayo es una línea en el espacio 3D, que se puede intersectar con el plano del suelo y obtener así un punto en el sistema de coordenadas del mundo. Para ello, si estamos interesados en obtener la distancia a la que se encuentra un vehículo detectado por el modelo de YOLO v3, deberemos reproyectar un punto de la línea horizontal inferior de la *bounding box* detectada, ya que este estará en el plano del suelo. Podemos ver en la Figura 3.12 gráficamente el proceso a realizar.

Teniendo en cuenta que el sistema de coordenadas de la cámara es distinto al del mundo, ya que para la cámara el eje Z apunta hacia delante, el eje X hacia la derecha y el eje Y

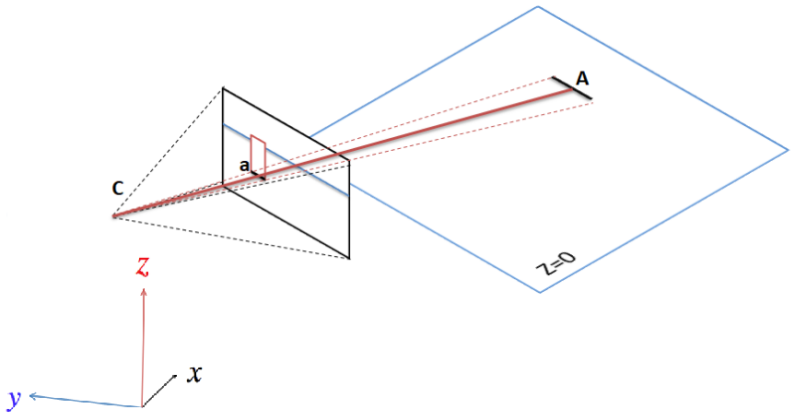


Figura 3.12: Explicación gráfica del proceso de reproyección a realizar para estimar la distancia de una detección junto a los ejes de coordenadas del mundo.

hacia abajo, crearemos la matriz T para poder convertir de un sistema a otro

$$T = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.1}$$

Considerando un punto tridimensional X proyectado en un plano de imagen como un punto bidimensional x

$$x = KT[R|t]X \tag{3.2}$$

entonces, podemos calibrar las coordenadas multiplicando por la izquierda $T^{-1}K^{-1}$.

$$T^{-1}K^{-1}x = T^{-1}K^{-1}KT[R|t]X = [R|t]X \tag{3.3}$$

Así obtendremos el rayo 3D expresado en el sistema de coordenadas de la cámara $\mathbf{r} = T^{-1}K^{-1}x$. La intersección del rayo $\mathbf{r} = (r_x, r_y, r_z)^t$ y el plano del suelo se puede obtener usando las coordenadas de Plücker para la línea y el plano (el plano tiene que estar expresado en el sistema de coordenadas de la cámara)

$$X' = \mathbf{L}\pi \tag{3.4}$$

donde $\pi = (a,b,c,d)^\top$ es la expresión del plano y $\mathbf{L}$ es el rayo en coordenadas de Plücker

$$\mathbf{L} = P_1P_2^t - P_2P_1^t \tag{3.5}$$

siendo $P_1 = (0,0,0,1)^t$ y $P_2 = (r_x, r_y, r_z, 1)^\top$ obtendremos X', un punto 3D homogéneo 4×1 que será la reproyección de x en el plano del suelo $Z = 0$.

Gracias a este método, podemos reproyectar los objetos que estén apoyados en el suelo, los objetos que estén por encima de la línea del horizonte no se podrán reproyectar correctamente, ya que no colisionarán debidamente con el plano del suelo. Para poder visualizar las reproyecciones, hacemos uso del visualizador 3D de RTMaps y mostramos a tiempo real los resultados que obtenemos de reproyectar el punto central inferior de las *bounding boxes*, como vemos en la Figura 3.13.

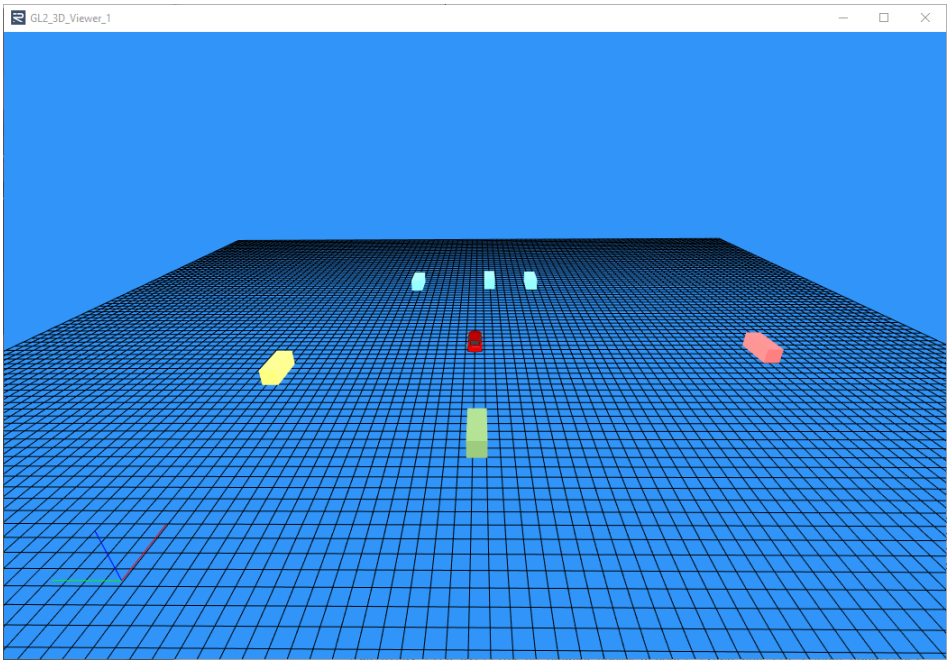


Figura 3.13: Visualizador 3D de RTMaps, mostrando la posición de las reproyecciones hechas como cubos rectangulares. El color azul es para las detecciones de la cámara frontal, amarillo para las izquierda, rojo para la derecha y verde para las trasera.

El ángulo de visión horizontal de cada cámara, como hemos mencionado previamente, es de 100º por lo que, como podemos ver en la Figura 3.14, existen regiones de solapamiento entre las cámaras laterales con las cámaras frontal y trasera. A la hora de reproyectar los vehículos detectados, deberemos tener en cuenta estas regiones para no tener reproyecciones duplicadas.

Por otro lado, cuando reproyectamos los vehículos, estamos siempre reproyectando la parte del vehículo más cercana a la cámara. Un coche estándar suele tener una longitud cercana a 4.5 metros y una anchura de unos 2 metros. Por ello, cuando las detecciones las realizamos desde la cámara frontal o trasera, aplicamos una corrección de ± 2 metros y en las cámaras laterales de ± 1 metro para acercarnos más al punto central del coche y reproyectar con mayor exactitud. Esta es una de las razones por las que en las regiones diagonales de las cámaras el error aumentará.

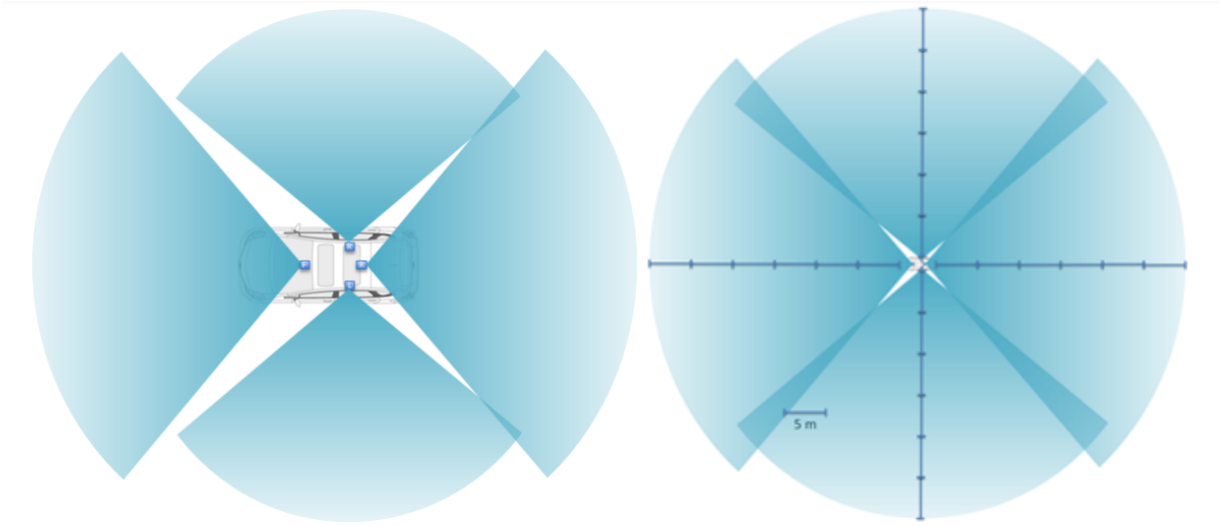


Figura 3.14: Campo de visión de cada cámara Sekonix de CarLOTA, mostrando las regiones de solapamiento entre cámaras a 30 metros.

3.5. Tracking

Para el *tracking* de las detecciones entre cámaras, el propio detector de YOLO asigna identificadores a cada *bounding box* nuevo que aparece en la imagen. La red neuronal que usa Vicomtech ha sido previamente tratada para que este *tracking* funcione también en las regiones de solapamiento entre cámaras, realizando así un *tracking* 2D a las imágenes. De esta manera, si a un *bounding box* detectado por la cámara frontal se le asigna un $id = 12$, y esta detección desaparece del rango de visión de la cámara frontal por la zona izquierda, esta misma detección aparecerá poco a poco en el campo de visión de la cámara izquierda. Al ocurrir esto se le asigna el mismo identificador $id = 12$ al *bounding box* de la cámara izquierda. En nuestro componente de reproyección, recibimos los *bounding boxes* de las detecciones junto a estos identificadores. En el caso de que dos *bounding boxes* tengan el mismo identificador, significará que esas detecciones se encuentran en una zona de solapamiento en el campo de visión entre las cámaras. En este caso, para reproyectar el vehículo en el visualizador 3D, usaremos el punto central obtenido haciendo la media de las posiciones de las reproyecciones de ambos *bounding boxes*.

Pese a todo, realizamos también un *tracking* 3D a los resultados obtenidos del *tracking* anterior, por si el identificador de algún *bounding box* falla y estando en las regiones de solapamiento no obtenemos el mismo identificador. Tras el primer *tracking*, tenemos como resultado un vector de objetos del tipo *RealObject* nativo de RTMaps representando a las reproyecciones de los vehículos con sus posiciones X, Y, Z en el visualizador 3D, que llamaremos $Objects$. Por cómo está programado el módulo de reproyección, tenemos un array $Objects$ de tamaño $N+1$, siendo N el número de reproyecciones a mostrar en el visualizador. El elemento número 0 del array corresponde al modelo 3D que usamos para

representar a CarLOTA. Los demás elementos están ordenados de forma que los primeros objetos del array corresponden a las detecciones de la cámara frontal, seguidos de la cámara izquierda, derecha y trasera, como vemos en la Figura 3.15.

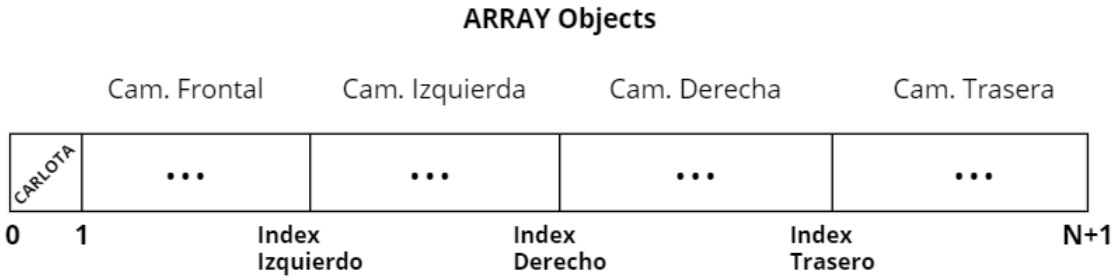


Figura 3.15: Array *Objects* de tamaño $N+1$ representando los objetos que obtenemos tras la reproyección y el filtro del tracking 2D.

Como vemos, tenemos unos índices en el array representando dónde empiezan y dónde acaban las detecciones de cada cámara. Para comprobar si hay un solapamiento entre las reproyecciones, hemos asignado un *threshold* de 4.5 metros entre dos vehículos. Utilizando estos índices, nos ahorraremos comprobar entre todas las detecciones si dos objetos se encuentran por debajo de nuestro *threshold*. Lo que haremos será usar esos índices para tratar nuestro array como un set de 4 subarrays, correspondiendo a las detecciones de cada cámara. Comprobaremos si hay solapamiento 3D entre la cámara frontal y las laterales y la cámara trasera con las laterales. En caso de que esto suceda, asignaremos el punto central entre ambas reproyecciones a un objeto y eliminaremos el otro, reasignando los índices que tengamos que actualizar tras ello. Podemos ver en la Figura 3.16 un ejemplo de solapamiento entre la cámara frontal (azul) e izquierda (amarillo) para detecciones con distintos identificadores. También podemos comprobar cómo en el caso de la detección de la cámara trasera (verde) y derecha (roja) no hay solapamiento, al no encontrarse el centro de reproyección entre ambas detecciones dentro del *threshold* de 4.5 metros.

3.6. Diagramas RTMaps

Para el testeo *offline* y uso a tiempo real de todos los componentes del proyecto, se han creado varios diagramas de RTMaps[2] que procedemos a explicar a continuación. Este software ha sido utilizado debido a la facilidad de trabajar a tiempo real que ofrece con todos los componentes del vehículo. Facilita en gran medida la conexión entre componentes y dispone de varias librerías centradas en conducción autónoma. Además, varios de los componentes creados previamente a este proyecto de reproyección han sido creados en este software.

[2] Para más información acerca del software RTMaps acudir al Anexo B.

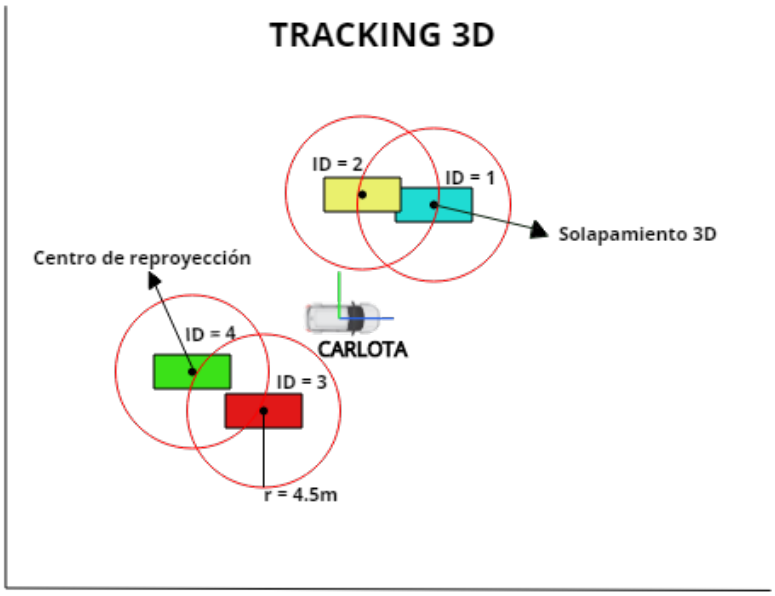


Figura 3.16: Vista de pájaro mostrando un ejemplo de solapamiento entre dos reproyecciones con identificadores distintos entre la cámara frontal y la izquierda. En el caso de las reproyecciones que vemos en la zona inferior izquierda, entre la cámara derecha y la trasera, las reproyecciones no están lo suficientemente cerca del *threshold* de 4.5 metros como para considerar un solapamiento.

3.6.1. Diagrama Offline

Este es el diagrama más básico del proyecto, y lo podemos ver en la Figura 3.17. En este diagrama hemos testeado el correcto funcionamiento de las grabaciones realizadas en la zona de Vicomtech.[3]

Esta es la función de cada componente del diagrama:

- **Video_Player**: Es un archivo .rec de una grabación realizada con CarLOTA. En este archivo se han grabado como datos las grabaciones de las cámaras, como *streamings* del tipo .imageOut. A su vez, se han grabado los *bounding boxes* de las detecciones de vehículos con YOLO v3 junto a sus identificadores.
- **Cam2World**: En este archivo de Python tratamos todos los datos obtenidos. Para ello, cargamos todos los parámetros de la calibración de las cámaras y vamos reproyectando los *bounding boxes* detectados por cada cámara a puntos 3D. Comprobamos que los *bounding boxes* no estén por encima de la línea del horizonte antes de esto para no tener datos erróneos. Dentro del script de Python dibujamos

[3] Para más información acerca de las grabaciones realizadas acudir al Anexo C.

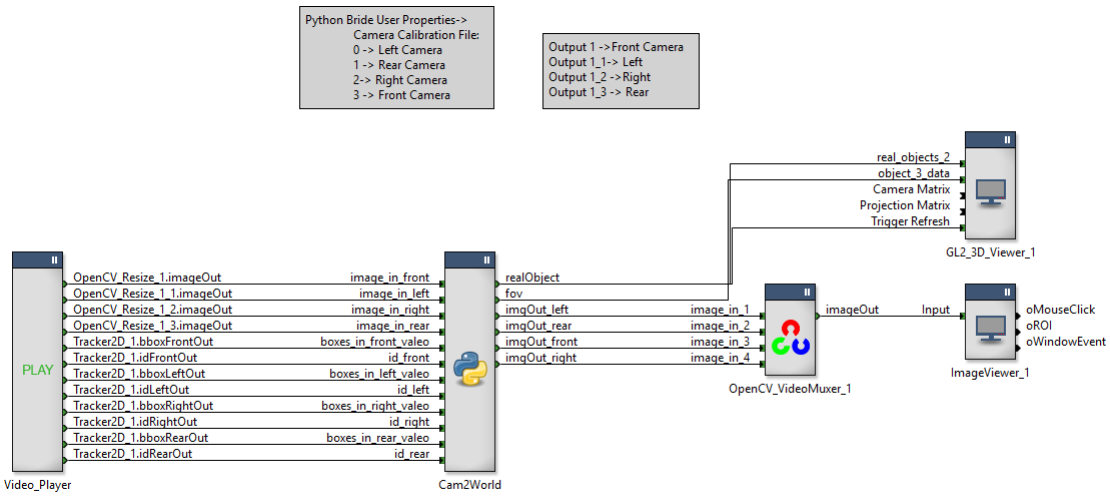


Figura 3.17: Diagrama de RTMaps para la visualización offline del entorno 3D y las cámaras de CarLOTA.

los *bounding boxes* en cada *stream* de vídeo para visualizar las detecciones con OpenCV, y gracias al proceso de reproyección, también dibujamos en la parte inferior de cada *bounding box* la distancia a la que se encuentra el vehículo detectado de CarLOTA. Por último, antes de enviar los resultados de la reproyección al visualizador 3D para poder dibujar la posición de los coches detectados, comprobamos que no haya detecciones en regiones de solapamiento de la cámara, para no dibujar dos coches en la misma posición en el visualizador 3D.

- **OpenCV_VideoMuxer**: Es un componente nativo de OpenCV para RTMaps, en el que podemos pasar varios streams de vídeo y juntarlos en un solo *stream*. En la parte superior derecha veremos la cámara frontal, en la superior izquierda la cámara izquierda, en la inferior izquierda la cámara trasera y en la inferior derecha la cámara derecha.

- **ImageViewer**: Es un componente nativo de RTMaps que nos permite visualizar en pantalla una ventana con el *streaming* de vídeo indicado, en este caso el output del VideoMuxer.

- **GL2_3D_Viewer**: Es el visualizador 3D de RTMaps, en el que dibujamos un modelo .obj de un coche para representar a CarLOTA, y también las reproyecciones de cada cámara como cubos coloreados. Por último, dibujamos el campo de visión de cada cámara tras haber quitado la distorsión a cada lente, lo cual reduce el ángulo de visión de cada cámara.

Gracias a este diagrama podemos visualizar lo que vemos en la Figura 3.18

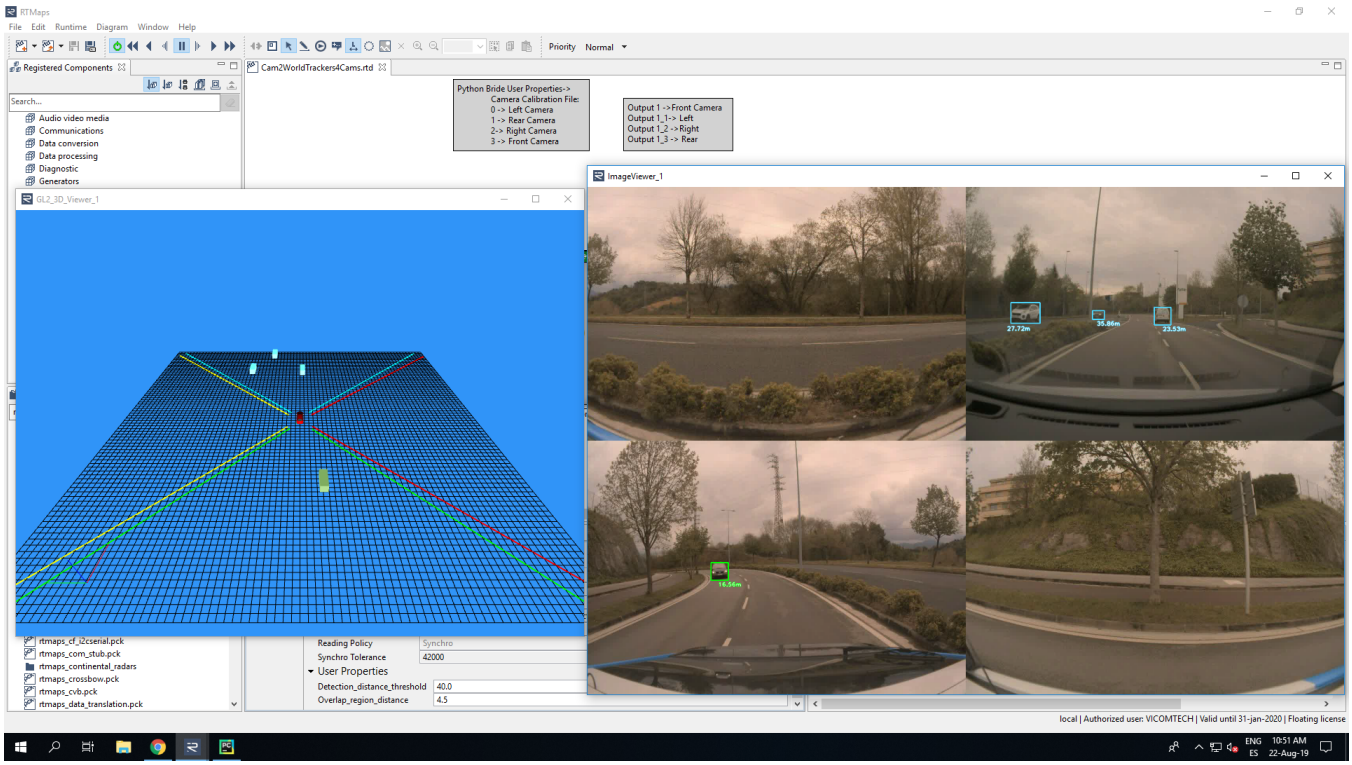


Figura 3.18: Output del diagrama, en el que podemos ver el visualizador 3D, los *streams* de vídeo de las 4 cámaras con sus detecciones de YOLO y las distancias de las detecciones.

3.6.2. Diagrama Offline con mapas y LiDAR

Pese a que la parte del proyecto correspondiente a este TFM se base en la reproyección de objetos sin uso de LiDAR, el proyecto abarca más ámbitos de la conducción autónoma y Vicomtech dispone de un LiDAR para el vehículo. Hemos añadido el LiDAR en algunas grabaciones, para comprobar visualmente la precisión de las reproyecciones.

A su vez, como el coche dispone de un localizador DGPS, podemos saber con precisión la posición del coche. Por tanto, si dispusiésemos de un mapa 3D de la zona, podríamos visualizar el movimiento del vehículo por el mapa. El problema es que generalmente, en conducción autónoma, se usan mapas del tipo *HD Maps*, que contienen información semántica y geométrica de la zona [41]. Estos mapas suelen ser de uso privado, debido a la dificultad y trabajo para crearlos. Sin embargo, hemos encontrado una alternativa gratuita, pese a no tener la misma precisión. Gracias a *Open Street Maps*, podemos obtener un fichero .osm de la zona que queremos modelar en 3D, y con el trabajo de OSM2World[4],

[4] http://osm2world.org/

podemos obtener un fichero .obj simulando aproximadamente los edificios y carreteras de la zona.

El origen de coordenadas del .obj lo proporcionan en latitud y longitud, al igual que los datos de posición del coche que obtenemos con el DGPS. Para facilitar el trabajo y comprensión del movimiento del vehículo por el mapa, hemos convertido estas coordenadas al sistema **UTM** (*Universal Transverse Mercator*), pasando de tener una representación en forma de esfera, como vemos en la Figura 3.19 a un plano bidimensional, como en la Figura 3.20 [42].

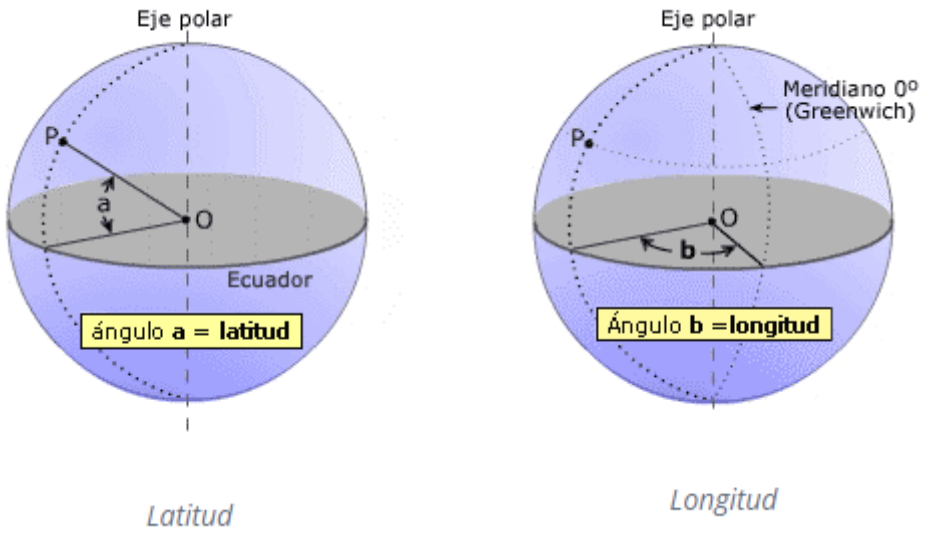


Figura 3.19: En la esfera izquierda vemos el ángulo que representa la latitud, mientras que en la esfera derecha vemos el ángulo que representa la longitud. Con estos dos ángulos, podemos obtener un punto en la superficie de la tierra representando la posición en la que nos encontramos.

Una vez tenemos las coordenadas del origen de coordenadas del .obj y de CarLOTA en UTM, vamos a tratar la posición de CarLOTA como el origen de coordenadas en todo momento, moviendo y rotando el fichero del mapa alrededor de este. Para ello, tendremos la posición de CarLOTA como $pos = (pos_x, pos_y, 0)$ y el punto que usaremos como origen de coordenadas $o = (o_x, o_y, 0)$ para crear el vector

$$t = (pos_x - o_x,\ pos_y - o_y,\ 0\ ) \tag{3.6}$$

y crearemos la matriz de transformación

$$M = \begin{bmatrix} 1 & 0 & 0 & t_0 \\ 0 & 1 & 0 & t_1 \\ 0 & 0 & 1 & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

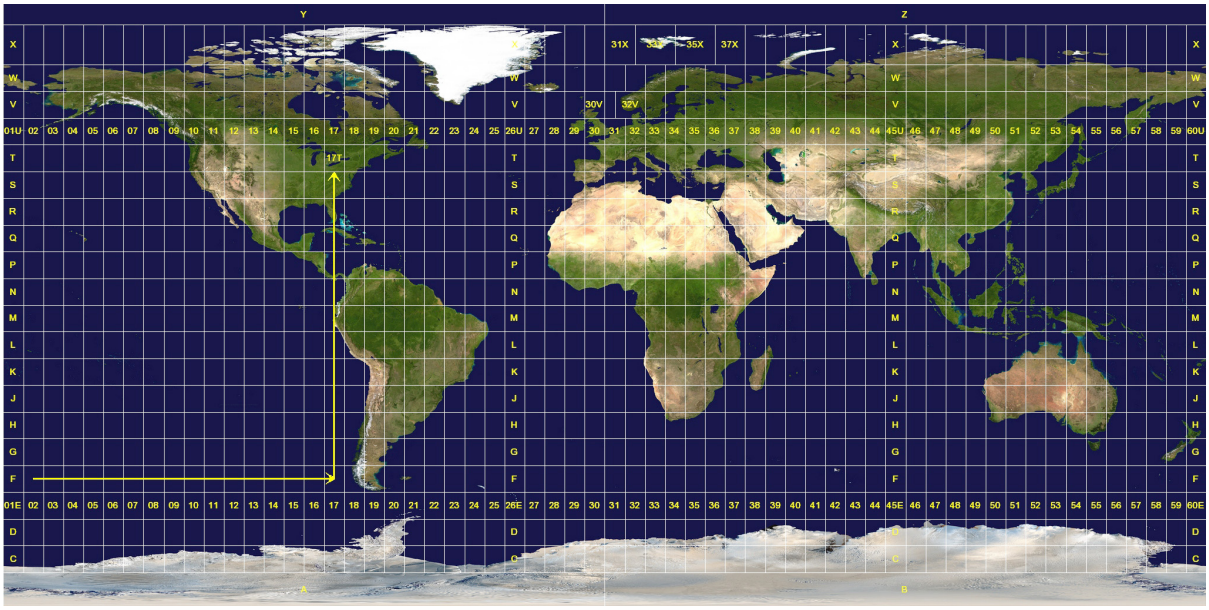


Figura 3.20: Representación de un mapa usando el sistema UTM, en el que podemos ver cómo está separado por 60 husos de 6º de longitud verticalmente y 20 bandas de 8º de longitud horizontalmente representadas por letras de la C a la X excluyendo a la I y la O.

Disponiendo gracias al DGPS el ángulo α al que se dirige el coche, podremos crear la matriz de rotación

$$rotz = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 & 0 \\ sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

y así poder ir trasladando y rotando el modelo del mapa alrededor de CarLOTA con la matriz M' que obtenemos de la multiplicación de

$$M' = rotz \times M \tag{3.9}$$

El diagrama de RTMaps que nos queda es el que vemos en la Figura 3.21. Este diagrama lo hemos usado para comprobar gráficamente las reproyecciones y añadir utilidad al visualizador 3D al poder ver un modelo 3D aproximado de la zona. Explicamos sus componentes a continuación.

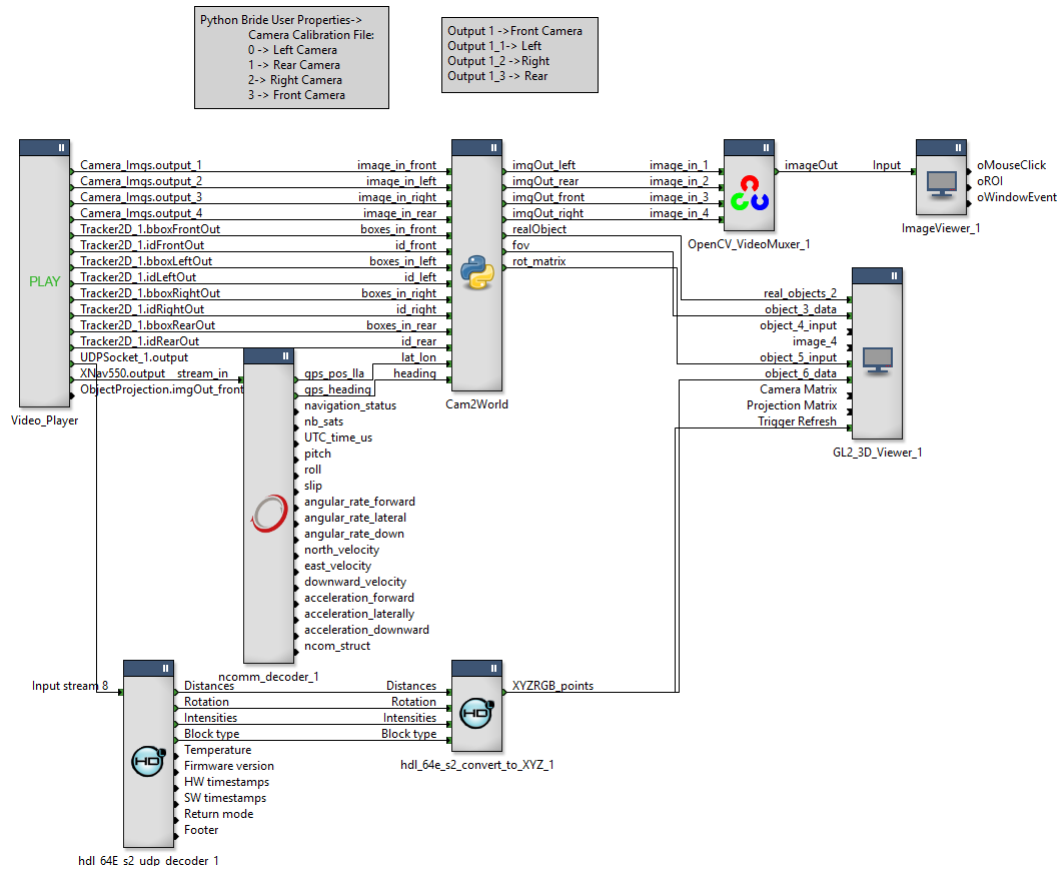


Figura 3.21: Diagrama de RTMaps para la visualización *offline* del entorno 3D junto al mapa y el LiDAR con las cámaras de CarLOTA.

- **Video_Player**: Disponemos de todos los elementos que tenían las anteriores grabaciones, más el output de una salida UDP con la información de la nube de puntos del LiDAR y también un *stream* de datos con la información del DGPS.
- **ncomm_decoder**: Es un componente de la empresa *OXTS* para leer e interpretar los datos del DGPS y así obtener la posición y dirección de CarLOTA.
- **hdl_udp_decoder**: Es un componente de la empresa *Velodyne* para interpretar la nube de puntos del *stream* UDP.
- **hdl_convert_to_XYZ**: Otro componente de *Velodyne* que interpreta la información obtenida de decodificar el *stream* UDP para obtener las coordenadas X, Y, Z de la nube de puntos generada por el LiDAR.
- **Cam2World**: Realiza la misma función que en el diagrama anterior, pero también realiza el proceso de transformación de latitud y longitud a UTM junto con las translaciones y rotaciones correspondientes del mapa .obj para el visualizador 3D.

- **OpenCV_VideoMuxer**: Es el mismo componente de OpenCV que usamos en el diagrama anterior para juntar los 4 *streamings* de video en uno.
- **ImageViewer**: Para poder visualizar en una ventana el *stream* de vídeo generado por el componente de OpenCV.
- **GL2_3D_Viewer**: Realiza las mismas funciones que en el diagrama anterior pero, también dibuja la nube de puntos generada con el LiDAR junto al fichero .obj representando al mapa de la zona.

Gracias a este diagrama podemos visualizar lo que vemos en la Figura 3.22

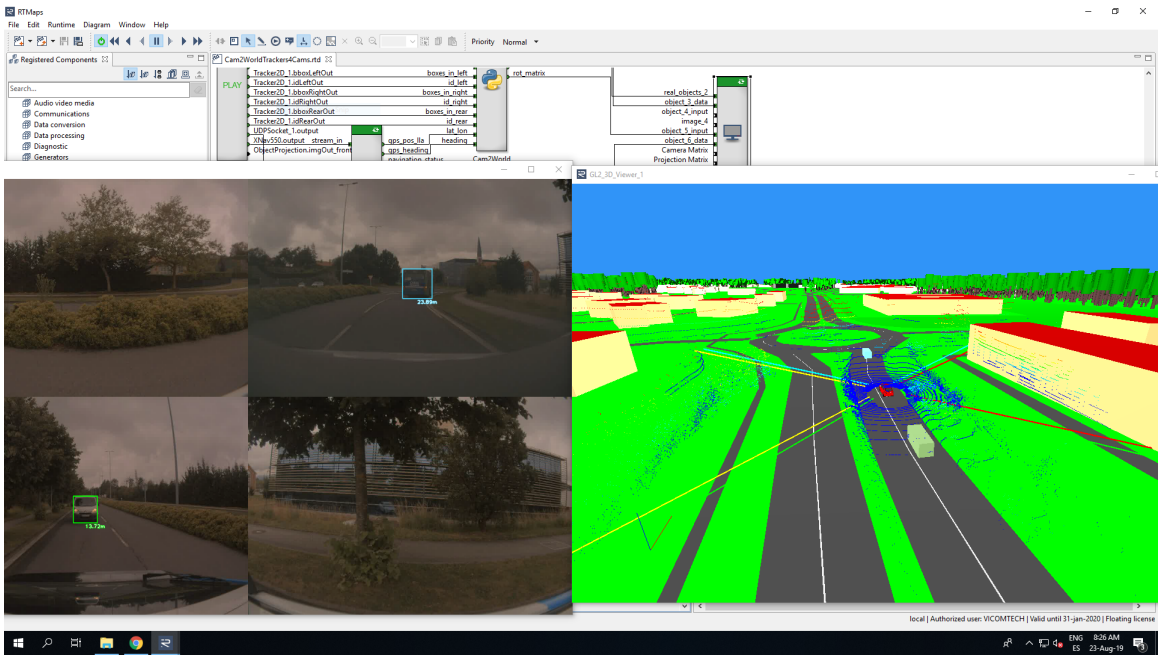


Figura 3.22: Output del diagrama de RTMaps para la visualización *offline* del entorno 3D, junto al mapa y el LiDAR con las reproyecciones de CarLOTA.

3.6.3. Diagrama a tiempo real

En este diagrama se encuentran todos los componentes usados en CarLOTA, tanto los creados por Vicomtech como los demás colaboradores del proyecto, para hacer funcionar todos los elementos del vehículo a tiempo real. Es el diagrama usado en la demo del ITS en Eindhoven[5]. El diagrama lo vemos en la Figura 3.23, y vamos a explicar a continuación su funcionamiento. Esta vez vamos a explicar la tarea a realizar en cada grupo de componentes, y no explicar su función uno a uno.

[5] Explicado con más detalle en el Anexo C.

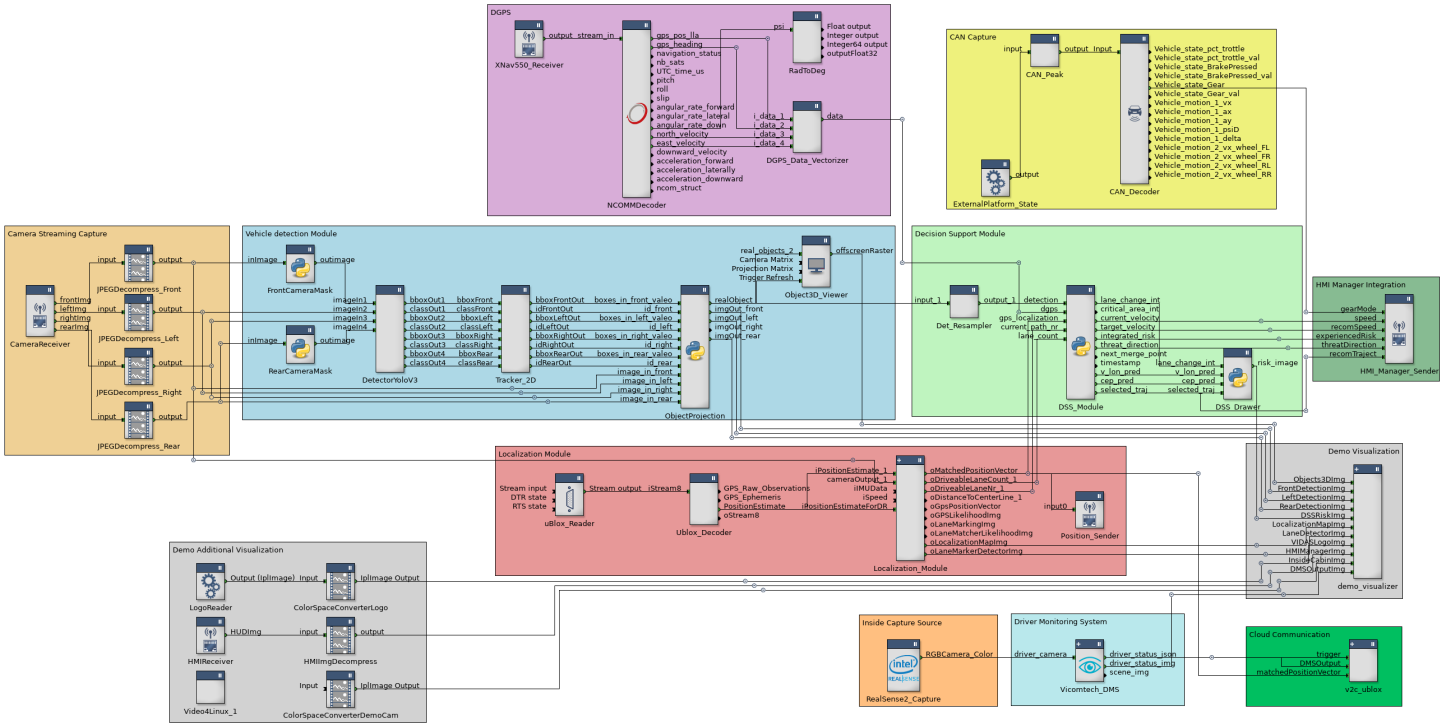


Figura 3.23: Diagrama de RTMaps para el funcionamiento a tiempo real de todos los componentes de CarLOTA.

- **Camera Streaming Capture**: Este grupo de componentes se ocupa de recibir las imágenes obtenidas de las cámaras desde la Drive PX 2, reconvertir su formato y mandarlo como output a los componentes necesarios.

- **Vehicle Detection Module**: En estos componentes realizamos la tarea de detección con YOLO v3, reproyección y *tracking* para mandarlos al visualizador 3D.

- **HMI Manager Integration**: *(Human Machine Integration)* con los datos que recibe de los demás componentes, iluminará una tira de LEDs en el caso de que el conductor pueda realizar un cambio de carril de forma segura.

- **Demo Additional Visualization**: Para la demostración realizada en Eindhoven se mostraba adicionalmente en pantalla el HMI y algunas imágenes con logos de los colaboradores.

- **DGPS**: Se encarga de obtener la señal del DGPS y transformarla para ser leída por los demás componentes.

- **Localization Module**: Es el módulo de TomTom usado para conocer el número de carriles que hay en la carretera sobre la que se circula si se dispone de HD Maps de la zona.

- **Inside Capture Source**: Se encarga de enviar las imágenes obtenidas de la cámara que enfoca al conductor para el Driver Monitoring System.
- **Driver Monitoring System** (DMS): Con las imágenes del conductor se monitorizará su estado para saber si está prestando atención a la tarea de conducción, se ha quedado dormido o está realizando algún movimiento anómalo.
- **Decision Support Module**: Módulo creado por Honda para saber si se puede realizar un cambio de carril de forma segura usando los datos de la reproyección.
- **CAN Capture**: Usado para saber en qué marcha está conduciendo el vehículo para mostrarlo en el HMI.
- **Cloud Communication**: Envía información del DMS y GPS a los servicios cloud de IBM.
- **Demo Visualization**: Es el visualizador en pantalla para *streamings* de vídeos, visualizador 3D y demás componentes con algún *output* de visualización dentro del diagrama.

Este diagrama no tiene incorporados los mapas en 3D que mencionábamos en el apartado anterior al haber sido usado en el ITS de Eindhoven, y la demo fuese previa a la programación del módulo de mapas 3D. Actualmente se está actualizando el diagrama para hacer funcionar todos los componentes junto a los mapas 3D.

4. CAPÍTULO

Análisis de los resultados

4.1. Dataset KITTI

Una vez descrito el método de reproyección, en este capítulo se presenta el trabajo realizado para evaluar su precisión. El principal problema de esto sería tener que etiquetar manualmente todos los vehículos que vemos a lo largo de una grabación, debido al alto coste de tiempo que esto conlleva. Si bien disponemos de un LiDAR que nos facilitaría esta tarea, crear un *dataset* de vehículos etiquetado queda fuera del alcance del proyecto, por el tiempo requerido y su dificultad. Por ello, se han buscado distintas alternativas siendo el *dataset* de **KITTI**[1] [22] el más completo que hemos encontrado para nuestro trabajo. Más en concreto con el subapartado de objetos 3D dentro del *dataset*, el cual cuenta con 15000 imágenes etiquetadas. Su vehículo de *testing* consta de un LiDAR y dos pares de cámaras estéreo con las que se puede obtener la profundidad de la escena. En nuestro caso usamos cámaras monoculares, por lo que no usaremos ni la información de profundidad de las cámaras estéreo ni la del LiDAR. El *dataset* que hemos utilizado consta de los siguientes atributos:

- **type:** Describe el tipo de objeto etiquetado, pudiendo tomar los siguientes valores: 'Car','Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' o 'DontCare'.
- **truncated**: Un valor decimal entre 0, no truncado, y 1, truncado, refiriéndose a que el *bounding box* de un objeto sobresalga del plano de imagen.
- **occluded**: Valores enteros de 0 a 3 indicando la visibilidad del objeto, 0 = totalmente visible, 1 = parcialmente visible, 2 = poco visible, 3 = desconocido.

[1] http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d

- **alpha**: Ángulo de observación del objeto $[-\pi, \pi]$.
- **bbox**: *Bounding box* 2D del objeto detectado en la imagen, conteniendo la posición del píxel superior izquierdo y el inferior derecho del *bounding box*.
- **dimensions**: Dimensiones 3D del objeto detectado en metros: altura, ancho y largo.
- **location**: Posición 3D en metros del objeto en coordenadas de la cámara: x, y, z.
- **rotation_y**: Rotación r_y alrededor del eje Y en coordenadas de la cámara $[-\pi, \pi]$.
- **score**: Valor decimal indicando el *confidence score* de la detección del objeto, cuanto más alta mejor.

Junto a estos datos también ofrecen las nubes de puntos del LiDAR, datos de profundidad de las cámaras estéreo y los datos de calibración con las matrices de intrínsecos y extrínsecos de las cámaras. Nosotros nos limitaremos a hacer pruebas con las imágenes y datos de calibración. Cabe mencionar que entre los 90 mejores resultados de este *dataset* ninguno usa cámaras monoculares, sino que se centran en el uso del LiDAR principalmente, o cámaras estéreo.

4.2. Análisis de reproyección para coches

Aplicando nuestro algoritmo de reproyección con los *bounding boxes*, y las anotaciones de posición de las anotaciones de KITTI, podremos comparar la precisión de nuestra reproyección. Primero vamos a analizar los vehículos con la etiqueta de tipo 'Car' a un rango de 40 metros del coche, que es el rango con el que se tiene pensado usar este proyecto. Hemos filtrado todos los *bounding boxes* que estén por encima de la línea del horizonte, ya que nuestro método no está pensado para reproyectar las detecciones en este caso. También hemos filtrado las detecciones con truncamiento y hemos obtenido 17195 vehículos etiquetados que procedemos a reproyectar.

En la Figura 4.1 podemos ver un *heatmap* de las posiciones reales de los coches etiquetados de KITTI en un rango de 40 metros, mientras que en la Figura 4.2 podemos ver la posición de esos mismos coches tras usar los *bounding boxes* de KITTI para reproyectar su posición.

Algo en lo que nos podemos fijar es que en la Figura 4.2, cuanto más se incrementa el valor en X, más se nota que son valores sintéticos, al verse líneas verticales en los que no se ha reproyectado ningún coche, a diferencia de los valores reales de la Figura 4.1. Esto se debe a que, según se va acercando la parte inferior del *bounding box* a la línea del horizonte en la imagen, más afecta esto a la reproyección en el eje X, haciendo que un solo píxel de diferencia reproyecte la detección a una distancia más lejana.

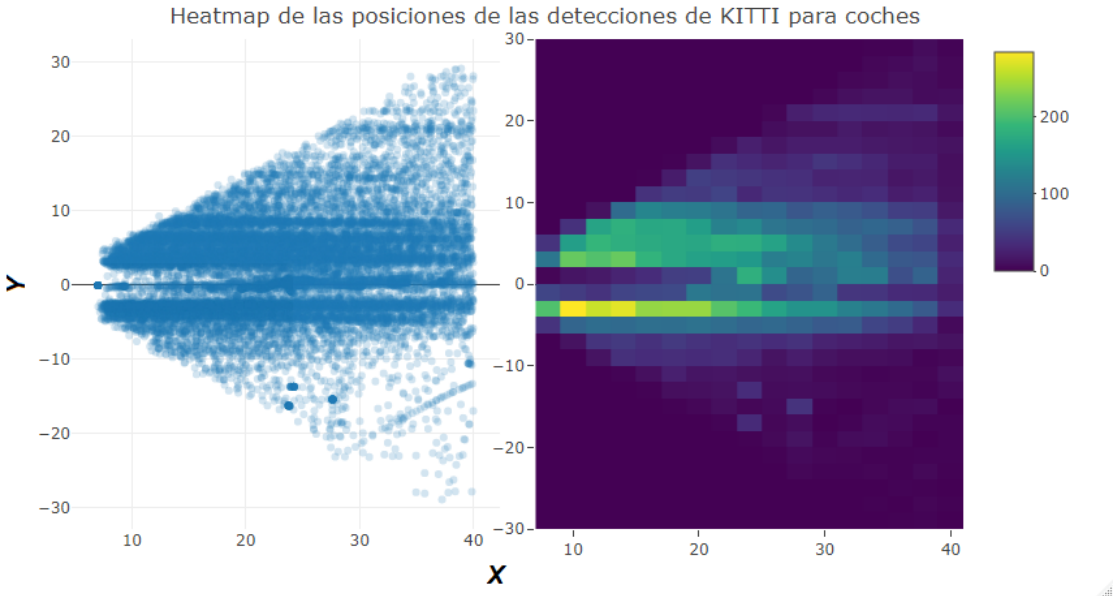


Figura 4.1: *Heatmap* con la posición de los coches del *dataset* KITTI, limitado a un rango de 40 metros y sin truncamiento.

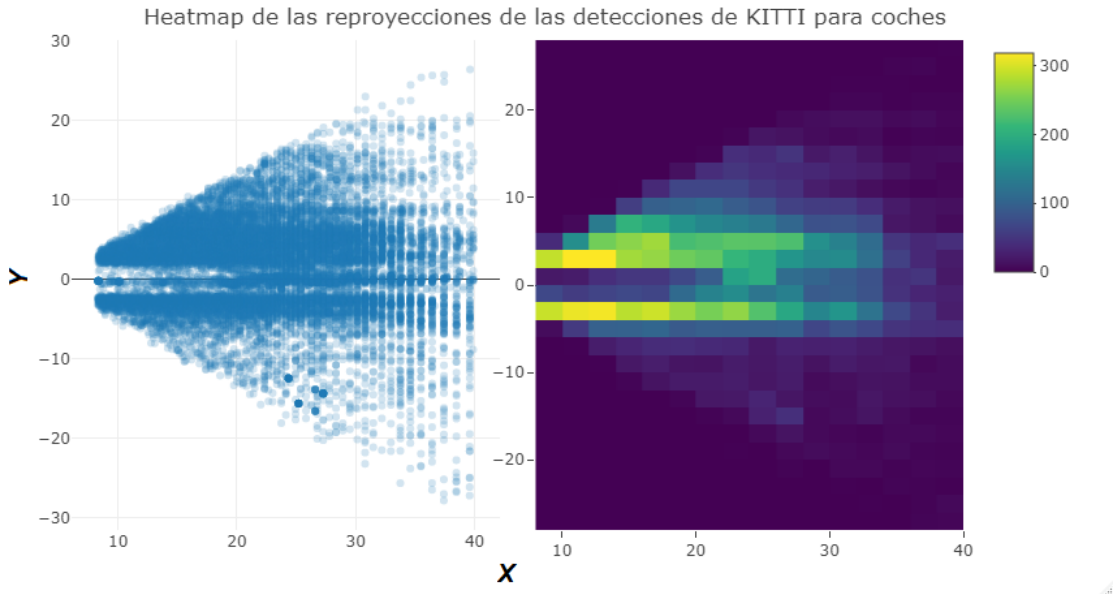


Figura 4.2: *Heatmap* con la reproyección de las posiciones de los coches del *dataset* KITTI, limitado a un rango de 40 metros y sin truncamiento.

Podemos ver en el gráfico 3D de la Figura 4.3 el error de reproyección que cometemos para cada vehículo, reproyectado en esas coordenadas X e Y mediante la distancia euclidiana $\varepsilon = \sqrt{(X^2+Y^2)}$.

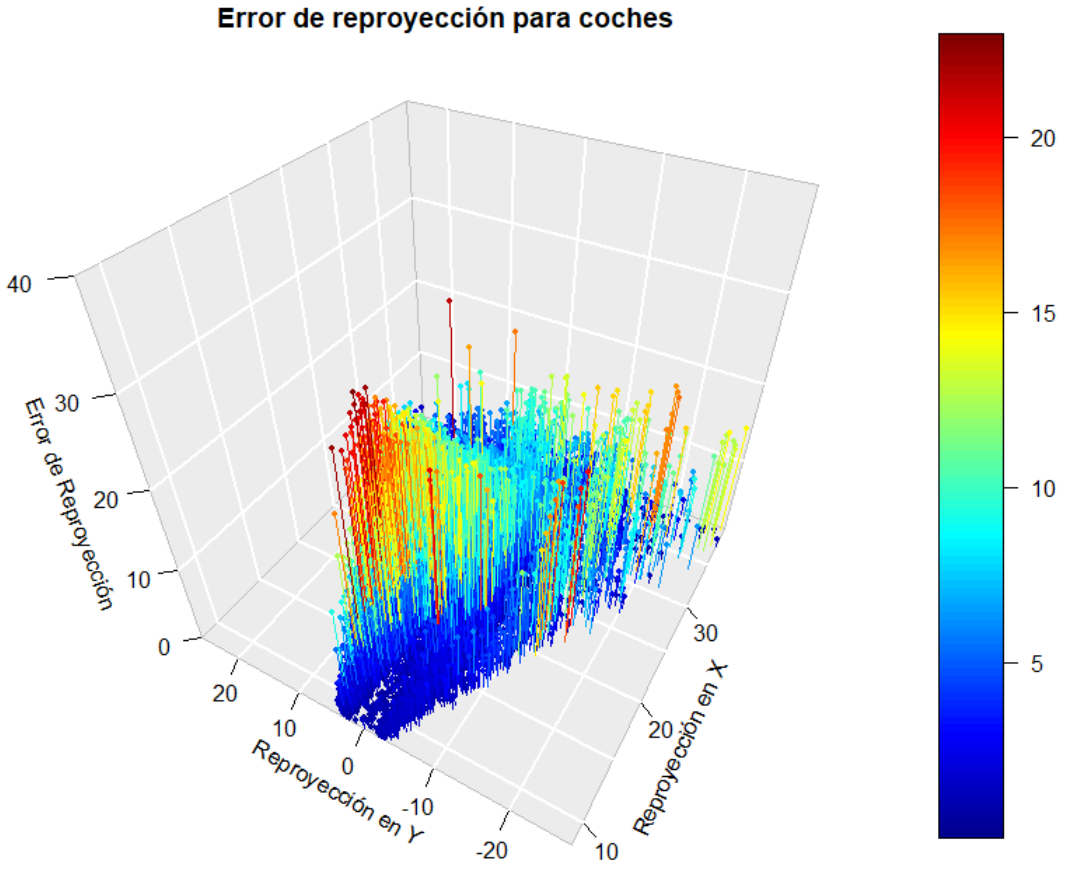


Figura 4.3: Gráfico 3D con el error de reproyección total para cada coche etiquetado en KITTI tras el filtro de datos.

El mayor error cometido en la reproyección ha sido de 22.94 metros, siendo este uno de los puntos de la zona central en la que se aprecia que hay un grupo de reproyecciones con mayor error. También se puede comprobar que en la zona más alejada de X hay algunos puntos con error de $\simeq 20$ metros. Para analizar qué problemas hemos tenido en la tarea de reproyección, podemos comprobar por separado qué error cometemos en cada coordenada de reproyección, para visualizar más claramente las zonas de error del gráfico.

Ahora, vamos a analizar la Figura 4.4, donde vemos el error cometido en X para las mismas reproyecciones del gráfico 3D anterior.

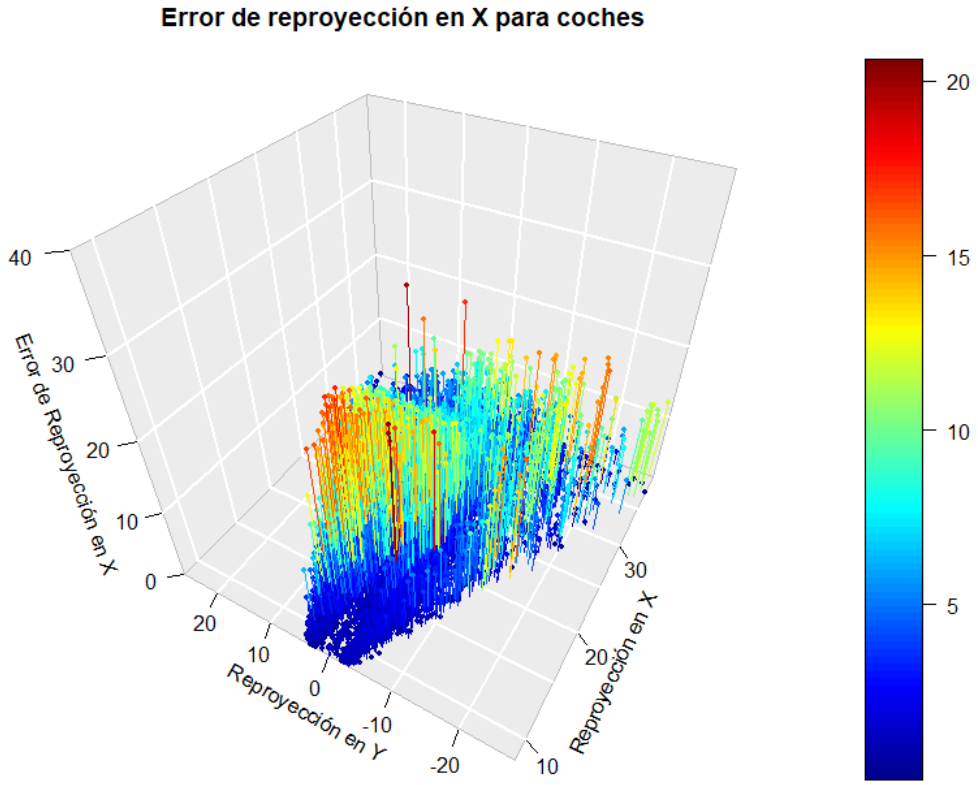


Figura 4.4: Gráfico 3D con el error de reproyección en X para cada coche etiquetado en KITTI tras el filtro de detecciones.

Para la coordenada X podemos ver cómo el error se acumula en las mismas zonas. En la Figura 4.5 podemos ver el valor de reproyección en X frente a su valor real, los puntos cercanos a la recta $X_{real} = X_{reproyectado}$ serán reproyecciones precisas, mientras que cuanto más se alejen mayor error tendrán. Los puntos por encima de la recta $X_{real} = X_{reproyectado}$ serán valores reproyectados mayores a los reales, mientras que los que están por debajo serán menores.

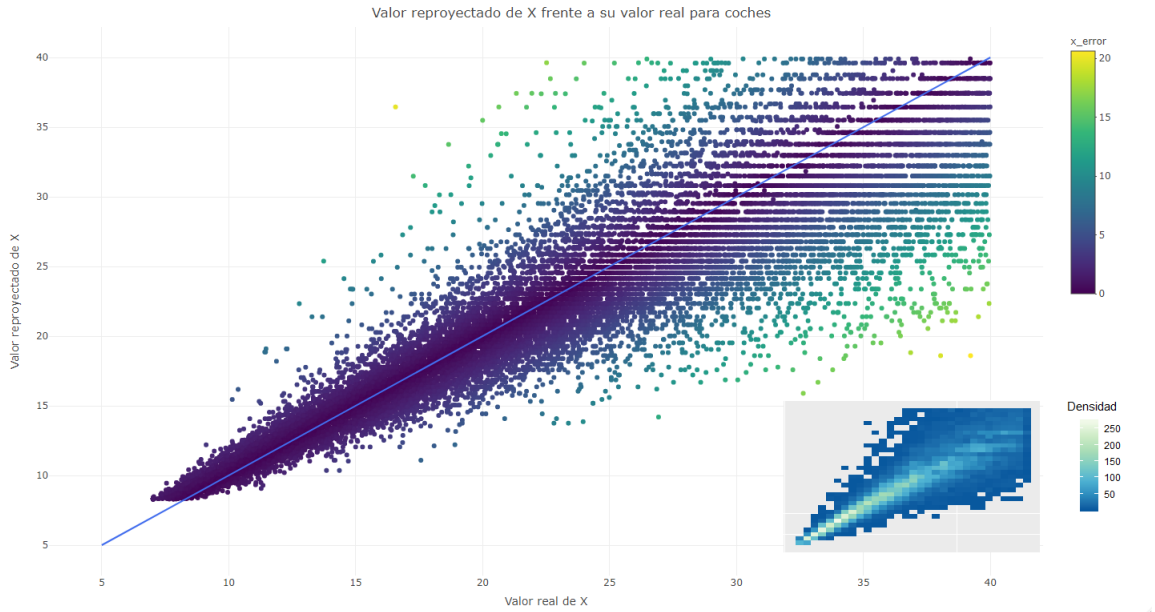


Figura 4.5: Diagrama de dispersión mostrando el valor reproyectado frente al real. Los puntos están coloreados según su error de reproyección en X, usando la paleta de colores superior derecha. En la parte inferior podemos ver un *heatmap* con la densidad de los puntos del diagrama de dispersión.

Podemos comprobar qué tipo de detecciones causan este error. En la Figura 4.6, generada en Python usando OpenCV y el script de reproyección, podemos ver los valores de reproyección obtenidos frente a los valores reales.

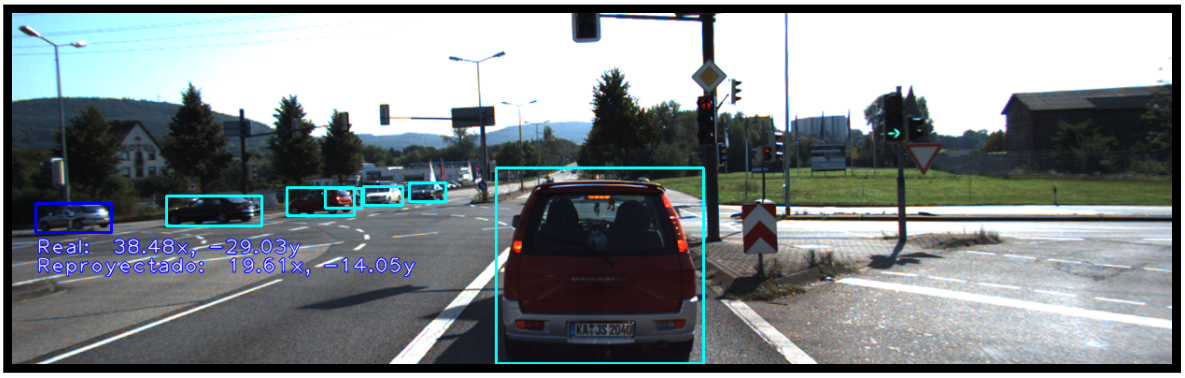
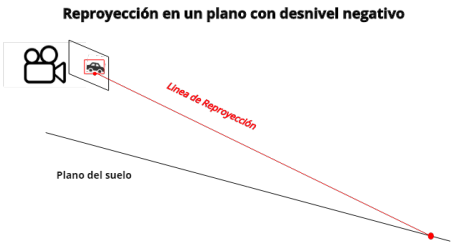


Figura 4.6: Imagen de validación del *dataset* KITTI, en el que comprobamos el error cometido en la reproyección frente a los valores reales de posición.

Podemos comprobar fácilmente que el plano del suelo en esta imagen muestra un desnivel negativo, lo que causa que nuestra reproyección en X sea notablemente inferior. Esto es debido a que, en el caso de que el plano fuese totalmente horizontal, el *bounding box*

estaría en una posición más alta. Al estar más bajo, la reproyección calcula que el objeto se encuentra más cerca. En la Figura 4.7 podemos comprobar cómo si fuésemos capaces de calcular el desnivel del plano del suelo este error mejoraría.

(a) En nuestro caso la reproyección siempre asume que el plano es horizontal al coche.



(b) Cambio en la distancia de reproyección en el caso de que el plano tuviese un desnivel negativo.

(c) Cambio en la distancia de reproyección en el caso de que el plano tuviese un desnivel positivo.

Figura 4.7: Fotos mostrando la diferencia en el proceso de reproyección en el caso de resolver la asunción del plano horizontal del suelo y poder calcularlo.

En el caso del error de reproyección en Y, podemos ver en la Figura 4.8 como es bastante menor que el que obtenemos en X, siendo las zonas laterales las que más error muestran. La zona central no consta de casi error de reproyección, lo cual es totalmente lógico. En el caso de que una detección en la zona central se vea afectada por un plano con desnivel, el eje X mostrará un error notable de reproyección, pero para Y el cambio es muy pequeño cuanto más se acerque a la zona central. La detección con mayor error en el eje Y es la misma que la de la Figura 4.6, debido a estar en un plano con desnivel negativo y en un lateral.

Los demás puntos que vemos en la zona izquierda con mayor error pertenecen a imágenes del *dataset* con *frames* de la misma grabación. Son los mismos coches, aunque en distinto

momento, los que han sido detectados, estando estos en un plano con desnivel y detectados en la zona lateral de la cámara.

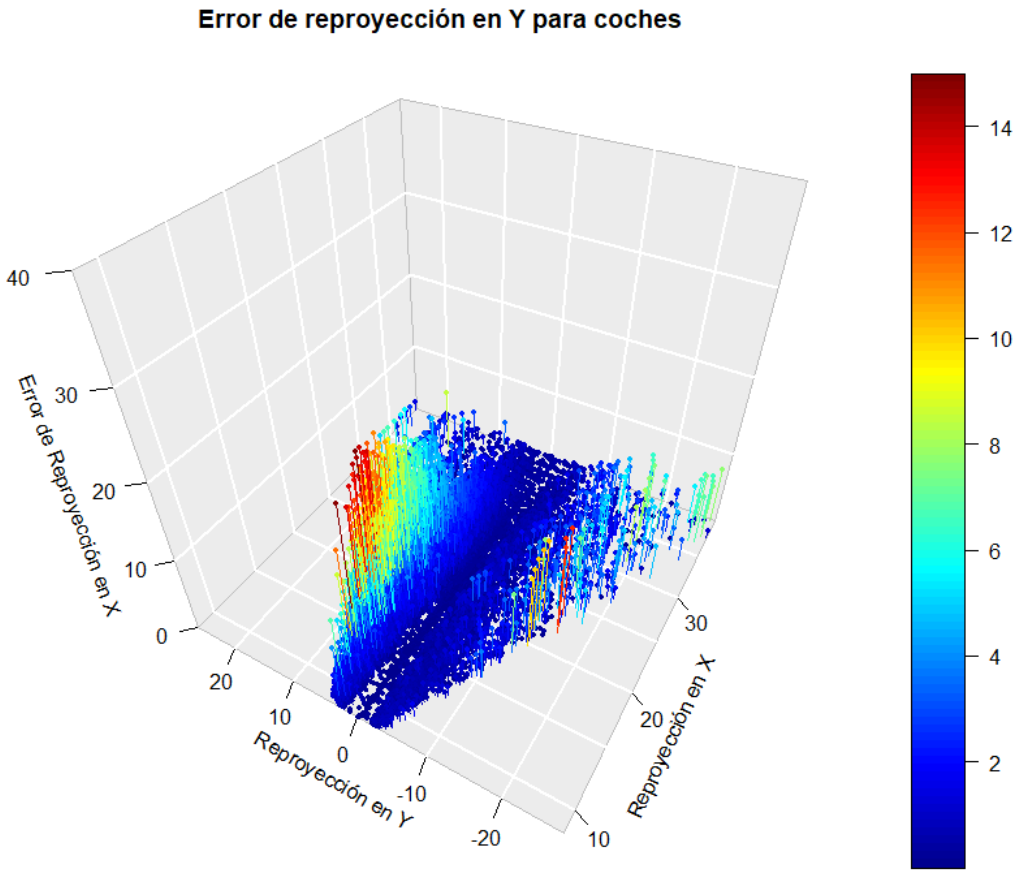


Figura 4.8: Gráfico 3D con el error de reproyección en Y para cada coche etiquetado en KITTI tras el filtro de detecciones.

Ahora vamos a analizar el diagrama de dispersión para el eje Y de la Figura 4.9, tal y como hacíamos con el eje X.

Podemos comprobar cómo la dispersión es mucho menor a la del eje X, teniendo la mayor zona de error en el cuadrante superior derecho, por debajo de la recta $Y_{real} = Y_{reproyectado}$. Estos puntos son reproyecciones que están por debajo del valor real de la detección, correspondiendo con los *frames* que veíamos de la grabación de la Figura 4.6.

En el caso de no haber realizado el filtro para las detecciones que se encuentran cercanas o por encima de la línea del horizonte, tendríamos reproyecciones con resultados mucho menos precisos. En el caso de que un *bounding box* esté en la línea del horizonte, al ser paralelo al plano, la reproyección daría un valor infinito.

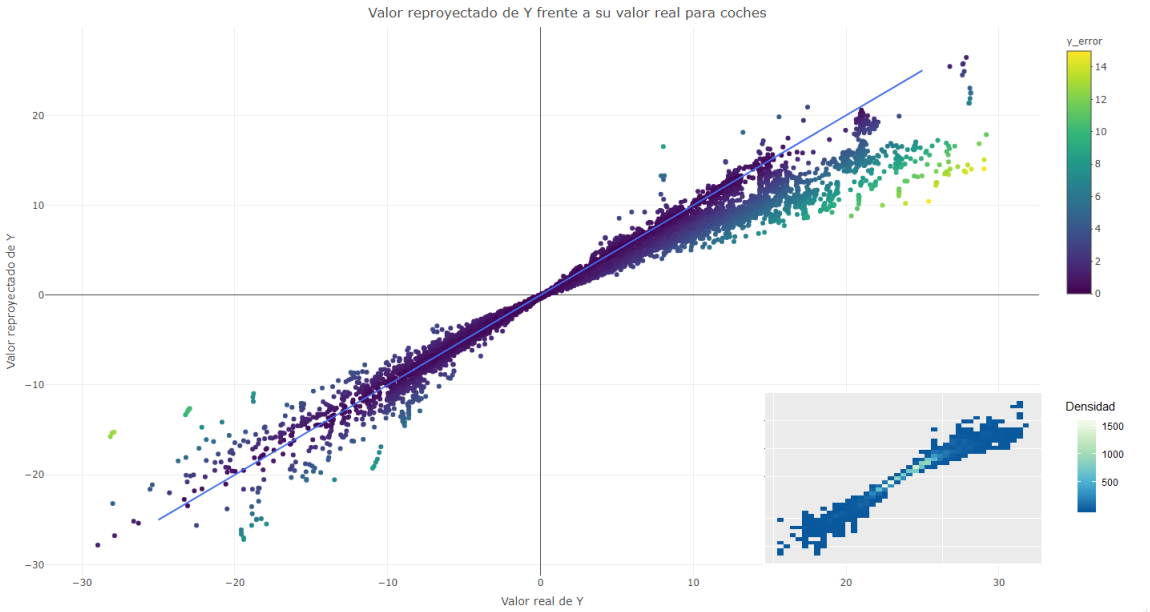


Figura 4.9: Diagrama de dispersión mostrando el valor reproyectado frente al real. Los puntos están coloreados según su error de reproyección en Y, usando la paleta de colores superior derecha. En la parte inferior podemos ver un *heatmap* con la densidad de los puntos del diagrama de dispersión.

En la Figura 4.10 podemos ver un caso filtrado en el que el *bounding box* está muy cercano a la línea del horizonte, por lo que sus valores son mucho mayores a los reales. La parte inferior de la detección de la furgoneta (en este caso no es de la clase 'Car') está en un plano con desnivel positivo, causando que el *bounding box* esté cercano a la altura de la línea del horizonte.

Esta imagen tiene ambos casos de desnivel del suelo. En la detección de la parte izquierda, la reproyección da valores más cercanos a los reales. En el caso de la detección derecha, como este está prácticamente en la línea del horizonte del centro de proyección, los valores son mucho más lejanos a los reales.

La media de error para el eje X es de 2.56 metros, para el eje Y de 1.03 metros y calculando la distancia euclídea para medir el error en ambos ejes el error sube a 2.89 metros.

4.3. Análisis de reproyección para peatones

En cuanto a la reproyección de peatones, es más importante aún filtrar los casos en los que se encuentren por encima o cerca de la línea del horizonte, ya que es más probable que estén subidos a una plataforma, rampa o cualquier elemento que les haga estar por encima del plano del suelo que un vehículo. También hemos filtrado los peatones poco visibles

Figura 4.10: Imagen de validación del *dataset* KITTI, en el que comprobamos el error cometido en la reproyección frente a los valores reales de posición en un *bounding box* cercano a la línea del horizonte.

para evitar casos como estar en puentes elevados, o que el etiquetado en la parte inferior no sea del todo preciso. Tras realizar el filtro, nos hemos quedado con 3053 detecciones de las 4005 originales en un rango de 40 metros para peatones. En la Figura 4.11, al igual que en las detecciones de coches, vamos a ver el error total de reproyección cometido con los peatones.

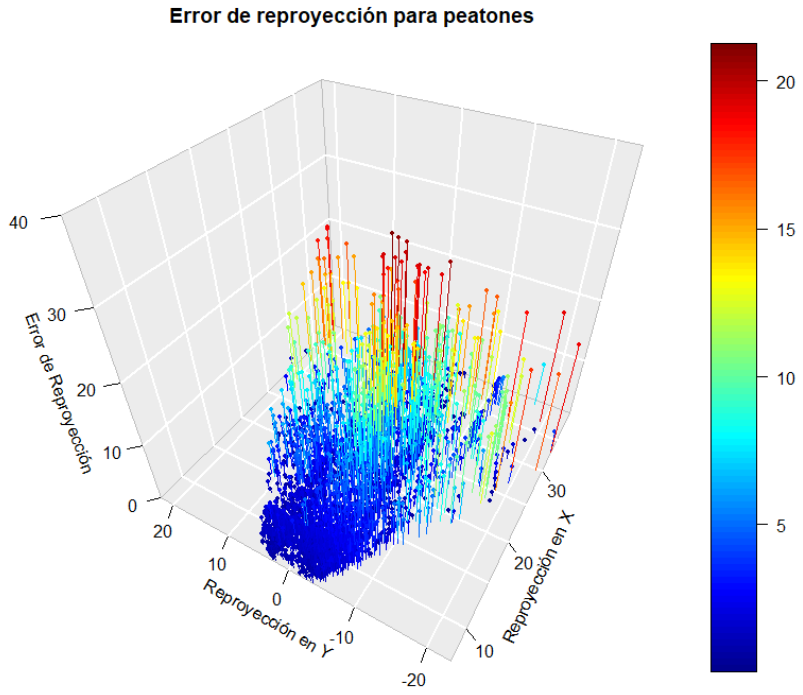


Figura 4.11: Gráfico 3D con el error de reproyección total para cada persona etiquetada en KITTI tras el filtro de detecciones.

Como recordamos, a la hora de reproyectar coches realizamos una pequeña corrección en la reproyección debido a que no estamos reproyectando el punto central real del vehículo, sino la parte más cercana del vehículo a la cámara. Los peatones, al ser más pequeños que un coche, no necesitarían de ningún tipo de corrección. En este caso podemos ver claramente en el gráfico 3D como según aumenta el valor real de X aumenta también el error de reproyección. Las detecciones cercanas a la cámara tienen valores muy pequeños de error. Ahora vamos a analizar el mismo gráfico, pero para su error de reproyección tan solo en el eje X, como podemos ver en la Figura 4.12.

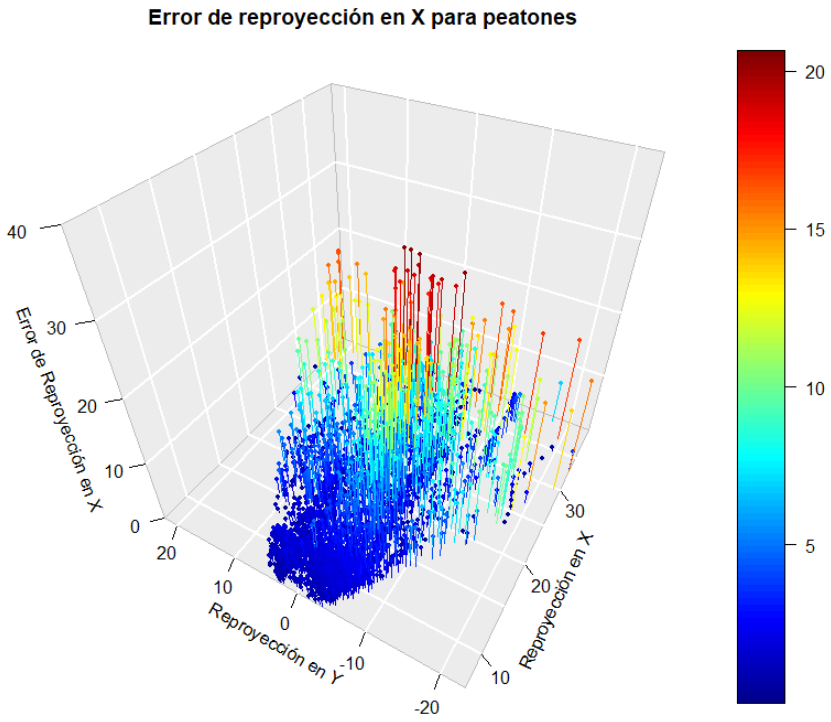


Figura 4.12: Gráfico 3D con el error de reproyección en X para cada persona etiquetada en KITTI tras el filtro de detecciones.

Podemos comprobar como el gráfico es muy parecido al del error de reproyección del gráfico 3D anterior, en el que comprobábamos el error total de reproyección de nuestro método. Esto es debido a que el error de reproyección en Y será muy pequeño para el caso de los peatones.

En el diagrama de dispersión que vemos en la Figura 4.13 podemos comprobar la reproyección frente a la posición real de los peatones en el eje X, y como la mayoría de datos con mayor error han sido reproyectados con valores mayores a los reales. Como hemos dicho previamente, los peatones en un entorno cotidiano tienen más probabilidad de estar por encima de la línea del horizonte, y no estrictamente en el plano del suelo, que un coche, ya que pueden ir por rampas, escaleras, estar subidos a algún objeto, etc.

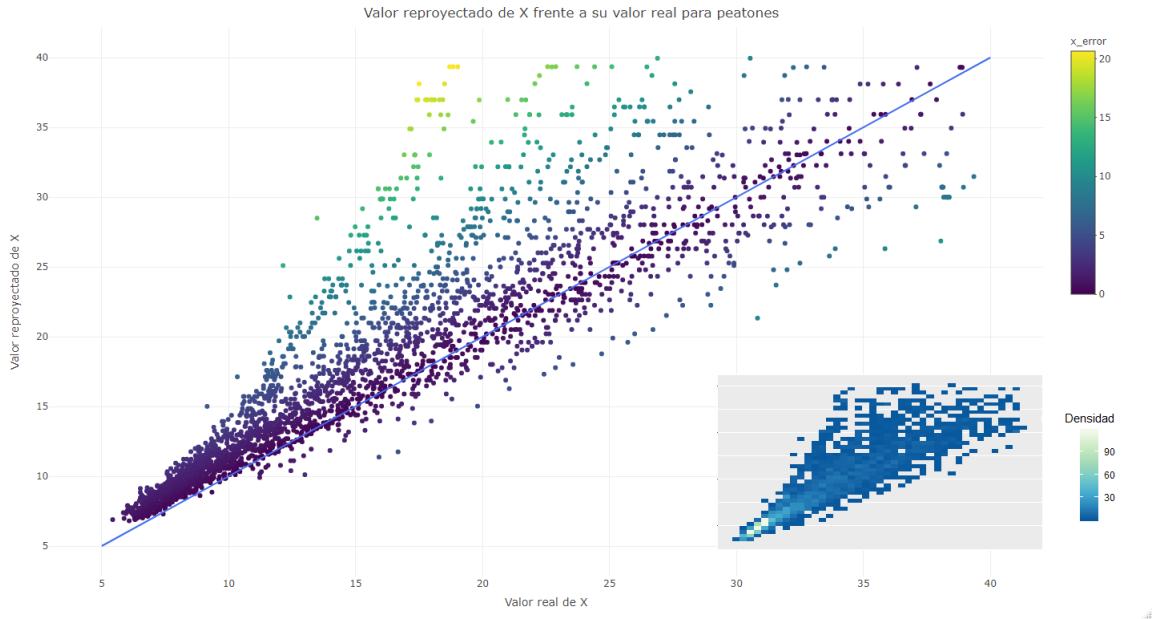


Figura 4.13: Diagrama de dispersión mostrando el valor reproyectado frente al real para peatones. Los puntos están coloreados según su error de reproyección en X, usando la paleta de colores superior derecha. En la parte inferior podemos ver un *heatmap* con la densidad de los puntos del diagrama de dispersión.

Como mencionábamos previamente, al ser los gráficos 3D anteriores muy parecidos implicará que el eje Y no tenga tanto error como el X. Podemos ver en la Figura 4.14 cómo esta asunción es totalmente correcta, al no aportar la reproyección en este eje casi error a las detecciones de peatones.

En el caso de los vehículos, había bastantes detecciones en la zona izquierda que se encontraban en un plano con desnivel, causando error tanto en X como en Y y dando peores resultados. En el caso de los peatones, la mayoría de reproyecciones erróneas están en la zona central de la cámara. Esto causa poco error en el eje Y, al no ser tan notorio el fallo en este eje si las detecciones se encuentran en en el centro de la imagen.

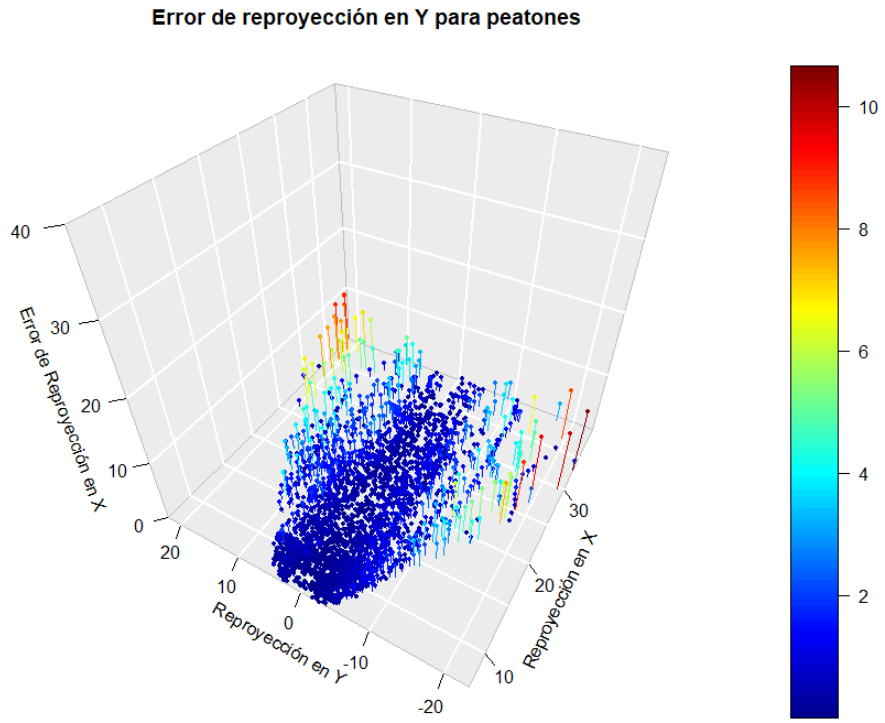


Figura 4.14: Gráfico 3D con el error de reproyección en Y para cada persona etiquetada en KITTI tras el filtro de detecciones.

Podemos comprobar cómo en el diagrama de dispersión que vemos en la Figura 4.15 las detecciones no muestran casi dispersión frente a la recta $Y_{real} = Y_{reproyectado}$, salvo en algunas excepciones.

Por último, cabe mencionar que la media de error en el eje X es de 2.92 metros y de 0.69 metros para el eje Y, mientras que el error total cometido calculado mediante la distancia euclídea es de 3.06 metros.

Hemos realizado el mismo análisis de datos anterior para coches y peatones sobre las clases 'Truck' y la clase 'Van' del *dataset* de KITTI. La cantidad de detecciones de estas clases es notablemente inferior a las anteriores, y ninguna de ellas aporta información adicional a lo ya analizado previamente. Aun así, en la Tabla 4.1 podemos ver las medias de error para cada clase analizada, incluyendo las clases 'Van' y 'Truck'.

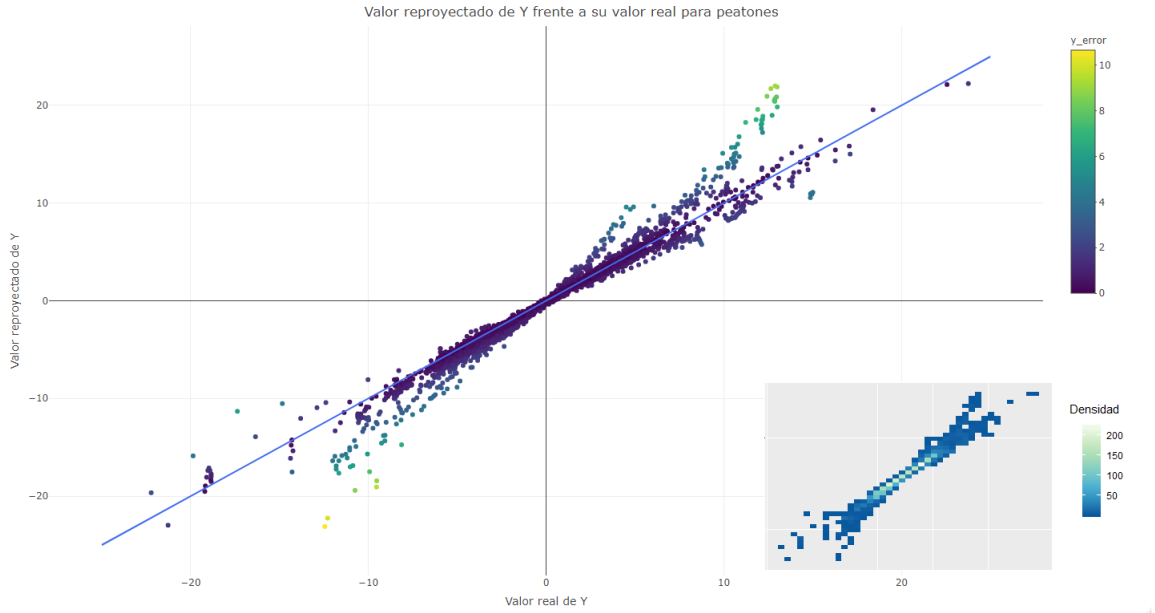


Figura 4.15: Diagrama de dispersión mostrando el valor reproyectado frente al real para peatones. Los puntos están coloreados según su error de reproyección en Y, usando la paleta de colores superior derecha. En la parte inferior podemos ver un *heatmap* con la densidad de los puntos del diagrama de dispersión.

	Error medio en X	Error medio en Y	Error medio total
Coches	2.56 m	1.03 m	2.89 m
Peatones	2.92 m	0.69 m	3.06 m
Furgonetas	2.73 m	1.16 m	3.07 m
Camiones	1.88 m	5.06 m	5.54 m
Todas las detecciones	3.6 m	1.17 m	3.93 m

Tabla 4.1: Error medio de reproyección para los distintos tipos de detecciones, analizados usando el *dataset* de KITTI.

5. CAPÍTULO

Conclusiones y trabajo futuro

Tras analizar esta alternativa de reproyección, que, a diferencia de otros métodos del estado del arte, no usa LiDAR, hemos llegado a las siguientes conclusiones.

En primer lugar, ha sido necesario familiarizarse con temas de conducción autónoma, desde los componentes usados, el estado del arte en cuanto a reproyección de objetos en entornos 3D, software de aplicaciones en tiempo real y el propio vehículo de testeo de Vicomtech CarLOTA. También se han tenido que estudiar temas de visión por computador relacionados con calibración de cámaras, geometría de proyección-reproyección y detección de objetos.

Hemos cumplido con los objetivos del proyecto y añadido características extra como mostrar la nube de puntos generada por el LiDAR o ser capaces de conseguir previamente mapas 3D de la zona sobre la que circula el vehículo, para así ver de forma más clara el movimiento y reproyección de este. El módulo ha sido usado en demostraciones reales, tal y como se describe en el Anexo C.

También hemos podido comprobar cómo la reproyección funciona adecuadamente en entornos con el suelo llano. El problema surge cuando el plano del suelo tiene algún tipo de desnivel, causando que la asunción de reproyección en un plano horizontal no funcione correctamente. En este caso, las reproyecciones, como hemos visto en el análisis de resultados, aparecerán más cerca o más lejos en el visualizador 3D que su posición en el mundo real.

Otro de los problemas que no hemos mencionado durante el proyecto es la pose de la reproyección de los vehículos en el visualizador 3D. Todas las reproyecciones están orientadas hacia el eje delantero del vehículo de testeo CarLOTA y no a su orientación en el mundo real.

Como trabajo futuro sería interesante estudiar formas de detectar el desnivel del plano sobre el que circula el vehículo, así como entrenar una red neuronal que sea capaz de

estimar la pose de los vehículos detectados en la imagen. Otros temas muy interesantes en líneas futuras de trabajo serían crear un *dataset* anotado con nuestras propias cámaras, sobre el que realizar el análisis de reproyección, al igual que con el *dataset* KITTI. Al haberse capturado este *dataset* con las mismas cámaras usadas en el proyecto, podríamos analizar con más detalle el comportamiento de la reproyección y también añadir a este análisis las otras tres zonas de visión, trasera y ambos laterales del vehículo. Por último, estudiar el uso de HD Maps o actualizar y generar los ficheros de Open Street Maps de la zona automáticamente para crear modelos 3D del mapa en tiempo real.

Anexos

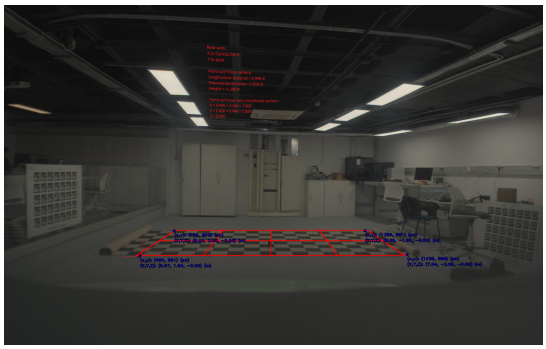
A. ANEXO

Verificación de la calibración.

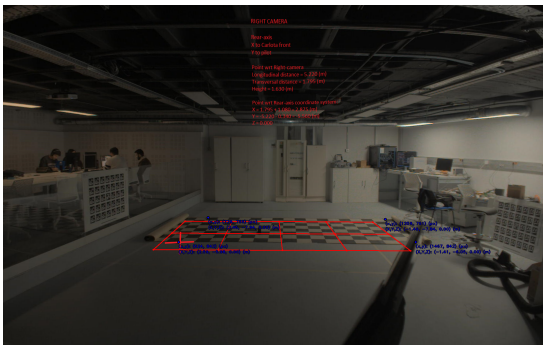
Para comprobar que las calibraciones que hemos realizado eran correctas se plantearon dos alternativas.

La primera de ellas consistía en llevar el coche a algún punto en el que el suelo fuese llano. Aquí, pondríamos conos de tráfico o algún otro objeto a distancias conocidas e iríamos sacando fotos desde las cámaras, para luego más tarde reproyectar la posición de estos y comprobar si la reproyección era precisa o no. El problema de este método era encontrar una superficie totalmente llana de mínimo 40 m^2 en el que no obstaculizásemos la circulación de los demás vehículos. A su vez, las medidas debían de ser totalmente exactas respecto al vehículo de pruebas, para poder comprobarlo con total precisión. Para cada cámara habría que o bien mover el vehículo con exactitud, o mover los conos al campo de visión de cada cámara, complicando aún más el proceso. En el caso de tener conos suficientes, se podrían colocar estos para cada cámara, pero esto implicaría encontrar una superficie llana de mínimo 80 m^2.

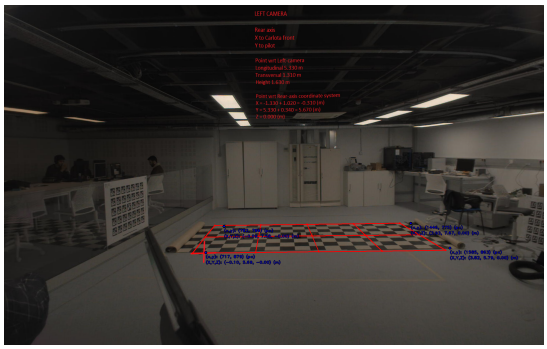
La segunda de ellas, y la que hemos realizado, consiste en lo siguiente: Usando una lona en el suelo con un patrón de cuadrados a modo de ajedrez, que disponía Vicomtech, de 2×4 m con cuadrados de 25 cm, hemos comprobado las distancias de reproyección entre los puntos interiores de los cuadrados, al conocer las distancias entre ellos previamente. Hemos usado el módulo de reproyección y OpenCV para comprobar gráficamente los resultados obtenidos, como podemos ver en la Figura A.1, y hemos realizado esto para las 4 cámaras. En las imágenes podemos comprobar cómo la calibración es precisa para la cámara frontal, pero para la cámara derecha no encajan tan bien los puntos de reproyección. En el futuro, se pretende usar datos obtenidos del LiDAR para poder conseguir mejores parámetros de calibración en las cámaras, y así ser capaces de realizar la tarea de reproyección con mayor precisión.



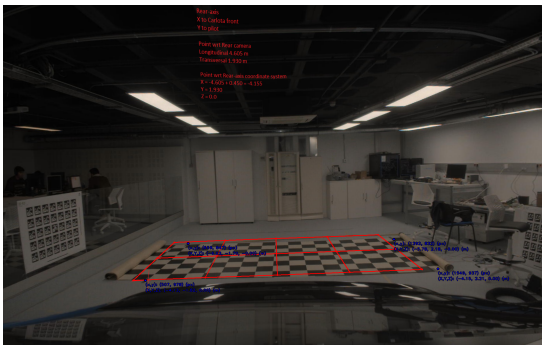
(a) Calibración para la cámara frontal.



(b) Calibración para la cámara derecha.



(c) Calibración para la cámara izquierda.



(d) Calibración para la cámara trasera.

Figura A.1: Verificación de las 4 cámaras para las calibraciones, usando la lona con forma de tablero de ajedrez.

B. ANEXO

RTMaps

El trabajo realizado ha sido desarrollado en RTMaps[1] (*Real Time Multisensor Applications*), software diseñado por la compañía Intempora. Gracias a este software hemos podido interconectar los distintos componentes de CarLOTA. En la Figura B.1, podemos ver un ejemplo de un diagrama de RTMaps.

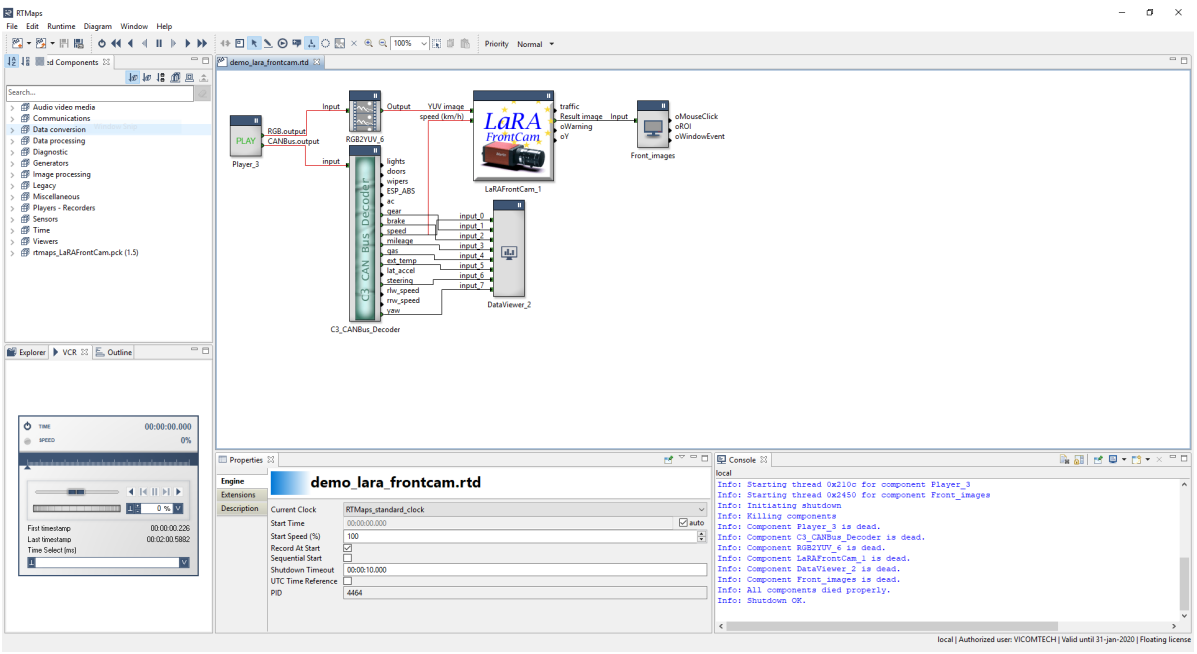


Figura B.1: Ejemplo de un diagrama de RTMaps con distintos componentes interconectados entre sí.

[1] https://intempora.com/products/rtmaps.html

El elemento más importante de RTMaps son los **componentes**. En el diagrama de RTMaps estos son las cajas que disponen de elementos de entrada y salida, por donde se envían información entre sí. Las entradas de un componente se mostrarán en la parte izquierda de este, y las salidas a la derecha. Un mismo componente puede enviar información a más de un componente y recibir información de varios componentes. RTMaps viene con componentes básicos ya instalados, como, visualizador de vídeo, visualizador de datos, reproductor multimedia, grabador de *streamings* de datos, lectores de periféricos, etc. También existen paquetes con componentes instalables como lectores de LiDAR, GPS, funciones de OpenCV, etc. En caso de que no existan componentes que realicen la tarea que necesitemos, podemos programar nuestros propios componentes haciendo uso de C++ o Python, como hemos hecho para el script de reproyección. Otro de los puntos fuertes de RTMaps es que permite grabar en archivos .rec todos los datos que nos convengan en una grabación en tiempo real, para después poder volver a reproducirla *offline*. Para ello, guarda las marcas de tiempo, o *Timestamp*, de todos los datos a grabar, para poder tratarlos después correctamente.

Otra de las características más importantes de RTMaps son los tipos de lectura de los datos de entrada de un componente. Cada *output* de un componente tiene asociado un ***buffer* circular**. Básicamente, cada *buffer* puede contener hasta *N* elementos (*N* imágenes, si el *output* es un vídeo, *N* vectores o matrices, *N* números enteros, etc.), por lo que el *output* escribe en su *buffer* circular, mientras que los *inputs* que leen los datos de salida de ese componente leerán su *buffer* como vemos en la Figura B.2.

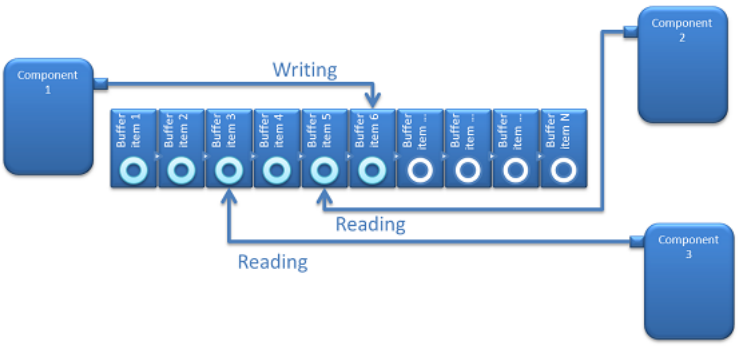


Figura B.2: Dos componentes leyendo la salida desde el *buffer* del componente 1.

Existen distintos métodos de lectura para los componentes. El primero de ellos es el **FIFOReader**. Como sabemos, FIFO son las siglas de *First In, First Out*. El principio de este se basa en lo siguiente:

- Si el *buffer* circular está vacío, o todos los datos del *buffer* han sido leídos, un nuevo intento de lectura de algún componente hará que se bloquee hasta que el componente de salida vuelva a escribir datos nuevos en el *buffer*.

- Si el *buffer* circular tiene datos que todavía no han sido leídos por los demás componentes, un intento de lectura devolverá el dato que más tiempo lleve en el *buffer* que no haya sido leído por ese componente. Una vez un dato ha sido leído por todos los componentes conectados a ese *output*, se borrará del *buffer*.

- Si el *buffer* circular se llena por completo, como podría pasar si un componente que lee no es lo suficientemente rápido leyendo frente a la velocidad de escritura del componente, entonces los nuevos datos se sobrescribirán encima de los datos más antiguos del *buffer*. Esto hará que el componente que lea del *buffer* nunca pueda leer ese dato que se ha perdido, y el siguiente intento de lectura devolverá el dato que más tiempo lleve en el *buffer*.

Este método funcionará perfectamente en el caso de que los componentes puedan leer más rápido que el componente de salida, haciendo que estén sincronizados en tiempo real. En el caso de que el *buffer* sea lo suficientemente grande y el componente lector tarde un poco en leer la información, podrá obtener todos los datos con un poco de latencia. Si por el contrario, no es capaz de seguir el ritmo de escritura del componente de salida, perderá información en el proceso de lectura. Es un buen método para procesar datos asíncronos en tiempo real, o para registrar datos.

El siguiente método es **LastOrNextReader**. Cuando el componente de lectura intenta leer de un *buffer* de salida y no hay datos, se bloqueará como con el método FIFO. Si por el contrario, hay datos sin leer, descartará todos menos el más reciente de todos. De esta forma, no tendremos nada de latencia a la hora de leer un *output*. Su funcionamiento es fácil de entender y su aplicación suele ser solamente para realizar un *display* de datos en pantalla de los datos más recientes.

Después tenemos el método **SamplingReader**. El lector recibirá siempre el último elemento que haya llegado al *buffer*, aunque ya lo haya leído previamente. Esto causará que, si el lector es más veloz que el componente de escritura, leerá varias veces el mismo dato. Por el contrario, si el componente de escritura es más veloz, perderá los últimos datos que hayan llegado al *buffer*. Normalmente se usa para tareas de remuestreo o para control de sistemas y actuadores.

Otro de los métodos de lectura que proporciona RTMaps es **Wait4NextReader**. El lector no tiene en cuenta el *buffer* circular del componente de escritura. Los datos se leen según llegan al componente de lectura. Un intento de lectura siempre esperará a que llegue un nuevo dato, y siempre bloqueará el *stream* hasta que termine de procesar el mismo. Si un dato nuevo llega antes de que se procese el dato previo, se descartará automáticamente. Si el lector es lo suficientemente rápido, los componentes estarán sincronizados a tiempo real y no se perderá información. Por el contrario, si el escritor es más rápido se perderá información en el proceso, pero en ningún momento habrá latencia, como pasaba con el método FIFO. Este método de lectura es perfecto para componentes que tienen que ser estrictamente a tiempo real y se pueden permitir perder datos sin ninguna consecuencia grave.

Por último, tenemos el método **NeverSkippingReader**. Es muy parecido al FIFO, pero en el caso de que el *buffer* circular esté lleno, no sobrescribirá ningún dato hasta que haya sido leído. Esto hará que se bloquee el componente de escritura hasta que tenga espacio en el *buffer* para escribir datos nuevos. Es el único método capaz de influir directamente en el componente de escritura, solo se debe usar en componentes críticos en los que no se pueda perder información. Su uso ha de ser considerado previamente, ya que puede bloquear el diagrama por completo en un uso a tiempo real. Normalmente se usa en el post-procesado de datos, para poder procesar toda la información obtenida.

C. ANEXO

Grabaciones realizadas

Para este proyecto se han realizado varias grabaciones con CarLOTA y una demostración real en un entorno controlado en el ITS (*Inteligent Transport System*) European Congress de 2019 en Eindhoven, Holanda, en el que se usaba el componente de reproyección en tiempo real.

C.1. Miramón

Las primeras grabaciones se realizaron en la zona de Miramón, Donostia, al tener el vehículo de pruebas CarLOTA en el garaje de Vicomtech. El propósito de estas grabaciones fue comprobar que todo el sistema de grabación y detección funcionaba, y hacer uso de estas detecciones *offline* en RTMaps para comprobar si la reproyección hacia su labor correctamente. Se realizaron grabaciones tanto con 4 cámaras como con 3 para preparar un posible caso de uso del proyecto. También hemos realizado grabaciones con LiDAR para probar la precisión de la reproyección. Ninguna de ellas siguió una ruta en concreto si no que nos hemos limitado a circular por la zona para tener datos que usar.

C.2. Simulación demostración Eindhoven

Este TFM forma parte del proyecto VI-DAS [1] (*Vision Inspired Driver Assistance Systems*), proyecto subvencionado por la UE y con colaboradores de toda Europa. Para la demostración que se iba a realizar en Eindhoven, se realizó un taller de integración en

[1] http://www.vi-das.eu/

Vicomtech con los demás participantes europeos del proyecto, incluyéndose entre otros TASS Siemens, IBM, Intel, TomTom, Honda, etc.

Durante el taller de integración, se pusieron en común todos los componentes creados por los participantes y se aseguró el correcto funcionamiento de los mismos. La demo a realizar consistía en un cambio de carril a la izquierda en un entorno controlado, usando como lugar para esta el Automotive Campus de Helmond, a las afueras de Eindhoven.

Para preparar este caso de uso realizamos grabaciones en dos lugares distintos durante el taller de integración. Primero en la zona de hospitales de Donostia, en la carretera que vemos en la Figura C.1, donde hay una incorporación a la izquierda en el carril, debido al comienzo de un carril bus. Junto al vehículo de testeo llevamos dos coches de Vicomtech para simular la demo, dejando espacio para la incorporación. En otras grabaciones no dejamos dicho espacio para comprobar que el vehículo indicaba al conductor que no se podía incorporar.

Figura C.1: Ruta marcada en Google Maps para la grabación simulando la incorporación al carril del ITS en Eindhoven.

El segundo tipo de grabaciones realizadas fueron en la incorporación desde una gasolinera a la autopista AP-8. Esto es debido a que es la única zona sobre la que TomTom contaba con información de HD Maps para realizar comprobaciones extra en cuanto al número y tamaño de carriles sobre la carretera en la que circulábamos. La incorporación se hizo desde la gasolinera para simular el cambio de carril a la izquierda. Podemos ver la zona en la Figura C.2.

Figura C.2: Ruta marcada en Google Maps para la grabación simulando la incorporación al carril en la AP-8 para el uso de HD Maps.

C.3. Demo ITS Eindhoven

La demostración fue realizada con éxito realizando varios casos de uso. Vamos a mostrar el que hemos mencionado previamente: incorporación al carril izquierdo con vehículos circulando por este, podemos ver la maniobra en la Figura C.3.

En la siguiente Figura C.4 podemos ver la visualización de componentes y su función. Estas eran las ventanas que se mostraban a los asistentes de la demo mientras se realizaba a tiempo real. Podemos ver el visualizador 3D junto a las reproyecciones de las cámaras entre ellos.

Por último, añadimos una Figura C.5 más. En esta vemos la visualización que obtenemos del diagrama en tiempo real, junto a CarLOTA en la parte derecha de la imagen, realizando el cambio de carril en la demostración.

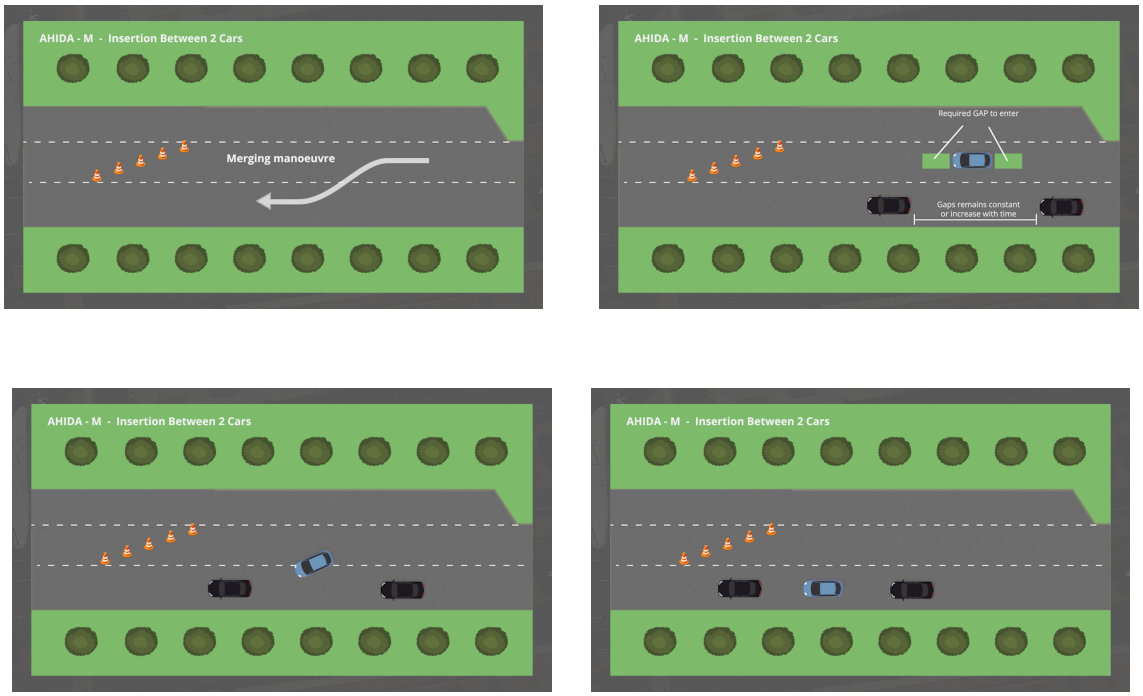


Figura C.3: Maniobra a realizar en uno de los casos de uso del ITS, cambio de carril a la izquierda entre dos vehículos.

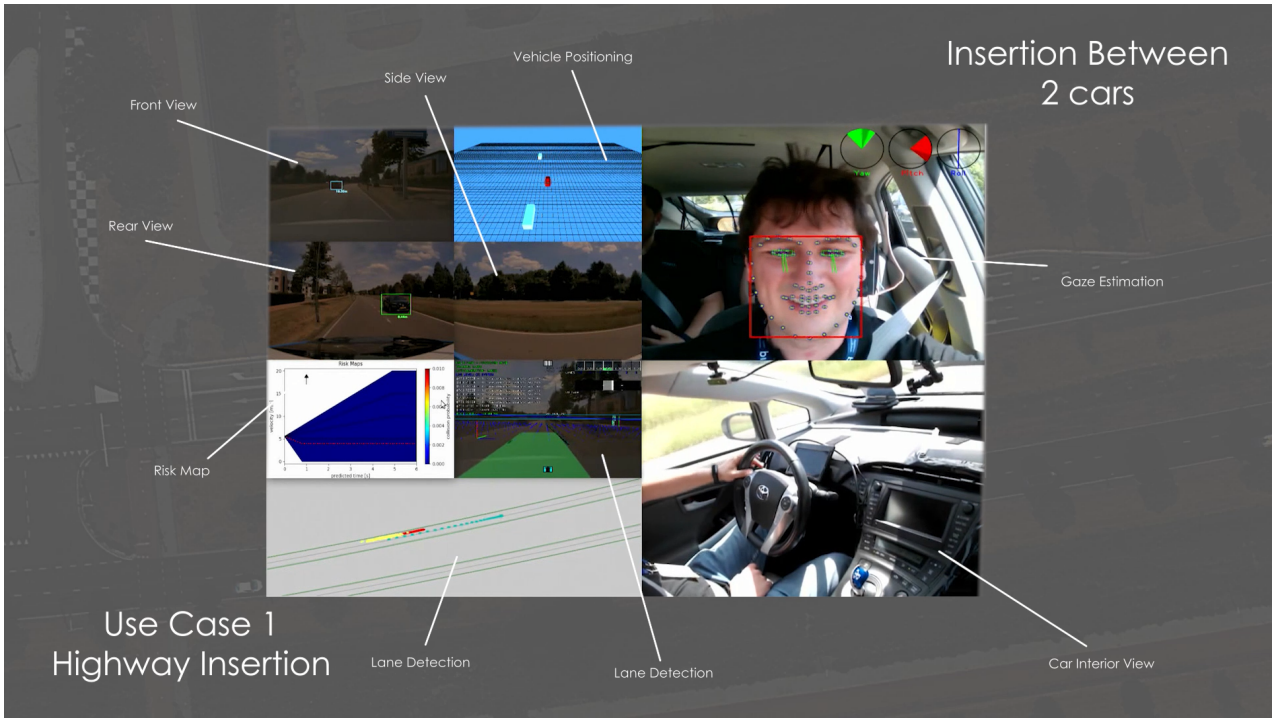


Figura C.4: Todas las ventanas de los componentes del diagrama en tiempo real usado en la demo del ITS.

Figura C.5: Visualización de las ventanas generadas por el diagrama en tiempo real de RTMaps junto a imágenes de CarLOTA incorporándose al carril izquierdo en el ITS.

Bibliografía

[1] L. Liu, W. Ouyang, X. Wang, P. W. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *CoRR*, vol. abs/1809.02165, 2018.

[2] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2 ed., 2003.

[3] Matlab, "What is camera calibration?." https://www.mathworks.com/help/vision/ug/camera-calibration.html.

[4] K. Bimbraw, "Autonomous Cars : Past , Present and Future - A Review of the Developments in the Last Century , the Present Scenario and the Expected Future of Autonomous Cars : Past , Present and Future," no. August, 2016.

[5] M. Xie, L. Trassoudaine, J. Alizon, M. Thonnat, and J. Gallice, "Active and intelligent sensing of road obstacles: Application to the european eureka-prometheus project," in *1993 (4th) International Conference on Computer Vision*, pp. 616–623, May 1993.

[6] L. Davis, D. DeMenthon, R. Gajulapalli, T. Kushner, J. LeMoigne, and P. Veatch, "Vision-based navigation: a status report," *Proc. Image Understanding Workshop*, vol. 1, pp. 153–169, 01 1987.

[7] E. D. Dickmanns, "Vehicles capable of dynamic vision: a new breed of technical beings?," *Artificial Intelligence*, vol. 103, no. 1, pp. 49 – 76, 1998. Artificial Intelligence 40 years later.

[8] A. D. NHTSA Systems, "Automated driving systems 2.0: A vision for safety," 2013.

[9] A. D. NHTSA Systems, "Traffic Safety Stats," no. February, pp. 2–3, 2015.

[10] A. Rosebrock, *Deep Learning for Computer Vision with Python: Starter Bundle*. PyImageSearch, 2017.

[11] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[12] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[13] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, 1969.

[14] P. J. Werbos *et al.*, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] A. Zelinsky, "Learning opencv—computer vision with the opencv library (bradski, g.r. et al.; 2008)[on the shelf]," *IEEE Robotics Automation Magazine*, vol. 16, pp. 100–100, Sep. 2009.

[18] P. Drap and J. Lefèvre, "An Exact Formula for Calculating Inverse Radial," no. 1, pp. 1–18, 2016.

[19] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[20] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, pp. 303–338, June 2010.

[21] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[22] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[23] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[24] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[25] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007, Oct 2017.

[26] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[27] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016.

[28] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from RGB-D data," *CoRR*, vol. abs/1711.08488, 2017.

[29] X. Du, M. H. A. Jr., S. Karaman, and D. Rus, "A general pipeline for 3d detection of vehicles," *CoRR*, vol. abs/1803.00387, 2018.

[30] K. Minemura, H. Liau, A. Monrroy, and S. Kato, "Lmnet: Real-time multiclass object detection on CPU using 3d lidar," *CoRR*, vol. abs/1805.04902, 2018.

[31] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *CoRR*, vol. abs/1608.07916, 2016.

[32] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. García, and A. de la Escalera, "Birdnet: a 3d object detection framework from lidar information," *CoRR*, vol. abs/1805.01195, 2018.

[33] S. Wirges, T. Fischer, J. B. Frias, and C. Stiller, "Object detection and classification in occupancy grid maps using deep convolutional networks," *CoRR*, vol. abs/1805.08689, 2018.

[34] H. Ha, S. Im, J. Park, H. Jeon, and I. S. Kweon, "High-quality depth from uncalibrated small motion clip," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5413–5421, June 2016.

[35] N. Kong and M. J. Black, "Intrinsic depth: Improving depth transfer with intrinsic images," in *IEEE International Conference on Computer Vision (ICCV)*, pp. 3514–3522, Dec. 2015.

[36] K. Karsch, C. Liu, and S. B. Kang, "Depth transfer: Depth extraction from video using non-parametric sampling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, pp. 2144–2158, Nov 2014.

[37] A. Saxena, M. Sun, and A. Y. Ng, "Make3d: Learning 3d scene structure from a single still image," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, pp. 824–840, May 2009.

[38] D. Hoiem, A. A. Efros, and M. Hebert, "Recovering surface layout from an image," *International Journal of Computer Vision*, vol. 75, no. 1, pp. 151–172, 2007.

[39] L. Ladicky, J. Shi, and M. Pollefeys, "Pulling things out of perspective," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 89–96, 2014.

[40] J. Redmon, A. Farhadi, and C. Ap, "YOLOv3 : An Incremental Improvement,"

[41] H. G. Seif and X. Hu, "Autonomous driving in the city" hd maps as a key challenge of the automotive industry," *Engineering*, vol. 2, no. 2, pp. 159 – 162, 2016.

[42] J. Varga, "Conversions between geographical and transverse mercator (utm, gauss-krueger) grid coordinates," *Periodica Polytechnica Civil Engineering*, vol. 27, no. 3-4, pp. 239–251, 1983.