UAH

Segmentación Semántica para Imágenes Fisheye basada en CNN

Master Universitario en Ingeniería de la Telecomunicación
Especialidad en Sistemas de Transporte Inteligente
Departamento de Electrónica

Presentado por:
Álvaro Sáez Contreras

Dirigido por:
Luis Miguel Bergasa Pascual

Alcalá de Henares, a 25 de septiembre de 2018

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Master Universitario en Ingeniería de la Telecomunicación: Especialidad en Sistemas de Transporte Inteligente

Trabajo Fin de Máster

Segmentación Semántica para Imágenes Fisheye basada en CNN

Autor: Álvaro Sáez Contreras

Director: Luis Miguel Bergasa Pascual

Tribunal:

Presidente: Daniel Pizarro Pérez

Vocal 1º: David Fernández Barrero

Vocal 2º: Luis Miguel Bergasa Pascual

Calificación: .....

Fecha: .....

"La fuerza de los valientes, cuando caen, pasa a la flaqueza de los que se levantan."
Miguel de Cervantes

Gracias mamá

Agradecimientos

La gratitud en silencio no sirve a nadie.

Gladys Bronwyn Stern

Este proyecto es el fruto de muchas horas de dedicación y trabajo no solo mías, sino también de todas las personas que se han implicado en el trabajo de forma activa o para darme apoyo en algún momento. A todos ellos me gustaría dedicarles unas palabras de agradecimiento:

En primer lugar, me gustaría agradecer a mi familia todo el apoyo que he recibido durante este año. En especial, quiero dedicar este trabajo a mi madre quién ha aguantado mis etapas de aislamiento y mal humor estoicamente y a mi tía, que siempre se ha interesado en mantenerme bien aconsejado.

A mi profesor y tutor Luis Miguel Bergasa, por encontrar una línea de trabajo tan prometedora, invertir tanto tiempo en ella y saber dirigirme por el camino adecuado. A pesar de lo difícil de este año, ha conseguido ayudarme a compatibilizar el proyecto y el trabajo con mis aventuras personales de manera llevadera. Aprovecho también para agradecer al resto de profesores (Pedro, Rafa, Elena) el tiempo invertido en mi y toda la ayuda prestada durante el curso.

A todos mis compañeros de laboratorio que han conseguido hacer que este año de trabajo haya sido tan llevadero. Gracias por las risas y las clases de hardware a los dos Carlos, Miguel, Felipe, Edu, Rafa, Kailun y a los Javis. También quiero tener unas palabras de agradecimiento para mis compañeros de Máster y de descansos Carlos y Miguel que han compartido conmigo los momentos más duros de este postgrado.

También quiero tener unas palabras de agradecimiento para todas las personas que han colaborado de forma activa en el desarrollo de este trabajo llevándome de paseo para grabar secuencias o ayudándome con el apartado hardware que tan bien se me da. Muchas gracias Marta, Ainhoa, Miguel, Felipe, Javi y Carlos.

Me gustaría volver a recordar mi agradecimiento y estima a los profesionales del Hospital Puerta del Hierro y de La Princesa que siguen insistiendo en arruinar mi carrera como modelo para mantenerme centrado en esta otra línea.

A todos esos amigos que se encargan de hacer viable el trabajo mediante los momentos de risas, relax y despreocupación. Muchas gracias a todos por hacer más facil y mejor la vida, espero seguir contando con vosotros en la etapa que se iniciará tras la entrega de este trabajo.

Resumen

Este trabajo fin de máster (TFM) está centrado en el desarrollo de un sistema de percepción basado en cámaras de campo de visión ultra-amplio de tipo *fisheye* para un prototipo de vehículo autónomo eléctrico. El método escogido para afrontar la comprensión del entorno a partir del sistema es el uso de segmentación semántica mediante redes neuronales convolucionales (CNN) dado que permite cubrir la mayoría de necesidades de un vehículo autónomo de manera unificada. Para ello se desarrollan nuevas arquitecturas de red, mecanismos de data-augmentation y datasets sintéticos específicos para abordar el problema generado por la fuerte distorsión.

Palabras clave: ojo de pez, distorsión, red neuronal, *DeepLearning*, vehículo autónomo.

Abstract

This work aims to develope a new simplified perceptive system for a full autonomous electric vehicle prototype based on ultra-wide field of view cameras. Semantic segmentation is chosen as the way to face scene-understanding as it satisfies most of autonomous vehicle needs in an unified way. For this purpose, specific elements to deal with distortion are designed, such as network architectures, data-augmentation techniques or synthetic fisheye datasets.

Keywords: fisheye, intelligent vehicle, CNN, DeepLearning, distortion.

Resumen extendido

Este trabajo desarrolla técnicas de DeepLearning adaptadas a cámaras de campo de visión ultra-amplio de tipo *fisheye* con objetivo de diseñar un sistema de percepción completo para un prototipo de vehículo autónomo eléctrico desarrollado por el grupo Robesafe de la UAH en colaboración con la universidad de Vigo.

Para afrontar la tarea de percepción del entorno del vehículo se opta por el uso de segmentación semántica, ya que permite la detección de todos los objetos de interés para un vehículo autónomo de forma unificada combinada con las cámaras de campo de visión ultra-amplio, dado que permiten minimizar el número de cámaras a utilizar. El trabajo consta de 3 bloques bien diferenciados:

- Un primer bloque teórico que incluye un amplio estudio del estado del arte de trabajos relacionados con cámaras de ojo de pez y segmentación semántica, así como el modelado matemático de una lente de ojo de pez genérica utilizada para la generación de datasets sintéticos con distorsión característica de este tipo de óptica. En este bloque, además, se realiza el diseño de una nueva arquitectura de red para la CNN de partida de este proyecto (ERFNet) con modificaciones específicas enfocadas al trabajo con imágenes que presentan grandes niveles de distorsión. El bloque abarca los capítulos 1,2,3 y 4.

- El segundo bloque se centra en el desarrollo de técnicas de DeepLearning enfocadas a cámaras *fisheye* entre las que se incluye la realización de entrenamientos con datasets sintéticos utilizando diferentes modelos de red, el desarrollo de nuevos métodos de data-augmentation para imágenes con distorsión incluida y la extracción de resultados para la comparación con otros trabajos similares del estado del arte. Este apartado concluyó con una publicación en el Intelligent Vehicles Symposium 2018 celebrado en China bajo el título *CNN-based Fisheye Image Real-Time Semantic Segmentation*. El bloque ocupa los capítulos 5 y 6.

- Un último bloque enfocado en la adaptación de las técnicas desarrolladas en los apartados anteriores a una cámara de ojo de pez real. Este apartado incluye la calibración de la cámara, la grabación de secuencias en el campus y en entornos urbanos y la integración y test en el prototipo de vehículo autónomo eléctrico de SmartElderlyCar. Este bloque se desarrolla íntegramente en el capítulo 8.

Como elemento adicional, el trabajo incluye un anexo que detalla el desarrollo del sistema de posicionamiento del vehículo basado en sistemas de geoposicionamiento GNSS, desarrollado en paralelo a este trabajo, que constituye la base del sistema de percepción del vehículo dado que la sincronización de datos de todos los sensores se basa en los tiempos del receptor GPS. El desarrollo de este sistema de posicionamiento también culminó con la publicación de un artículo en el congreso internacional Workshop of Physical Agents (WAF) 2018 a celebrar en Madrid en noviembre con título *Positioning system for an electric autonomous vehicle based on the fusion of Multi-GNSS RTK and Odometry by using an Extented Kalman Filter*.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Presentación

El desarrollo de vehículos autónomos, así como de tecnologías e investigaciones asociadas a ellos, se ha incrementado exponencialmente en los últimos años. Este crecimiento ha propiciado el aumento de la incorporación de cámaras en todo tipo de vehículos incluyendo no únicamente los de naturaleza autónoma sino también en los guiados por humanos para proporcionar información adicional al conductor o a los que incorporan los sistemas de asistencia a la conducción para constituir los denominados ADAS (Advanced Driver Assistance Systems). Estos sistemas tienen por objetivo facilitar la conducción mediante el suministro de asistencia o de información útil adicional al piloto, consiguiendo así un incremento de la seguridad del propio vehículo y de la carretera. El grupo Robesafe de la propia UAH ha desarrollado múltiples ADAS que realizan tareas como la detección de líneas de carretera [1], la detección automática de peatones [2] o la percepción del nivel de cansancio del piloto [3].

Las cámaras son un tipo de sensor óptimo para las aplicaciones de conducción autónoma, dado que ofrecen una fuente visual de información fácilmente procesable en tiempo real gracias a los avances computacionales de los últimos años, a partir de la que se pueden extraer con sencillez datos útiles como imágenes de puntos ciegos en los que no existe visión directa para el conductor o información semántica de alto nivel utilizada para detectar elementos tales como carriles, señales de tráfico u otros vehículos presentes en la escena. Además, ofrecen otras ventajas como su reducido coste económico, su bajo consumo de energía o su compatibilidad con la mayoría de sistemas de percepción dado que se trata de sensores pasivos. No obstante, el uso de cámaras simples suele complementarse con el de sensores que permiten recuperar la información de profundidad de la escena como cámaras estéreo, sistemas lidar o radar y con sistemas de geoposicionamiento que permiten localizar los vehículos como GPS, GLONASS y Galileo o sistemas alternativos basados en redes móviles como LTE o UMTS.

Dado que los vehículos autónomos requieren conocer su entorno completo en tiempo real, la información ofrecida por una única cámara es insuficiente. Para solventarlo, los sistemas de percepción que incorporan constan de múltiples cámaras que permiten abarcar prácticamente los 360 grados que los rodean. El uso de más de una cámara incrementa notablemente la complejidad del diseño de los sistemas, debido a la necesidad de incorporar tareas adicionales como las de sincronización en la adquisición de los datos y la de calibración extrínseca del conjunto de cámaras para conseguir que los datos obtenidos sean combinables y el sistema sea funcional. La dificultad del diseño y de su posterior implementación real es proporcional al número de cámaras utilizadas, por lo que la reducción de este parámetro al máximo es un objetivo final del diseño.

La principal estrategia para afrontar la reducción del número de cámaras que componen el sistema de visión de un vehículo autónomo es utilizar cámaras con ángulo de visión amplio (con campos de visión cercanos a 180 grados) o de ángulos de visión ultra-amplios (que abarcan 180 grados o más), superiores siempre al campo de visión estrecho proporcionado por las cámaras tradicionales que proporcionan campos de visión inferiores a los 90 grados.

El tipo de cámaras con campo de visión ultra-amplio más extendido consta de una óptica denominada de ojo de pez o *fisheye*. Con este tipo de dispositivos, teóricamente dos cámaras serían suficientes para cubrir el entorno del vehículo completo. Sin embargo, en diseños reales ya implementados como en [4] se recurre a entre 3 y 4 cámaras para asegurar la robustez del diseño mediante redundancia en la cobertura del entorno.

A cambio de esta simplificación del sistema de percepción final, el uso de este tipo de óptica introduce un problema adicional: la fuerte distorsión que presentan las imágenes generadas. Ésta obliga a utilizar estrategias que permitan lidiar con ella para preservar la robustez de los algoritmos y técnicas de visión empleados.

1.2 Motivación

La conducción autónoma es una tarea difícil y desafiante que requiere soluciones complejas en labores de percepción del entorno. Para afrontar el problema se utilizan las informaciones obtenidas desde múltiples sensores, que son fusionadas para conseguir la comprensión total del entorno. Este conjunto de sensores, además, tiende a ser redundante por motivos de seguridad. No obstante, dadas las múltiples ventajas que ofrecen las cámaras (nombradas en la sección anterior), las últimas investigaciones del estado del arte han enfocado sus esfuerzos en soluciones basadas en visión.

Los procesos de adquisición de datos deben ser seguidos por tareas de más alto nivel tales como la detección, identificación y clasificación de los distintos objetos o regiones presentes en la escena que posibilitan la posterior toma de decisiones. Con dichas tareas se consigue identificar la zona conducible en la escena, el resto de participantes en el tráfico como otros vehículos o peatones, las diferentes señales de tráfico que rigen la circulación, etc.

Las tareas planteadas anteriormente y el conjunto de necesidades de un vehículo autónomo se pueden afrontar de manera unificada mediante el uso de técnicas de segmentación semántica [5], que se han popularizado notablemente en los últimos años gracias a la evolución de frameworks de Deep Learning como Caffe [6], Tensorflow [7], Torch [8] o Pytorch, al desarrollo y abaratamiento de las llamadas Unidades de Procesamiento Gráfico (GPUs), que permiten el entrenamiento de redes neuronales en tiempos razonables incluso en dispositivos embebidos montados a bordo de un vehículo y a la publicación de grandes datasets para el entrenamiento de las redes [9].

Sin embargo, la dificultad de estas actividades puede verse incrementada en determinadas circunstancias como, por ejemplo, en los entornos que son de naturaleza especialmente dinámica e impredecible como los urbanos o en situaciones que requieren mayores volúmenes de información para ser gestionadas correctamente como los cruces, las rotondas o las intersecciones. Esta creciente necesidad de gestionar situaciones complejas de manera eficiente ha incrementado el interés por las cámaras de campo de visión ultra-amplio como las de ojo de pez ya que permiten afrontar dichas circunstancias de manera sencilla como ilustra la figura 1.1 gracias a su mayor campo de visión.

No obstante, en la actualidad la integración de estos dispositivos en vehículos de naturaleza autónoma se limita a los de nivel 1 ó 2 introducidos por la agencia americana NHTSA [10], con el simple objetivo

Figura 1.1: Vista de una intersección desde una cámara fisheye

de proporcionar información adicional al conductor o de alertar de peligros inminentes como una posible colisión. Para integrar las cámaras en vehículos de orden superior capaces de tomar decisiones de manera autónoma es necesario poder extraer datos útiles de más alto nivel a partir de las imágenes capturadas como, por ejemplo, la información semántica.

A pesar de las virtudes que ofrecen las técnicas de Deep Learning como la segmentación semántica, su aplicación a cámaras de óptica no convencional no está muy extendida debido a las dificultades que implica la integración de ambos elementos. Ésta plantea tres grandes problemas que no se dan en los algoritmos de Deep Learning tradicionales:

- La dificultad que plantea gestionar la fuerte distorsión introducida por la óptica de las cámaras de ángulo de visión amplio, que distorsiona fuertemente las formas y texturas de los elementos presentes en la imagen modificando su apariencia y haciendo inviable utilizar las técnicas y algoritmos tradicionales.

- La ausencia de grandes datasets de imágenes anotados a nivel de píxel capturados a partir de este tipo de dispositivos que permitan realizar entrenamientos alternativos para las redes adaptados a este nuevo tipo de óptica.

- La gran cantidad de recursos computacionales requeridos y la complejidad de implementar algoritmos o aplicaciones funcionales en tiempo real, debido a la alta carga que exigen las técnicas de Deep Learning.

La adaptación de las técnicas de Deep Learning existentes a imágenes obtenidas por cámaras de ángulo de visión amplio conseguiría afrontar los dos grandes problemas planteados (satisfacción integral de las necesidades de un vehículo autónomo y gestión de situaciones específicas de mayor complejidad) de manera unificada y, previsiblemente, de forma muy eficiente al estar la solución propuesta basada en una única tarea.

Esta línea de trabajo enlaza con el desarrollo del proyecto SmartElderlyCar del grupo Robesafe de la Universidad de Alcalá, llevado a cabo en colaboración con la Universidad de Vigo.

1.3 Marco de trabajo: SmartElderlyCar

Smart Elderly Car Project propone la investigación en tecnologías para el desarrollo de un coche eléctrico autónomo que asista al sector de población de personas mayores en entornos fundamentalmente urbanos. La propuesta es disruptiva ya que plantea técnicas novedosas aplicadas sobre un futuro coche eléctrico, en donde España es referencia internacional, y focalizadas a un sector de la población con necesidades crecientes. El sistema propuesto está basado en las más avanzadas técnicas de percepción sensorial a bordo del vehículo y de mapeado, control y planificación de trayectorias en entornos dinámicos y cambiantes.

Las estadísticas muestran que el 68 % de la Unión Europea (UE), incluyendo estados asociados, viven en áreas urbanas. Según la Organización Mundial de la Salud, casi un tercio de la población mundial vivirá en ciudades en el año 2030. Conscientes de este problema, el Libro Blanco del Transporte publicado por la Comisión Europea en 2011 indicaba que nuevas formas de movilidad debían ser propuestas hacia soluciones sostenibles de personas y mercancías de forma segura. Respecto a la seguridad, estableció el ambicioso objetivo de reducir a la mitad el número global de muertes en las carreteras de la UE entre 2010-2020. Sin embargo, el objetivo no va a ser fácil, solo en 2014 más de 25.700 personas murieron en las carreteras de la UE (reducción del 18 %), además, hay estudios que muestran que los accidentes mortales se incrementan con la edad a partir de los 65 años. La solución a esta problemática reside en el desarrollo de la conducción autónoma, que además también se presenta como uno de los grandes retos de la industria actual.

La existencia de vehículos autónomos fiables y asequibles económicamente creará un gran impacto en la sociedad afectando a aspectos medioambientales, demográficos, sociales y económicos y presentando todas las ventajas y avances mencionados anteriormente. La conducción autónoma ha atraído gran atención recientemente en los grupos de investigación y en la industria, debido a los espectaculares anuncios de varias empresas sobre previsiones de entrada al mercado. Sin embargo, sus predicciones interesadas parecen muy optimistas. Un organismo más científico, como es IEEE, ha predicho recientemente que para el año 2040 la mayoría de vehículos que circulen por autopistas serán autónomos. La conducción en entornos urbanos tardará más en llegar, debido a su mayor complejidad e incertidumbre.

En el contexto de la segmentación semántica de una escena urbana se han propuesto varias aproximaciones novedosas [11] que han proporcionado una gran mejora hacia una buena solución para este problema. Sin embargo, la mayoría de estos algoritmos trabajan en el plano imagen sin tener en cuenta información relevante como la estructura 3D general o la geometría del objeto. En cuanto a la aproximación del mapeado 3D de una escena urbana, ha sido extensivamente utilizada. Sin embargo, al igual que ocurre con la segmentación semántica, las aproximaciones de mapeo 3D no tienen en cuenta información importante como la categoría de los objetos, la cual es necesaria para entender el complejo entorno urbano y la iteración entre sus componentes. De hecho, la mayoría de los sistemas referenciados elaboran mapas 3D usando varios sensores lidar y después incorporan la información semántica en el mapa en una tediosa fase de anotación manual. Motivado por los inconvenientes de estas aproximaciones parciales, algunos trabajos recientes [12] [13] han explorado la idea de combinar ambos para crear reconstrucciones 3D semánticas para entornos exteriores, las cuales suponen una solución avanzada al problema del entendimiento de escenas urbano. Sin embargo, su uso en aplicaciones reales está todavía limitado debido a la falta de precisión de las actuales aproximaciones y a su extremado coste computacional. Otro problema sin resolver es el mantenimiento de un mapa útil a largo plazo. La actualización del mapa después de grandes cambios de la escena es un problema abierto ya que los actuales métodos de reconocimiento de lugares son incapaces de funcionar con cambios visuales extremos debidos a los cambios de estaciones y otros efectos ambientales [14].

Con estos antecedentes, dos de los retos que se están abordando en este proyecto son precisamente

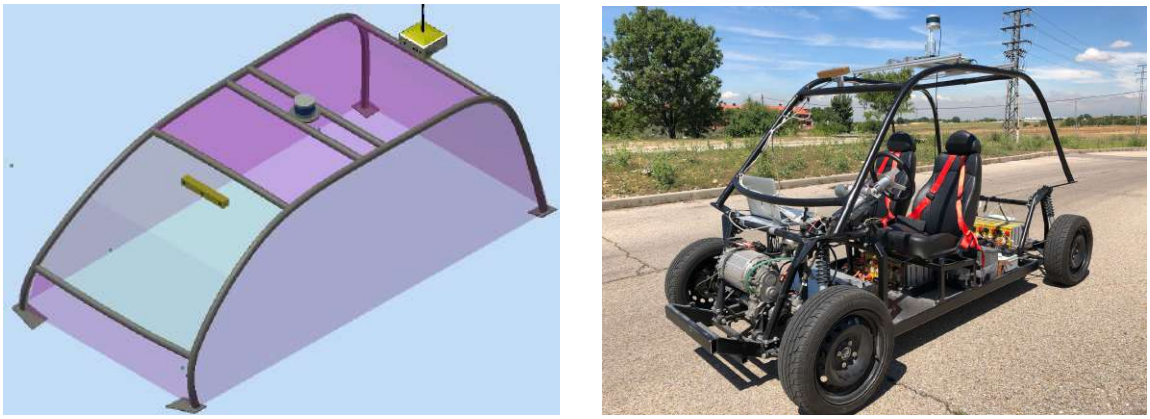


Figura 1.2: Esquema básico de los sensores de percepción de SmartElderlyCar e implementación real

la fusión de información proveniente de lidar 3D y cámaras para la creación de mapas 3D semánticos aplicables a la navegación autónoma de un vehículo real, y la obtención de mapas 3D. En esta línea ya se han realizado algunos trabajos preliminares [14] [15] que han constituido el punto de partida de esta investigación.

La estructura del vehículo se encuentra ya implementada junto con los soportes para los distintos sensores con los que ya se han realizado pruebas de adquisición de datos a bordo para realizar las simulaciones preparatorias de los casos de uso que se pretenden testear en los próximos meses.

Los sensores que componen el sistema de percepción inicial son:

- Una cámara estéreo Bumblebee XB3, posicionada en la parte frontal del vehículo, por encima de la luna y en la zona exterior.
- Un lidar Velodyne Puck Lite de 16 haces, elevado sobre la parte central del vehículo con visión del entorno completo del coche.
- Un sistema GPS diferencial situado en la parte trasera del vehículo que obtendrá correcciones a través de una estación base dedicada instalada en el propio edificio de la escuela con una antena GNSS de anillos concéntricos.
- Un sistema de odometría basado en encoders situado en las ruedas del coche, que fusionará la información obtenida con una IMU.

Dado que el único sensor que proporciona información sobre las regiones laterales y trasera del vehículo es el lidar (que además solo genera información de profundidad) la incorporación de un sistema de cámaras que abarque dichas regiones y que proporcione información semántica resulta atractiva siempre que no introduzca un aumento notable de la complejidad del sistema original, para lo que las cámara *fisheye* resultan óptimas.

1.4 Trabajo relacionado

Los primeros trabajos que buscaban incorporar cámaras *fisheye* a tareas de reconocimiento, identificación y segmentación de objetos o regiones tomaban como aproximación para afrontar el problema la eliminación de la distorsión que presentan las imágenes originales para, así, poder aplicar los algoritmos y métodos

tradicionales. Así, estos trabajos realizaban una corrección de las imágenes obtenidas por las cámaras para después aplicar sobre ellas algoritmos de detección tradicionales como *Local Binary Pattern* (LBP) [16] o *Deformable Part Models* (DPM) [17].

Otros trabajos enfocados en una línea similar persiguieron utilizar las ecuaciones matemáticas del modelo proyectivo de una cámara *pinhole* tradicional para generar imágenes corregidas sobre las que el trabajo fuera más sencillo [18] [19].

A pesar de que las dos líneas de trabajo previas demostraron ser capaces de obtener buenos resultados en diversas tareas, terminaron siendo descartadas debido a su fuerte dependencia de los parámetros de calibración de las cámaras que obligan a realizar calibraciones muy precisas de manera periódica, al consumo de recursos computacionales y de tiempo que exigen las etapas de gestión de la distorsión y a la degradación de la calidad de la imagen original en la que ciertas regiones desaparecen o pierden mucha resolución aunque los parámetros de los que se dispone sean muy precisos.

Como alternativa a estos trabajos las últimas líneas del estado del arte persiguen adaptar los métodos de detección existentes a las imágenes con distorsión originales, salvando las dificultades mencionadas. Para ello, el reciente trabajo presentado en [20] se centra en generar datasets con distorsión característica de las cámaras *fisheye* a partir de datasets existentes como ETH [21]. En el caso anterior se partía de un modelo de imagen de perspectiva esférica para definir la relación entre las imágenes originales y las distorsionadas. El objetivo es llevar a cabo entrenamientos alternativos adaptados a las nuevas cámaras.

En el campo de la segmentación semántica no son muchos los trabajos que han explorado esta línea. En el inicio de este TFM ninguna publicación había conseguido lograr una segmentación semántica de calidad en imágenes obtenidas a partir de una cámara *fisheye* real y tan solo [22] había conseguido mostrar resultados de segmentación semántica correcta en imágenes sintéticas. Para generarlas, se recurría al dataset CityScapes [9] y a la definición de una relación matemática entre la posición de los píxeles en las imágenes originales con otra ubicación en las nuevas imágenes distorsionadas basada en uno de los posibles modelos matemáticos bajo los que se diseñan las lentes *fisheye*. Esta publicación constituye la línea base de partida de este trabajo y su posterior evolución [23] se solapó nuestra investigación.

1.5 Objetivos

Siguiendo la línea de trabajo del proyecto SmartElderlyCar de la UAH, el objetivo inicial de este TFM será el de incorporar un nuevo sistema de percepción basado en cámaras *fisheye* en este vehículo, con objeto de mejorar el sistema de percepción visual inicial diseñado para este proyecto, que en el momento de inicio de este trabajo solo constaba de una cámara estéreo frontal con un campo de visión de 45 grados. Dicha integración deberá ser precedida de una adaptación de las técnicas utilizadas en el proyecto en la actualidad, para poder aprovechar el nuevo tipo de imágenes adquiridas con las mejoras que conllevarán en los resultados.

Dichas técnicas son, fundamentalmente, la obtención de imágenes semánticas mediante redes neuronales convolucionales (CNN) que posteriormente son fusionadas con la información de profundidad del lidar.

El alcance del trabajo abarca:

- El estudio y modelado matemático de la óptica de ojo de pez para permitir el correcto desarrollo del trabajo.

- La generación de datasets sintéticos con distorsión específica de ojo de pez basados en el modelo matemático desarrollado previamente a partir de otros convencionales de uso extendido para posibilitar los posteriores entrenamientos.
- El proceso de estudio inicial de la red ERFNet y de posibles redes alternativas, para realizar una selección de la mejor red de segmentación semántica para la aplicación final.
- El diseño de un entrenamiento específico para la red final escogida valorando varios procesos y mecanismos de mejora como distintos tipos de data augmentation genéricos o específicos para cámaras *fisheye*.
- El desarrollo de una aplicación (orientada a funcionar en tiempo real) capaz de realizar segmentación semántica de las imágenes sintéticas generadas.
- La evaluación de la posibilidad de modificar la arquitectura básica inicial de la red en base a pruebas con variaciones de la red original inspiradas en propuestas específicas para este tipo de imágenes.
- La selección y compra de las cámaras utilizadas para la obtención de imágenes así como su integración en el vehículo prototipo.
- El proceso de calibración del sistema de las cámaras adquiridas.
- La adaptación de la red entrenada para la generación de imágenes semánticas del entorno del vehículo utilizando la nueva óptica incorporada que incrementará el campo de visión del sistema empleado y, por tanto, la cantidad de información obtenida para analizar la escena
- El testeo de la aplicación final y de las técnicas desarrolladas tanto mediante el uso de imágenes sintéticas generadas a partir de otras bases de datos ya existentes (como KITTI o Cityscapes) como mediante imágenes obtenidas a través de las cámaras seleccionadas.

Dado que los resultados cuantitativos obtenidos a lo largo del TFM han sido buenos, un objetivo adicional ha sido la comparativa constante con los distintos trabajos similares publicados en el estado del arte y la presentación de los resultados obtenidos en publicaciones internacionales.

1.6 Estructura

Este trabajo está estructurado en 8 capítulos. Tras este primer capítulo de introducción se presentan otros 3 de naturaleza teórica: el segundo capítulo analiza en detalle la óptica de las cámaras de campo de visión ultra-amplio así como diferentes modelos de cámara de ojo de pez; el tercero desarrolla conceptos fundamentales sobre las redes neuronales convolucionales, estudia la arquitectura original de la ERFNet tomada como línea base para este trabajo y propone un arquitectura alternativa para la red y el cuarto capítulo explica el mecanismo utilizado para generar datasets sintéticos en base a las ecuaciones matemáticas obtenidas en el primer capítulo y justifica la selección del dataset utilizado para la generación de imágenes. El quinto capítulo estudia diferentes tipos de data-augmentation específicos para imágenes de ojo de pez y presenta los resultados de los primeros experimentos realizados. El sexto capítulo evalúa los resultados de diferentes redes sobre los datasets generados y el séptimo se centra en adaptar las técnicas desarrolladas a una cámara real. El último capítulo expone las conclusiones extraídas del proyecto y las futuras líneas de trabajo a desarrollar.

Capítulo 2

Geometría Fisheye

2.1 Introducción

Desde la década de los 90 se han propuesto múltiples maneras de modelar las lentes ópticas con el objetivo de poder satisfacer las necesidades de tareas como la reconstrucción de entornos 3D o la creación de imágenes panorámicas mediante *stitching*. En particular, los modelados basados en polinomios paramétricos son actualmente muy utilizados, dado que son computacionalmente muy ligeros y ofrecen buenos resultados, en tareas como la evaluación de lentes fotográficas, la visión computarizada o en aplicaciones de robótica. El caso de las lentes de ojo de pez es especialmente complicado, dado que este tipo de óptica presenta múltiples variantes y no hay una única versión genérica que la identifique.

Este capítulo persigue la obtención de un modelo matemático que permita describir el funcionamiento de una cámara o lente de ojo de pez. A modo de introducción, se realiza un análisis de la geometría de la óptica de las cámaras tradicionales, que es seguido por uno de la óptica asociada a las cámaras *fisheye*. Además, se exponen los distintos métodos de clasificación de estas cámaras así como de los diferentes modelos matemáticos que permiten diseñar este tipo de lentes u otras similares de ángulo de visión muy amplio.

2.2 Modelo cámara pinhole

El modelo más extendido para caracterizar las propiedades de las cámaras tradicionales es el de la cámara *pinhole* o estenopeica. Se trata de un modelo extremadamente simple que consiste en un único espacio estanco a la luz con un pequeño orificio denominado estenopo que permite el paso de la luz y de un material fotosensible capaz de captar la luz que atraviesa el orificio de entrada. En este material se conforma la imagen final captada por la cámara, manteniendo una relación de escala con la escena del mundo real captada proporcional a la denominada distancia focal de la cámara. Este parámetro se corresponde con la distancia existente entre el orificio de la cámara y el material sobre el que se compone la imagen. La figura 2.1 ilustra el esquema básico del modelo.

En la figura, un objeto de tamaño $\mathbf{X}$ situado a una distancia $\mathbf{Z}$ del centro de la cámara representado por $\mathbf{C}$, se proyecta sobre el plano imagen con centro de coordenadas $\mathbf{p}$ en el punto x. La distancia focal $\boldsymbol{f}$ en esta situación se corresponde con la distancia que separa $\mathbf{C}$ y $\mathbf{p}$. Esta proyección puede resumirse mediante la ecuación 2.1.

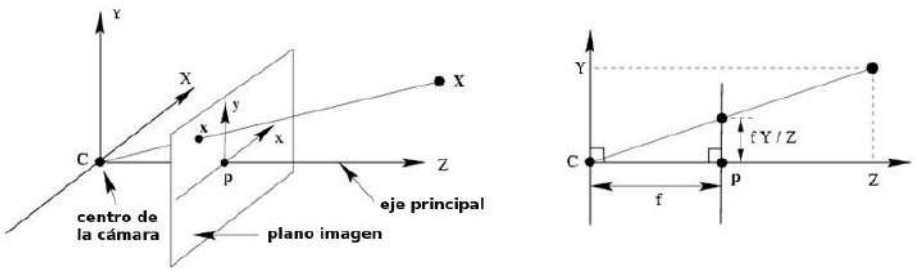


Figura 2.1: Modelo genérico de una cámara estenopeica.

$$-x = f\frac{X}{Z} \tag{2.1}$$

2.3 Óptica Fisheye

La óptica característica de ojo de pez genera campos de visión ultra-amplios (de incluso más de 180 grados) al doblarse la luz incidente en la lente, permitiendo el paso de rayos que en otros casos (como en el modelo tradicional *pinhole*) no consiguen entrar en el sensor con la consecuente pérdida de información. La entrada de luz en una lente *fisheye* y en una cámara tradicional aparecen representadas en la 2.2:

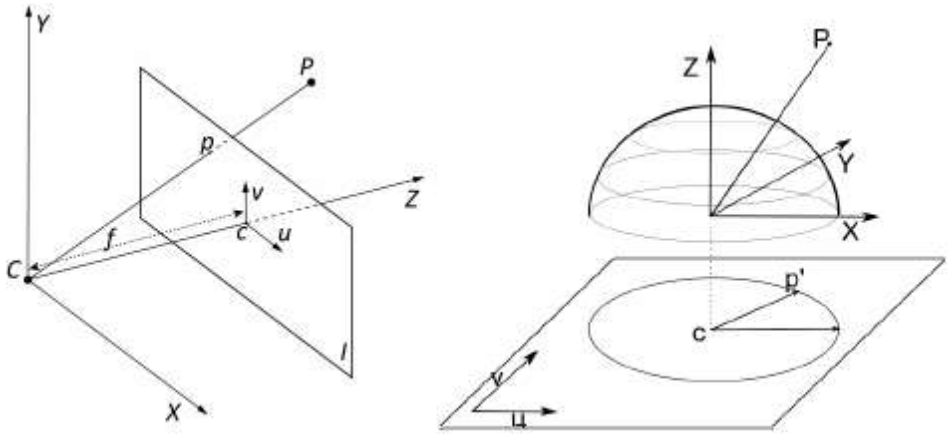


Figura 2.2: Comparativa modelos de ojo de pez en base al campo de visión.

En la figura anterior los ejes X, Y y Z representan los ejes de coordenadas del mundo real, los ejes u y v los ejes de la imagen generada, c el punto principal de la imagen (centro),P el rayo incidente del mundo real que se proyecta sobre el plano de la imagen y p' la coordenada final de la imagen sobre la que queda proyectado.

Como consecuencia del modo en el que la luz se curva al entrar en la lente, la imagen final generada (como se puede observar en la figura anterior) tiene forma circular y, además, presenta una distorsión característica que aumenta al alejarse de la zona central de la imagen siendo mucho más intensa en las zonas periféricas del círculo proyectado. Esta distorsión es mayoritariamente de naturaleza radial al igual que otros tipos de distorsiones bien conocidas como la de barril o la de cojín, aunque puede contener componentes tangenciales de menor intensidad. La distorsión altera las formas y texturas de los elementos presentes en la imagen final y es la causante de que los algoritmos de detección tradicionales no puedan ser aplicados a sistemas de percepción basados en este modelo óptico. Una cuadrícula con distorsión genérica de ojo de pez aplicada sobre ella presenta la siguiente apariencia:

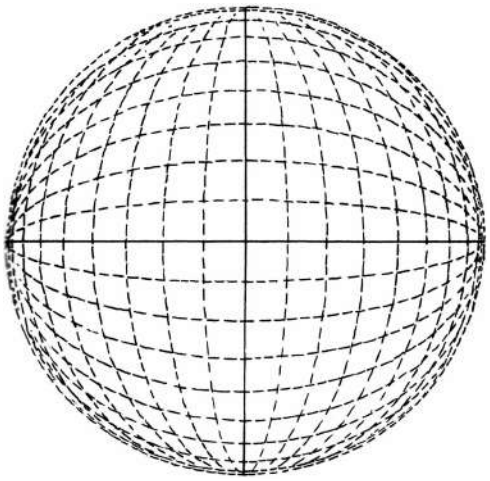


Figura 2.3: Cuadrícula de 10 x 10 con distorsión de ojo de pez.

Como se puede observar en la figura, los cuadrados centrales presentan una distorsión de carácter leve, mientras que los del borde de del círculo han perdido su apariencia original por completo. Las zonas periféricas de la imagen son, por tanto, las zonas críticas a tratar.

La distorsión expuesta en la figura es, sin embargo, de naturaleza genérica dado que no existe un tipo único de ojo de pez y la distorsión varía dependiendo del tipo estudiado. Las dos siguientes secciones de este capítulo estudian los tipos de ojo de pez existentes en base al modelo proyectivo de la cámara y en base al tipo de imagen final generada.

2.4 Clasificación según modelo proyectivo

El modelo proyectivo de una cámara define cómo se conforma la imagen en su correspondiente plano en base a la forma en la que los rayos de luz llegan al sensor óptico. Para el modelo de cámara *pinhole* se sigue la siguiente ecuación matemática:

$$\rho_{pinhole} = ftan(\theta) \tag{2.2}$$

En la ecuación anterior se relacionan el ángulo de entrada de la luz en la lente con respecto al eje principal θ y la distancia focal de la cámara f con la distancia existente entre el punto principal de la imagen y el punto final en el que se proyecta el rayo incidente en la imagen.

Para las cámaras de ojo de pez no existe un modelo proyectivo matemático unívoco que describa la relación sino que existen un conjunto de modelos matemáticos empleados por los fabricantes para modelar sus lentes. Algunos de los más extendidos son los modelos de la proyección del ángulo equisólido (*equisolid angle projection*) que se refleja en la ecuación 2.6 o el *fisheye* equidistante representado en la ecuación 2.3, pero existen otros modelos menos utilizados que se basan en proyecciones esféricas tradicionales como el modelo estereográfico o el ortográfico, descritos por las ecuaciones 2.4 y 2.5 respectivamente. Sus ecuaciones asociadas aparecen a continuación:

$$\rho_{equidistante} = f\theta \tag{2.3}$$

$$\rho_{estereografica} = 2ftan(\theta/2) \tag{2.4}$$

$$\rho_{ortogonal} = fsin(\theta) \tag{2.5}$$

$$\rho_{equisolid} = 2fsin(\theta/2) \tag{2.6}$$

Como se ha mencionado anteriormente, la distorsión que presentan los diferentes tipos de cámaras, a pesar de ser de la misma naturaleza, es distinta. En las imágenes generadas por los diversos tipos de óptica, la distancia entre el punto central de la imagen y la proyección de cada uno de los rayos que entran en el sensor varía dependiendo del modelo utilizado. Esto aparece ilustrado en la figura 2.4, en la que el trazo verde representa la proyección en una cámara estenopeica, el morado el de un ojo de pez estereográfico, el azul uno equidistante, el amarillo uno ojo de pez de ángulo equisólido y el rojo uno de tipo ortográfico.

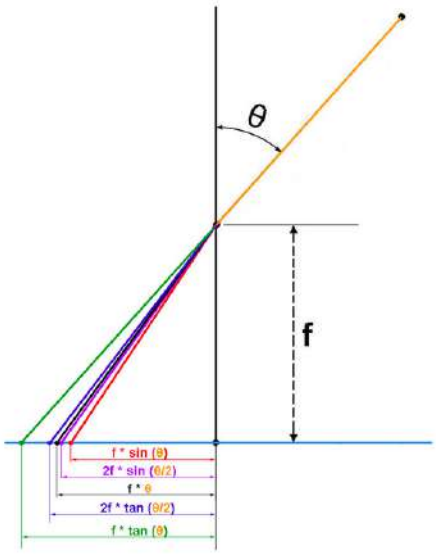


Figura 2.4: Proyección de la imagen de los diferentes modelos de ojo de pez.

Una manera extendida de cuantificar la distorsión que introducen los distintos tipos de lentes es representar la desviación de los píxeles de la imagen con respecto al centro óptico en función de el ángulo de incidencia de la luz en el dispositivo con respecto al eje óptico principal. Ésta desviación se referencia a la distancia focal de las cámaras para poder realizar comparativas estandarizadas. En la figura 2.5 se representa esta distorsión para los modelos expuestos anteriormente. En ella el eje de ordenadas representa el cociente entre la proyección del rayo incidente en la lente y la distancia focal de la cámara y el de abscisas el ángulo de llegada de la luz a la lente que varía entre 0 y π.

El primer el modelo que aparece en la figura es el de la cámara tradicional *pinhole* en el que la longitud del rayo de proyección se dispara asintóticamente antes de alcanzar los 90 grados debido al reducido campo de visión de este tipo que hace que la luz con ángulos de incidencia mayores se pierda y no aparezca en la imagen proyectada final. Los otros cuatro modelos que aparecen son los *fisheye* tradicionales.

El primer modelo *fisheye* que se observa en la gráfica es el estereográfico, que matemáticamente es similar al *pinhole*. Sin embargo, su radio de proyección no se dispara asintóticamente hasta los 180 grados, lo que permite obtener un campo de visión generalmente inferior a dicho ángulo aunque mucho más amplio que el que ofrece la cámara de óptica tradicional. El modelo de la cámara *fisheye* equidistante presenta una distorsión homogénea con respecto al ángulo de incidencia de la luz en la lente (la ecuación matemática para la proyección es una recta). Matemática y computacionalmente es el modelo más sencillo de representar y, además, permite obtener un campo de visión muy amplio (mayor de 180 grados), motivo por el que es el tipo de ojo de pez más extendido, tanto a nivel de fabricantes de lentes como a nivel de modelado en aplicaciones de visión artificial. El modelo del ángulo equisolid es el que mayores campos

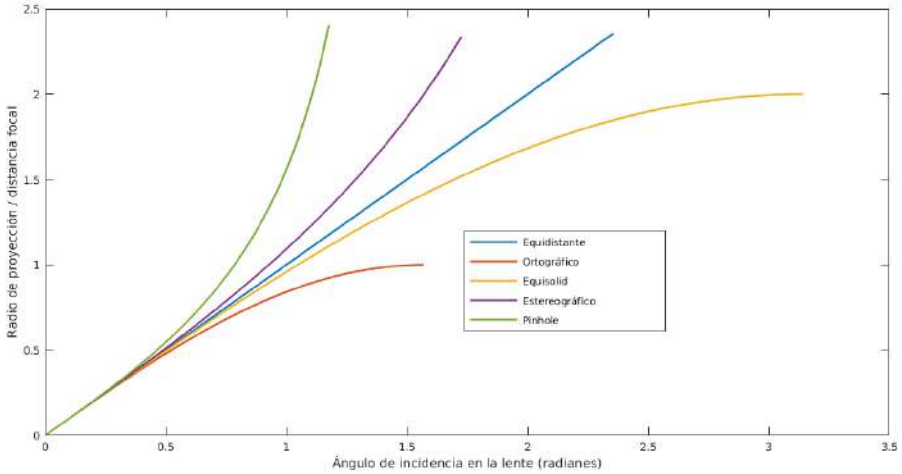


Figura 2.5: Distorsión para los diferentes modelos de ojo de pez.

de visión puede llegar a proporcionar, siendo incluso muy superiores a los 180 grados ofrecidos por la mayoría del resto de modelos (hasta 235 grados). Por el contrario, el último modelo ortográfico solo permite alcanzar campos de visión inferiores a 180 grados.

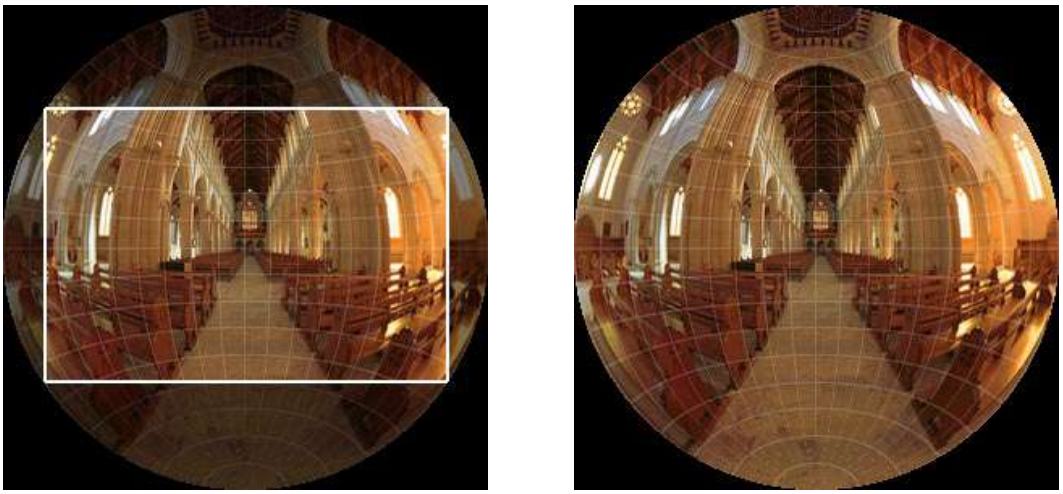
2.5 Clasificación según el campo de visión

En base al campo de visión de una cámara podemos distinguir dos tipos de ojo de pez: el ojo de pez circular, en el que la imagen capturada abarca casi por completo, exceptuando una regiones vacías que aparecen en los bordes y esquinas, el marco de la imagen y el ojo de pez completo o *fullframe fisheye*, en el que no existen regiones negras dado que la imagen circular abarca el marco del sensor por completo, perdiéndose parte de la información en la captura de la imagen. No obstante, ambos tipos siguen las mismas ecuaciones de proyección y, por tanto, el tipo circular puede transformarse en el modelo completo actuando sobre el sensor: el tipo circular puede transformarse en el completo utilizando un sensor más pequeño, un adaptador o lente con zoom.

La diferencia final entre ambos tipos es que el modelo *fullframe* consigue el campo de visión de 180º a lo largo de su diagonal completa mientras que el campo de visión de la versión circular no consigue dicho valor ni en el campo de visión horizontal ni en el vertical que suelen rondar los 150 y los 100 grados respectivamente. El tipo circular tiende a utilizarse en aplicaciones de imagen amateur como la fotografía paisajística o de interiores, dado que a pesar de la pérdida de información no presenta las regiones negras exteriores. Los dos tipos aparecen ilustrados en la figura 2.6:

2.6 Modelado mediante parámetros de calibración

La calibración de una cámara de ojo de pez, a diferencia de una calibración tradicional, implica dos procesos principales: la determinación de un modelo proyectivo adecuado para la cámara estudiada y el cálculo de los parámetros específicos que caracterizan la cámara. Sin embargo, la mayoría de herramientas de calibración disponibles se limitan a utilizar un modelo proyectivo matemático sencillo y a calcular un conjunto de parámetros que minimicen el error para ese modelo, lo que desemboca en resultados de mala calidad.



(a) Modelo de ojo de pez fullframe. (b) Modelo de ojo de pez circular.

Figura 2.6: Modelos de ojo de pez en base al campo de visión.

2.6.1 Parámetros intrínsecos, extrínsecos y coeficientes de distorsión

El uso de los diferentes modelos matemáticos proyectivos disponibles para afrontar la gestión de la distorsión que introducen este tipo de cámaras es la estrategia más recurrida en el estado del arte actual y la escogida en este trabajo aunque puede ser complementada mediante el uso de los parámetros intrínsecos de la cámara. La ecuación 2.7 muestra los parámetros intrínsecos y extrínsecos de una cámara tradicional ideal:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.7}$$

En la anterior ecuación la primera matriz representa las coordenadas de un píxel en la imagen (u,v), junto con una tercera coordenada (1) que identifica el plano imagen. El valor s añadido al comienzo de la ecuación es una constante de escala que indica que ambos lados de la igualdad, a pesar de ser proporcionales, no tienen por qué ser iguales.

La primera matriz del otro lado de la igualdad representa los parámetros intrínsecos de la cámara que incluyen la coordenada en X de la distancia focal f_x, la coordenada en Y de la distancia focal f_y, el coeficiente de *skew* γ (que define el ángulo existente entre los ejes) y las coordenadas del centro de la imagen en X e Y x_0 e y_0.

La segunda matriz contiene los parámetros de calibración extrínsecos que indican el posicionamiento y orientación de la cámara en la escena. La matriz incluye una submatriz de rotación 3x3 construida a partir del producto de 3 matrices de rotación que indican el ángulo de giro de la cámara con respecto a cada uno de los tres ejes de coordenadas y una submatriz de traslación de 3x1 que indica el desplazamiento en unidades del mundo real con respecto al origen 3D de coordenadas cartesianas. Las matrices de rotación con respecto a cada uno de los ejes aparecen en las ecuaciones 2.8 2.9 y 2.10.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\gamma) & -sin(\gamma) \\ 0 & sin(\gamma) & cos(\gamma) \end{bmatrix} \tag{2.8}$$

$$R_y = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \tag{2.9}$$

$$R_z = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

La tercera y última matriz representa las coordenadas de un punto genérico (*x,y,z*) que se proyecta en la imagen, junto con una cuarta coordenada añadida para adaptar los tamaños de las matrices. En la práctica, los valores presentados anteriormente difieren levemente dado que las matrices de las cámaras reales no contienen elementos con valor 0 ó 1 sino con valores muy cercanos a ellos.

La obtención de este conjunto de parámetros es simple mediante el uso de un objeto fácilmente identificable y de geometría conocida, como un tablero de ajedrez para las cámaras tradicionales, pero es más compleja para el caso de las cámaras con campo de visión ultra-amplio, dado que los mecanismos para calibrar tradicionalmente no son válidos para estas cámaras debido a su modelo proyectivo y a su amplio ángulo de visión: un objeto situado en el mismo plano de la cámara es perceptible para las cámaras con campo de visión ultra-amplio debido a que son capaces de capturar objetos a 180 grados. No obstante, esto implica ubicar el objeto en el plano $Z = 0$ en la ecuación 2.1, lo que matemáticamente es inviable, dado que el modelo tradicional ubicaría dichos objetos en el infinito. Los mecanismos para obtener los parámetros de calibración intrínsecos de una cámara *fisheye* difieren por tanto de los utilizados habitualmente.

Sin embargo, dado que no ofrecen información sobre la distorsión introducida, este conjunto de parámetros no es suficiente para modelar el funcionamiento de la cámara. La distorsión de una cámara *pinhole* habitualmente se modela mediante 5 coeficientes: 3 para modelar la distorsión radial y otros 2 para la tangencial. Las ecuaciones utilizadas para corregir los dos tipos de distorsiones principales de una cámara estándar son las siguientes:

$$x_{corregida} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.11}$$

$$y_{corregida} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.12}$$

$$x_{corregida} = x + [2p_1 xy + p_2(r^2 + 2x^2)]\theta \tag{2.13}$$

$$y_{corregida} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{2.14}$$

Los parámetros k_1, k_2 y k_3 de las ecuaciones 2.11 y 2.12 modelan la distorsión radial de la cámara, mientras que los parámetros p_1, p_2 en 2.13 y 2.14 se ocupan de la tangencial. Por otro lado, la variable θ representa el ángulo de conformado entre el píxel analizado y los ejes principales de la imagen.

En las cámaras de campo de visión ultra-amplio existen varias aproximaciones propuestas para afrontar el problema de la caracterización de la distorsión, pero no existe una solución exacta y el camino a seguir depende de las necesidades de la tarea a realizar. A continuación, se plantean varias de las opciones viables a la hora de modelar la distorsión mediante los parámetros extraíbles de la calibración de las cámaras y para la obtención de los parámetros intrínsecos.

2.6.1.1 Aproximación mediante parámetros tradicionales (OpenCV)

La librería de visión computarizada OpenCV proporciona una herramienta propia para llevar a cabo la calibración y extracción de parámetros de cámaras de ojo de pez. Esta herramienta es una aproximación sencilla que busca calcular los parámetros intrínsecos a partir de un modelo de cámara *pinhole* estándar y añadir a posteriori la distorsión característica del ojo de pez. El modelo de cámara tradicional utilizado es el siguiente:

Se toma como punto de partida la ecuación 2.7 para describir cómo una imagen de una escena se forma proyectando los puntos del espacio 3D en el plano imagen. Esta ecuación puede reescribirse como 2.15

$$sm' = A[R|t]M' \tag{2.15}$$

donde:

- M' es una matriz 4x1 (X,Y,Z,1) que contiene las coordenadas de un punto 3D del espacio en unidades del mundo real.
- m' es una matriz 3x1 (u,v,1) que contiene las coordenadas de un punto proyectado en píxeles.
- A es la matriz de parámetros intrínsecos de la cámara.
- R es la matriz de parámetros extrínsecos de rotación de la cámara.
- t es la matriz de parámetros extrínsecos de traslación de la cámara.

Las matrices R y t permiten trasladar las coordenadas del punto del mundo real a otro sistema fijo con respecto a la cámara, siempre que $z \neq 0$. Está transformación viene descrita por la ecuación 2.16.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{2.16}$$

Las coordenadas normalizadas finales del punto con respecto al sistema de referencia de la cámara pueden escribirse como 2.17 2.18:

$$x' = \frac{x}{z} \tag{2.17}$$

$$y' = \frac{y}{z} \tag{2.18}$$

El radio de dicha coordenada con respecto al punto principal de la imagen puede calcularse como 2.19:

$$r = (x')^2 + (y')^2 \tag{2.19}$$

Finalmente, estas coordenadas junto con los parámetros intrínsecos de la cámara permiten calcular la posición del píxel que ocupará dicho punto en la imagen según las ecuaciones 2.20 y 2.21:

$$u = f_x x' + c_x \tag{2.20}$$

$$v = f_y y' + c_y \tag{2.21}$$

donde los valores c_x y c_y designan el centro óptico de la imagen. No obstante, estás coordenadas pueden no ser precisas si la cámara utilizada tiene distorsión. La mayoría de cámaras presentan distorsión de naturaleza radial y en menor medida componentes tangenciales. Por ello, en ocasiones es necesario añadir otra corrección a las coordenadas obtenidas en las ecuaciones 2.17 y 2.18. Esta corrección se realiza en base a los coeficientes de distorsión $(k_1, k_2, k_3, k_4, k_5, k_6, p_1, p_2, s_1, s_2, s_3, s_4)$ que devuelve la herramienta. Los parámetros s_n modelan la denominada distorsión del prisma delgado, consecuencia de la ligera inclinación de la lente de la cámara o del sensor de la imagen, los parámetros p_n se corresponden con los de distorsión tangencial del modelo básico de cámara y los k_n son los coeficientes ampliados utilizados para modelar la distorsión radial que en el caso de OpenCV se corresponden con los seis coeficientes de un cociente polinómico (el modelo básico solo tiene 4 coeficientes). Matemáticamente se realiza de manera similar a las ecuaciones 2.11, 2.12, 2.13 y 2.14 como se indica en 2.22 2.23.

$$x'' = x'(\frac{1 + k_1r^2 + k_2r^4 + k_3r^6}{1 + k_4r^2 + k_5r^4 + k_6r^6}) + 2p_1x'y' + p_2(r^2 + 2(x')^2) + s_1r^2 + s_2r^4 \tag{2.22}$$

$$y'' = y'(\frac{1 + k_1r^2 + k_2r^4 + k_3r^6}{1 + k_4r^2 + k_5r^4 + k_6r^6}) + p_1(r^2 + 2(y')^2) + 2p_2x'y' + s_1r^2 + s_2r^4 \tag{2.23}$$

Nuevamente, estas coordenadas pueden transformarse en píxeles mediante el uso de los parámetros intrínsecos de la cámara según 2.24 y 2.25:

$$u = f_xx'' + c_x \tag{2.24}$$

$$v = f_yy'' + c_y \tag{2.25}$$

Una vez caracterizado el modelo de cámara tradicional utilizado en OpenCV, se realiza la adición de distorsión característica de la cámara de ojo de pez, descrita matemáticamente mediante la ecuación 2.26.

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \tag{2.26}$$

Esta distorsión puede añadirse a las coordenadas originales que definen la posición del objeto proyectado en la imagen mediante las ecuaciones 2.27 y 2.28

$$x_{fisheye} = \frac{\theta_d}{r}x \tag{2.27}$$

$$y_{fisheye} = \frac{\theta_d}{r}y \tag{2.28}$$

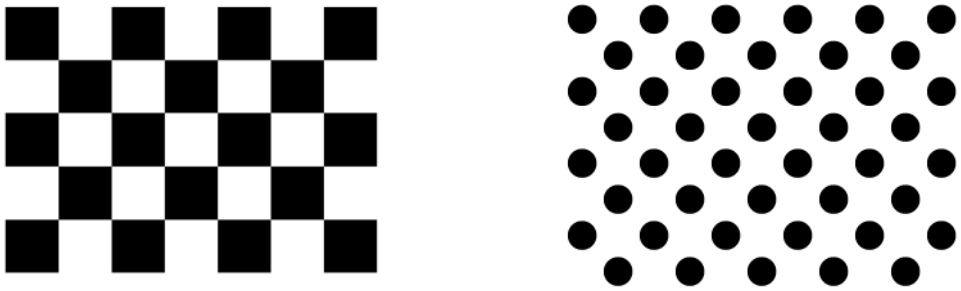
Finalmente, del mismo modo que se realizaba en el modelo original de cámara estenopeica se puede realizar una conversión a coordenadas de la imagen final generada mediante las ecuaciones 2.29 y 2.30

$$u_{fisheye} = f_x(x_{fisheye} + \gamma y_{fisheye}) + c_{fx} \tag{2.29}$$

$$v_{fisheye} = f_yyy + c_{fy} \tag{2.30}$$

El método de calibración de la cámara de OpenCV está basado en [24]. Para poder aplicarse requiere de un patrón de calibración de dimensiones geométricas conocidas. La opción más recurrida es utilizar un tablero de ajedrez, pero existen alternativas como patrones de círculos de dimensiones conocidas. La propia documentación de OpenCV proporciona imágenes para generar dichos patrones, mostrados en la figura 2.7.

La propia toolbox de calibración también ofrece herramientas para eliminar la distorsión de la cámara calibrada y rectificar las imágenes de entrada. En lel capítulo 7 de este trabajo se ofrece un ejemplo



(a) Patrón de calibración cuadrado. (b) Patrón de calibración circular.

Figura 2.7: Modelos de patrón de calibración.

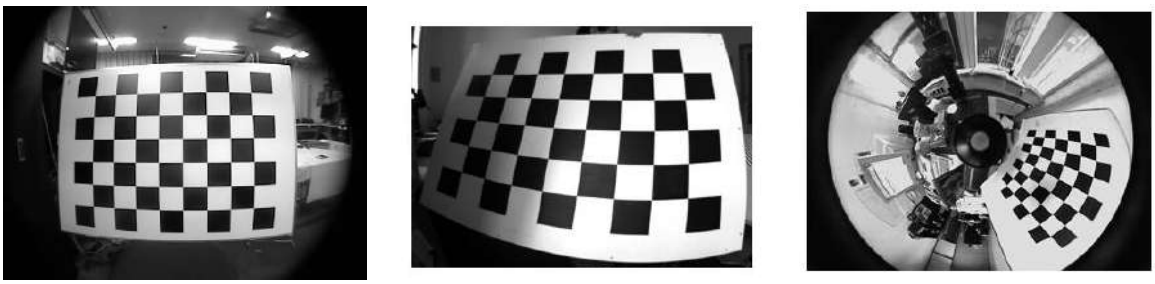
de uso de esta herramienta, aunque los resultados finales no son buenos por lo que es necesario evaluar alternativas.

2.6.1.2 Aproximación de Scaramuzza

Las calibraciones llevadas a cabo en el software de MATLAB son más precisas que las ofrecidas por OpenCV. En el caso de las cámaras de ojo de pez, las propias librerías del software proporcionan una herramienta para calibrar cámaras *fisheye* a partir de la versión de 2017. Sin embargo, existen otras toolboxes que suponen mejoras alternativas dado que utilizan modelos más elaborados como el propuesto por Scaramuzza [25].

La herramienta proporcionada por esta toolbox puede utilizarse para calibrar un extenso conjunto de cámaras de amplio campo de visión incluyendo omnidireccionales, de ojo de pez o de ángulo de visión amplio 2.8. En general, puede considerarse como una opción viable para afrontar la calibración de cualquier cámara con una componente de distorsión radial grande.

Debido a la incapacidad del modelo de la cámara *pinhole* para describir el comportamiento de las cámaras con ángulos de visión amplios (incluso incluyendo coeficientes de distorsión de manera adicional al modelo como con la aplicación de OpenCV), Scaramuzza propone un nuevo modelo de cámara específico destinado a gestionar campos de visión amplios.



(a) Fisheye. (b) Ángulo de visión amplio. (c) Omnidireccional.

Figura 2.8: Modelos de cámara procesables en la toolbox de Scaramuzza.

El modelo propuesto no solo realiza una corrección de la posición final de los píxeles en base a un modelo de distorsión, sino que realiza el cálculo de un vector $(x, y, z)^T$ que apunta desde un punto de referencia a una esfera de imagen en la dirección del rayo de luz entrante para cada píxel $(u, v)^T$ de la imagen de ojo de pez. La figura 2.9 describe la forma de construir la esfera imagen:

Para modelar la distorsión radial de la cámara se define una función basada en un polinomio de Taylor cuyos coeficientes están formados a partir los parámetros de calibración intrínsecos de la cámara y de la

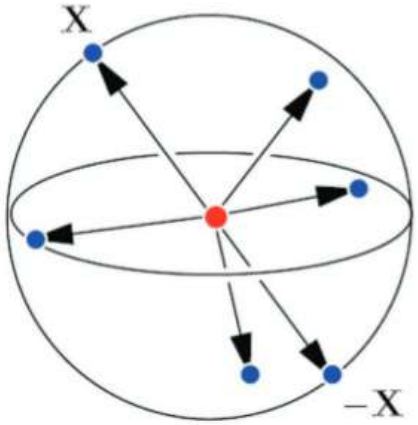


Figura 2.9: Modelo esférico de Scaramuzza.

distancia de cada uno de los píxeles de la imagen al punto principal de la misma. Esta función sigue la ecuación 2.31:

$$g(p) = (a_0 + a_1 p + a_2 p^2 + a_3 p^3 + ... + a_n p^n) \tag{2.31}$$

siendo:

$$p = \sqrt{u^2 + v^2} \tag{2.32}$$

La función definida en 2.31 es necesaria para poder hacer uso del modelo de proyección esférico definido anteriormente, dado que permite que siempre se pueda realizar la proyección de los puntos desde el sistema de coordenadas de la cámara como un punto de un rayo específico. Esto se muestra en la ecuación 2.33:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = s \begin{bmatrix} u \\ v \\ g(p) \end{bmatrix} + t \tag{2.33}$$

La ecuación 2.31 debe ser resuelta para poder proyectar los puntos en la imagen y, por tanto, obtener las coordenadas (u,v) finales de cada punto. Computacionalmente resolver esta ecuación es un proceso complejo e ineficiente, por lo que la toolbox de Scaramuzza define una aproximación basada en un polinomio de Taylor inverso (ecuación 2.34) que define la posición radial de los píxeles en la imagen (su distancia con respecto al centro) en función del ángulo de llegada del rayo a la cámara con respecto al eje de coordenadas z (ecuación 2.36).

$$f(\alpha) = (b_0 + b_1 \alpha + b_2 \alpha^2 + b_3 \alpha^3 + ...b_n \alpha^n) \tag{2.34}$$

$$r = (x)^2 + (y)^2 \tag{2.35}$$

$$\alpha = arctan(z/r) \tag{2.36}$$

A partir de dichas funciones los puntos del sistema de referencia de la cámara son proyectables a una imagen mediante las ecuaciones 2.37 y 2.38:

$$u = \frac{x}{r} f(\alpha) \tag{2.37}$$

$$v = \frac{y}{r} f(\alpha) \tag{2.38}$$

En el capítulo 7 se realiza la calibración de una cámara real mediante una toolbox basada en el modelo de Scaramuzza.

Capítulo 3

Generación de un dataset sintético basado en CityScapes

3.1 Introducción

Uno de los principales motivos del éxito de los técnicas basadas en el uso de DeepLearning y de las redes neuronales convolucionales es la aparición de grandes datasets de imágenes anotadas para permitir el entrenamiento de redes con diferentes finalidades. Éstas incluyen tareas como la detección de objetos, la clasificación de imágenes, la estimación del flujo de una escena, el cálculo de la disparidad de una secuencia estéreo, el seguimiento de objetos, la estimación de la odometría a partir de imágenes, la segmentación de elementos o regiones incluso a nivel de píxel. Para esta última finalidad, la aparición de grandes datasets ha sido fundamental debido al enorme esfuerzo que supondría tener que anotar un gran volumen imágenes a nivel de píxel.

Estos datasets cubren escenarios de diferente naturaleza: desde entornos cerrados e interiores como COCO [26], a otros abiertos con escenas de múltiple naturaleza (urbana, suburbana, áreas rurales o autopistas) como [27] o entornos muy específicos, como los urbanos de CityScapes [9], lo que permite seleccionar el más adecuado para el objetivo final. Además, estos datasets suelen contener imágenes tomadas en diferentes condiciones, lo que desemboca en una mejor capacidad de generalización por parte de las redes entrenadas.

En los últimos años, de manera paralela a la publicación de datasets anotados, se han comenzado a publicar también simuladores que permiten generar imágenes sintéticas acompañadas de un ground truth que constituyen otra posible fuente de datos para abordar la tarea de entrenamiento de las redes.

Este capítulo aborda el análisis de los datasets y simuladores utilizados durante el desarrollo del trabajo, las métricas asociadas a cada uno de ellos para evaluar los resultados obtenidos en la fase de test y el análisis del procedimiento utilizado para la generación de datasets sintéticos adaptados a la nueva óptica de ojo de pez a utilizar.

3.2 CityScapes

Cityscapes es un dataset de imágenes urbanas grabadas desde una cámara montada en un vehículo en 50 ciudades alemanas diferentes. El dataset incluye un total de 5000 imágenes con anotaciones a nivel de píxel de alta calidad y otras 20000 con ground-truth de menor calidad. El conjunto de imágenes con

anotaciones finas se encuentra dividido en 3 conjuntos menores: el primero de ellos es el usado para entrenamiento, que tiene un total de 2975 imágenes y del que se proporciona tanto el ground-truth como las imágenes reales; el segundo conjunto es el de validación, compuesto por un total de 500 imágenes de las que también se proporciona el ground-truth y el último conjunto es el de test, con un total de 1525 imágenes de las que no se ha publicado el ground-truth, pero para el que se proporciona una herramienta de evaluación online.

Se trata de un dataset con un número de imágenes de un orden de magnitud mayor que las anteriores propuestas que, además, incluye imágenes tomadas en condiciones muy diversas, incluyendo diferentes horas del día, distintos meses y estaciones del año y diferentes condiciones climatológicas persiguiendo así proporcionar mejores capacidades de generalización a las redes. El dataset fue creado principalmente con dos grandes objetivos:

- Evaluar el rendimiento de los algoritmos de visión orientados a la comprensión semántica de escenas urbanas, ya sea a nivel de píxel o a nivel de instancia.
- Impulsar las investigaciones que buscan aprovechar grandes volúmenes de datos débilmente anotados, como por ejemplo los entrenamientos de redes neuronales profundas.

No obstante, de manera complementaria a las imágenes básicas capturadas y a sus correspondientes anotaciones, orientadas a tareas de segmentación semántica, se proporcionan otros tipos de datos que pueden resultar útiles:

- Coordenadas GPS extraídas durante la captura de datos.
- Datos de ego-motion extraídos del sistema de odometría incluido en el vehículo
- Temperatura ambiente obtenida a partir de un sensor incorporado en el vehículo.
- Imágenes adicionales capturadas desde otra cámara para componer un par estéreo y poder extraer información de profundidad de la escena. .

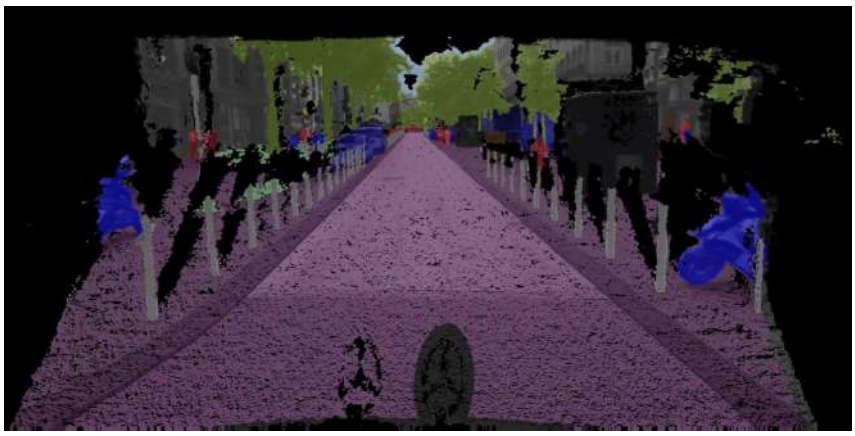


Figura 3.1: Ejemplo de imagen de profundidad en Cityscapes.

- Imágenes adicionales que preceden y siguen a las básicas para añadirles un breve contexto temporal.

De manera adicional al conjunto de imágenes y datos proporcionados por el dataset, también se ofrece software escrito en Python para poder gestionar con facilidad los datos anotados, una herramienta

para anotar imágenes siguiendo las etiquetas definidas en el dataset y un servidor web para evaluar los resultados obtenidos en los conjuntos de test y de validación.

El dataset es completamente gratuito para finalidades académicas o no comerciales y puede descargarse de manera sencilla desde su página web, aunque algunos elementos solo son descargables previo registro en la web y bajo petición.

3.2.1 Clases y política de etiquetado

El dataset se etiqueta con un total de 30 clases agrupadas en 8 grandes grupos. Sin embargo, del total de 30 clases sólo 20 son utilizadas durante el entrenamiento de redes debido a las políticas de evaluación definidas en las herramientas proporcionadas. A continuación se adjuntan las clases utilizadas para entrenamiento y los grupos a los que pertenecen, junto con una breve descripción de la misma proporcionada por la documentación de la base de datos que persigue evitar ambigüedades:

(a) Imagen original. (b) Imagen etiquetada.

Figura 3.2: Ejemplos de imágenes y ground-truth de Cityscapes.

- **Plano:** a este grupo pertenecen todas las clases asociadas al suelo:
  - **Carretera:** parte del suelo por la que se desplazan los vehículos, incluyendo todas las marcas dibujadas en ellos como señales o las líneas que delimitan los carriles.
  - **Acera:** parte del suelo destinada al desplazamiento de peatones y de ciclistas incluyendo los bordillos, tradicionalmente situadas a los lados de la carretera, pero también se incluyen otras zonas a las que los vehículos no tienen acceso.
- **Humano:** en este grupo se incluyen todas las clases asociadas a las personas incluidas en la imagen:
  - **Peatón:** esta clase incluye todos los humanos presentes en la escena que no se estén desplazando con ayuda de algún elemento adicional, así como todos los objetos que transporte como maletas, mochilas o carteras.
  - **Ciclista:** incluye todos los humanos que se desplace mediante la ayuda de algún elemento como bicicletas, motocicletas, scooters, patines, monopatines, sillas de ruedas o coches descapotables.

- **Naturaleza:** grupo en el que se incluyen las clases asociadas a terrenos no generados por el humano y los elementos naturales presentes en la escena:
  - **Vegetación:** incluye todos los tipos de vegetación vertical presentes en la escena, con la única excepción de aquella que crece adherida a un edificio, que se etiqueta como tal a no ser que ocupe una superficie mayor del 20 % del mismo.
  - **Terreno:** contiene todos los tipos de hierba, cualquier tipo de vegetación horizontal, arena o tierra por los que los vehículos no puedan circular.
- **Vehículo:** engloba todos los elementos utilizados por el humano para desplazarse presentes en la escena:
  - **Coche:** incluye todos los coches o jeeps de forma continua presentes en la escena.
  - **Camión:** agrupa camiones (incluyendo tanto el vehículo como la carga que transporta) y camionetas.
  - **Autobuses:** clase que incluye todo vehículo de transporte público capaz de llevar a un grupo grande de personas.
  - **Bicicleta:** incluye las bicicletas presentes en la escena (sin incluir el conductor).
  - **Motocicleta:** motocicletas, ciclomotores o scooters (sin incluir el conductor).
  - **Tren:** engloba los trenes y tranvía presentes en la imagen.
- **Construcción:** incluye toda estructura artificial presente en la escena:
  - **Edificio:** agrupa casas, rascacielos, paradas de transporte cubiertas, garajes o aparcamientos cubiertos.
  - **Muro:** cualquier muro independiente de los edificios de la escena.
  - **Valla:** incluyen todas las vallas presentes en la escena, incluyendo sus agujeros.
- **Cielo:** grupo con una única clase que incluye el cielo abierto y los cables finos presentes en la escena situados delante de él, como los de la electricidad.
- **Objetos:** incluye el conjunto de elementos destinados a permitir o a regular la circulación:
  - **Poste:** cualquier poste fino (orientado mayoritariamente de manera vertical) destinado a sostener diferentes elementos que se etiquetan también como poste si no están incluidas en alguna otra clase de las presentadas.
  - **Señal de tráfico:** engloba todas las señales presentes en la escena para regular la circulación y para proveer información a los conductores y peatones sin incluir sus postes.
  - **Semáforo:** incluye los semáforos sin tener en cuenta el poste.
- **Sin clasificar:** incluye todos los elementos y estructuras que no encajen con ninguna de las clases presentadas anteriormente. Esta clase no se utiliza en el entrenamiento y es ignorada en los procesos de evaluación.

Las 10 clases restantes no utilizadas para el entrenamiento incluyen elementos como caravanas, plazas de aparcamiento, vías de tren, túneles, puentes, guardarraíles o grupos de postes. Éstas clases aparecen en las imágenes etiquetadas, pero son convertidas a la clase *Sin clasificar* por los scripts proporcionados para gestionar los datos de entrada.

La política de etiquetado del dataset es muy estricta y no permite que los objetos anotados tengan agujeros a través de los que sean visibles otros objetos: si un elemento es visible a través de otro objeto de primer plano (como la ventanilla de un vehículo o el espacio entre las ramas de un árbol) éste se clasifica como el objeto del primer plano. Ésta define también un código de colores para representar cada una de las clases mostradas anteriormente que viene detallado en los scrips proporcionados para complementar las imágenes. Este código se muestra en la figura 3.3. .

Figura 3.3: Código de colores de Cityscapes.

3.2.2 Métricas de evaluación

Para evaluar los resultados obtenidos sobre el dataset se establece una métrica basada en el índice de Jaccard [28]. Este valor mide la similitud entre dos conjuntos de datos de cualquier naturaleza. La técnica utilizada es calcular el cociente de la intersección de ambos conjuntos entre la unión de los mismos. La formulación para dos conjuntos arbitrarios A y B es la siguiente:

$$J(A, B) = |A \cap B|/|A \cup B| \tag{3.1}$$

Se trata de la métrica utilizada en el desafío PASCAL VOC [29], habitualmente conocida como *Intersection-over-Union* (IoU). La manera de definir este valor viene dado por la ecuación 3.2

$$IoU = TP/(TP + FP + FN) \tag{3.2}$$

donde para cada categoría o clase el valor TP (*True Positives*) representa el número de píxeles asociados a dicha clase o categoría etiquetados correctamente, FP (*False Positives*) los píxeles asociados a otras categorías pero etiquetados como la evaluada y FN (*False Negatives*) los píxeles asociados a la categoría estudiada pero etiquetados con otra errónea.

La evaluación final para el subconjunto de test se realiza considerando todos los píxeles asociados a

cada una de las clases o categorías dentro de dicho grupo de manera conjunta.Una vez obtenido el valor final para dada clase se realiza el cálculo de la media aritmética de IoU_{clase} o de $IoU_{categoría}$ para dar el resultado final sobre el conjunto de test. Todos los píxeles asociados a clases que no se incluyen dentro de las de entrenamiento se deben etiquetar en la clase *Sin Clasificar* durante las tareas de entrenamiento y de evaluación para ser ignorados.

3.3 Generación de un dataset artificial

Uno de los grandes problemas para aplicar las técnicas de DeepLearning existentes en el estado del arte sobre cámaras de óptica no convencional como las de ojo de pez (planteados en el capítulo 1) es la ausencia de datasets de gran tamaño anotados a nivel de píxeles capturados desde este tipo de cámaras. La solución más aceptada en el estado del arte para afrontar el problema es la generación y uso de datasets sintéticos a partir de otros ya existentes.

Cityscapes supone un dataset óptimo para el estudio del funcionamiento de las cámaras de campo de visión ultra amplio, debido a que los entornos urbanos son altamente dinámicos y presentan situaciones difíciles de gestionar con cámaras tradicionales, como las que se dan en rotondas, intersecciones o cruces. Además, se trata de un dataset de uso muy extendido por lo que es fácil poder establecer comparativas con otros trabajos realizados previamente.

Del estudio previo presentado en el capítulo 2 se puede concluir que la manera más acertada de modelar la geometría presentada por una cámara de ojo de pez es mediante el uso de un modelo proyectivo específico que se adapte bien a la cámara a utilizar posteriormente. El modelo más extendido por las cámaras de ojo de pez venía descrito por la ecuación 2.3, que modelaba las cámaras de ojo de pez equidistantes. Dado que las imágenes del dataset original fueron obtenidas mediante una cámara *pinhole* tradicional, para poder definir una relación de transformación entre el dataset original y el nuevo sintético hay que recurrir también a la ecuación 2.2. Así, el punto de partida lo constituye el siguiente sistema de ecuaciones:

$$\begin{aligned} \rho_{fisheye} &= f\theta \\ \rho_{pinhole} &= ftan(\theta) \end{aligned} \tag{3.3}$$

La relación de transformación perseguida debe de establecer una correspondencia entre los píxeles de la imagen original $P_c(x_c, y_c)$ y los de la transformada $P_f(x_f, y_f)$ en función de los parámetros genéricos de una cámara. Despejando el ángulo de entrada en la cámara θ en ambas ecuaciones obtenemos el siguiente sistema:

$$\begin{aligned} \theta_{fisheye} &= \frac{\rho_{pinhole}}{f} \\ \theta_{pinhole} &= arctan(\frac{\rho_{pinhole}}{f}) \end{aligned} \tag{3.4}$$

Además, también se conoce la expresión que describe la distancia del punto principal de cada una de las dos imágenes a un píxel genérico en la misma, que viene dada por una sencilla distancia euclídea entre el centro de las distorsión de la imagen (u_{px}, v_{py}) y las coordenadas del píxel estudiado (x_p, y_{py}):

$$R_{pinhole} = \sqrt{(x_p - u_{px})^2 + (y_p - v_{py})^2} \tag{3.5}$$

$$R_{fisheye} = \sqrt{(x_f - u_{fx})^2 + (y_f - v_{fy})^2} \tag{3.6}$$

El procedimiento propuesto en [22] plantea obtener una relación a partir de las ecuaciones 3.6, 3.5 y el sistema 3.4, empleando la igualación de los ángulos de entrada en la lente de los rayos en ambos modelos para ello. Con estas 4 ecuaciones se puede deducir la ecuación 3.7:

$$\sqrt{(x_p - u_{px})^2 + (y_p - v_{py})^2} = ftan(\frac{\sqrt{(x_f - u_{fx})^2 + (y_f - v_{fy})^2}}{f}) \tag{3.7}$$

Teniendo en cuenta que la distorsión tangencial es habitualmente reducida, por lo que el ángulo de proyección de los píxeles se conserva para ambos modelos, se puede considerar también la ecuación 3.8, en la que m refleja el ángulo generado entre la proyección de un píxel de la imagen y el eje principal de ésta:

$$m = \frac{y_p}{x_p} = \frac{y_f}{x_f} \tag{3.8}$$

A partir de las 2 ecuaciones planteadas previamente se pueden obtener las coordenadas que ocuparían las asociadas a un píxel de una imagen tradicional en otra de ojo de pez. No obstante, la resolución de dicho sistema es computacionalmente compleja dado que las coordenadas aparecen elevadas al cuadrado y la solución final es doble y hay que discriminar uno de los valores obtenidos. Una solución simple es trabajar utilizando coordenadas polares, indicando la localización de un píxel como en la ecuación 3.9:

$$\begin{aligned} u_f &= R_f cos(\theta) \\ v_f &= R_f sin(\theta) \end{aligned} \tag{3.9}$$

siendo R_f el radio del píxel con respecto al centro de la imagen (que constituye además el centro de la distorsión, dado que ésta es radial) y θ el ángulo que conforma el radio del píxel con el eje u de la imagen. Trabajando con coordenadas polares las ecuaciones presentadas anteriormente se pueden simplificar a las ecuaciones 3.10 y 3.11:

$$R_c = ftan(\frac{R_f}{f}) \tag{3.10}$$

$$arctan(\theta) = \frac{y_p}{x_p} = \frac{y_f}{x_f} \tag{3.11}$$

y dado que tanto θ como el radio asociado a cada píxel en la imagen sin distorsionar son conocidos, éstos se pueden calcular para la imagen distorsionada en función del parámetro f que representa una distancia focal genérica para la cámara simulada. Una vez calculados, se pueden utilizar para calcular la posición final de los píxeles de la imagen original en la sintética de ojo de pez mediante las ecuaciones 3.9.

A la hora de aplicar la transformación, hay que prestar atención a dos consideraciones sensibles: en primer lugar, el centro de la distorsión es el de la imagen, por lo que hay que aplicar un desplazamiento en las coordenadas de la imagen igual a la mitad de sus columnas en el eje u y al de sus filas en el v, para que el punto central de la imagen se corresponda con el origen de coordenadas (0,0). En segundo lugar, hay que estudiar por separado el caso de los píxeles que pertenecen al ángulo $\theta = \pi/2$, dado que el valor de la tangente para este caso es infinito lo que computacionalmente genera problemas. Para solventar el problema, tras la estimación de la longitud del radio asociado a los píxeles en la imagen de ojo de pez, el valor obtenido representa la componente en v y la de u es 0. Además, existe otro caso especial que se incluye dentro de este y es el del centro de la imagen. Este punto debe mantener su posición y no debe desplazarse, dado que constituye el centro de la distorsión.

Utilizando el desarrollo matemático expuesto y Cityscapes se generó un dataset sintético de imágenes de ojo de pez tomando como distancia focal base $f_0 = 159$ con el objetivo de poder comparar los resultados con los obtenidos por otros métodos del estado del arte. Para ello, se distorsionan tanto las imágenes RGB como el ground-truth asociado a ellas, utilizando exactamente la misma transformación tanto para el conjunto de entrenamiento como para el de validación, pero no para el de test dado que sus imágenes de ground-truth no se encuentran liberadas.

Las imágenes originales de entrada son de alta resolución (2048x1024 píxeles). El proceso de transformación introduce una pérdida de escala muy importante dependiente de la distancia focal escogida para el proceso. Para el caso de $f_0 = 159$ la resolución de salida de las imágenes es de 451 x 403 lo que significa que no solo se ha modificado el valor del tamaño de la imagen, sino que también se ha modificado el *aspect-ratio* de la misma (la relación entre los tamaños horizontal y vertical de la imagen) que ahora toma por valor 1,1191 en vez de 2 como en la original. A medida que el valor f seleccionado aumenta también lo hace el tamaño de la imagen de salida junto con su *aspect-ratio*, pero la distorsión introducida es menor. Sin embargo, este tamaño no es adecuado, dado que para ser utilizada por nuestra red se exige que estos valores sean múltiplos de 8.

La selección del tamaño final de la imagen de salida se debe tomar en base a tres criterios:

- Se debe buscar mantener el *aspect-ratio* obtenido en la transformación en la medida de lo posible para no penalizar la forma de los elementos presentes en la imagen.
- Se debe buscar obtener una escala mayor sin que esto implique una pérdida notable de resolución o una degeneración de las texturas de los objetos.
- En el proceso posterior de entrenamiento se combinarán imágenes con diferentes distorsiones, por lo que se debe de escoger un tamaño con un *aspect-ratio* que se aproxime al obtenido con otros parámetros de distorsión similares.

El tamaño final escogido para las imágenes es de 640x576 (*aspect-ratio* = 1,111). Para alcanzarlo se aplica un proceso de re-escalado utilizando aproximación al elemento más cercano para el ground-truth e interpolación bilineal para las imágenes RGB.

El proceso de transformación deja zonas vacías en los bordes de la imagen, mayoritariamente en las zonas cercanas a las esquinas. Éstas son análogas a las zonas que aparecen en las imágenes de ojo de pez circular.

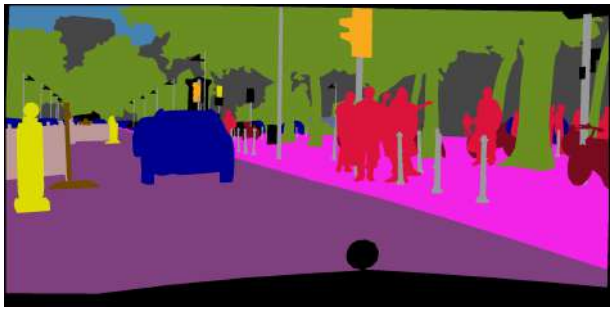
En las figuras 3.4 y 3.5 se muestran ejemplos del resultado final de la transformación final, para las imágenes RGB y para su ground-truth asociado.

3.4 Simuladores

El uso de simuladores como fuente de generación de datos útiles para el entrenamiento de redes está cobrando fuerza, dado que eliminan la laboriosa tarea de etiquetado a nivel de píxel, proporcionan una fuente inagotable de imágenes, permitiendo además seleccionar los elementos que aparecen en ella, lo que habilita la posibilidad de incidir en el entrenamiento de las clases menos representadas en el dataset, permiten seleccionar ciertas características de las condiciones que aparecen en la imagen (como el nivel de iluminación o las condiciones climatológicas) o de los elementos que aparecen en la escena y todo ello de manera completamente automatizada. Por ello, en este trabajo se lleva a cabo un estudio sobre la posibilidad de incorporar imágenes obtenidas desde un simulador en el dataset sintético generado.

(a) Imagen original

(b) Imagen distorsionada



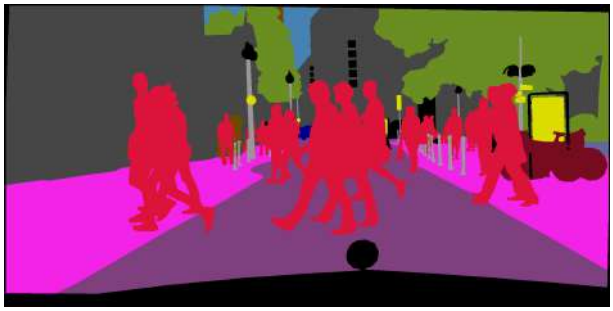
(c) Ground-Truth original

(d) Ground-Truth distorsionado

Figura 3.4: Ejemplo 1 de imágenes distorsionadas

(a) Imagen original

(b) Imagen distorsionada



(c) Ground-Truth original

(d) Ground-Truth distorsionado

Figura 3.5: Ejemplo 12 de imágenes distorsionadas

3.4.1 CARLA

El simulador CARLA [30] fue presentado en 2017 con el objetivo de apoyar el desarrollo, entrenamiento y validación de los sistemas de conducción autónomos en entornos principalmente urbanos. El simulador permite el control de un vehículo dentro de un mapa cerrado que es ampliamente configurable dado que se pueden controlar los elementos que aparecen en él, las condiciones meteorológicas 3.6, el comportamiento de los agentes presentes en la escena, el número de sensores utilizados para percibir el entorno así como su posición y orientación en el vehículo e incluso propiedades avanzadas como la resolución que ofrecen o la tasa de adquisición de imágenes.

Figura 3.6: Diferentes condiciones meteorológicas simulables en CARLA.

.

La puesta en funcionamiento del simulador es compleja dado que requiere de la instalación previa de Unreal Engine 4 (UE4) y la descarga de un conjunto de paquetes de texturas desde la tienda de EpicGames (de manera gratuita). Una vez instalados los requisitos previos para el simulador se puede compilar su código fuente disponible de manera gratuita en la web.

Tras la instalación del simulador se tiene acceso a dos aplicaciones: una en la que el usuario puede controlar un coche mediante el teclado a través del mapa y otra no controlada en la que el coche se desplaza de manera autónoma a través del mapa adaptándose a las reglas de circulación tradicionales en la medida de lo posible. Esta segunda se encuentra implementada mediante un cliente y un servidor de Python. En ambos casos, la extracción de datos se puede realizar mediante la adición de sensores que proporcionan información del entorno del coche. CARLA proporciona acceso a los siguientes sensores:

- **Cámara RGB**: es una cámara estándar que proporciona imágenes en color. Son configurables tanto su posición como su resolución finales. En la figura 3.7a se muestran varias imágenes tomadas en diferentes condiciones climatológicas.
- **Cámara de profundidad**: devuelve la profundidad en metros detectada en la escena codificada en una imagen RGB. El propio simulador facilita herramientas para extraer los valores. La figura 3.7 muestra un ejemplo de imágenes original y decodificada.
- **Cámara de segmentación semántica**: proporciona las etiquetas asociadas a los píxeles presentes en las imágenes codificadas en el canal rojo de una imagen RGB. En la figura 3.8 se muestra un ejemplo de una imagen RGB de entrada y de otra que ya contiene las etiquetas de color de

CityScapes. Además, la política de etiquetado del simulador es bastante similar a la del dataset Cityscapes.

(a) Imagen con condiciones de sol (b) Imagen tomada con condiciones de lluvia

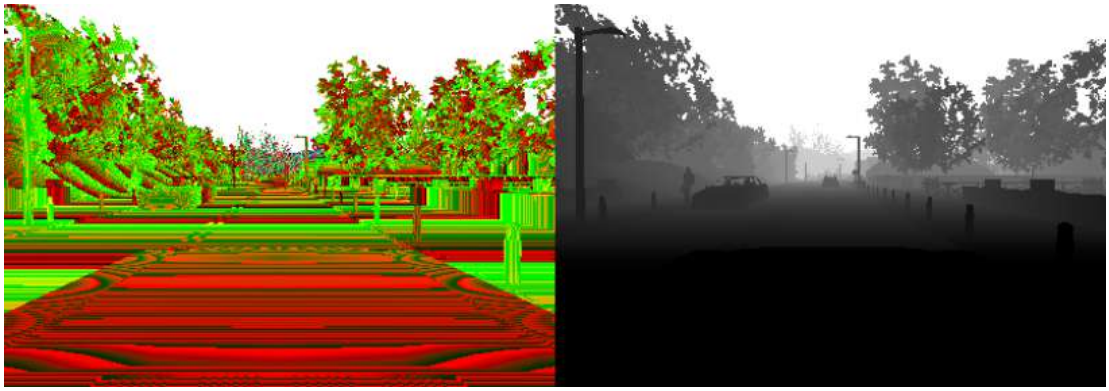


Figura 3.7: Imágenes de profundidad original y decodificada.

(a) Imagen original del simulador (b) Imagen segmentada

Figura 3.8: Ejemplo de segmentación semántica mediante el simulador

.

En la última versión del simulador (publicada en junio de 2018) se incluye además la posibilidad de incorporar un sensor lidar en el coche que proporciona nubes de puntos dispersas del mapa recorrido.

A pesar de que el test del simulador verificó su potencial, finalmente se optó por no incluir imágenes asociadas a él en el dataset final generado, debido a que las etiquetas utilizadas para la segmentación no son las mismas que en CityScapes y alguna de ellas es incompatible con las del primer dataset, dado que son usadas para referencias a un mismo objeto. A continuación se proporciona un pequeño estudio sobre las clases utilizadas en ambos datasets.

El número de etiquetas existentes en CARLA es de 13, de las cuales 2 designan elementos que no se deben clasificar o indefinidos. Las etiquetas del simulador incluyen edificio, valla, carretera, peatón, poste, línea de carretera, acera, vegetación, vehículo, muro, señal de tráfico, nada y otro. Por otro lado, las clases de CityScapes aparecen descritas detalladamente en la sección 3.2.1.

Cabe destacar que CityScapes incluye muchas más clases que CARLA por varios motivos: en primer lugar muchos elementos presentes en escenas urbanas no se encuentran implementados en el simulador,

Tabla 3.1: Clases incompatibles entre CARLA y CityScapes

Elemento observado	Clase Asociada CARLA	Clase Asociada CityScapes	Compatibilidad final
Línea de carretera	Línea de carretera	Sin clase asociada	Compatibles
Ciclista	Sin clase asociada	Ciclista	Compatibles
Poste	Poste	Poste	Incompatibles parcialmente
Terreno	Vegetación	Terreno	Incompatibles
Cielo	Otro	Cielo	Incompatibles
Camión	Coche	Camión	Incompatibles.
Tren	Sin clase asociada	Tren	Compatibles
Autobús	Sin clase asociada	Autobús	Compatibles
Motocicleta	Sin clase asociada	Motocicleta	Compatibles
Peatón	Peatón	Peatón	Incompatibles parcialmente

como por ejemplo las bicicletas, los trenes, los autobuses o las motocicletas; la política de etiquetado del simulador agrupa muchas clases consideradas como diferentes en CityScapes en una única como por ejemplo para el caso de los camiones que son incluidos dentro de la clase coche, y por último, el hecho de que el simulador solo considere escenarios completamente urbanos elimina la posibilidad de aparición de ciertas clases de CityScapes como la clase terreno.

La consecuencia de estos factores es que las etiquetas de las imágenes provenientes de ambas fuentes sean diferentes. Esto supone un problema a la hora de combinar imágenes extraídas de ambos orígenes, dado que pueden existir clases incompatibles o solapadas lo que desembocaría en un problema al aplicar técnicas de DeepLearning.

La tabla 3.1 muestra todas las clases que son interpretadas de forma distinta en los datasets, la manera en la son interpretadas y la compatibilidad final estimada para la clase.

Muchos de los casos presentados pueden terminar siendo compatibles con pequeñas modificaciones, pero perdiendo información útil para el entrenamiento. Las posibles soluciones para cada uno de los casos se exponen a continuación:

- **Línea de carretera**: para salvar la compatibilidad de este elemento bastaría con etiquetar las líneas de carretera como carretera, siguiendo la política de CityScapes, con lo que se conseguiría compatibilizar la clase.
- **Ciclista, Autobús, Motocicleta, Tren**: este conjunto de clases no existen en CARLA, pero al no encontrarse implementadas en el simulador no aparecen en las imágenes, por lo que no suponen un problema y terminan siendo compatibles. Sin embargo, al no aparecer en las imágenes no puede obtener ningún beneficio del simulador. Dado que son las clases menos representadas en CityScapes son para las que a priori resultaba más interesante el uso de un simulador.
- **Cielo, Camión, Terreno**: dado que las clases cielo, camión y terreno no se encuentran implementadas en el simulador pero los píxeles asociados a ellas se asignan a otras clases (a *otro*, a *coche* y a *vegetación* respectivamente) éstas son incompatibles y no existe una manera trivial de solucionarlo.

- **Peatón**: la política de etiquetado de CityScapes incluye todos los elementos portados por los peatones tales como bolsas o maletas mientras que CARLA los etiqueta como *Otro*, por lo que la diferencia en políticas de etiquetado desembocará en una degradación del proceso de entrenamiento.
- **Postes**: del mismo modo que en el caso anterior la política de etiquetado varía en ambos datasets para los postes: en el caso de CARLA ningún elemento adherido al poste se considera como tal, si no que se clasifica como *Otro* o como *Sin clasificar*, mientras que en CityScapes se incluían como poste siempre que su tamaño no fuera muy superior al del poste.

El objetivo final del uso de un simulador es el de mejorar la capacidad de las redes para generalizar resultados, consiguiendo datos en situaciones con condiciones alternativas y mejorando el entrenamiento para las clases menos representadas en los datasets utilizados para el entrenamiento mediante la generación de más datos sintéticos. El hecho de que las clases menos representadas en CityScapes no aparezcan en CARLA reduciría notablemente los beneficios de incluir imágenes extraídas de CARLA en el dataset sintético generado. No obstante, el hecho más nocivo es el de que existan elementos iguales designados con distintas etiquetas en ambos datasets, lo que constituye un impedimento prácticamente insalvable para combinar imágenes de ambas fuentes.

Por ello finalmente se optó por no incluir imágenes de CARLA en el dataset sintético generado, dado que no existirían grandes beneficios y se perjudicaría el entrenamiento de las clases mencionadas anteriormente.

Capítulo 4

Redes Neuronales Convolucionales: ERFNet

4.1 Introducción

La aparición de las redes neuronales convolucionales supuso una importante mejora con respecto a los resultados obtenidos con los clasificadores tradicionales existentes hasta el momento. A pesar de ser prácticamente inviables inicialmente debido al excesivo consumo de recursos computacionales que implicaban, a la escasez de bases de datos para entrenamientos y a los problemas que presentaban debido al escaso desarrollo teórico existente, la aparición de dispositivos hardware como las GPUs, de grandes bases de datos anotadas y de extensos estudios teóricos que proponían soluciones para los problemas existentes han permitido un espectacular avance en los últimos años.

Este capítulo abarca un pequeño análisis de la historia de las redes neuronales, el desarrollo de conceptos fundamentales de las CNNs junto con un breve análisis del estado del arte de CNNs aplicadas a segmentación semántica, una descripción detallada de la CNN que supone la línea base de este trabajo (ERFNet) y la justificación e introducción de las modificaciones propuestas sobre la misma con el objetivo de mejorar su rendimiento ante el nuevo desafío presentado en este trabajo.

4.2 Historia

Los primeros modelos informáticos para redes neuronales fueron propuestos en la década de 1940 por Warren McCulloch y Walter Pitts, basándose en el estudio de la fisiología y mecanismo de funcionamiento básico de las neuronas del cerebro junto con la teoría de computación de Alan Turing. Dichos modelos describían el estado de una neurona en función de la actividad de las neuronas vecinas. Otros estudios posteriores como los de Donald Hebb profundizaban en el análisis de la interacción entre varias neuronas mediante modelos artificiales a través de sus sinapsis en los que se modelaban dichas conexiones a partir de pesos matemáticos.

Sin embargo, no comenzaron a ser implementados hasta la década posterior en la que Marvin Minsky comienza a construir la primera neurocomputadora. En paralelo, Farley y Wesley A. Clark empezaron a desarrollar en el MIT las primeras máquinas que incorporaban redes de Hebb (redes de aprendizaje no supervisado). Sin embargo, la aportación más importante fue la de Frank Rosenblatt quién en 1959 creó el primer perceptrón, un algoritmo de reconocimiento de patrones basado en una red de aprendizaje de

computadora de dos capas usando una simple suma y la resta. El perceptrón constaba de un total de 400 fotoceldas que estaban conectadas de manera aleatoria a 512 unidades de tipo neurona. Al proporcionarle un patrón de entrada al perceptrón, éste era capaz de clasificarlo según su categoría de forma correcta. Así consiguió clasificar con éxito las letras del abecedario.

En las décadas de los 60 y 70 se utilizaron las redes neuronales existentes para afrontar por primera vez problemas reales: Stephen Grossberg diseñó una red a partir de elementos discretos cuya actividad variaba con un tiempo que satisface ecuaciones diferenciales continuas, para resolver actividades tales como el reconocimiento continuo del habla y el aprendizaje del movimiento de los brazos de un robot. De forma paralela, Bernard Widrow y Marcial Hoff desarrollaron los modelos de red Adaptive Linear Elements (ADALINE) y Multiple ADALINE (MADALINE) que fueron utilizados para eliminar el eco presente en las líneas telefónicas.

En 1969 Marvin Misnsky y Seymour Papert publicaron un trabajo que profundizaba en el estudio del perceptrón, llevando a cabo un desarrollo matemático más profundo y estudiando la posibilidad de crear modelos de perceptrones multicapa. Como conclusión general se resaltaba la gran limitación del perceptrón a nivel matemático, debido a la incapacidad de implementar operaciones matemáticas triviales como la operación lógica XOR. Esta publicación desvió el interés de este tipo de algoritmos hasta la década de los 80 dado que se asumió que no eran el camino a seguir debido a sus carencias.

Al mismo tiempo que el perceptrón era descartado se crearon los primeros clasificadores lineales que sumaban los valores presentes en sus entradas de manera ponderada con unos pesos obtenidos mediante un entrenamiento previo. John Hopfield, posteriormente, los combinaría con funciones de energía y modificaría su arquitectura básica para adaptarlos a labores de optimización y de reconstrucción de patrones.

El impulso final para el asentamiento de las redes neuronales fue la recuperación de una idea propuesta en 1974 por Paul Werbor para el aprendizaje de las redes: Rumelhart, Hunton y Williams redescubren el algoritmo de *back-propagation* o de propagación hacia atrás. La expansión de este algoritmo junto con el desarrollo de las tecnologías de fabricación de hardware (como microchips) desembocó en la construcción de las primeras redes neuronales implementadas en silicio en torno a 1986.

A partir del año 2009 los resultados de las redes neuronales existentes comenzaron a sobrepasar claramente a los obtenidos por los clasificadores tradicionales. Las mejores técnicas del estado del arte en actividades como reconocimiento de patrones y aprendizaje de máquina (*machine learning*) pasaron a estar basadas en redes neuronales que se vieron impulsadas por el desarrollo de hardware específico para el entrenamiento de las redes como las unidades de procesamiento gráfico (GPU) y de grandes bases de datos anotadas que permitían entrenamientos complejos en tiempos asumibles.

4.3 Fundamentos

El elemento básico que compone las redes neuronales artificiales es la neurona. El diseño de las neuronas artificiales está funcional y estructuralmente inspirado en el de las reales que forman el cerebro humano. Estas tienen una forma y un tamaño variable pero presentan siempre las mismas subdivisiones anatómicas.

4.3.1 Estructura

La estructura básica de una neurona está compuesta por un cuerpo o soma que contiene el núcleo de la célula y es el encargado de coordinar todas las actividades de la neurona; un conjunto de dendritas que son estructuras ramificadas que parten del soma y que tienen por función captar estímulos de

otras células adyacentes a través de las sinapsis; una estructura alargada unida al cuerpo o soma de la neurona que permite enviar señales a otras células cercanas denominada axón y las sinapsis, que son las interconexiones entre diferentes neuronas. Cada neurona posee entre 10.000 y 100.000. La interacción entre neuronas tiene naturaleza química, realizándose la propagación de la información mediante la liberación de neurotransmisores, aunque tiene efectos eléctricos cuantificables. Cuando una neurona es estimulada con la suficiente intensidad a través de sus dendritas genera un impulso que se propaga por toda la célula, liberándose neurotransmisores a través del axón. La frecuencia e intensidad de los impulsos emitidos por las neuronas depende del nivel de excitación de sus dendritas y sigue una función de naturaleza sigmoide.

Las neuronas artificiales presentan elementos análogos a las partes de las células biológicas que se encargan de implementar una función matemática que actúa como clasificador:

- **Entradas**: se trata de un vector de valores que contienen los datos proporcionados por otras neuronas. Se corresponden con las dendritas de las neuronas biológicas.
- **Pesos sinápticos**: representan la importancia otorgada a las diferentes interconexiones entre neuronas. Son los valores a entrenar durante el entrenamiento y pueden equipararse al volumen de neurotransmisores presentes en una sinapsis.
- **Función de clasificación**: esta parte se corresponde con el cuerpo celular. Se trata de una función matemática que implementa el clasificador de la neurona. Por simplicidad se suele recurrir a uno lineal, pero de forma genérica la función de clasificación puede ser descrita por la ecuación 4.1 en la que w_{ij} representan los pesos entrenados y x_i representa el vector de valores de entrada.

$$y = f(\sum(w_{ij}x_i + b_i)) \tag{4.1}$$

- **Función de activación**: es una función matemática que puede presentar múltiples formas, aunque se suele recurrir a una función sigmoidea de la forma indicada en la ecuación 4.2. Otras posibilidades son funciones basadas en la tangente hiperbólica o la función ReLu dadas por las ecuaciones 4.3 y 4.4 respectivamente.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

$$tanh(x) = 2\sigma(2x) - 1 \tag{4.3}$$

$$f(x) = max(0, x) \tag{4.4}$$

La figura 4.1 ilustra la comparativa entre el modelo de neurona biológico y el real. Las redes

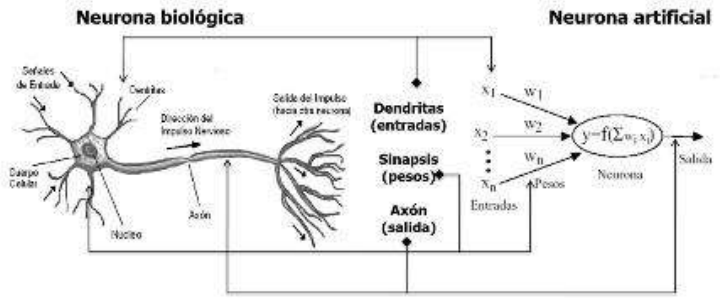


Figura 4.1: Comparativa entre neuronas real y artifical.

neuronales se conforman mediante la agrupación de neuronas para formar capas, que a su vez se agrupan secuencialmente conectando las distintas neuronas mediante sus pesos de entrada de los elementos de una

capa con las salidas de la capa previa. Generalmente el esquema básico de una red se representa mediante 3 tipos de capas: la capa de entrada que compone la interfaz inicial de la red con el exterior, las capas ocultas que son invisibles desde el exterior y que realizan las tareas de análisis de los datos de entrada y la capa de salida que proporciona la clasificación final realizada. El esquema genérico se ilustra en la figura 4.2.

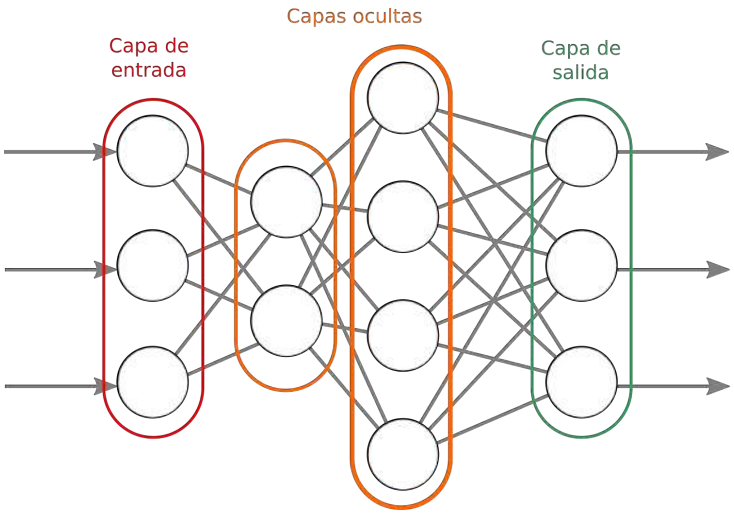


Figura 4.2: Arquitectura de una red neuronal genérica.

El número de capas ocultas incluidas depende del objetivo de la aplicación: cuantas más capas sean incluidas más compleja será la función que la red es capaz de aproximar pero a cambio su entrenamiento se vuelve más complejo. Si los datos manejados no son demasiado complejos son preferibles las redes pequeñas para evitar problemas de *overfitting* (sobreajuste a una solución concreta que desemboca en una mala capacidad para generalizar) pero para tareas complejas las redes con muchas capas permiten encontrar soluciones de mejor calidad.

4.3.2 Bloques y aprendizaje

Una red neuronal puede ser matemáticamente interpretada como la composición de un determinado número de funciones de la forma mostrada en la ecuación 4.5.

$$f(x) = f_L(...f_2(x; w_1); w_2)...), w_L) \tag{4.5}$$

donde cada función f_l toma como entrada un dato x_l y un vector de parámetros w_l para dar lugar a un dato de salida x_{l+1}. Las funciones utilizadas son un elemento del diseño de la red, mientras que el conjunto de parámetros $w = (w_1, ..., w_L)$ son aprendidos por la propia red durante el proceso de aprendizaje.

En el campo específico de la visión artificial, las redes neuronales tienen por objetivo el análisis de imágenes. Por ello, las funciones y datos de la red tienen una estructura adicional. Los datos $x_1, ..., x_n$ a procesar son imágenes con más de una dimensión (por lo general son arrays de píxeles de 2 dimensiones). Así, cada dato x_i de entrada será un array de $MxNxK$ elementos representando M el alto de la imagen, N el ancho y K el número de canales por cada píxel. El dato x_1 representará la imagen original, mientras que el resto de datos $x_2, ..., x_n$ representarán mapas de características intermedios derivados de la misma.

4.3.2.1 Capas convolucionales

Para el caso de las redes neuronales convolucionales (CNNs) las redes tienen peculiaridades adicionales. En ellas, las funciones f_l aplican operaciones de convolución, que son locales e invariantes a la traslación, a los datos de entrada. Una manera de ejemplificar estos operadores sería mediante la aplicación de un banco de filtros lineales al dato de entrada x_l. Estos operadores constituyen la unidad elemental para componer las capas básicas de las CNNs.

Sin embargo, el uso de estos filtros puede presentar problemas en determinadas situaciones: la aplicación de estos operadores sobre un dato puede no ser viable debido a las dimensiones del dato a procesar, como consecuencia del paso o *stride* con el que los filtros barren la imagen. La condición para que la aplicación de un filtro convolucional sea posible viene dada por la ecuación 4.6. Ésta expone que el cociente entre la resta de las dimensiones del dato a procesar N con la correspondiente dimensión del filtro F y el paso o *stride* S aplicado debe ser un número entero.

$$\frac{(N-F)}{S} + 1 \in \mathbb{Z} \tag{4.6}$$

La figura 4.3 muestra un ejemplo de capa convolucional con los parámetros mencionados anteriormente representados en ella.

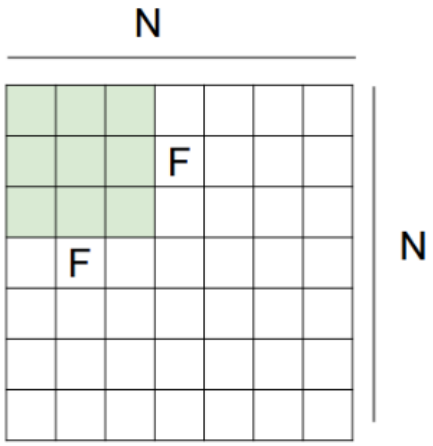
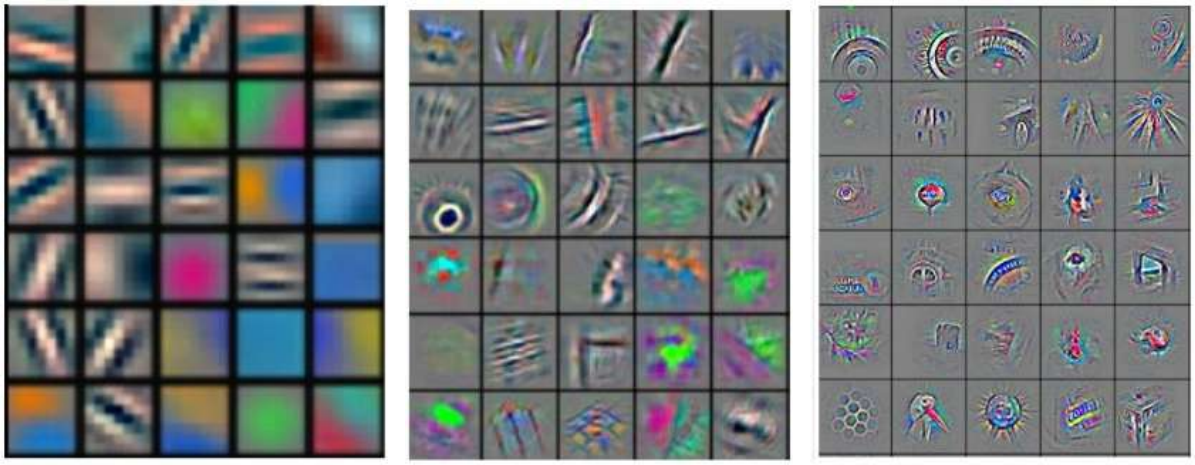


Figura 4.3: Ejemplo de filtro convolucional.

En segundo lugar, los operadores pueden devolver como resultado mapas de características que poseen dimensiones diferentes a las de la entrada original, lo que dificulta el procesamiento de los datos. Para evitar la modificación de estas dimensiones y asegurar que las operaciones siempre sean realizables se puede recurrir a una operación de *zero-padding*, consistente en aumentar el tamaño de la imagen mediante la adición de ceros en los bordes para conseguir una salida en las capas con el mismo tamaño que la entrada.

La base del aprendizaje de las CNNs es el ajuste de los pesos asociados a las diferentes componentes de los filtros. La figura 4.4 muestra gráficamente los pesos aprendidos por 3 conjuntos de capas de una CNN:

La sub-figura 4.4a se corresponde con los pesos aprendidos por las primeras capas de la red, que terminan convergiendo a filtros extractores de características de bajo nivel como detectores bordes o esquinas. Las capas medias de la red detectan características de mayor nivel 4.4b como texturas o formas y, por ultimo, en los pesos aprendidos por las capas más profundas se observan características de alto nivel 4.4c en las que pueden incluso visualizarse algunas de las clases buscadas.



(a) Características de bajo nivel (b) Características de nivel medio (c) Características de alto nivel

Figura 4.4: Representación de los filtros aprendidos por una CNN

4.3.2.2 Capas de activación

La mayoría de datos naturales siguen distribuciones no lineales. La distribución más repetida es la Gaussiana, aunque existen otras habituales como la Rayleigh. La presencia de datos de naturaleza no lineal dificulta la tarea de clasificación que deja de ser abordable simplemente mediante el uso de funciones lineales como las convoluciones planteadas en el apartado anterior. Las redes requieren, por tanto, la incorporación de capas adicionales o más complejas capaces de realizar clasificar datos que siguen distribuciones no lineales complejas.

Una manera sencilla de implementar un clasificador no lineal es seguir el modelo de una neurona real que incorpora una función de activación no lineal tras la etapa de clasificación implementada mediante el clasificador lineal. Las ecuaciones 4.4, 4.2 y 4.3 expusieron distintos ejemplos de funciones que pueden implementar la etapa de activación no lineal. Así, las etapas de convolución de las CNNs tienden a estar seguidas por capas de activación no lineales entre las que destaca la capa Relu que es la más recurrida al ser la más simple.

4.3.2.3 Capas de pooling

Los bloques de *pooling* también son un elemento habitual e importante en las CNNs. Se trata de bloques que suelen seguir a las etapas convolucionales y que reducen el tamaño de los mapas de características tratados siguiendo un criterio seleccionado durante el diseño obteniendo así múltiples beneficios:

- La disminución del tamaño de los mapas de características hace las representaciones de las imágenes menos pesadas y más manejables, sin perder demasiada información si son diseñadas adecuadamente.
- El aumento del tamaño del campo receptivo de los filtros de las capas posteriores a la de *pooling*. Este aumento desemboca en una mejora del análisis de la información contextual de la escena debido a que los campos receptivos pasan a abarcar regiones más amplias.
- El uso de etapas de *pooling* supone la incorporación de no linealidades adicionales que resultan beneficiosas para la tarea de clasificación.

El tipo de capa de *pooling* más característico en las redes neuronales convolucionales es el *max-pooling*, que conserva solo el mayor elemento del mapa de características en un determinado área. La formulación matemática es la expuesta en la ecuación 4.7.

$$y_{i,j,k} = max(y_{i',j',k} : i \leq i' < i + p, j \leq j' < j + p) \tag{4.7}$$

La selección del elemento con mayor valor ayuda a conservar la información más relevante dentro de los mapas de características, por lo que este tipo específico de capa de *pooling* consigue todos los beneficios expuestos anteriormente sin perder apenas información para el análisis.

4.3.2.4 Capas de normalización

Las etapas de normalización son otras capas importantes para conseguir que la extracción de información de las imágenes sea eficaz. Estos bloques operan de manera individual sobre cada uno de los canales de las imágenes realizando una normalización del color de los canales en base a varios posibles criterios como podrían ser el número de imágenes cargadas en memoria durante el entrenamiento (*batch*), la región de la imagen estudiada o los histogramas de color de la imagen a nivel de dataset completo.

4.3.2.5 Propagación de la pérdida

El proceso de entrenamiento de una red neuronal consiste en el ajuste iterativo de los pesos de cada uno de los elementos que componen la red. Durante este proceso, la última capa de la red, que se denomina capa de pérdidas o de clasificación, realiza el cálculo de un valor denominado pérdida que representa la diferencia entre el valor real obtenido a la salida de la red y el ground-truth empleado. El objetivo del entrenamiento de la red es minimizar esta pérdida, para lo que se usa una técnica denominada "propagación hacia atrás" o *back-propagation*. Ésta se implementa mediante un método de optimización basado en el gradiente. El cálculo de un gradiente para una CNN puede realizarse mediante la regla de la cadena, dada por la ecuación 4.8:

$$\frac{\partial f}{\partial w_l}(x_0; w_1, ..., w_L) = \frac{\partial f_L}{\partial x_{L-1}}(x_{L-1}; w_L) \times ... \times \frac{\partial f_{l+1}}{\partial x_l}(x_l; w_{l+1}) \times \frac{\partial f_l}{\partial w_l}(x_{l-1}; w_l) \tag{4.8}$$

Sin embargo, la aplicación de esta regla para el cálculo de derivadas cuando se trabaja con imágenes no es adecuada, dado que el número de elementos resultantes en las derivadas puede resultar extremadamente grande al ser las funciones y los pesos entrenados bidimensionales. Para conseguir realizar el cálculo de este gradiente de forma eficiente los *frameworks* de DeepLearning proporcionan múltiples técnicas. La más básica consiste en la transformación de los datos de entrada de dos dimensiones (tensores) en datos de una dimensión (vectores). Esta transformación se refleja en la ecuación 4.9, en la que el operador vec convierte un tensor a un vector mediante la reorganización de sus elementos en una única dimensión.

$$\frac{\partial vec(f)}{\partial vec^T(w_l)} = \frac{\partial vec(f_L)}{\partial vec^T(x_L)} \times \cdots \times \frac{\partial vec(f_{l+1})}{\partial vec^T(x_l)} \times \frac{\partial vec(f_l)}{\partial vec^T(w_l)} \tag{4.9}$$

A partir de este planteamiento, los diferentes *frameworks* ofrecen mecanismos simples para el cálculo de la salida de una red a partir de una entrada $y = f(\mathbf{x}; \mathbf{w})$ y las derivadas necesarias para propagar una pérdida a través de la arquitectura de una red $\langle \mathbf{p}, f(\mathbf{x}; \mathbf{w}) \rangle$

4.4 Segmentación semántica

En el ámbito de la visión artificial, las redes neuronales fueron inicialmente planteadas para afrontar la tarea de clasificación, es decir, para predecir categorías de objetos únicos a partir de una imagen. Sin embargo, rápidamente surgió interés en adaptar las arquitecturas existentes para que generaran resultados de clasificación a nivel de píxel y así fueran útiles en tareas como la segmentación semántica.

Los mayores esfuerzos para conseguir esta adaptación se han basado en las *Fully Convolutional Networks* (FCN) [31] que fueron propuestas específicamente con este objetivo. El mecanismo escogido para ello fue convertir las CNNs en redes completamente convolucionales y en realizar un conjunto de operaciones de *upsampling* sobre los mapas de características obtenidos hasta alcanzar el tamaño de entrada de las imágenes originales con el objetivo de obtener la clasificación a nivel de píxel deseada.

Esta primera aproximación no resulta suficiente, dado que debido a los procesos de *dowsampling* incluidos en las redes originales (que persiguen incrementar la información contextual empleada al analizar la imagen) las salidas obtenidas son de poca precisión (*píxeles gruesos*). Para alcanzar precisión a nivel de píxel buscada, estas salidas requieren de un proceso de refinamiento adicional, para lo que se ha propuesto una solución principal: combinar los mapas de características obtenidos en las primeras capas de la red (que conservan prácticamente la totalidad de la resolución de la imagen) con las salidas finales obtenidas mediante conexiones entre ambas capas.

Una primera implementación de esta solución fue la aportada por SegNet [32]. Esta red presentaba una arquitectura revolucionaria con una topología de encoder-decoder simétrica en la que el decoder realizaba un *upsampling* más fino utilizando para ello índices de *max-pooling* extraídos en base al encoder. Esta aproximación consiguió reducir significativamente el coste computacional y de memoria de la red ya que ésta no exige guardar todos los mapas de características generados. Otra solución fue la aportada por redes como Deeplab [33] que utilizan una etapa adicional consistente en un campo aleatorio condicional (CRF) para refinar la salida. Esta opción también se ha intentado integrar dentro de la propia estructura de la red (CRFasRNN) [34]. No obstante, esta etapa introduce un retardo importante dado que computacionalmente es muy pesada, por lo que su uso no es recomendable en aplicaciones orientadas a funcionar en tiempo real.

La excesiva carga computacional de las propuestas basadas en CRFs despertó el interés en modelos más ligeros capaces de llevar a cabo segmentación semántica en tiempo real. Una de las primeras propuestas para ello fue ENet [35] que surgió como una alternativa eficiente para permitir la implementación de la segmentación semántica en tiempo real. Con dicho propósito se inspiró en ideas planteadas en ResNet [36] y utilizó múltiples capas residuales de tipo *bottleneck* (la estructura de este tipo de capas se analiza en detalle en la sección 4.5) que pueden ser aprovechados en etapas tanto de *downsampling* como de *upsampling*. Sin embargo, difiere de propuestas previas como SegNet en otros aspectos como la estructura simétrica encoder-decoder y aboga por un encoder más grande que el decoder, dado que considera que las capas iniciales de la red no contribuyen directamente a la clasificación sino que deben actuar mayoritariamente como extractores de características. Esta red consiguió ser funcional en tiempo real, pero a cambio de una gran pérdida de precisión en sus resultados. En la misma línea de trabajo se propusieron otras arquitecturas que consiguieron superar los resultados obtenidos por ENet, como LinkNet [37] que buscaba vincular las capas del encoder con las del decoder y SQNet [38] que empleaba convoluciones dilatadas paralelas que ayudaban a determinar mejor los contornos de los objetos al emplear información de bajo nivel de las capas inferiores del encoder. PSPNet fue la red que presentó mejores resultados [39] utilizando un decoder estudiando la información proporcionada por el encoder combinando bloques de distintos tamaños para analizar la imagen a distintas resoluciones y así beneficiar la información contextual. Los resultados obtenidos por esta red fueron superiores a los presentados por todas las redes

anteriores, pero el tiempo de procesamiento que requería era extremadamente alto, superando el segundo para imágenes de gran tamaño.

El uso de capas residuales (propuestas inicialmente en [36]) se ha vuelto habitual, dado que solventan el problema de la degradación del gradiente al ser propagado a través de un gran número de capas y permiten el uso de redes neuronales profundas. Trabajos como [40] (DeepLab2) integran en redes planteadas anteriormente (DeepLab) redes de capas residuales (ResNet-101) para obtener resultados de clasificación extremadamente precisos. RefineNet [41] utilizaba un sistema de refinamiento final que aprovechaba toda la información disponible durante el proceso de *downsampling* y conexiones residuales para proporcionar predicciones de alta precisión. En [42] (FRRN) se combina una red de capas residuales de arquitectura similar a ResNet con un módulo de extracción de contexto y en [43] (LRR) se proponía una pirámide laplaciana que combinaba las características extraídas a diferentes escalas.

Todas estas propuestas integran capas residuales en redes con resultados excelentes, pero distan por mucho de ser viables en tiempo real. Como línea base para este trabajo se parte del uso de una red que emplea dichas capas pero que consigue ser funcional en tiempo real.

4.5 ERFNet

La red neuronal convolucional ERFNet (Efficient Residual Factorized Convnet) [44], desarrollada por el grupo Robesafe de la UAH, supuso una propuesta disruptiva, dado que presentaba una red con resultados similares a los del estado del arte pero computacionalmente mucho más ligera permitiendo afrontar tareas muy pesadas como la segmentación semántica en tiempo real, incluso superando los 10 frames por segundo en dispositivos embebidos (Jetson TX2). Así, se consolidó rápidamente como la CNN con el mejor compromiso eficiencia computacional-rendimiento.

El éxito de la propuesta subyace en la adopción de otro enfoque a la hora de afrontar el problema de la segmentación semántica: las mejores soluciones del estado del arte del momento ([45] [33] [42] [46]) estaban centradas en obtener ligeras mejoras en sus resultados a costa de aumentar la complejidad de la red y, por tanto, el volumen de recursos computacionales requerido convirtiéndolos en modelos inviables para aplicaciones en tiempo real. Por otro lado, las propuestas centradas en mejorar la eficiencia de las redes para conseguir que fueran funcionales en tiempo real utilizaban estrategias que desembocaban en una clara degradación del rendimiento como reducir el número de parámetros utilizados de forma muy agresiva ([35] [38]).

ERFNet busca un punto equidistante entre ambos tipos de solución, basado en la propuesta de un nuevo tipo de capa residual diseñada para ser especialmente eficiente al ser factorizable pero manteniendo la precisión final alcanzada.

El resultado es una red extremadamente eficiente, cuya implementación ocupa en torno a 8 MB en memoria pero con resultados cercanos a los de las mejores redes del estado del arte. La figura 4.5 muestra los resultados que la red es capaz de alcanzar en CityScapes.

.

4.5.1 Arquitectura

La arquitectura de ERFNet sigue un modelo de encoder-decoder similar a los expuestos anteriormente, pero basado en una nueva versión de las capas residuales de tipo *non-bottleneck* rediseñadas para ser especialmente eficientes. Lo habitual en las redes que tienen como objetivo prioritario la eficiencia

Figura 4.5: Resultados ERFNet en CityScapes.

es recurrir a las capas residuales de tipo *bottleneck*, dado que reducen el tamaño de los mapas de características generados para agilizar los tiempos de computación aunque a cambio de una degradación de la precisión obtenida. El primer tipo de capa aporta mapas de características más anchos que, en tareas de predicción densas como la segmentación semántica, desembocan en mejores resultados. Por ello, a pesar de la reducción de la eficiencia computacional que supone el uso de este tipo de capas, con objetivo de mantener una clasificación final muy precisa ERFNet apuesta por una re-implementación del segundo tipo de capas.

La nueva versión propuesta busca compensar la pérdida de eficiencia construyendo las convoluciones con kérneles de una dimensión(1D) [47] completamente factorizables los cuales son computacionalmente más rápidos, implican menor número de parámetros pero dan lugar a mapas de características de mejor calidad. El esquema básico de este tipo de capa aparece reflejado en la figura 4.6, junto con el esquema de una capa residual tradicional y de una capa *non-bottleneck* básica sin kérneles de una dimensión factorizables. .

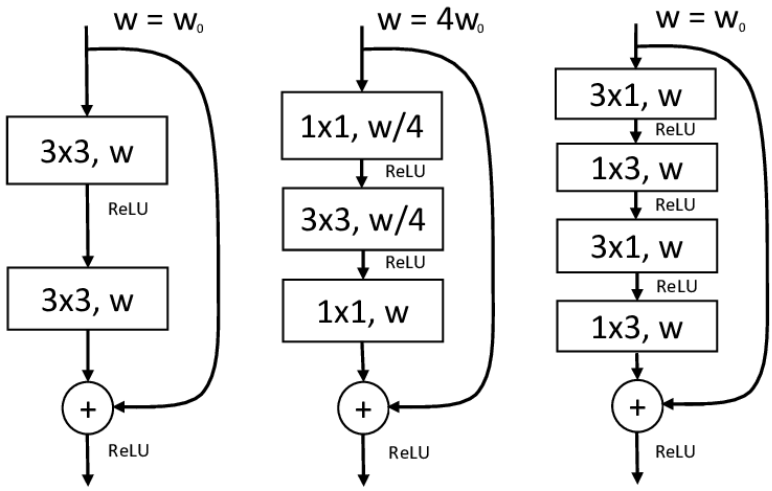


Figura 4.6: Esquema de propuesta de capa *non-bottleneck*.

La arquitectura de la red se constituye a partir de la agrupación secuencial de capas como la expuesta anteriormente junto con bloques de *downsampling* en el encoder y de bloques deconvolucionales en el

decoder que permiten recuperar la resolución de la imagen de entrada. En total, el bloque codificador se compone de un total de 16 capas que incluyen 13 capas residuales y 3 capas de *downsampling* y el bloque decodificador contiene otras 7 capas que incluyen 3 capas deconvolucionales y 4 capas residuales. La figura 4.7 muestra los mapas de características generados para una imagen genérica de dimensiones 640x576 como las utilizadas posteriormente en los entrenamientos realizados, distinguiendo los mapas asociados a los segmentos del encoder y del decoder.

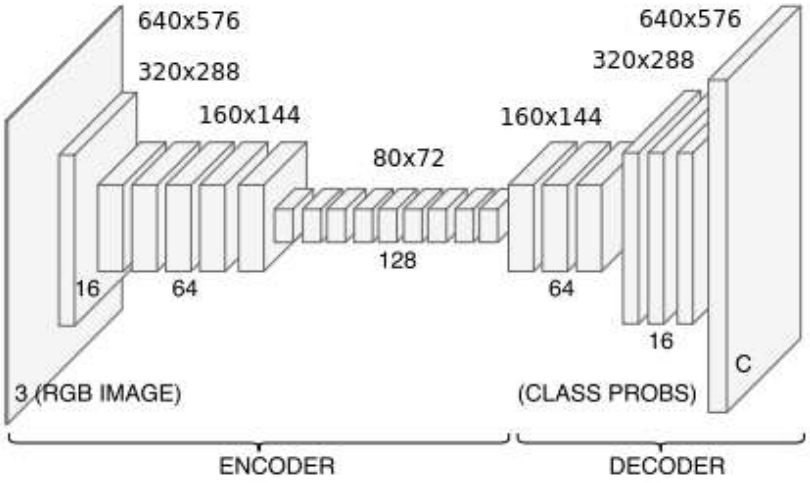


Figura 4.7: Mapas de características generados para una imagen de entrada de 640x576

.

La tabla 4.1 muestra detalladamente la organización de las capas que componen la arquitectura de la red. De las 23 capas totales, las 16 primeras capas pertenecen al bloque del encoder y las 7 últimas componen el bloque del decoder.

4.6 Modificaciones de la arquitectura original

Los principales items relevantes en el aprendizaje de una CNN son la apariencia de los elementos de las clases que, a su vez, engloba múltiples items menores tales como color, textura, partes que componen los elementos, etc; la forma de los objetos que componen las clases y la información contextual presente en la imagen. Este último item engloba también varios items menores entre los que se incluyen:

- La relación existente entre las diferentes clases: la tendencia de los distintos objetos de las imágenes a agruparse siguiendo ciertos patrones (como por ejemplo ciclistas o bicicletas o peatones y acera) puede ser representativa a la hora de identificar objetos de apariencia similar.
- La información global de la imagen: muchos tipos de objetos diferentes presentan apariencias iguales y la habilidad para identificarlos reside en la capacidad para identificar el tipo de escena presentada en la imagen. Es el caso presentado al tener que identificar muros y casas, edificios y rascacielos o montañas y colinas.
- Las diferentes escalas de objetos presentes en la imagen: el estudio de las imágenes por regiones y de manera global proporciona la capacidad de clasificar con mejor precisión objetos con tamaños muy pequeños que de otra manera serían muy complicados de identificar (a pesar de que pueden tener gran valor semántico como los semáforos o las señales de tráfico) u objetos con tamaños muy grandes que exceden el tamaño de los campos receptivos de los filtros de la red por lo que requieren otros mecanismos para ser identificados.

Tabla 4.1: Organización de las capas de ERFNet

Capa	Tipo de capa	Mapas de características	Resolución de salida
1	**Bloque Dowsampling**	16	320x288
2	**Bloque Dowsampling**	64	160x144
3-7	5 x **Non-bottleneck 1D**	64	160x144
8	**Bloque Dowsampling**	128	80x72
9	**Non-bottleneck 1D**	128	80x72
10	**Non-bottleneck 1D**	128	80x72
11	**Non-bottleneck 1D**	128	80x72
12	**Non-bottleneck 1D**	128	80x72
13	**Non-bottleneck 1D**	128	80x72
14	**Non-bottleneck 1D**	128	80x72
15	**Non-bottleneck 1D**	128	80x72
16	**Non-bottleneck 1D**	128	80x72
17	**Bloque Deconvolucional**	64	160x144
18-19	2 x **Non-bottleneck 1D**	64	160x144
20	**Bloque Deconvolucional**	16	320x288
21-22	2 x **Non-bottleneck 1D**	16	320x288
23	**Bloque Deconvolucional**	C	640x576

- La posición de los objetos presentes en la escena: la posición ocupada dentro de la imagen de un cierto objeto resulta relevante a la hora de llevar a cabo la tarea de clasificación, dado que ciertas clases con apariencias muy cambiantes como puede ser el cielo, por lo general, siempre aparecen asociadas a una determinada región que puede beneficiar su identificación.

La forma y apariencia (que se ve modificada principalmente por la alteración de las texturas) de los objetos que figuran en las imágenes de ojo de pez se ven claramente alteradas al ser éstos deformados y distorsionados. De manera adicional, un mismo objeto puede presentar diferentes apariencias y formas en una misma imagen en función de la región en la que esté situado.

Éstos son dos de los tres elementos especialmente relevantes en el aprendizaje de las redes neuronales que se verá previsiblemente penalizado por la diversidad de apariencias y requerirá una mejor capacidad de generalización. Trabajos previos como [22] destacaban este hecho y resaltaban la importancia de utilizar arquitecturas que primasen el tercer item influyente en el entrenamiento de las CNNs.

Partiendo de esta idea y del conjunto de propuestas expuestas en la sección 4.4, en este trabajo se propone una modificación de la arquitectura original de ERFNet, incorporando el decoder piramidal de la PSPNet con objetivo de beneficiar la información contextual presente en las imágenes, que no se ve degradada e incluso puede resultar más relevante dado que la apariencia de los objetos está estrechamente relacionada con su posición en la imagen.

El decoder de PSPNet incluye un conjunto de bloques que analizan los mapas de características proporcionados por el encoder a diferentes resoluciones para generar otros mapas de distintos tamaños en función de la resolución evaluada. Éstos son procesados por capas deconvolucionales que los re-escalan

a un tamaño común en el que son concatenados junto con el último mapa generado por el encoder. El mapa compuesto por la concatenación de los anteriores es procesado por capas deconvolucionales que proporcionan la imagen final clasificada. Para el decoder piramidal implementado se han utilizado 4 bloques con escalas de 1, 1/2, 1/4 y 1/8.

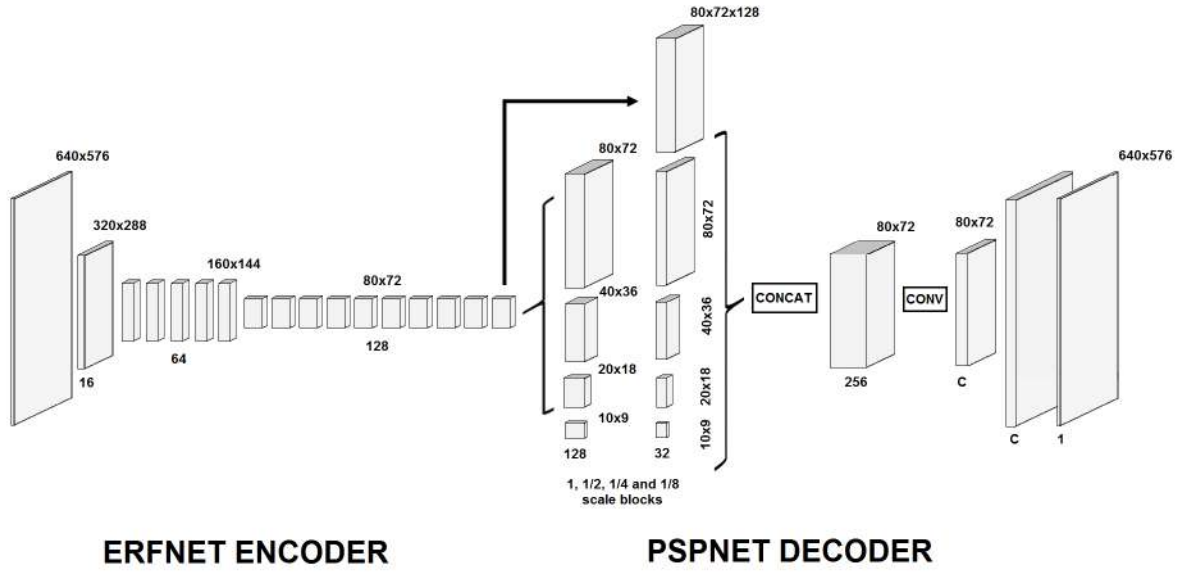


Figura 4.8: Arquitectura de ERFNet con modelo piramidal de PSPNet

.

Capítulo 5

Entrenamiento y data-augmentation para fisheye

5.1 Introducción

La capacidad para generalizar de una red neuronal es definida como la habilidad para extrapolar resultados obtenidos en una determinada situación (la del entrenamiento) a otras con diferentes condiciones. Ésta puede verse claramente mejorada durante el proceso de entrenamiento mediante el uso de *data-augmentation*, es decir, a partir de técnicas que permiten la incorporación de imágenes adicionales durante el entrenamiento a partir de modificaciones fijas o aleatorias tales como desplazamientos, rotaciones, cambios de escala o *aspect-ratio*, en la iluminación o en los colores de la imagen a partir de las imágenes de las que se disponía originalmente.

Este capítulo expone los resultados de los primeros experimentos realizados que abarcan el análisis de la influencia de la distorsión incluida en las imágenes sobre el rendimiento de la red, la comparativa de los resultados obtenidos en los primeros entrenamientos con los de otros métodos del estado de arte y el estudio del beneficio aportado por el uso de diferentes tipos de *data-augmentation* implementados durante el entrenamiento. Además, se incluye una nueva propuesta de *data-augmentation* que no había sido implementada con anterioridad.

5.2 Entrenamiento sobre el dataset sintético

En el capítulo 3 se generó un dataset sintético de imágenes de ojo de pez a partir del modelo proyectivo de una cámara fisheye equidistante, de una distancia focal que establecía el nivel de distorsión introducido y de el dataset de imágenes urbanas CityScapes.

La distancia focal inicial escogida era $f_0 = 159$ y era la utilizada previamente en otros trabajos como [22]. Este parámetro focal introduce una distorsión muy fuerte pero, para el tamaño de las imágenes de CityScapes originales, devuelve otras prácticamente circulares muy similares a las proporcionadas por una cámara real. Por ello, fue considerado como apropiado para constituir la línea base de trabajo para realizar comparativas y escoger la mejor arquitectura para avanzar en la línea de investigación. Como punto de partida se realiza un entrenamiento comparativo para ese mismo parámetro de distorsión con las dos arquitecturas propuestas en el capítulo 4.

Todos los entrenamientos realizados se llevan a cabo en el framework para DeepLearning *Pytorch* que ha conseguido un papel destacado en los últimos años gracias a su gran eficiencia y a utilizar un lenguaje de programación muy extendido (Python).

El entrenamiento para ambas arquitecturas se realiza de forma similar. En primer lugar, éste se divide en dos etapas: una primera en la que se entrena el encoder de la red de forma aislada tomando como ground-truth la imagen etiquetada adaptada al tamaño de salida de los mapas de características del bloque codificador tras tres etapas de downsampling (el ancho y el alto del ground-truth original divididos entre 8) y una segunda en la que se entrena la red completa para generar segmentación semántica. La división del entrenamiento en dos partes independientes persigue simplificar el proceso de ajuste de los pesos de la red, dado que reduciendo el número de parámetros para ajustar simultáneamente se reduce la dimensionalidad del problema y, por tanto, también lo hace su complejidad. En la segunda etapa, el ajuste de los pesos del decoder es más simple ya que se parte de un encoder con pesos ya entrenados. Los resultados obtenidos, además, son ligeramente mejores en el primero de los casos que realizando un entrenamiento de la arquitectura completa en una única etapa. El número de épocas utilizado es de 90 para cada una de las dos etapas (180 en total).

El modelo se entrena utilizando la optimización de Adam del método del gradiente estocástico descendente [48], con un tamaño de batch de 6 (máximo valor que permite la memoria de la GPU utilizada), con un momentum de 0.9 y un parámetro de decaimiento de los pesos de 2e−4. La tasa de aprendizaje inicial se fija en 5e−4 y no se modifica durante el entrenamiento más que mediante los dos parámetros presentados anteriormente. Para establecer la función de pérdidas usada en el entrenamiento se usa la propuesta de [35], tomando como función para establecer el peso de cada clase $w_{clase} = \frac{1}{ln(c+p_{clase})}$, siendo p_{clase} la probabilidad de aparición de cada clase, obtenida mediante el análisis de los histogramas asociados a cada clase. El valor para la constante c se fija mediante un análisis empírico en 1.10. Se ha testeado el entrenamiento de un modelo partiendo de un encoder sin pre-entrenamiento y de un encoder pre-entrenado en Imagenet [49] para demostrar el beneficio de este segundo método.

Durante el proceso de entrenamiento solo se utilizan las imágenes pertenecientes al conjunto de entrenamiento, dado que los resultados se testearán sobre el conjunto de validación.

Como *data-augmentation* básico en todos los entrenamientos realizados en esta sección y en las posteriores se incluyen rotaciones aleatorias en la imagen, así como desplazamientos de entre 0 y 2 píxeles también de manera aleatoria.

La tabla 5.1 refleja los resultados obtenidos en este dataset por las dos arquitecturas básicas propuestas sin ningún tipo de pre-entrenamiento por un modelo de ERFNet con un encoder preentrenado en Imagenet comparadas con la mejor propuesta anterior del estado del arte: la OPPNet. Esta red es una red con una estructura de encoder-decoder similar a ERFNet, que consta de un bloque codificador encargado de extraer características y otro de decodificación basado en el módulo piramidal de PSPNet con 4 bloques de análisis con kérneles de 1x1, 2x2, 3x3 y 6x6.

Los datos presentados reflejan como la ERFNet supera claramente los resultados obtenidos por las mejores redes evaluadas anteriormente. La mejora del modelo sin pre-entrenamiento es del 3 % sobre la OPPNet y la del modelo que incorpora un encoder pre-entrenado es del 3.7 %. La nueva arquitectura propuesta con el modelo de decoder piramidal consigue resultados superiores a los previos del estado del arte, pero no consigue superar los de la arquitectura original de ERFNet. Sin embargo, cabe destacar que para algunas de las clases menos representadas como *Muro, Valla o Bicicleta* los resultados de esta arquitectura son ligeramente superiores (para el caso de la clase *Bicicleta* el incremento en la mejora es de más de un 12 %).

A pesar de que la naturaleza de las imágenes utilizadas en CityScapes y en Imagenet sea distinta (y

Tabla 5.1: IoU de clase (%) obtenido en el dataset sintético de f_0.

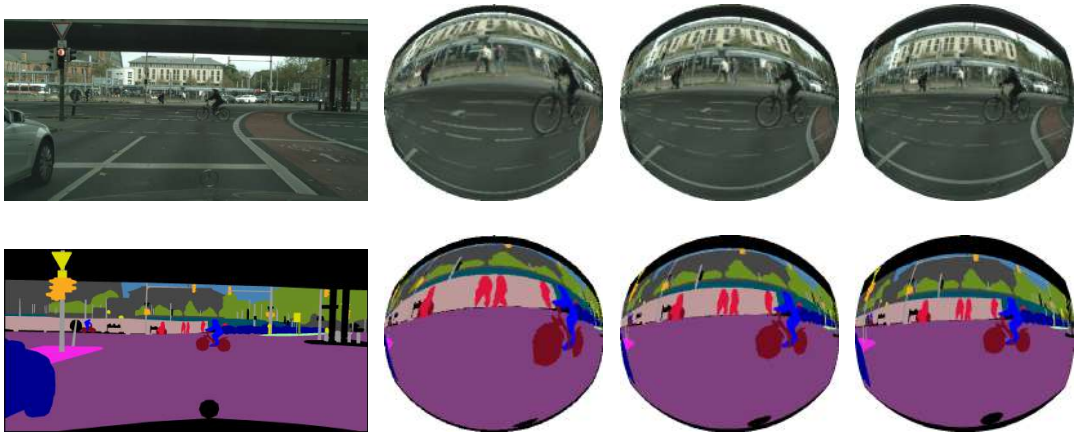
Clase	OPPNet	ERFNet	ERFNetPSP	ERFNet Pre-entrenada
Carretera	96.5	96.8	96.6	**96.9**
Acera	61.4	65.7	64.5	**65.8**
Edificio	78.4	79.3	77.4	**80.3**
Muro	23.7	30.2	**33.1**	31.7
Valla	22.8	24.5	22.9	**29.1**
Poste	24.6	**35.1**	28.5	34.7
Semáforo	28.4	32.8	30.2	**34.7**
Señal de tráfico	41.1	49.5	44.4	**51.5**
Vegetación	82.5	84.6	82.2	**84.9**
Terreno	39.1	45.6	45.9	**47.9**
Cielo	87.2	**87.9**	85.7	87.1
Peatón	63.3	67.9	62.2	**68.9**
Ciclista	34.2	46.3	41.9	**48.2**
Coche	85.8	87.4	86.0	**87.7**
Camión	40.2	42.8	42.6	**43.8**
Autobús	56.7	**66.1**	58.8	63.3
Tren	**39.2**	34.7	33.2	17.9
Motocicleta	**41.2**	38.4	21.9	38.6
Bicicleta	53.9	42.4	54.4	**56.5**
IoU (%)	52.6	55.6	53.3	**56.3**

más aún las del dataset sintético generado), la mejora del modelo con encoder pre-entrenado responde a la tendencia de los filtros de las primeras capas de las redes a actuar como detectores de bordes o de esquinas que son elementos comunes en la mayoría de tipos de imágenes. Utilizar un modelo pre-entrenado agiliza la convergencia del entrenamiento y, por lo general, desemboca en mejores resultados con independencia del tipo de imágenes analizadas.

A partir de este resultado se escoge la ERFNet pre-entrenada como línea base en el resto de experimentos propuestos en este capítulo.

5.3 Influencia de la distorsión

Como segundo experimento de este capítulo se plantea el estudio de la influencia de la distorsión sobre el rendimiento de las redes. Para ello se generan dos datasets alternativos al original con un parámetro de distorsión mayor y otro menor determinados por $f_1 = 96$ y $f_2 = 242$ respectivamente. Las imágenes generadas en ambos datasets son re-escaladas al tamaño de las del original (640x576) para permitir combinarlas en un mismo entrenamiento y ser utilizadas para testear distintos métodos de *data-augmentation* en apartados futuros. La figura 5.1 muestra un ejemplo de las diferentes distorsiones aplicadas sobre una misma imagen.



(a) Imágenes originales (b) $f_1 = 96$ (c) $f_0 = 159$ (d) $f_2 = 242$

Figura 5.1: Ejemplos de imágenes y de ground-truth de los 3 datasets con diferentes distorsiones

Como se puede observar en la figura anterior los parámetros de distorsión grandes (asociados a distancias focales pequeñas) generan imágenes más circulares, con un nivel de zoom mayor y con peor resolución, mientras que las distancias focales grandes devuelven imágenes menos circulares, con menor zoom y con mejor resolución.

5.3.1 Resultados Cuantitativos

Para llevar a cabo la comparativa en los resultados de la red se realizan 2 entrenamientos adicionales sobre los nuevos datasets utilizando los mismos parámetros empleados en el experimento anterior tanto para la etapa de entrenamiento como para la de test de los resultados. Además, se testean los resultados que obtendría la red ERFNet original entrenada en CityScapes sin modificar en el test conjunto de validación con $f_0 = 159$. Los resultados de los 4 experimentos se presentan en la tabla 5.2.

El primer resultado reflejado en la tabla es que la red con el entrenamiento básico en CityScapes original no es capaz de abordar la distorsión correctamente, siendo la pérdida de precisión de un 18.1 % con respecto del modelo entrenado en el dataset sintético.

Los resultados mostrados en la tabla también reflejan que el rendimiento de la red se degrada a medida que la distorsión aumenta. Entre los modelos con mayor y menor distorsión la diferencia en el rendimiento es de 13,6 %. También se muestra que esta degradación es especialmente brusca las clases con menor representación en el dataset como *Tren* (27.6 a 0.9), *Autobús* (66.1 a 4.2) o *Valla* (21.3 a 8.5).

La degradación del rendimiento de la red tiene dos motivos principales: el primero de ellos es la pérdida de resolución de las imágenes en función de la distorsión añadida al generar los datasets. Cuanto mayor

Tabla 5.2: Influencia de la distorsión sobre los resultados de la red

Clase	**ERFNet Original**	**ERFNet** $f_1 = 96$	**ERFNet-Pre**	**ERFNet** $f_2 = 242$
Carretera	84.1	95.6	96.9	97.4
Acera	51.2	56.9	65.8	70.4
Edificio	60.6	73.2	80.3	83.8
Muro	19.1	20.7	31.7	28.3
Valla	8.5	16.8	29.1	38.0
Poste	20.0	27.5	34.7	38.9
Semáforo	19.4	23.0	34.7	39.6
Señal de tráfico	32.1	40.8	51.5	56.2
Vegetación	77.8	78.8	84.8	86.8
Terreno	36.3	36.1	47.9	48.7
Cielo	85.6	82.6	87.1	89.4
Persona	43.2	60.4	68.9	71.0
Ciclista	29.7	37.3	48.2	49.2
Coche	63.6	83.5	87.7	89.0
Camión	33.5	24.6	43.8	51.2
Autobús	4.2	49.9	63.3	63.4
Tren	0.9	6.9	17.9	39.1
Motocicleta	21.9	20.9	38.6	42.8
Bicicleta	35.0	49.7	56.5	60.0
IoU (%)	38.2	46.6	56.3	60.2

es la distorsión introducida menor es la resolución final de la imagen lo que dificulta el aprendizaje de la red. El segundo motivo es que cuánto mayor es la distorsión introducida mayor es la variabilidad en la apariencia de los objetos que aparecen en las imágenes, lo que obliga a la red a obtener una mejor capacidad de generalización durante el entrenamiento, complicando su aprendizaje.

De manera adicional, se realizan otros 4 entrenamientos con valores de distorsión menores para ver cómo evoluciona la precisión de la red de manera gráfica. Se toman como distancias focales valores que dan lugar a imágenes con valores de anchura y altura divisibles por 8 (o valores muy cercanos a ello) $f_3 = 303$, $f_4 = 424$, $f_5 = 484$ y $f_6 = 602$. Los resultados de estos entrenamientos junto con el rendimiento de la ERFNet original en CityScapes se reflejan en el gráfico de la figura 5.2.

Como se puede observar en la figura anterior, eliminar la distorsión gradualmente hace que los

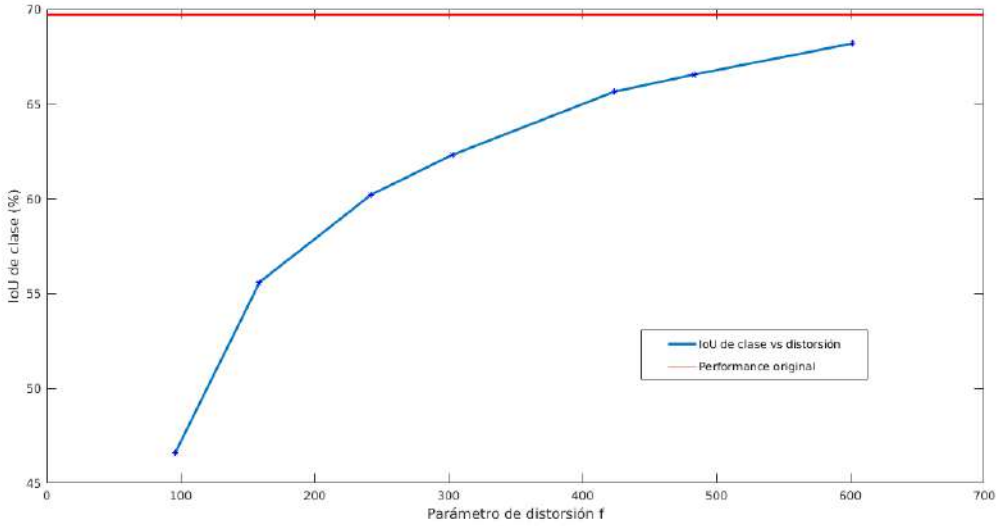


Figura 5.2: Performance de la red frente a parámetro de distorsión

resultados de la red se aproximen asintóticamente a los valores obtenidos por la red en el dataset original, como por otro lado cabe esperar.

5.3.2 Resultados Cualitativos

La figura 5.3 muestra dos ejemplos de la segmentación generada por las redes entrenadas en este apartado: la primera columna se corresponde con los resultados de la red ERFNet entrenada en el dataset de CityScapes original y testeada sobre el dataset sintético definido por f_0. Las otras 3 columnas se corresponden con las redes entrenadas en los 3 datasets sintéticos generados f_0, f_1 y f_2.

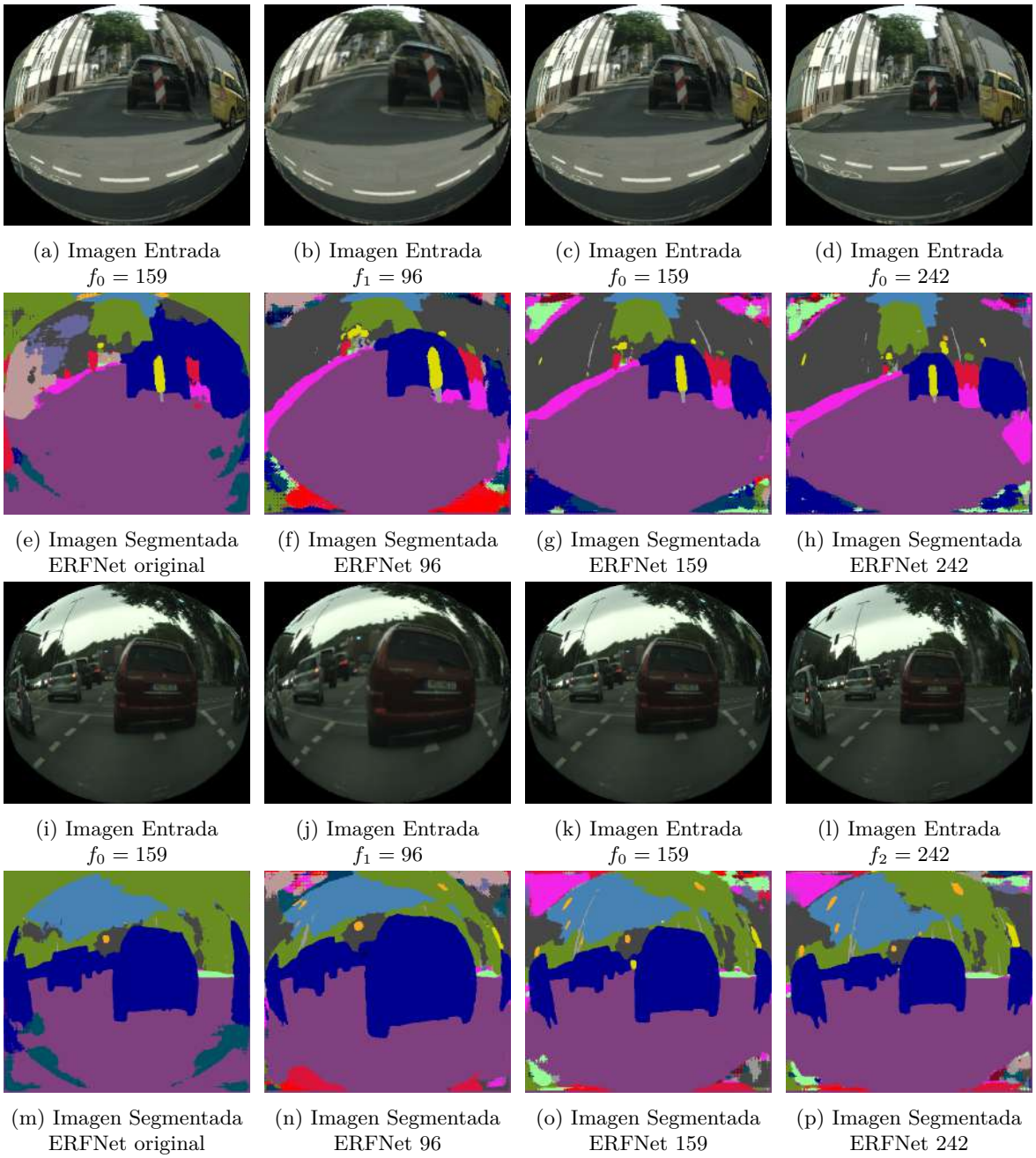
La figura 5.3 muestra claramente como la red original es incapaz de gestionar la distorsión añadida, especialmente la de las zonas periféricas, en las que la segmentación es de muy mala calidad (figuras 5.3e y 5.3m). Los resultados obtenidos con entrenamientos específicos para fisheye son claramente superiores, mejorando especialmente para las zonas con mayor distorsión y para las clases más pequeñas. Para este tipo de clases, además, se puede observar como aparecen mejor identificadas a medida que se reduce la distorsión (aumentando a la vez la resolución de la imagen). Esto se ve claramente reflejado en las señales de tráfico y en los semáforos de las figuras 5.3n 5.3o y 5.3p.

5.4 Data-Augmentation

5.4.1 Fixed Zoom-Augmentation

Los datasets con $f_1 = 96$ y $f_2 = 242$ generados en el apartado anterior pueden utilizarse para aplicar una técnica de *data-augmentation* propuesta en [22] combinándolos con el dataset sintético original con $f_0 = 159$ para dar lugar a un dataset con mayor número de imágenes y mayor variedad de distorsiones que ayudan a mejorar la capacidad de generalización de las redes en las que la técnica ha sido testeada. El método fue denominado como *zoom-augmentation*.

La selección de los parámetros de distorsión utilizados para generar las imágenes adicionales del dataset se realiza en base al *aspect-ratio* de las imágenes de salida de la transformación. Los dos valores



(a) Imagen Entrada $f_0 = 159$ (b) Imagen Entrada $f_1 = 96$ (c) Imagen Entrada $f_0 = 159$ (d) Imagen Entrada $f_0 = 242$

(e) Imagen Segmentada ERFNet original (f) Imagen Segmentada ERFNet 96 (g) Imagen Segmentada ERFNet 159 (h) Imagen Segmentada ERFNet 242

(i) Imagen Entrada $f_0 = 159$ (j) Imagen Entrada $f_1 = 96$ (k) Imagen Entrada $f_0 = 159$ (l) Imagen Entrada $f_2 = 242$

(m) Imagen Segmentada ERFNet original (n) Imagen Segmentada ERFNet 96 (o) Imagen Segmentada ERFNet 159 (p) Imagen Segmentada ERFNet 242

Figura 5.3: Ejemplos de segmentación semántica para diferentes distorsiones

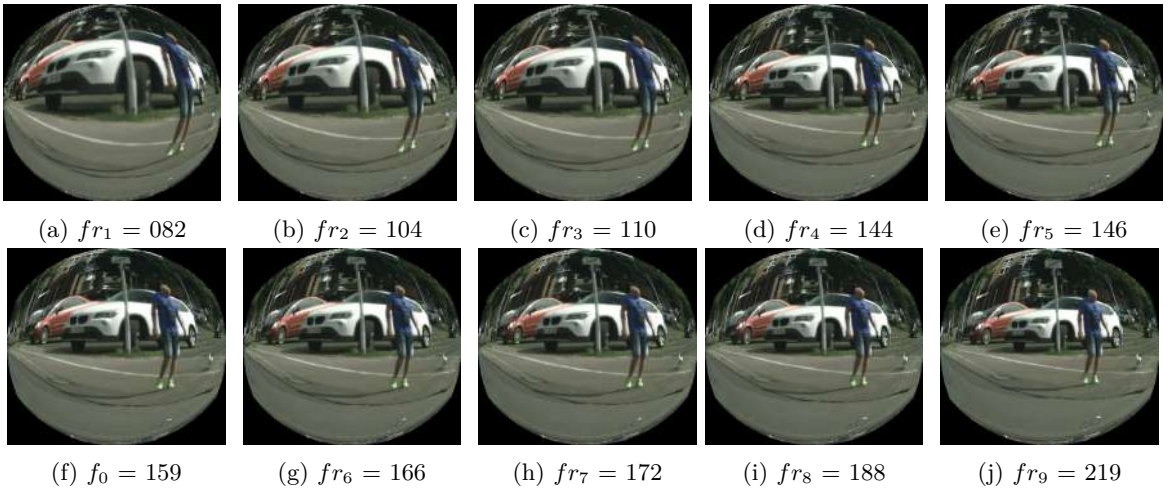
son escogidos específicamente para no alterar el *aspect-ratio* de las imágenes de salida más de 0.1 con respecto al parámetro tomado como línea base $f_0 = 159$.

5.4.2 Random Zoom-Augmentation

Como mecanismo alternativo a la generación de imágenes mediante valores de distorsión fijos calculados empíricamente de forma previa, en este trabajo se evalúa la posibilidad de utilizar distorsiones aleatorias para cada una de las imágenes empleadas en el entrenamiento. Esta técnica de *data-augmentation* se denominó *random-zoom-augmentation* y fue implementada en C++ mediante OpenCV, dado que permite operar a nivel de píxel de forma eficiente.

Para la implementación de la nueva técnica se utilizaron 9 imágenes adicionales con distorsiones

aleatorias junto con la de la distorsión original ($f_0 = 159$). Se define un límite máximo y otro mínimo para las posibles distorsiones, que se seleccionan para ser ligeramente menos restrictivos que los establecidos por la versión fija del algoritmo: $f_{inf} = 80$ y $f_{sup} = 250$. La distribución aleatoria seleccionada para la generación de valores es una Gaussiana con media centrada en la distorsión de referencia y $\sigma^2 = 40{,}0$. La figura 5.4 muestra un ejemplo de las distorsiones utilizadas para una imagen en el entrenamiento.



(a) $fr_1 = 082$ (b) $fr_2 = 104$ (c) $fr_3 = 110$ (d) $fr_4 = 144$ (e) $fr_5 = 146$

(f) $f_0 = 159$ (g) $fr_6 = 166$ (h) $fr_7 = 172$ (i) $fr_8 = 188$ (j) $fr_9 = 219$

Figura 5.4: Ejemplos de imágenes distorsionadas con valores aleatorios

Así, el entrenamiento final se lleva a cabo con un total de 29703 imágenes diferentes con las mismas dimensiones que en el resto de entrenamientos.

5.4.3 Adaptación de dominio

Como técnica complementaria se propone el uso de múltiples tipos de *data-augmentation* adicionales [50] para preparar a la red para diferentes dominios con apariencias distintas al utilizado para el entrenamiento. Este *data-augmentation* intensivo incluye la variación del aspect ratio de las imágenes de entrada, el uso de *random-cropping* (consistente en seleccionar de forma aleatoria regiones cuadradas de las imágenes y utilizarlas en el entrenamiento) y la adición de color-jittering consistente en la variación aleatoria del brillo, el contraste y la saturación de los colores presentes en la imagen (de manera uniforme). Se trata de un conjunto de transformaciones orientadas a la preparación del entrenamiento para su aplicación al caso del mundo real, pero como se puede comprobar en la siguiente sección la técnica proporciona también beneficios a nivel cuantitativo. Las principales técnicas aplicadas se encuentran representadas en las siguientes figuras.

El primer par de figuras 5.5 representa la aplicación de una rotación (representada en la figura 5.5b) a una imagen utilizada para entrenar (figura 5.5a). Las rotaciones introducidas son aleatorias y varían entre -π y π.

En el segundo par de imágenes 5.6 se ilustra tanto la aplicación de una modificación del contraste (en este caso se ve incrementado) como la de una técnica de mirroring que invierte la imagen (figura 5.6b). La magnitud de la modificación del contraste es aleatoria y puede tanto incrementarlo como disminuirlo.

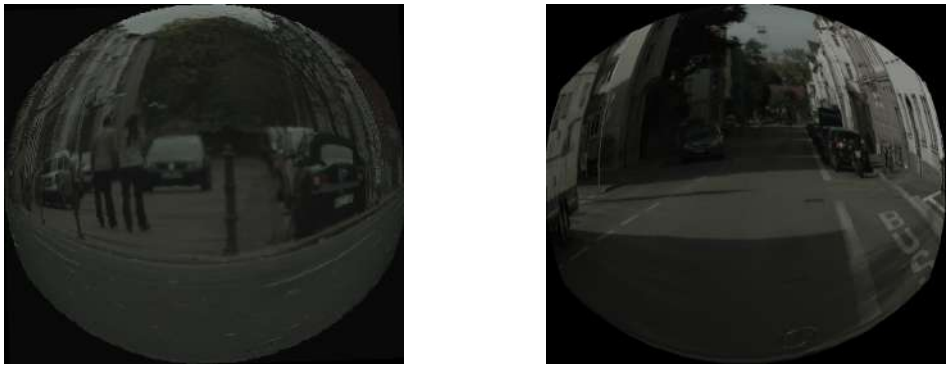
El tercer par de imágenes muestran la modificación del brillo de la imagen de entrada de manera aleatoria (figura 5.7). La primera figura 5.7a muestra la reducción del brillo de una imagen mientras que la figura 5.7b muestra el aumento del mismo.

(a) Imagen de Entrenamiento 1 (b) Ejemplo de adición de rotación

Figura 5.5: Ejemplos de aplicación de rotaciones aleatorias

(a) Imagen de entrenamiento 2 (b) Contraste y mirroring

Figura 5.6: Ejemplo de modificación del contraste y de aplicación de mirroring



(a) Ejemplo de modificación del brillo 1 (b) Ejemplo de modificación del brillo 2

Figura 5.7: Ejemplo de modificación del brillo

Por último, el cuarto par muestra el uso de random cropping durante el entrenamiento representado en la figura 5.8. En el proceso se selecciona un área de la imagen que se re-escala al tamaño completo de la imagen (640x576) (figura 5.8b) junto con su ground-truth asociado y se utiliza para el entrenamiento.

5.4.4 Resultados Cuantitativos

Las tres técnicas planteadas anteriormente fueron testeadas por separado y combinadas sobre el conjunto de validación del dataset con $f_0 = 159$ sobre las dos arquitecturas de red para ERFNet expuestas

(a) Imagen de entrenamiento 3 (b) Ejemplo de Random-Cropping

Figura 5.8: Ejemplo de aplicación de random-cropping

anteriormente y, además, se compararon con la mejor propuesta del estado del arte (OPPNet) combinada también con el mismo tipo de *data-augmentation*.

La tabla 5.3 refleja los resultados finales de las distintas técnicas aplicadas sobre el entrenamiento básico. Las tres primeras columnas representan resultados para la técnica de *zoom-augmentation* fijo (*****), la cuarta representa el resultado de la nueva propuesta de *zoom-augmentation* aleatorio (**rnd**) y las dos últimas son el resultado de la combinación de las técnicas de adaptación de dominio y del *zoom-augmentation* fijo (******).

Los resultados presentados en la tabla muestran como todas las técnicas introducidas mejoran notablemente los resultados de las redes:

- El mejor modelo previo del estado del arte era OPPNet*, a pesar de ser superado por todas los modelos propuestos en este trabajo, es el segundo que más beneficiado se ve por el *data-augmentation*, alcanzando una mejora del 1,9 % con el uso de distorsiones fijas adicionales.

- El entrenamiento original de ERFNet* (con encoder pre-entrenado en Imagenet) obtiene una mejora del 0,7 % con la adición de distorsiones adicionales fijas, manteniéndose como la mejor arquitectura también para entrenamientos con este tipo de *data-augmentation*.

- El modelo de ERFNet con decoder piramidal (ERFNetPSP*) es el que mayor mejora presenta debido a la incorporación de *data-augmentation*, alcanzando un incremento del IoU de clase del 2,3 % para la técnica de *zoom-augmentation* fija.

- El uso complementario de las técnicas de adaptación de dominio supone una mejor adicional en el rendimiento que en el caso de la ERFNet se dispara hasta el 3,2 % llegando el IoU de clase hasta un notable **59,3** % y en el caso de la ERFNetPSP es de 3,8 % alcanzando un 58,3 %.

- La nueva propuesta de uso de distorsiones aleatorias (rnd ERFNet) alcanza un resultado similar al de las fijas tanto a nivel cualitativo como cuantitativo, quedándose tan solo un 0,2 % por debajo de el rendimiento del método alternativo (ERFNet*).

La propuesta de *zoom-augmentation* aleatoria introduce el beneficio de evitar el estudio previo sobre qué distorsiones son óptimas para aplicar la primera técnica por lo que puede considerarse como una alternativa clara al primer método. Por este motivo y por ser una propuesta propia de este trabajo, se considerará como la técnica básica de *data-augmentation* (junto con la de adaptación de dominio) utilizada en el resto del trabajo.

Tabla 5.3: IoU de clase (%) para diferentes redes y técnicas de *data-augmentation*

Técnica	**OPPNet***	**ERFNet***	**ERFNetPSP***	**rnd ERFNet**	**ERFNetPSP****	**ERFNet****
Método Augmentation	**Fixed**	**Fixed**	**Fixed**	**Random**	**Fixed y dominio**	**Fixed y dominio**
Carretera	96.7	96.9	96.8	96.9	**97.2**	97.1
Acera	63.5	66.8	64.8	67.6	**68.8**	67.6
Edificio	79.6	80.3	79.4	80.7	81.3	**81.5**
Muro	26.9	34.4	32.7	31.1	31.9	**35.0**
Valla	25.4	23.8	25.5	21.9	**27.5**	26.3
Poste	25.6	36.3	31.2	36.2	32.9	**37.3**
Semáforo	30.6	36.2	34.1	37.4	36.5	**38.8**
Señal tráfico	44.0	50.1	46.4	49.1	49.8	**52.4**
Vegetación	83.2	**85.0**	84.7	84.7	83.8	**85.0**
Terreno	43.0	47.9	46.4	46.1	**49.2**	48.1
Cielo	88.8	87.3	87.8	88.5	88.0	**88.9**
Persona	65.7	69.0	67.8	68.5	69.7	**69.8**
Ciclista	39.4	47.6	45.5	48.8	**50.4**	50.2
Coche	86.7	87.7	87.5	87.7	**89.0**	**89.0**
Camión	48.6	47.6	50.0	42.7	**60.1**	56.6
Bus	55.3	64.6	65.1	67.8	66.3	**71.5**
Tren	**37.2**	22.9	23.5	24.9	18.7	24.9
Motocicleta	40.1	41.8	38.1	41.5	**46.5**	45.2
Bicicleta	55.4	57.0	56.2	58.4	59.5	**60.5**
IoU (%)	54.5	57.0	55.9	56.8	58.3	**59.3**

5.4.5 Resultados del Modelo con Decoder Piramidal

Los resultados del modelo ERFNetPSP, a pesar de ser más cercanos a los obtenidos por el modelo original de ERFNet en los entrenamientos iniciales, siguen siendo inferiores. Sin embargo, en publicaciones previas se había defendido la importancia de primar la información contextual con este tipo de óptica y se habían aportado datos cuantitativos [22] de la mejora que suponía utilizar arquitecturas con módulos similares al de PSPNet. Para encontrar el motivo del fracaso de esta nueva arquitectura en estos primeros entrenamientos se visualizaron los mapas de características presentes a la salida del encoder tanto para la arquitectura con el decoder piramidal como para la original. Con dicho fin se añadió una capa convolucional adicional tras la última capa del encoder para generar mapas de características representables, como muestra la figura 5.9.

A pesar de que la etapa de entrenamiento del encoder es común en los entrenamientos de ambas arquitecturas, dado que en la segunda etapa se entrena el encoder de manera conjunta con el decoder, se ha independizado la representación de los mapas de características generados por ambas arquitecturas dado que son distintos. La figura 5.10 representa estos mapas para varias imágenes utilizadas en el entrenamiento:

En la figura anterior, las imágenes de entrada están representadas en las sub-figuras 5.10a y 5.10d,

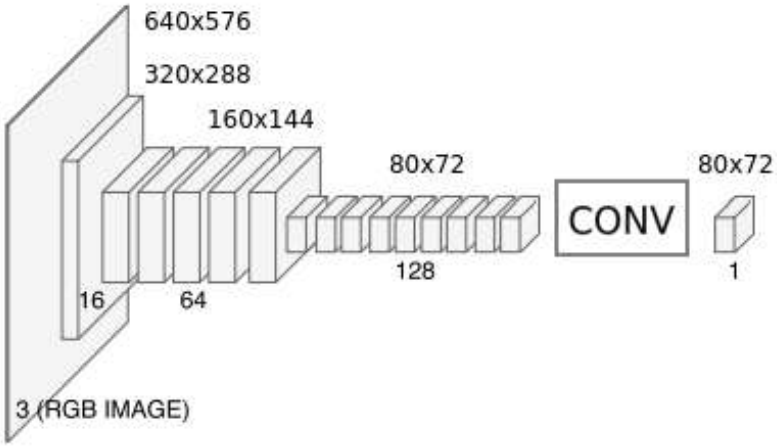
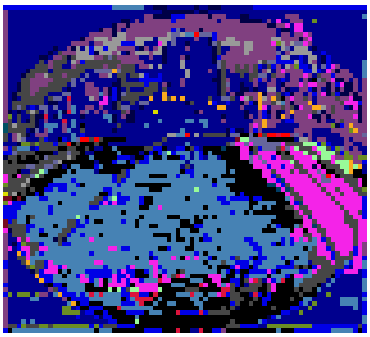


Figura 5.9: Punto de obtención de la representación de las características

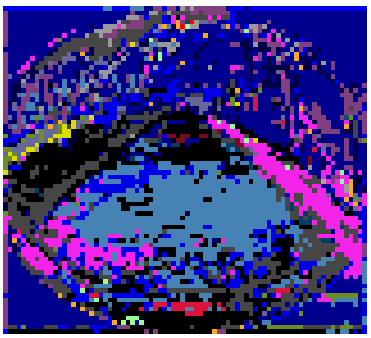
(a) Imagen Original 1



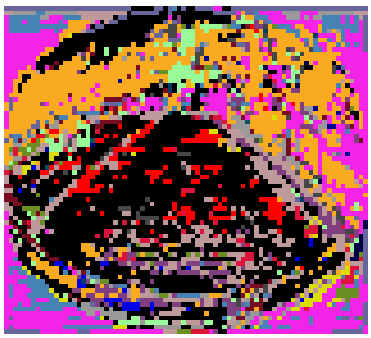
(b) Encoder ERFNet

(c) Encoder ERFNetPSP

(d) Imagen Original 2



(e) Encoder ERFNet



(f) Encoder ERFNetPSP

Figura 5.10: Mapas de características generados por las dos arquitecturas para varias imágenes de entrada

las sub-figuras 5.10b y 5.10e contienen los mapas de características para la arquitectura ERFNet y las sub-figuras 5.10c y 5.10f para la versión con el decoder piramidal. Las imágenes asociadas a los mapas se encuentran re-escaladas, dado que su resolución real es de 80x72 píxeles y, en ellas, los colores presentados en los mapas de características no están asociados con los definidos por CityScapes sino que representan valores numéricos escalares.

Como se puede observar, en los mapas de características generados por el encoder se pueden identificar fácilmente las formas de todos los elementos presentes en la imagen y los colores utilizados prácticamente permiten distinguir las diferentes clases existentes. Los mapas generados por ambas arquitecturas son similares, pero el generado por la ERFNet original es más homogéneo: las regiones extensas de la imagen

como la carretera o la acera en ambos ejemplos presentan una textura homogénea, mientras que la arquitectura con el modelo piramidal da lugar a mapas más heterogéneos en el que las clases menos representadas se identifican con mayor facilidad (como se ve en los postes representados en la figura 5.10c), pero no así para las clases más grandes.

A la vista de las imágenes se puede afirmar que el modelo piramidal conserva mejor la información contextual, dado que las clases que tienden a estar presentes en una determinada región de la imagen (como el cielo, que habitualmente está en la superior) o las clases menos representadas en las que, generalmente, la información contextual resulta más relevante son mejor identificadas en los mapas de características. El hecho de que los resultados finales de la arquitectura sean inferiores a los de la original se corresponden con la reducción de la escala producida por los 3 procesos de downsampling presentes en el encoder. La reducción del tamaño de la imagen hasta 80x72 píxeles perjudica la información contextual ya que a partir de una resolución tan baja es muy complejo extraer información útil. Como se podrá ver en experimentos posteriores, la diferencia en el rendimiento de las redes se reduce a medida que la resolución de las imágenes de entrada aumenta, hasta que llega un punto en el que el rendimiento de la red con el modelo piramidal termina siendo superior a la de la arquitectura original.

5.4.6 Resultados Cualitativos

En las siguientes figuras se presentan resultados cualitativos para todas las redes entrenadas en este capítulo. La primera figura 5.11 presenta cuatro imágenes RGB de entrada acompañadas de sus respectivos ground-truths:

Figura 5.11: Imágenes de entrada y GT ejemplo

La figura 5.12 muestra la segmentación generada por cada una de las 6 redes entrenadas para las 4 imágenes anteriores: la primera fila (a) contiene los resultados para el entrenamiento básico sin *data-augmentation*; la segunda fila (b) contiene los resultados de aplicar la nueva propuesta de *data-augmentation* aleatorio; la tercera (c) y cuarta fila (d) contienen los resultados del modelo con decoder piramidal sin y con técnicas de adaptación de dominio respectivamente junto con distorsiones adicionales fijas y la quinta (e) y sexta fila (f) contienen los entrenamientos análogos (*zoom-augmentation* y *zoom-augmentation* junto con adaptación de dominio) pero para el modelo de ERFNet original.

Las figuras 5.13a, 5.14a y 5.15a exponen ejemplos de segmentación al detalle obtenidos para dos imágenes de test utilizando los 6 entrenamientos realizados. Los resultados mostrados corroboran los

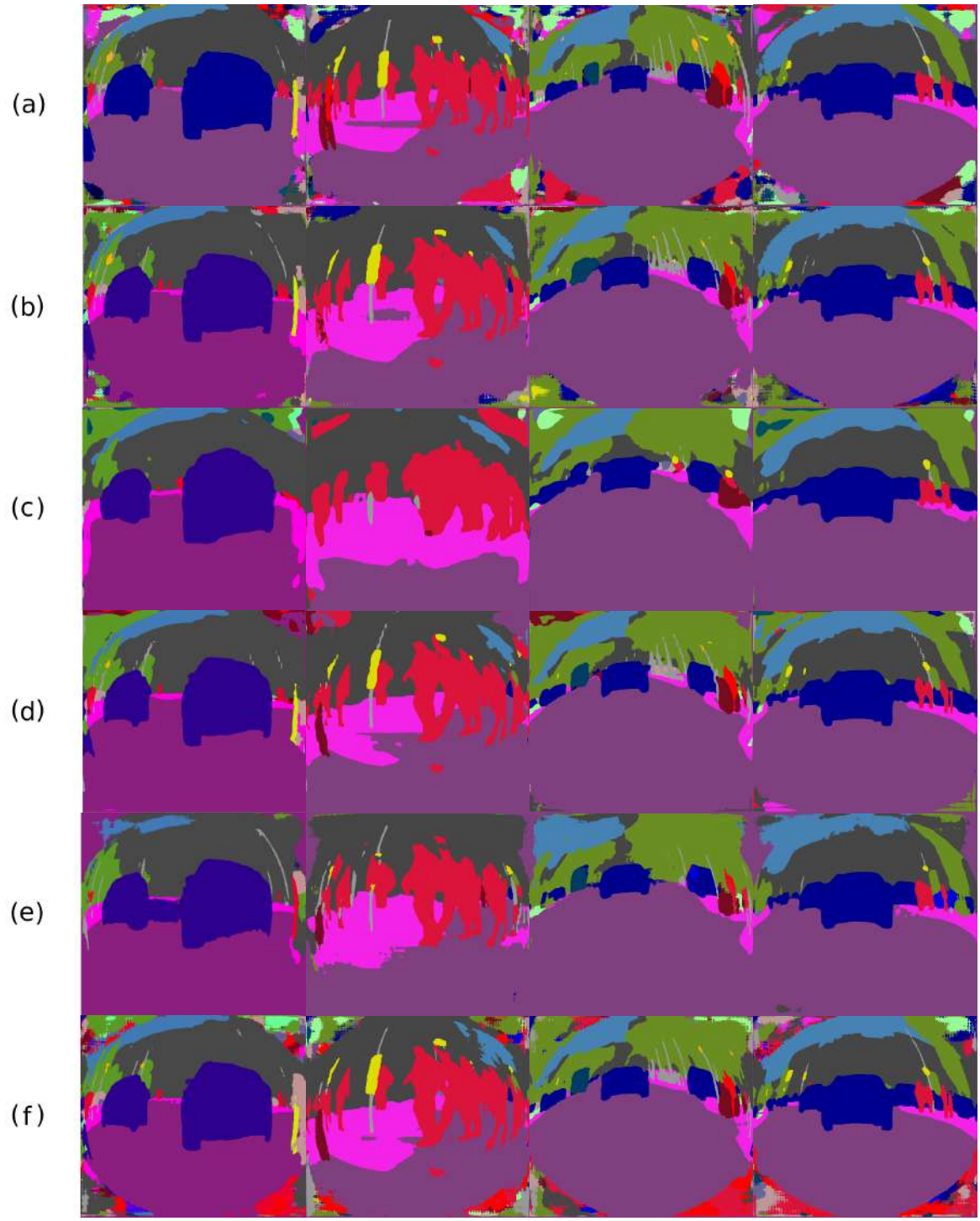


Figura 5.12: Ejemplos de segmentación semántica con las distintas redes entrenadas: (a) ERFNet, (b) ERFNet y random z-aug, (c) ERFNetPSP, (d) ERFNetPSP, Fixed z-aug y adaptación de dominio, (e) ERFNet y Fixed z-aug (e) ERFNet, Fixed z-aug y adaptación de dominio

cuantitativos presentados en la sección anterior:

- La red con el entrenamiento adaptado a $f_0 = 159$ es capaz de gestionar la distorsión introducida de manera correcta, pero apenas es capaz de identificar las clases menos representadas como las señales de tráfico en 5.13c, el camión en 5.14c o el semáforo en 5.15c.
- La red con el *data-augmentation* que incluye distorsiones aleatorias muestra mejor capacidad para

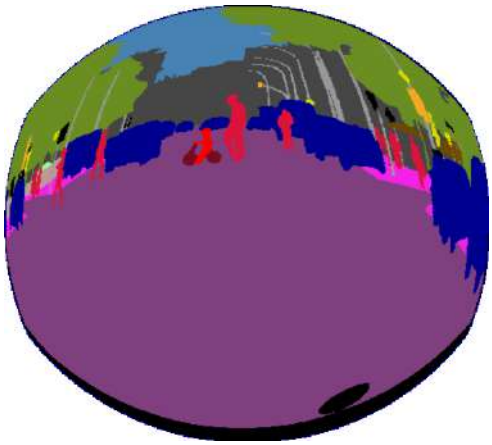
identificar las clases menos representadas. En 5.14d la red ya es capaz de identificar el camión de la imagen y los postes, señales de tráfico y semáforos aparecen correctamente detectados.

- El modelo con decoder piramidal con distorsiones fijas ofrece una salida a nivel de píxel "grueso", que proporciona objetos con formas geométricas muy regulares sin ser capaz de detectar las clases menores. Sin embargo, el modelo que además incorpora la adaptación de dominio en el entrenamiento es el que más beneficiado se ve por el uso de *data-augmentation*: las figuras 5.13f, 5.14f y 5.15f muestran una segmentación semántica de muy buena calidad aunque aún con bordes demasiado regulares que no se ajustan bien a los contornos de los objetos. A cambio apenas genera errores de píxeles aislados como el resto de modelos.

- Los dos últimos modelos presentados son los asociados a la arquitectura ERFNet original. El primero (5.13g, 5.14g y 5.15g) muestra una segmentación de calidad similar a la presentada por el modelo con decoder piramidal junto con técnicas de adaptación de dominio pero siendo capaz de detectar las formas de los elementos de la escena de forma refinada. El modelo con las técnicas de adaptación de dominio (5.13h, 5.14h y 5.15h) muestra una capacidad para segmentar superior. En la figura 5.14h se puede observar la precisión de la red a la hora de detectar el triciclo del centro de la imagen como bicicleta y el niño montado encima como ciclista.

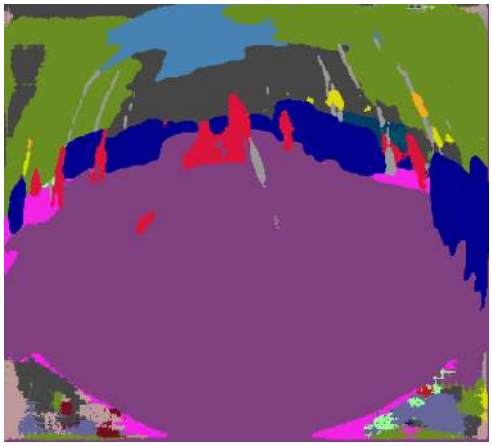
La mejora del último entrenamiento con respecto del mejor del estado del arte previo es de un 4,8 %, lo que representa una mejoría más que destacable. Esta mejora junto con la propuesta de la nueva técnica de *data-augmentation* desembocaron en una publicación [51] en el IEEE Intelligent Vehicles Symposium presentado en junio de 2018 en Changshu (Suzhou, China) con título *CNN-based Fisheye Image Real-Time Semantic Segmentation.*

(a) Imagen de entrada 1

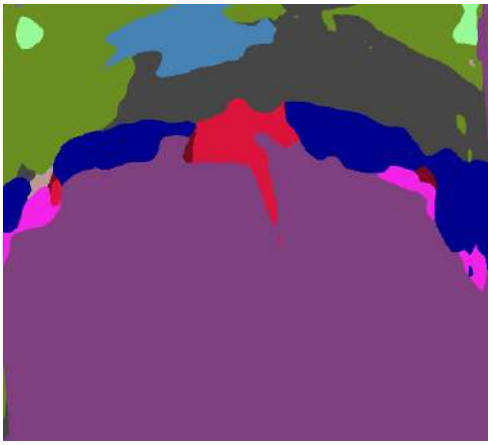


(b) Ground-Truth

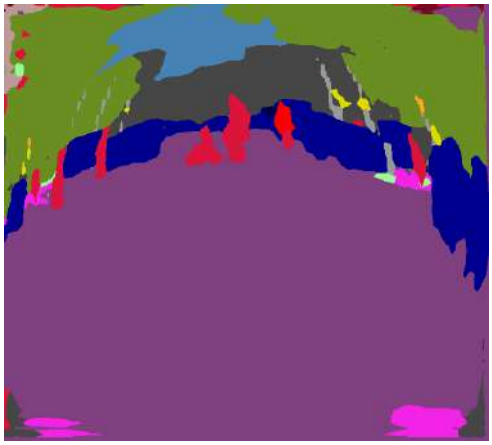
(c) ERFNet $f_0 = 159$



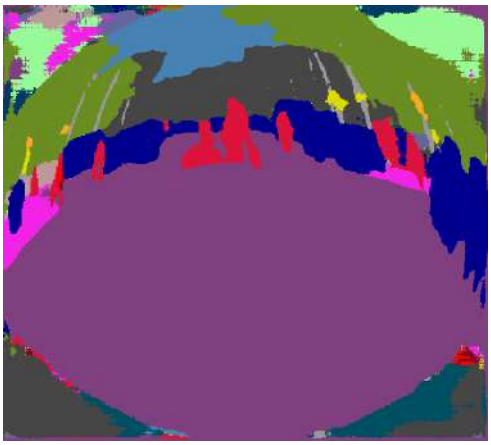
(d) ERFNet con rnd ZA



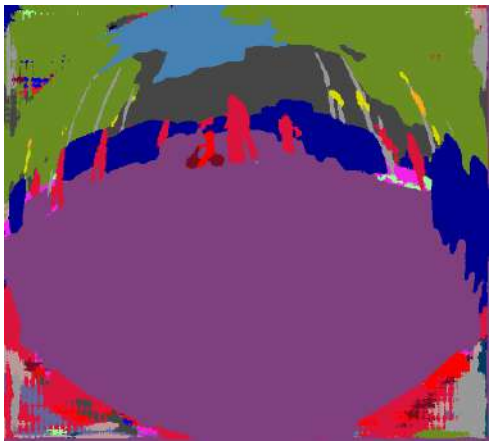
(e) ERFNetPSP con f-ZA



(f) ERFNetPSP con f-ZA y dominio



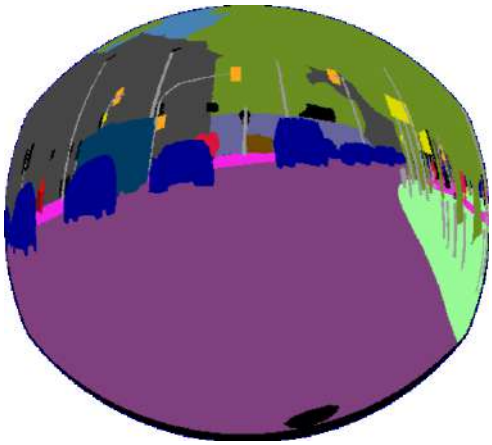
(g) ERFNet con f-ZA



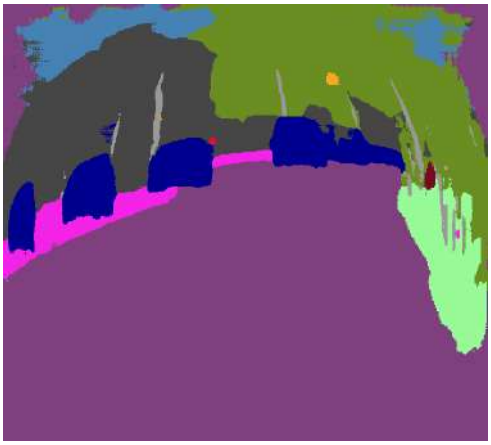
(h) ERFNet con f-ZA y dominio

Figura 5.13: Ejemplo 1 de segmentación de las redes entrenadas

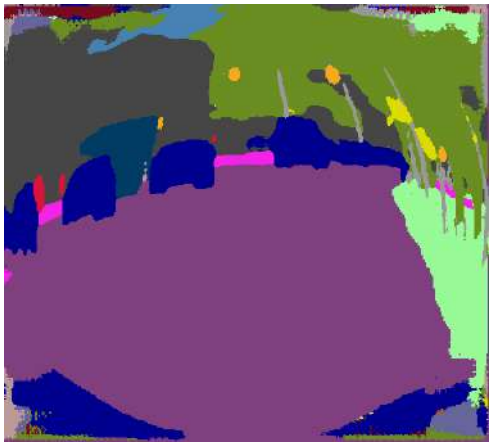
(a) Imagen de entrada 2



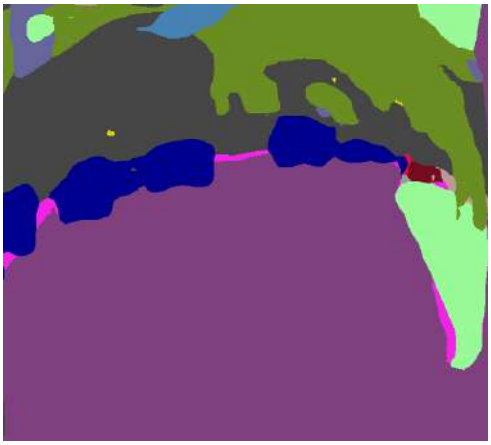
(b) Ground-Truth



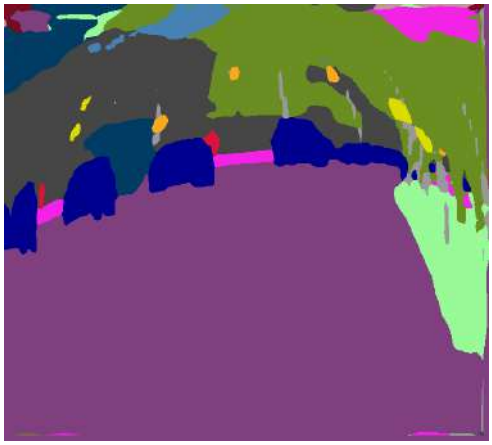
(c) ERFNet $f_0 = 159$



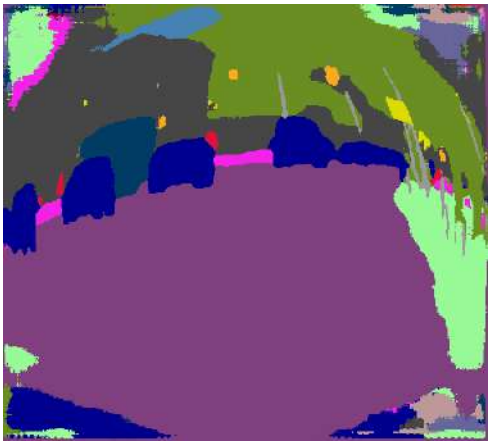
(d) ERFNet con rnd ZA



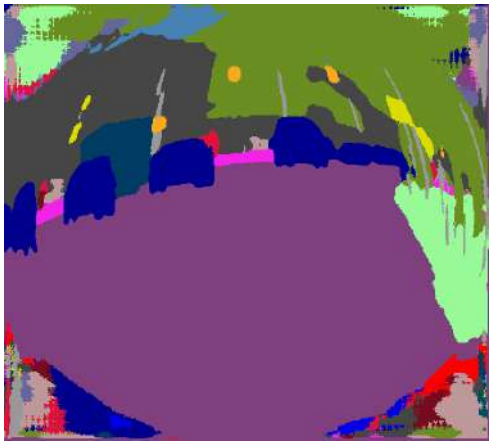
(e) ERFNetPSP con f-ZA



(f) ERFNetPSP con f-ZA y dominio



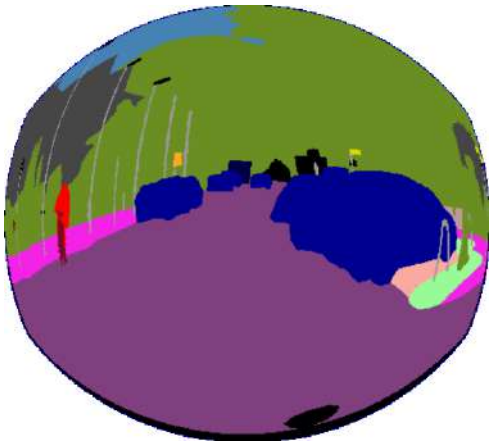
(g) ERFNet con f-ZA



(h) ERFNet con f-ZA y dominio

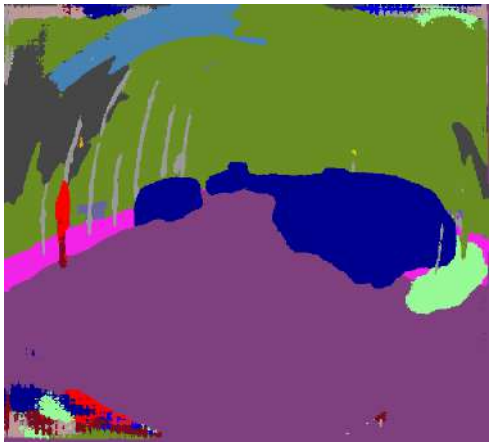
Figura 5.14: Ejemplo 2 de segmentación de las redes entrenadas

(a) Imagen de entrada 3



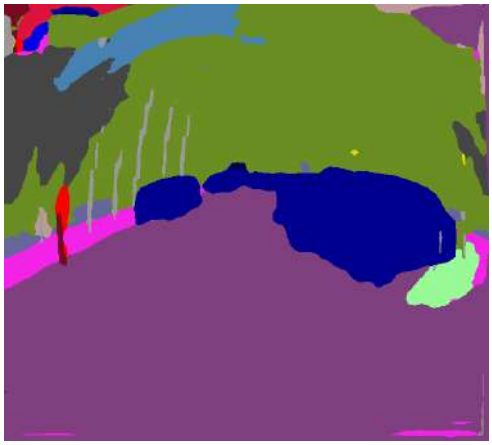
(b) Ground-Truth

(c) ERFNet $f_0 = 159$

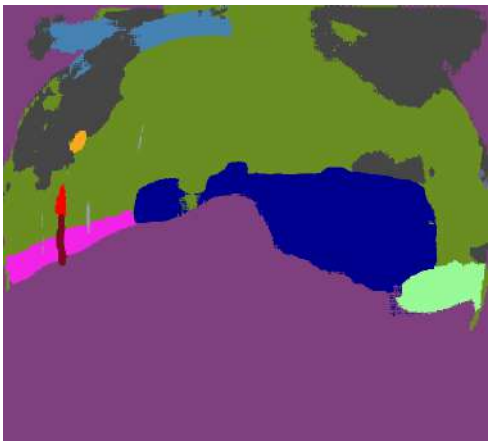


(d) ERFNet con rnd ZA

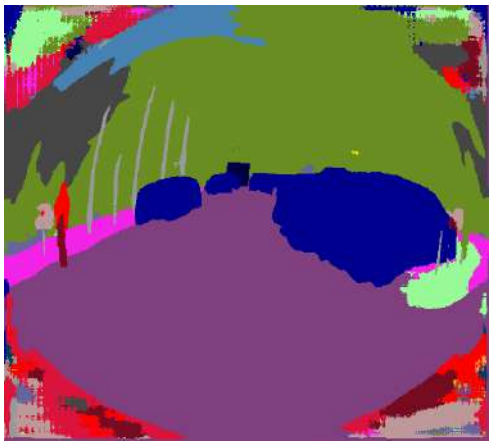
(e) ERFNetPSP con f-ZA



(f) ERFNetPSP con f-ZA y dominio



(g) ERFNet con f-ZA



(h) ERFNet con f-ZA y dominio

Figura 5.15: Ejemplo 3 de segmentación de las redes entrenadas

Capítulo 6

Arquitecturas Alternativas

6.1 Introducción

En el capítulo anterior se presentaron los resultados obtenidos sobre los datasets sintéticos generados y se realizó una comparativa con otros trabajos similares del estado del arte. Sin embargo, el número de redes utilizadas para realizar la segmentación no era muy grande dado que pocas arquitecturas han sido testeadas sobre datasets sintéticos de estas características.

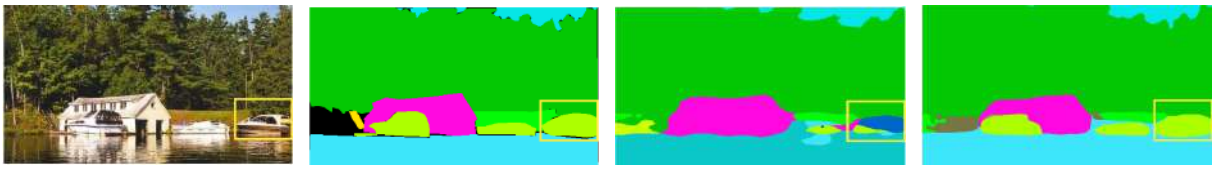
Este capítulo analiza una serie de redes alternativas a las utilizadas previamente que incluyen algún elemento de interés para el trabajo, así como una nueva variación en el entrenamiento llevado a cabo hasta ahora.

6.2 PSPNet

Pyramid Scene Parsing Network (PSPNet) [39] es una red neuronal convolucional que proporciona una segmentación a nivel de píxel de muy alta calidad a cambio de un coste computacional grande y de un tiempo de procesado elevado. La red consiguió el primer puesto en el desafío de segmentación de imágenes de ImageNet 2016 y actualmente mantiene posiciones destacables en rankings de datasets muy conocidos como CityScapes (top 11) o VOC2012 (top 10).

La red presenta una arquitectura enfocada al análisis de la información contextual de las imágenes, bajo la idea de que la precisión a la hora de realizar una segmentación a nivel de píxel puede ser incrementada notablemente utilizando información de la imagen a nivel global [52]. Para ello, se recurre al uso de la agrupación de pirámides espaciales que analizan la escena a diferentes escalas o resoluciones que al ser combinadas a posteriori relacionan las diferentes regiones de la imagen [53]. Así se solucionan ciertos problemas presentados al segmentar objetos de apariencias similares pero pertenecientes a diferentes clases. Además, el trabajo también incluía una nueva propuesta para la optimización del entrenamiento mediante el control de la pérdida propagada.

El problema mencionado anteriormente se ve ilustrado por la figura 6.1 en la que se puede observar como una FCN tradicional confunde el barco presente en la primera imagen con un coche (figura 6.1a) debido a la similitud en la apariencia de ambos elementos (forma, textura, color, etc). Sin embargo, la arquitectura de PSPNet evita este problema al analizar la imagen tanto de forma local como a nivel global, lo que le permite relacionar el barco con la región inferior en la que hay presente una masa de agua para realizar una clasificación a nivel de píxel correcta (figura 6.1d).



(a) Imagen Original (b) Ground Truth (c) Salida FCN estándar (d) Segmentación PSPNet

Figura 6.1: Segmentación errónea debido a la similitud entre clases

La arquitectura de la red aparece representada en la figura 6.2. Está compuesta por dos módulos principales (similares a un modelo encoder-decoder tradicional de otras arquitecturas):

- Un primer bloque extractor de características consistente en una Resnet-101 con convoluciones dilatadas [54] [55] para extraer los mapas que servirán como entrada al segundo bloque. El modelo de red utilizado en este bloque es una arquitectura muy pesada con una red muy profunda que proporciona mapas de características de muy buena calidad. Sin embargo, este primer bloque puede ser sustituido por otras arquitecturas alternativas como la propuesta en el capítulo 5 utilizando el encoder de ERFNet.

- Un módulo jerárquico que analiza las distintas sub-regiones de la imagen a diferentes escalas mediante diferentes capas de *pooling*, agrupando las características de los mapas de entrada bajo diferentes escalas de pirámide. El nivel más grueso es el primero, que en la figura aparece representado en rojo y que analiza los mapas de manera íntegra. Los siguientes niveles de módulo (amarillo, azul y verde) dividen los mapas de características en diferentes subregiones y formas y agrupan la representación para ubicaciones distintas. La salida de los diferentes niveles del módulo contienen mapas con distintos tamaños que deben ser agrupados. Para mantener el tamaño del mapa de características se aplica una capa convolucional de 1x1 tras los niveles piramidales para que el tamaño final de la representación de los mapas obtenida sea de $\frac{1}{N}$, siendo N el nivel que define la pirámide empleada para analizar el mapa. Los mapas son después re-dimensionados a la escala perseguida utilizando interpolación bilineal y concatenados para dar lugar a un mapa conjunto. Tras la obtención de este mapa se ubica una última capa convolucional que da lugar a la imagen clasificada a nivel de píxel.

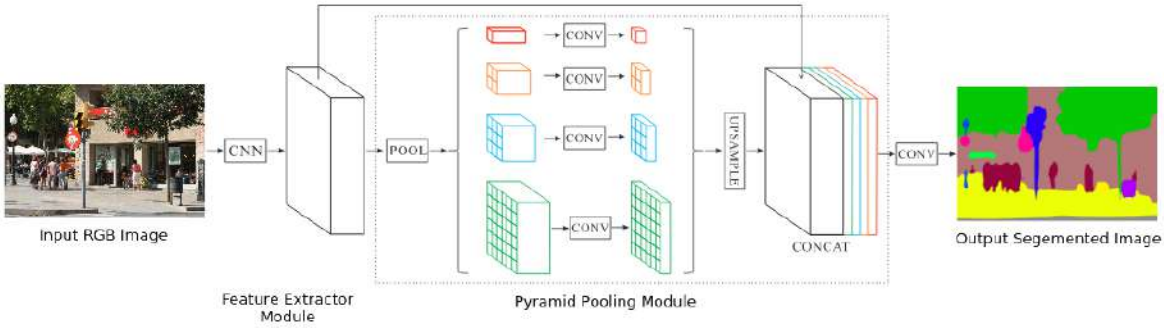
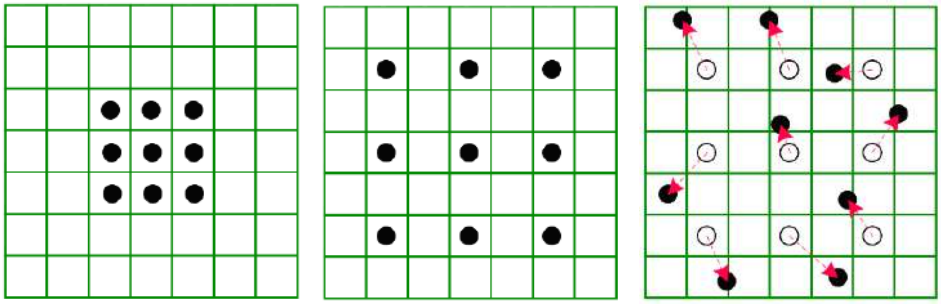


Figura 6.2: Esquema del módulo de análisis piramidal de PSPNet

Para emplear esta red se ha utilizado una re-implementación en Pytorch de la red original que utiliza como bloque inicial una ResNet 101 con convoluciones deformables 2D [56]. Las convoluciones dilatables introducen un nuevo parámetro que altera el espacio entre los elementos del kernel de las neuronas modificando su campo receptivo mediante una dilatación que siempre mantenía el *aspect-ratio*

del kernel. En contraposición a las dilatables de la arquitectura inicial, las deformables incorporan 2 parámetros que permiten modificar el espacio de los elementos del kernel en 2 dimensiones, pero de manera independiente, modificando el *aspect-ratio* del campo perceptivo, además de otros dos parámetros adicionales que permiten desplazar el kernel de la convolución también en dos dimensiones. La figura 6.3 muestra los kérneles para una convolución de cada uno de los tipos anteriores.



(a) Kernel de una convolución estándar

(b) Kernel de una convolución dilatada

(c) Kernel de una convolución deformable

Figura 6.3: Diferentes tipos de convolución

El segundo bloque de la re-implementación se corresponde con el módulo piramidal original de 4 niveles que también fue empleado en la arquitectura ERFNetPSP.

Para el entrenamiento no se dispone de un modelo pre-entrenado de la red en Imagenet, por lo que el entrenamiento parte de cero. La optimización del método de propagación del gradiente para esta arquitectura es el del gradiente estocástico descendiente (SGD) [57] pero con parámetros diferentes a los del entrenamiento original de la red (obtenidos de forma empírica). Como learning-rate se utiliza 5e-3, con momentum de 0.99 y decaimiento de los pesos de 2e-4. Debido al gran tamaño de la red se reduce el batch utilizado para la carga de imágenes hasta 2 con objeto de que quepa en la memoria de la GPU utilizada (Titan-X).

6.3 DRNs

Las redes dilatables con capas residuales (Dilated Residual Networks o DRNs) [58] tienen como enfoque principal preservar la resolución espacial de las imágenes a clasificar. Se trata de una línea [59] que va en contraposición a múltiples trabajos del estado del arte que emplean procesos de *downsampling* para generar mapas de características representativos y, a la vez, reducir la carga computacional de la red. Sin embargo, este nuevo enfoque defiende que reducir las dimensiones de las imágenes de entrada daña significativamente la información extraíble de los mapas de características generados, lo cual resulta especialmente crítico en tareas de etiquetado a nivel de píxel como la segmentación semántica. La línea seguida es similar a la de la PSPNet: se busca primar la información contextual y la relación de los objetos de las imágenes con su entorno y localización en la escena. Para ello se recurre a las convoluciones dilatables que respetan mejor las dimensiones de las imágenes de entrada y a las capas residuales que permiten conectar capas con mayores volúmenes de información con capas más profundas. Con ello se persigue identificar mejor objetos que no son espacialmente dominantes o que son de pequeño tamaño, por lo que tienden a ser más difíciles de identificar. Como arquitectura base se utiliza el modelo de ResNet original mostrado en la figura 6.4.

La arquitectura presentada por este conjunto de redes se inspira en [36]. Partiendo de la idea básica de que únicamente eliminar los procesos de *downsampling* tendría consecuencias negativas (dado que se

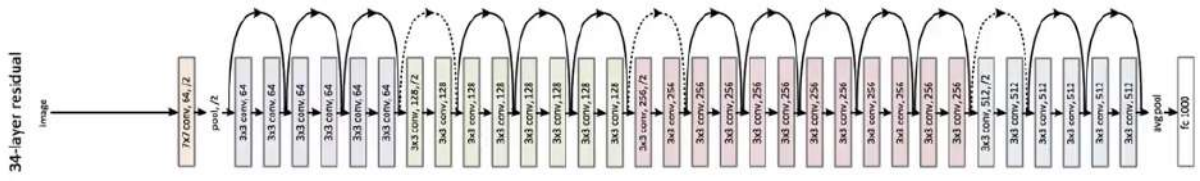
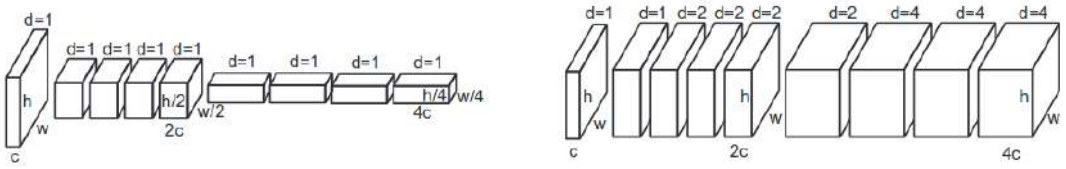


Figura 6.4: Esquema básico de la arquitectura de ResNet

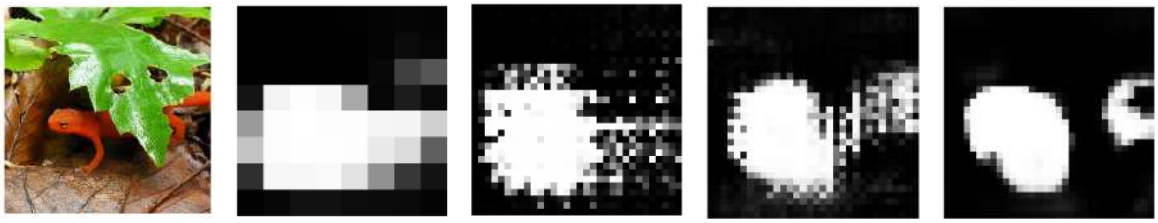
reduciría el campo perceptivo de las capas posteriores) se propone la idea complementaria de incorporar convoluciones dilatables con mayores campos receptivos. Tomando como punto de partida el modelo básico de una ResNet se elimina el stride introducido en las últimas capas convolucionales que realizaba el análisis de las columnas y filas bien pares o bien impares de la imagen reduciendo la escala de los mapas de características. El proceso anterior penaliza el campo receptivo de las capas posteriores, lo que degrada el aprovechamiento de la información contextual ya que las diferentes zonas de la imagen dejan de ser relacionadas entre sí. Para afrontar este problema se incorporan convoluciones dilatables con factor de dilatación d proporcional a la pérdida de campo receptivo producido al eliminar el stride de las convoluciones. Un ejemplo de los mapas de características obtenidos para la arquitectura original de ResNet y de una DRN se puede observar en la figura 6.5.



(a) Mapa de características de ResNet (b) Mapa de características de DRN

Figura 6.5: Comparativa capas DRN y Resnet

Además del uso de convoluciones dilatadas, las DRNs mantienen el planteamiento original de ResNet con respecto al uso de capas residuales que ayudan a propagar la información a lo largo de múltiples capas de manera correcta en redes profundas. El resultado de la combinación de ambas técnicas son mapas de activación claramente más precisos para tareas de clasificación de imágenes y segmentaciones semánticas más precisas que muchos modelos previos del estado del arte (como por ejemplo el propio modelo base de ResNet). La figura 6.6 muestra una comparativa entre los mapas de activación generados por diferentes redes para una misma imagen de entrada. Se incluyen los mapas para una ResNet en la figura 6.6b y para 3 DRNs producto de utilizar diferentes versiones de ResNet en las figuras 6.6c,6.6d y 6.6e. La figura 6.6c se corresponde con una Resnet 18, la figura 6.6d con una Resnet 26 y la figura 6.6e con el mismo modelo pero incorporando convoluciones dilatadas.



(a) Entrada (b) ResNet-18 (c) DRN-A-18 (d) DRN-B-26 (e) DRN-C-26

Figura 6.6: Mapas de activación para diferentes redes

En la figura anterior los mapas de características presentados persiguen identificar el animal de la

figura. En ellos se muestra claramente como las DRNs proporcionan mapas de activación mucho más precisos que la ResNet original que da una salida de píxel gruesa. Este comportamiento de los mapas de características ya había sido estudiado previamente en trabajos como [60], aunque al implementación de las DRNs fue al primer trabajo que, de manera específica, buscaba aprovechar este tipo de mapas más precisos.

Para la evaluación de este tipo de redes se aprovecha un repositorio de Pytorch proporcionado por los propios autores del trabajo que contiene modelos para 7 DRNs generadas a partir de varias arquitecturas de ResNet. De entre ellos se utilizará la arquitectura DRN-D-54 (generado a partir de ResNet-50) que presenta resultados de muy alta calidad a cambio de un coste computacional moderado. Para el entrenamiento se utiliza un tamaño de batch de 4 con constante de aprendizaje de 5e-4, momentum de 0.90 y decaimiento de los pesos de 1e-4.

6.4 SegNet

SegNet [61] es una arquitectura de red diseñada específicamente para abordar la tarea de segmentación de imágenes a nivel de píxel de manera eficiente.

La red presenta una arquitectura de encoder-decoder junto con una capa de clasificación a nivel de píxel. La figura 6.7 ilustra la estructura básica de la red. El bloque de codificación consta de 13 capas convolucionales que se corresponden con las primeras capas de una red VGG16 [62] sin modificar. Utilizar un bloque sin modificaciones extraído de otra red permite aprovechar pesos pre-entrenados para esa misma red en otros datasets (Imagenet). El modelo descarta el uso de capas convolucionales completamente conectadas con el objetivo de dar lugar a mapas de características con mayor resolución, con lo que además se reduce el número de parámetros utilizados en el encoder que desciende de 134M a 14.7M siendo notablemente inferior al de múltiples arquitecturas de uso muy extendido como DeepLab. El bloque de decodificación está compuesto por 13 capas deconvolucionales análogas a las del encoder. Finalmente la salida de la red se obtiene mediante una capa de clasificación multiclase soft-max que proporciona probabilidades de clase a nivel de píxel.

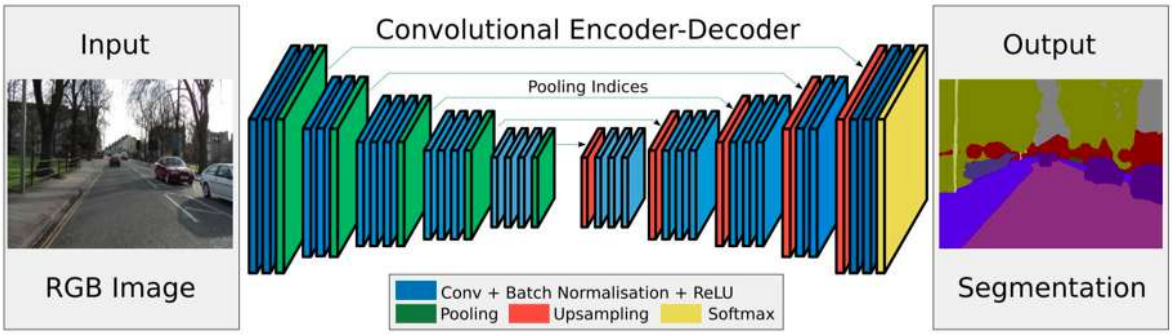


Figura 6.7: Esquema básico de la arquitectura de Segnet.

El bloque encoder genera mapas de características utilizando capas convolucionales que incorporan etapas de *batch-normalization* [63] [64] seguidas de la aplicación de otra etapa sucesiva de ReLU ($max(o, x)$) a nivel de píxel. Como se muestra en la figura 6.7, el bloque de codificación también incorpora 5 capas de *max-pooling* con ventana de 2x2 y paso de 2 que tienen por objetivo conseguir que la red aprenda a identificar pequeños desplazamientos en los objetos de la imagen. El proceso de *sub-sampling*, como se ha expuesto anteriormente, consigue ampliar los campos receptivos de las capas posteriores mejorando la

información contextual que son capaces de extraer. Sin embargo, la pérdida de información producida en el proceso no es beneficiosa para la segmentación por lo que es necesario capturar y almacenar la información perdida de los mapas de características antes del sub-sampling. Debido al consumo de memoria requerido para llevar a cabo dicho almacenamiento, la red solo guarda los índices de *max-pooling* del encoder (las posiciones de las características con máximo valor en los mapas) para cada mapa. Con esta técnica basta con 2 bits de memoria para cada ventana de 2x2 aplicada con lo que el consumo final se reduce significativamente.

El bloque decoder contiene capas que se encargan de re-dimensionar los mapas de características del encoder utilizando los índices almacenados en la etapa anterior. Estas capas dan lugar a mapas de características dispersos que necesitan etapas posteriores para completarlos. Dichas etapas son implementadas mediante bancos de filtros convolucionales entrenables que tienen como salida mapas de características densos. Estas capas convolucionales también incluyen etapas de batch-normalization. La última capa se corresponde con el clasificador a nivel de píxel soft-max que devuelve la imagen segmentada.

Para el entrenamiento de esta red se recurre a una re-implementación en Pytorch de características idénticas al modelo original implementado en el framework Caffe. Como parámetros de entrenamiento se toman también los del modelo original: tasa de aprendizaje de 0.1 y momentum de 0.9 con ajuste estocástico del método del gradiente.

6.5 Re-implementaciones ERFNet

Para complementar las redes entrenadas, se utilizarán los datos publicados en [23], donde se presentan varias re-implementaciones de la estructura original de ERFNet con modificaciones que persiguen mejorar los resultados de la arquitectura original. Todos los modelos planteados presentan dos diferencias fundamentales en la arquitectura con respecto a la red original: en primer lugar se incluyen capas de batch-normalization después de todas las capas convolucionales y, en segundo lugar, las capas deconvolucionales pasan a utilizar ventanas de 2x2 con stride de 2.

Se plantean tres modelos distintos, con arquitecturas similares pero utilizando diferentes tipos de convoluciones: ERFNet re-implementada, RDCNet y FRDCNet.

- **ERFNet re-implementada**: se trata de una variante de la red ERFNet implementada en el lenguaje MXNet [65] en la que las capas convolucionales adoptan las dos modificaciones expuestas.
- **RDCNet**: se trata de un modelo de ERFNet re-implementada en MXNet con las dos modificaciones básicas y con convoluciones deformables como la de la figura 6.3c incorporadas en ciertas capas del modelo. La diferencia introducida sobre el modelo estándar de capa convolucional deformable es que el elemento central del kernel de la convolución permanece fijo y no se le otorga libertad para desplazarse, como se muestra en la figura 6.8. Al nuevo tipo de convolución se le denomina convolución deformable restringida. Además, incorpora un estudio sobre la influencia en el rendimiento de la red del número de capas con convoluciones deformables utilizadas.
- **FRDCNet**: el modelo RDCNet supone romper con uno de los pilares de la construcción de ERFNet, al dejar de utilizarse capas factorizables de una dimensión para emplear kérneles de 2 dimensiones. Por ello, el mismo trabajo propone una capa alternativa en la que el nuevo tipo de convolución se lleva a cabo mediante la agrupación de filtros de una dimensión factorizables con objetivo de mantener la eficiencia computacional de la red. La figura 6.9 muestra un ejemplo de construcción del encoder de FRDCNet para 2 capas convolucionales deformables restringidas.

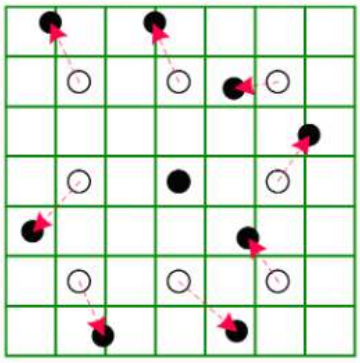
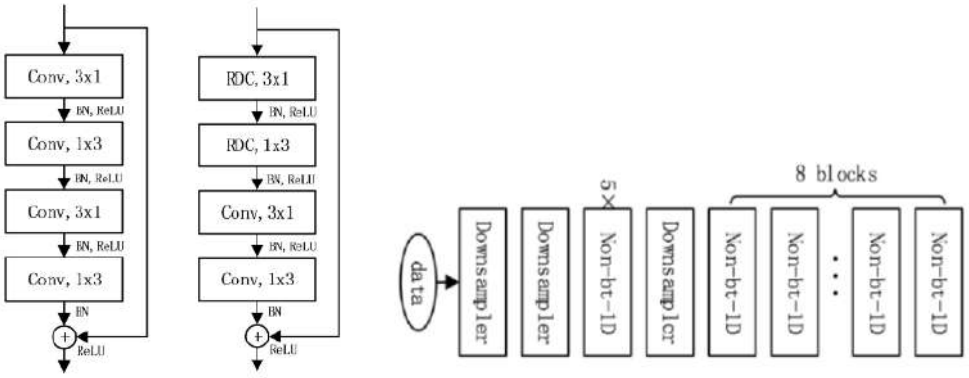


Figura 6.8: Ejemplo de convolución deformable restringida.



(a) Bloque Non-bt-1D de ERFNet

(b) Bloque Non-bt-1D FRDCNet

(c) Encoder de FRDCNet para dos capas deformables

Figura 6.9: Ejemplo de incorporación de capas convolucionales deformables restringidas.

El siguiente apartado de este capítulo realiza una comparativa entre todas las redes presentadas. Los datos de las dos últimas redes son los publicados por el propio trabajo, dado que los autores no proporcionan acceso ni al software ni a las arquitecturas utilizadas en él [23].

6.6 Entrenamiento con Clase Adicional

Los bordes presentes en las imágenes generadas para los datasets sintéticos (figuras 5.13a y 5.14a) también se dan en las imágenes obtenidas mediante una cámara de ojo de pez real. Como se puede observar en las figuras 5.13 y 5.14 estas zonas son clasificadas de manera aleatoria por las redes entrenadas hasta el momento, debido a que ninguna de las clases utilizadas en el entrenamiento engloba dichas regiones.

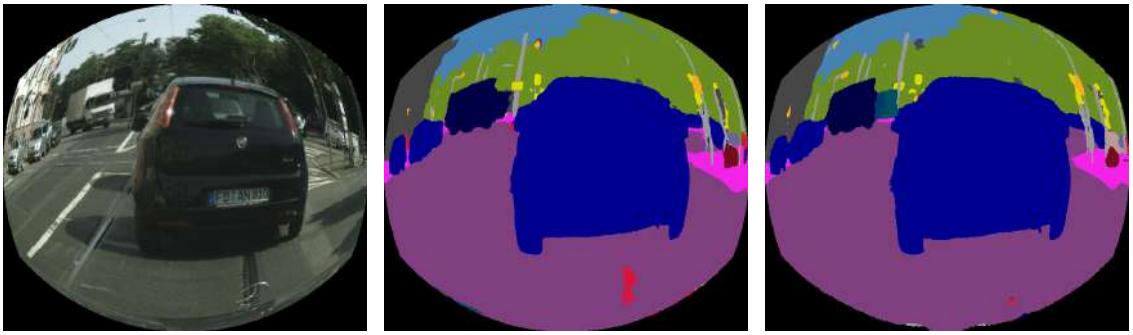
A nivel de resultados cuantitativos la presencia de estas regiones no tiene, a priori, ninguna importancia dado que a la hora de evaluar el IoU obtenido los píxeles asociados a esas zonas son ignorados. Sin embargo, a nivel cualitativo se puede observar como estas zonas tienen segmentaciones muy heterogéneas e irregulares, como en las figuras 5.13g y 5.14g, que debido al aprendizaje de información contextual producido durante el proceso de entrenamiento podrían influir negativamente sobre la clasificación de las zonas adyacentes a dichos bordes. Sin este aprendizaje de información relacional, la aplicación de una única máscara o región de interés a la salida de la red sería suficiente para evitar la extracción de conclusiones erróneas, pero dado que el proceso es inherente al aprendizaje de la red es necesaria la búsqueda de una alternativa.

Una posible solución es la inclusión de una clase adicional en las imágenes durante el entrenamiento de las redes que se corresponda con dichas regiones. El objetivo es que la red aprenda a identificar correctamente dichas zonas, con lo que la información contextual de los bordes no se degradaría e incluso podría verse beneficiada. Esta técnica fue implementada y testeada sobre ERFNet aprovechando el dataset generado en el experimento anterior utilizando los mismos parámetros de entrenamiento y todo el *data-augmentation* presentado hasta el momento menos el *zoom-augmentation*, dado que modifica la forma de estas regiones que en una cámara real sería constante. El *data augmentation* utilizado incluye, por tanto, *random-cropping*, *color jittering*, rotaciones y desplazamientos aleatorios, modificaciones del *aspect-ratio* de las imágenes, *mirroring* y cambios de escala aleatorios.

En la tabla 6.2 de la sección posterior se proporcionan datos cuantitativos que muestran el ligero empeoramiento en el rendimiento de la red al emplear esta clase adicional. Para un análisis de 20 clases el IoU obtenido es ligeramente superior (62,2 %), pero si se analizan los píxeles de las clases relevantes se obtiene un resultado ligeramente inferior al del modelo entrenado con 19 clases (60,3 %).

Como se puede observar, los resultados de la red son notablemente inferiores debido a la degradación de los valores obtenidos para clases como *Tren* o *Camión* y a las confusiones producto de la aparición de esta nueva clase, dado que un problema de clasificación en 20 clases es más complejo que uno de 19. Sin embargo, existe mejoría en ciertas clases (notable en algunos casos como el de la clase *Poste*) y, a nivel cualitativo, se pueden observar ciertos beneficios por el uso de esta técnica.

La figura 6.10 muestra una comparativa en los resultados obtenidos al segmentar mediante una red entrenada para 19 clases y otra para 20. La primera imagen (figura 6.10a) es la de entrada, la segunda figura (6.10b) expone los resultados de la segmentación para un entrenamiento de 19 clases seguida de la aplicación de una región de interés y la tercera imagen (figura 6.10c) muestra los resultados para un entrenamiento de 20 clases. En las dos segmentaciones pueden percibirse cambios en las zonas cercanas a los bordes.



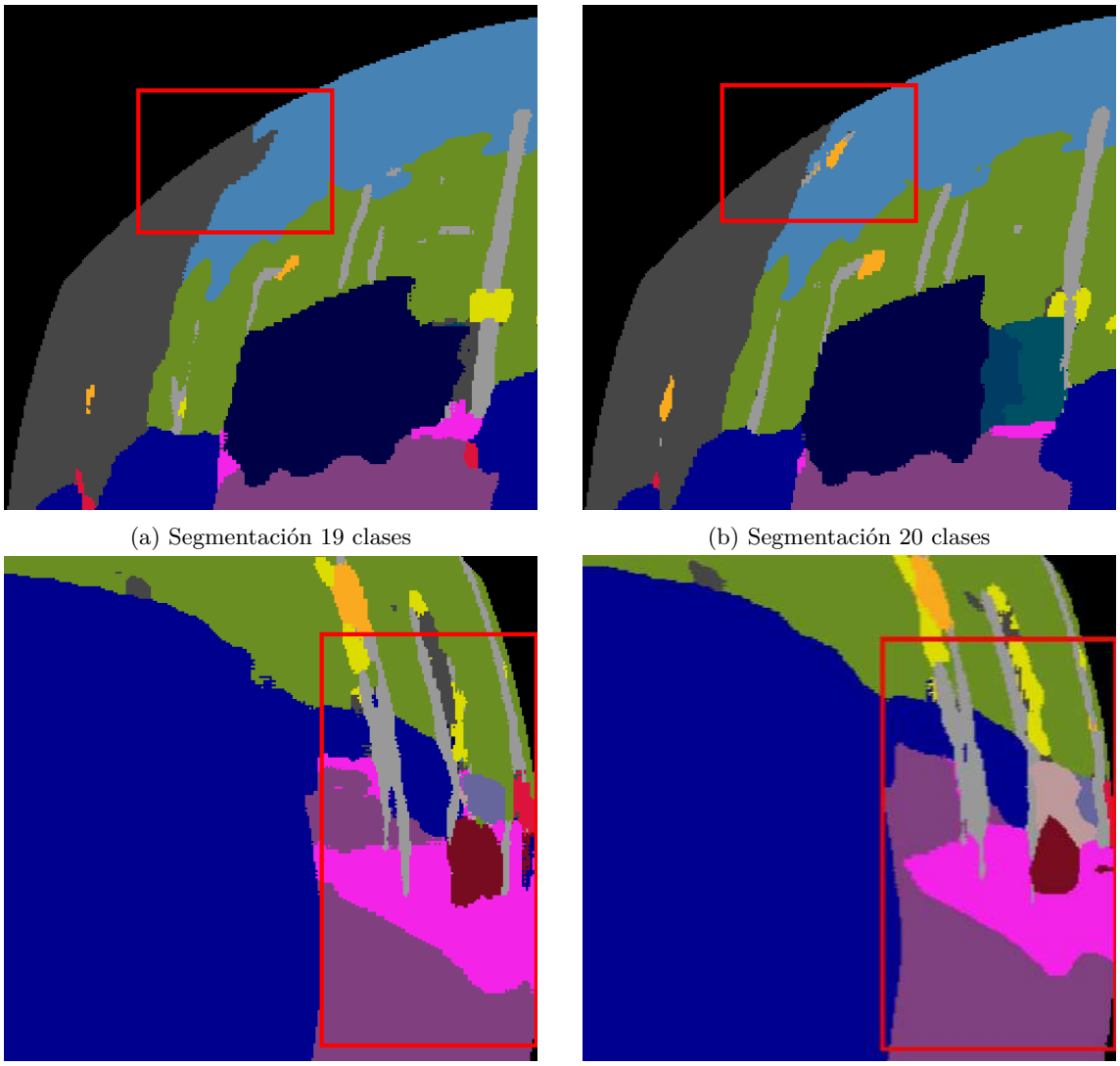
(a) Imagen de Entrada (b) Segmentación con 19 clases (c) Segmentación con 20 clases

Figura 6.10: Comparativa entrenamiento con clase adicional

A pesar del empeoramiento expuesto en la tabla 6.2 en ciertas imágenes, como la mostrada anteriormente, se percibe una mejora a nivel cualitativo al segmentar objetos de clases menores muy cercanos a los bordes. La imagen 6.11 muestra varios detalles de las imágenes anteriores en los que la diferencia en los resultados se hace perceptible:

En el primer detalle de la figura anterior (figuras 6.11a y 6.11b) se puede ver cómo el primer entrenamiento con 19 clases no es capaz de identificar el semáforo adyacente al borde de la imagen y lo asigna a la clase limítrofe que se corresponde con el edificio que, además, era la clase asociada al borde previamente a la aplicación de la región de la ROI.

En el segundo detalle (figuras 6.11c y 6.11d) se puede observar como las formas de los elementos



(a) Segmentación 19 clases

(b) Segmentación 20 clases

(c) Segmentación 19 clases

(d) Segmentación 20 clases

Figura 6.11: Detalles de la diferencia entre las dos segmentaciones

de la zona izquierda de la imagen están claramente mejoradas. En dicho borde, la persona y el poste que aparecen son identificados correctamente cuando en el entrenamiento con 19 clases los objetos eran mezclados con otras clases como coche o bicicleta. Además, la forma de todos los elementos de la región está claramente mejorada como en el caso de la acera o del resto de postes del área.

En cuanto a la detección de la nueva clase, en ambas imágenes se observa cómo la red la identifica de forma casi perfecta dado que su aspecto y forma no presentan variaciones en las imágenes. Sin embargo, la forma no es totalmente perfecta y es mejor el uso de una ROI, como demuestran los resultados cuantitativos expuestos anteriormente.

6.7 Resultados Cuantitativos

Para testear las redes entrenadas, se utiliza el dataset empleado en [23] con el objetivo de llevar a cabo una comparativa justa. En el trabajo se recurre a un dataset sintético de imágenes de ojo de pez de 640x576 definido por el parámetro de distorsión $f = 240$. Este valor da lugar a imágenes de mejor resolución y con menor distorsión que en los experimentos anteriores, con lo que los resultados de las redes serán,

previsiblemente, mejores.

La tabla 6.1 aporta datos sobre la carga computacional y sobre la eficiencia de las redes estudiadas. En ella se recogen datos de tiempo de proceso (para imágenes de 640x576 sobre una única Titan X), de espacio ocupado en memoria por el modelo y sobre el IoU de clase obtenido sobre el dataset caracterizado por $f = 240$. A continuación se detallan las conclusiones más relevantes proporcionadas por la tabla:

- El primer resultado destacable de la tabla es que el modelo de ERFNet con decoder piramidal es el que mejor IoU obtiene, incluso superando ligeramente los del modelo ERFNet base (0,12 %) al contrario que sucedía en apartados anteriores. Este hecho tiene como causa la mayor resolución de las imágenes de entrada que conserva mejor la información contextual al respetar más los objetos pequeños y los bordes y transiciones entre elementos presentes en la escena, lo que da ventaja al modelo piramidal como se preveía en las conclusiones del entrenamiento comparativo entre ambos modelos anterior. Ambas redes obtienen resultados parejos para la mayoría de clases, pero el modelo piramidal destaca claramente en algunas como *Camión* o *Valla* con menor representación en el dataset. En la comparativa del siguiente capítulo se muestra cómo la diferencia entre los resultados de ambas arquitecturas es aún mayor al aumentar más la resolución de las imágenes de entrada. El entrenamiento con 20 clases obtiene un IoU mayor debido a los buenos resultados de la nueva clase asociada a los bordes, pero el IoU obtenido analizando únicamente las 19 clases originales es ligeramente menor.

- Los modelos re-implementados en Pytorch obtienen resultados inferiores a lo que cabía esperar. En concreto, los resultados de PSPNet y de DRNet son especialmente bajos en comparación con la performance de esas mismas redes publicada para otros datasets. Los motivos principales de esta bajada de rendimiento son la no disponibilidad de una red pre-entrenada que, por ejemplo, en el caso del modelo piramidal supone una mejora del IoU de más del 2 %, el uso de un entrenamiento adaptado a ERFNet (salvo los parámetros utilizados, que se modifican de acuerdo a los expuestos en las publicaciones de cada red), el uso de un modelo re-implementado que no es óptimo debido a las pequeñas diferencias en las arquitecturas y dado que el cambio de lenguaje de programación incluye pequeñas variaciones y debido a que el *data-augmentation* utilizado no está diseñado específicamente para cada uno de los modelos. En el caso de PSPNet, además, el entrenamiento en CityScapes incluía el uso adicional de 20.000 imágenes anotadas a nivel de píxel grueso y de las 500 pertenecientes al conjunto de validación.

- Los arquitecturas más rápidas estudiadas son las derivadas de ERFNet, siendo la re-implementación de ERFNet en MXNet la más rápida de todas las expuestas (0,016 s) al aumentar el paso de las capas convolucionales de 1 a 2. FRDCNet a pesar en incluir capas convolucionales factorizables (que presumiblemente son más eficientes) obtiene resultados ligeramente inferiores (56,10 %) aunque mejores que el modelo base ERFNet (55,10 %). Segnet obtiene un tiempo de proceso rápido (0,07 s) aunque mayor que el publicado en el paper original para imágenes de mayor escala (0,06 s en CityScapes original). Por último, los modelos más pesados son PSPNet y DRNet que superan las dos décimas y la décima de segundo para procesar una imagen respectivamente.

- El espacio ocupado en memoria está claramente relacionado con el tiempo de proceso. El modelo más ligero es el de ERFNet original seguido por el que incluye el decoder piramidal. Del resto de modelos derivados de ERFNet no se publican datos de volumen de memoria ocupado. PSPNet ocupa casi 300MB, dado que el primer bloque de la red es una ResNet 101 que consta de un número muy elevado de capas. La red de capas residuales analizada supera los 140MB al partir también de una ResNet. El caso de SegNet es especialmente llamativo, dado que a pesar del reducido tiempo

Tabla 6.1: Comparativa eficiencia-performance de las diferentes redes.

Arquitectura	**IoU (%)**	**Tiempo de Proceso (s)**	**Espacio Memoria (MB)**
ERFNet	61,53	0,020	**8,3**
ERFNet 20 clases	62,17-(60,31)	0,020	**8,3**
ERFNetPSP	**61,65**	0,022	10,0
PSPNet	59,18	0,220	287,2
RDCNet	57,90	0,018	-
DRNet	57,56	0,150	141,5
FRDCNet	56,10	-	-
ERFNet (MX)	55,10	**0,016**	-
SegNet	50,12	0,070	117,9

de proceso que tiene (está en torno a 14 fps) el modelo ocupa 117 MB en memoria. Los datos del tamaño de las arquitecturas derivadas de ERFNet no se muestran dado que no están disponibles en las publicaciones asociadas a las mismas.

La segunda tabla expuesta 6.2 muestra datos detallados sobre los resultados obtenidos por cada red a nivel de IoU clase.

Los resultados de la anterior tabla muestran como los modelos de ERFNet de 19 y 20 clases y ERFNetPSP se reparten la mayoría de mejores resultados de clase, a excepción de la clase *Tren* en la que destaca PSPNet. A pesar de que ERFNet obtenga resultados ligeramente mejores en la mayoría de clases, el modelo con decoder piramidal es el que mejor precisión final obtiene gracias a la elevada diferencia de rendimiento en ciertas clases puntuales. Por detrás de estos dos modelos se encuentran PSPNet y DRNet que a pesar de ser penalizados por un entrenamiento específico para ERFNet proporcionan buenos resultados de segmentación. PSPNet llega incluso a obtener la mejor precisión para la clase Tren. En último lugar se encuentra el modelo de SegNet que, como cabía esperar, da lugar a una segmentación de peor calidad.

6.8 Resultados Cualitativos

En las siguientes figuras se exponen ejemplos de resultados de segmentación semántica para las redes entrenadas en este capítulo. La primera figura presentada 6.12 introduce las cuatro imágenes para las que se mostrarán resultados acompañadas de sus respectivos Ground-Truth. La siguiente figura 6.13 muestra los resultados obtenidos por las 6 redes entrenadas en este capítulo: la fila (a) muestra los resultados para el modelo de ERFNet entrenado con 20 clases; la segunda fila (b) muestra los resultados para el modelo con el decoder piramidal (ERFNetPSP); la tercera fila (c) presenta los resultados para el modelo original de ERFNet entrenado con 19 clases; la cuarta fila (d) contiene los resultados de la re-implementación

Tabla 6.2: IoU de clase (%) para las redes estudiadas

Clase	ERFNet 20 Clases	ERFNetPSP	ERFNet	PSPNet	DRNet	SegNet
Carretera	**97.4**	97.3	**97.4**	97.2	97.0	96.8
Acera	70.2	70.1	**70.8**	68.7	67.7	65.1
Edificio	83.8	83.2	**83.9**	83.2	82.4	79.6
Muro	34.7	35.9	**37.2**	34.9	34.6	25.7
Valla	31.6	**31.8**	29.4	31.3	31.4	19.9
Poste	**40.7**	37.2	39.2	31.4	30.3	29.6
Semáforo	42.2	**42.3**	41.9	37.5	36.5	29.1
Señal tráfico	55.2	54.7	**55.3**	48.7	49.6	36.6
Vegetación	**87.1**	86.3	86.8	85.7	85.4	84.3
Terreno	52.4	52.3	**53.2**	47.7	47.3	42.9
Cielo	89.5	**90.4**	89.8	89.5	88.9	89.3
Persona	69.1	68.7	**69.7**	66.7	66.4	58.4
Ciclista	47.9	48.2	**49.4**	44.6	43.3	29.1
Coche	88.7	88.8	**89.1**	88.3	87.2	85.3
Camión	52.3	**64.2**	56.2	57.4	51.3	37.6
Autobús	69.4	74.2	**76.1**	67.7	65.5	48.9
Tren	30.5	41.6	42.3	**45.0**	35.4	17.6
Motocicleta	40.9	41.1	**41.4**	40.1	36.0	25.8
Bicicleta	60.1	**60.2**	59.9	59.1	57.4	49.8
Bordes	**99.6**	-	-	-	-	-
IoU (%)	62.2 (60.3)	**61.7**	61.5	59.2	57.6	50.1

de PSPNet con convoluciones deformables en Pytorch; la penúltima fila (e) presenta los resultados para DRNet-54 y la última fila contiene la segmentación devuelta por SegNet.

Como observaciones generales, los mejores resultados se obtienen para los modelos de ERFNet con decoder piramidal (ERFNetPSP) y el modelo ERFNet entrenado con 20 clases que son capaces de identificar clases muy poco representadas (como *Camión*) o que espacialmente no son dominantes (como *Señal de tráfico*). El modelo entrenado para PSPNet devuelve también resultados de buena calidad, pero menos precisos a la hora de detectar los contornos y formas de los objetos. Por último, los modelos de DRN y SegNet proporcionan segmentaciones de peor calidad en las que se realiza una detección pobre de las formas que, además, va acompañada de múltiples confusiones entre clases.

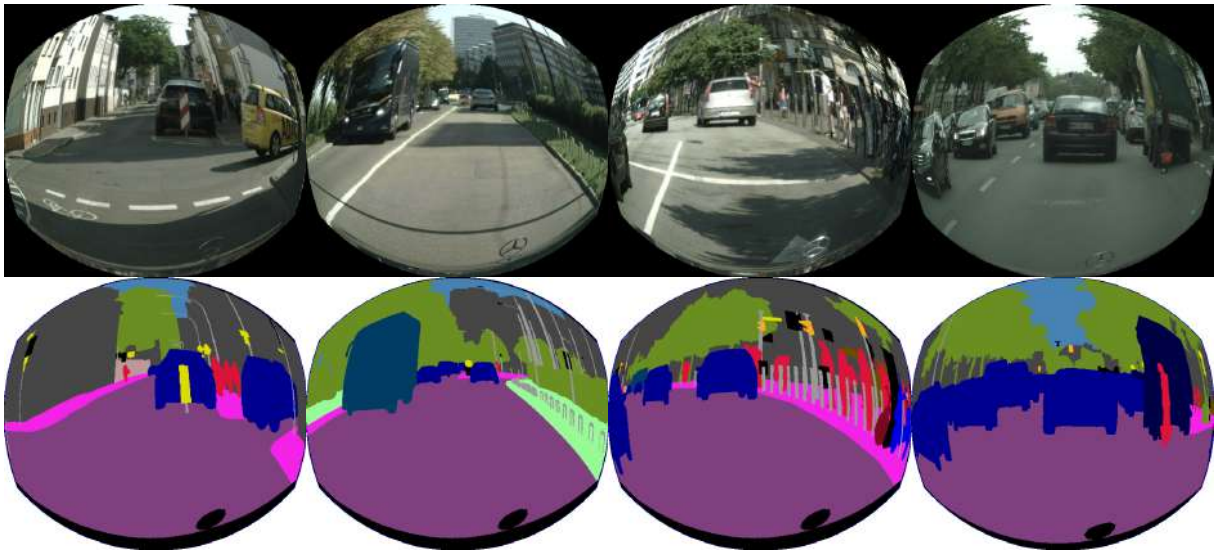


Figura 6.12: Imágenes de entrada y GT ejemplo

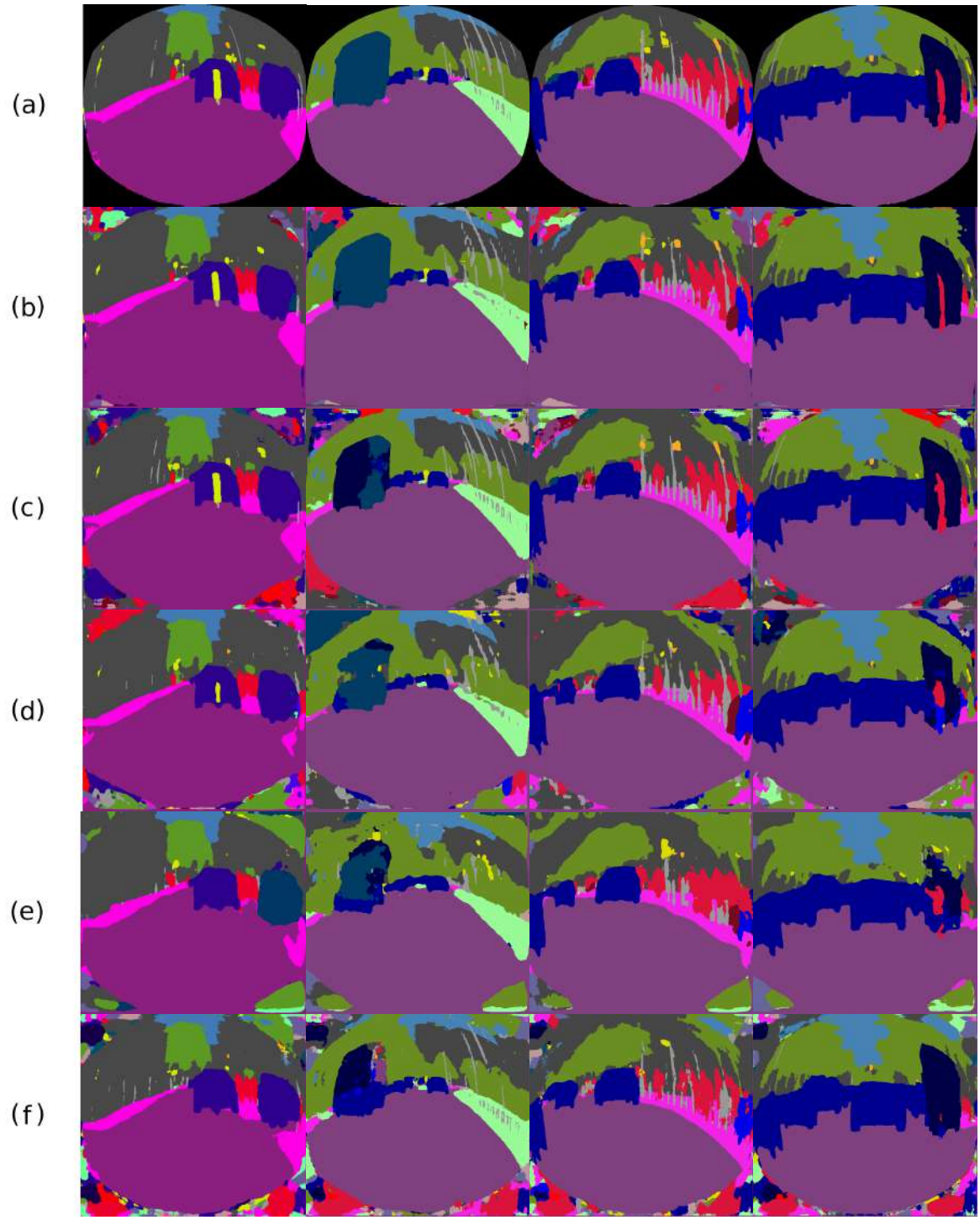


Figura 6.13: Ejemplos de segmentación semántica con las distintas redes entrenadas: (a) ERFNet 20 clases, (b) ERFNetPSP, (c) ERFNet, (d) PSPNet, (e) DRNet, (f) SegNet

Capítulo 7

Adaptación y Test en una Cámara Real

7.1 Introducción

En los capítulos anteriores se expusieron los resultados obtenidos sobre los datasets sintéticos generados y se realizó una comparativa con otros trabajos similares del estado del arte, así como con otras arquitecturas de red re-implementadas en Pytorch para este trabajo. Sin embargo, los resultados obtenidos no fueron testeados en imágenes adquiridas directamente por una cámara real por lo que su valor es mayoritariamente teórico. Para comprobar la validez de las arquitecturas y técnicas utilizadas se procedió a la compra de una cámara de ojo de pez.

Este capítulo aborda la adquisición y calibración de la cámara de ojo de pez real, la grabación de secuencias en entornos urbanos para el posterior test de la cámara, la adaptación de los entrenamientos y técnicas de *data-augmentation* utilizados en los experimentos anteriores a la nueva cámara y el test final sobre el conjunto de secuencias grabadas en el campus y en zonas cercanas de la ciudad.

7.2 Adquisición de la cámara

La adquisición de una cámara con óptica *fisheye* tuvo por objetivo su integración en el sistema de percepción del prototipo eléctrico de SmartElderlyCar, por lo que los principales criterios de selección fueron:

- **Tamaño de la cámara**: las dimensiones de la cámara debían ser reducidas con objetivo de permitir una integración sencilla dentro de la estructura del vehículo, de tal forma que la cámara pudiese ser instalada y retirada del coche con facilidad.
- **Fácil integración**: la cámara debía poder ser conectada al sistema de percepción del vehículo de manera sencilla, utilizando los recursos disponibles, lo que limitaba las posibilidades a un puerto serie USB dado que la conexión Firewire 80 disponible se encuentra ocupada por la cámara principal Bumblebee XB3 del vehículo.
- **Campo de visión**: este trabajo se encuentra centrado en el estudio de las cámaras con campo de visión ultra amplio, por lo que la cámara escogida debía presentar un campo de visión de al menos 180 grados.

- **Precio**: el precio final de la cámara debía ser razonable y ajustarse al presupuesto disponible para el proyecto.

Con estos requisitos se realizó una búsqueda de una cámara que se adaptara a estos puntos y se seleccionó una cámara USB de bajo coste y consumo fabricada por SONY con modelo ELP-USBFHD01M-BL180. La cámara es la presentada en la figura 7.1. A continuación se detallan las especificaciones más relevantes del dispositivo:

- **Definición**: 1080P Full-HD.
- **Resolución Máxima**: 1920 x 1080 (2.0 Megapíxel).
- **Campo de visión**: 180 grados con óptica de ojo de pez.
- **Conectividad**: USB con protocolos USB2.0 HS/FS y USB1.1 FS.
- **Marca**: SONY ELP.
- **Sensor**: 1/2.7ÇMOS OV2710.
- **Imágenes por segundo**: 30fps a 1080P, 60fps a 720P y 120fps a 480P.
- **Formato de compresión vídeo**: MJPEG.
- **Dimensiones(longitud x anchura x grosor)(mm)**: 41 x 41 x 21.
- **Fuente de Alimentación (V)**: 5V.
- **Consumo(w)**: 5V/150mA.
- **Compatibilidad**: Android y la mayoría de sistemas operativos.
- **Precio Final**: 40$.

Figura 7.1: Cámara adquirida para el proyecto.

La cámara adquirida es un dispositivo Plug-and-Play que no requiere de la instalación de software específico para su uso y que es compatible con Ubuntu (el sistema operativo del ordenador utilizado en el proyecto) por lo que su integración en el sistema de percepción del vehículo resultó inmediata.

7.3 Grabación de Secuencias

Con objetivo de testear el conjunto de técnicas desarrolladas en este trabajo se grabaron varias secuencias en el campus de la UAH y en zonas de la ciudad de Alcalá de Henares cercanas a él. La cámara fue montada en la parte delantera del vehículo en diferentes posiciones para cada una de las secuencias con objetivo de testear en qué posición se obtenía la mejor segmentación. La figura 7.2 muestra varios ejemplos del montaje de la cámara a bordo, llevado a cabo mediante el uso de un soporte para un teléfono móvil.

Figura 7.2: Ejemplos de posicionamiento de la cámara a bordo de vehículos reales

El recorrido escogido para la grabación de las secuencias incluyó tanto zonas representativas del campus como zonas urbanas con situaciones desafiantes para los vehículos autónomos tales como cruces, ceda el paso, rotondas, intersecciones o pasos de peatones. Las grabaciones se realizaron a 10 imágenes por segundo y a la máxima resolución permitida (1920 x 1080). La figura 7.3 muestra ejemplos de imágenes de las secuencias grabadas, tanto en el campus como en entornos urbanos.

Figura 7.3: Ejemplos de imágenes pertenecientes a secuencias urbanas y de campus

7.4 Calibración de la Cámara

Como paso previo al uso de la cámara se realizó una calibración de la misma empleando para ello el software para óptica de ojo de pez proporcionado por MATLAB a partir de la versión de 2017. Éste utiliza un modelo basado en la aproximación de Scaramuzza expuesta en el capítulo 2 modificado según el trabajo [66]. Para ello se grabó una secuencia de 32 imágenes utilizando un patrón similar a un tablero de ajedrez impreso sobre papel y adherido a una superficie rígida. La figura 7.4 muestra un ejemplo de varias imágenes empleadas en la calibración.

(a) Ejemplo de imagen utilizada para la calibración 1 (b) Ejemplo de imagen utilizada para la calibración 2

Figura 7.4: Ejemplo de imágenes empleadas en el proceso de calibración

El proceso de calibración en la toolbox se encuentra completamente automatizado, siendo solo necesaria la carga de las imágenes y las llamadas a las funciones proporcionadas por las librerías para llevarlo a cabo. El primer paso necesario es la llamada al fichero encargado de realizar la detección de las esquinas del tablero, para lo que es necesario introducir previamente el número de esquinas a detectar mediante el número de filas y columnas del damero (7x4). Para la detección de esquinas solo se utilizan las casillas interiores del patrón, ignorándose las filas y columnas de los bordes, dado que limitan con el tablero y pueden confundir al detector. Una vez seleccionadas las esquinas a detectar el proceso es automático. La imagen 7.5 muestra un ejemplo de detección de esquinas.

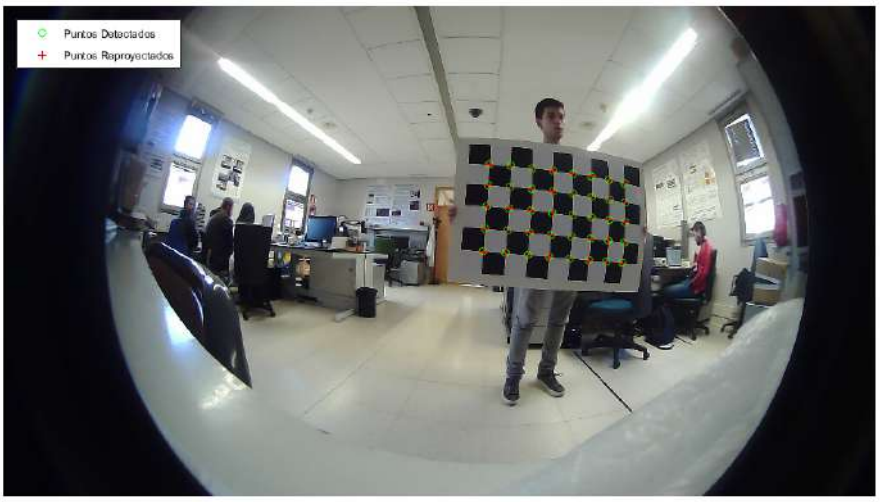


Figura 7.5: Ejemplo de detección de esquinas en el tablero

A continuación se deben introducir las dimensiones del tablero en unidades reales, que en nuestro caso

eran de 10x10 cm. A partir de este dato el programa es capaz de determinar una posición en el mundo real para cada una de las esquinas del tablero, al ser sus dimensiones conocidas, mediante una llamada a otra función. Conociendo estos puntos, el conjunto de parámetros perseguidos ya es determinable mediante el uso de ecuaciones geométricas.

El siguiente paso, por tanto, es la estimación de los parámetros intrínsecos de la cámara y los extrínsecos con respecto al tablero en cada una de las imágenes de la secuencia a partir de los puntos obtenidos con la función anterior. Para ello se realiza una última llamada a una función que devuelve ambos conjuntos de datos almacenados en una única estructura. Los parámetros de calibración extrínsecos para cada imagen de la secuencia (que determinan la posición del tablero en el espacio 3D) aparecen reflejados en la figura 7.6. En cuanto a los intrínsecos, aparecen adjuntados en el Apéndice A de este trabajo.

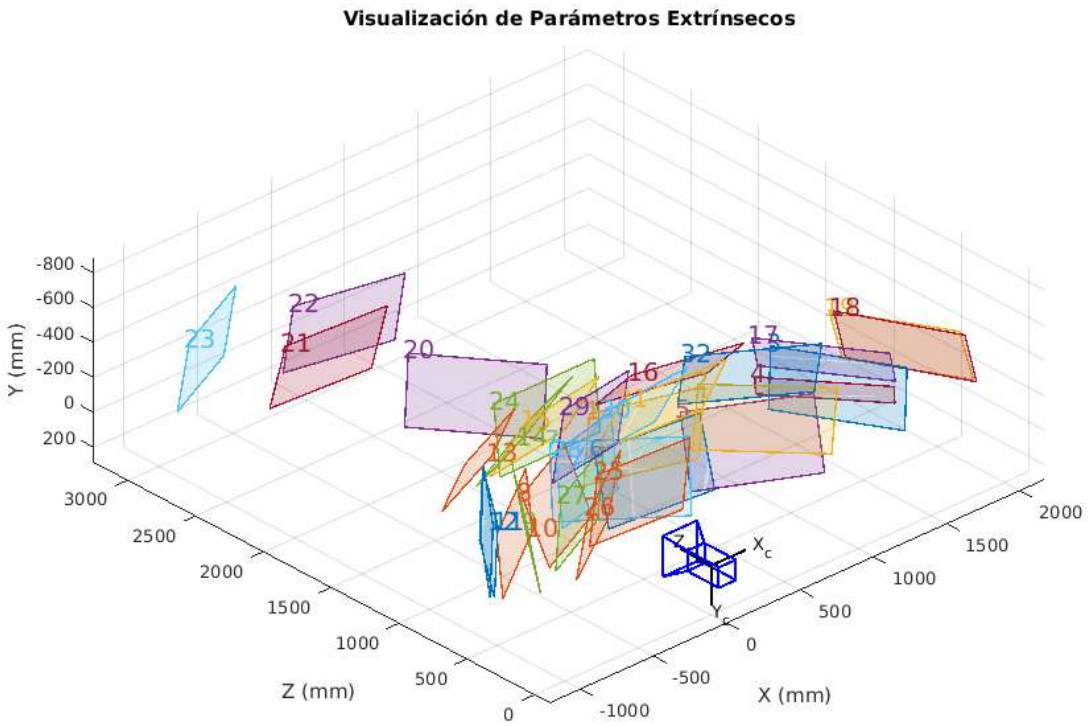


Figura 7.6: Parámetros extrínsecos estimados para la secuencia

El último paso para completar la calibración es el de realizar una estimación de lo precisa que ha sido mediante una re-proyección de los puntos 3D asociados a las esquinas del tablero aprovechando los parámetros calculados. El error cometido en cada una de las imágenes tras la calibración aparece reflejado en la gráfica presentada en la figura 7.7. Como se puede observar, el error cometido estimado tras la calibración es muy pequeño, de en torno a 0,43 píxeles. Este error es claramente mayor en las zonas en las que la distorsión es pequeña (zonas centrales), lo que se debe a que el modelo de cámara utilizado por Scaramuzza (se utiliza un modelo de proyección esférico) no encaja completamente con el que presenta la cámara real. Por este motivo, las re-proyecciones de píxeles para las zonas centrales de la imagen presentan un error mayor que en el resto de zonas.

Con el conjunto de parámetros determinados se pueden modificar las imágenes capturadas mediante la cámara de ojo de pez para obtener imágenes rectificadas y libres de distorsión. Un ejemplo se muestra en la figura 7.8.

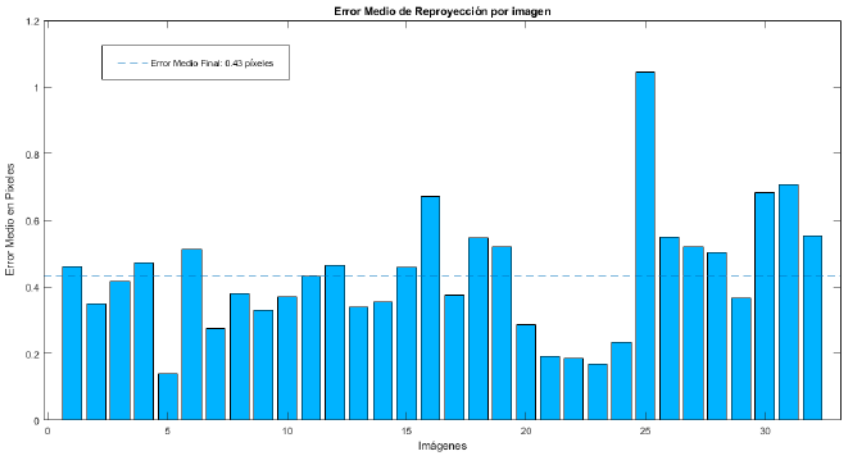


Figura 7.7: Error cometido en la re-proyección de píxeles tras la calibración

(a) Imagen original con distorsión (b) Imagen corregida sin distorsión

Figura 7.8: Ejemplo de eliminación de la distorsión en una imagen

La figura 7.8b muestra cómo los parámetros de calibración no permiten eliminar la distorsión de manera completamente correcta. En ella, las filas y columnas del tablero son rectas al haberse eliminado la distorsión que introduce la lente en la zona central de la imagen. Sin embargo, se puede observar una clara disminución de la calidad de la imagen que pierde mucha resolución en las zonas menos cercanas al centro en las que, además, no se recupera por completo la forma original de los objetos que tiende a alargarse y deformarse ligeramente debido a la diferencia entre el modelo proyectivo utilizado por Scaramuzza y el de la cámara real. En segundo lugar también es destacable la pérdida de mucha información de la imagen al no conservarse las zonas adyacentes a los bordes. Por otro lado, como desventaja adicional, el tiempo de procesado requerido es superior a 5 segundos (que podría ser reducible al estar implementada en MATLAB) lo que la hace completamente inviable para aplicaciones orientadas a funcionar en tiempo real. Estos hechos comprueban a nivel práctico las grandes desventajas presentadas a nivel teórico por la eliminación de la distorsión de las imágenes de ojo de pez como método de afrontar el problema tratado en este trabajo. La figura 7.9 muestra un ejemplo de cómo una imagen en un entorno urbano real se degrada al eliminar la distorsión que presenta.

En este punto se descartó una posible línea alternativa de trabajo que planteaba la generación de imágenes con distorsiones específicas adaptadas a la cámara adquirida. El principal obstáculo para seguir esta línea es la ausencia de toolboxes de calibración con modelos proyectivos adaptados a los diferentes tipos de óptica de ojo de pez existentes. Al utilizar la mayoría de toolboxes modelos basados en proyecciones esféricas, las distorsiones que son capaces de modelar no se adaptan a la cámara real adquirida. Además, hay que añadir los problemas aparecidos al deshacer la distorsión. La pérdida de

(a) Imagen en entorno urbano con distorsión (b) Imagen urbana corregida sin distorsión

Figura 7.9: Ejemplo de eliminación de la distorsión en una imagen de una de las secuencias grabadas

información de los bordes y la disminución de la resolución y escala dañan gravemente las imágenes obtenidas al deshacer la distorsión y, previsiblemente, esta degradación de la calidad de la imagen también se daría en el proceso inverso de añadir distorsión a la imagen.

La línea escogida para adaptar los entrenamientos previos a la nueva cámara es la de continuar con un modelo de proyección genérico pero con un entrenamiento más específico centrado en la cámara de ojo de pez.

7.5 Adaptación del entrenamiento

Las redes entrenadas en los capítulos previos del trabajo proporcionaron malos resultados sobre las secuencias grabadas debido a varios factores:

- **Escala de la imagen**: las imágenes capturadas tienen un tamaño de 1920x1080, mientras que las usadas en el entrenamiento eran de 640x576. Esto implica tanto una variación de la escala como del *aspect-ratio* entre ambos conjuntos de imágenes. A pesar de que las CNNs aprenden a identificar objetos con independencia de su tamaño y *aspect-ratio*, las redes presentan el mejor rendimiento para el tamaño de imagen para el que han sido entrenadas. El rendimiento en la tarea de segmentación se reduce al realizar cambios de escala tan grandes.

- **Resolución**: las imágenes de entrada en la etapa de entrenamiento tienen una resolución baja debido al proceso de mapeado durante su generación. Esta disminución de la resolución hace muy difíciles de identificar muchos objetos en las imágenes incluso para un humano, por lo que resulta crítica en la pérdida de rendimiento en el cambio de imágenes.

- **Modificación de dominio**: el conjunto de imágenes utilizadas en el entrenamiento se corresponden con ciudades alemanas en las que el aspecto de muchos objetos difiere de la apariencia de elementos análogos en las imágenes del campus, como por ejemplo, las aceras, los edificios o el cielo. Este cambio de dominio influye también negativamente sobre la segmentación final.

- **Modelo de distorsión**: las imágenes generadas para los datasets sintéticos aprovechaban un modelo de proyección genérico para las cámaras con óptica de ojo de pez equidistante (el modelo más extendido por los fabricantes). Sin embargo, el modelo de proyección y el de distorsión no se ajustan perfectamente al presentado por la cámara real, por lo que el rendimiento de la red se claramente penalizado.

- **Iluminación**: el campo de visión de 180 grados de la cámara tiene una gran desventaja: el volumen de luz capturada es muy grande lo que, en combinación con la forma curvada de la lente, da lugar a brillos no deseados en las imágenes capturadas 7.10 que confunden fuertemente a la red dando lugar a segmentaciones de muy mala calidad.

Figura 7.10: Ejemplo de reflejo en la imagen.

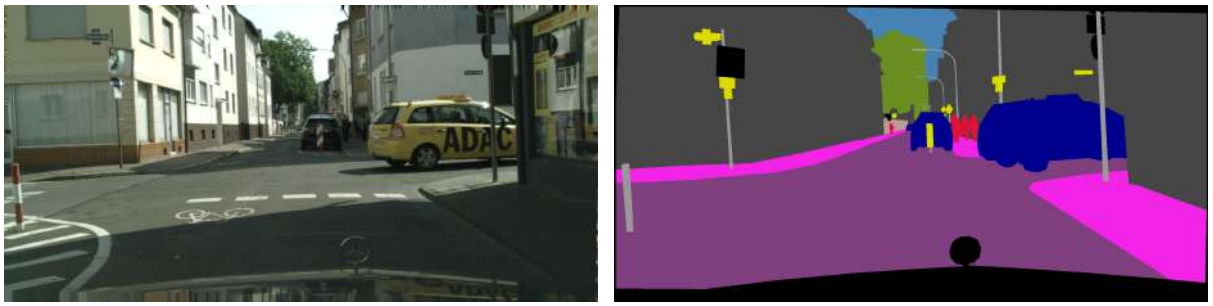
Con objetivo de mejorar los resultados obtenidos en la segmentación por las redes anteriores, se diseñó un nuevo entrenamiento basado en CityScapes y adaptado en la medida de lo posible a la cámara a utilizar y se aplicó sobre las dos arquitecturas utilizadas en el trabajo.

Las imágenes generadas para el nuevo entrenamiento tienen un tamaño de 1120x792, mucho mayor que en los anteriores casos, teniendo además un *aspect-ratio* de 1,414 muy similar al de las imágenes a segmentar sin incluir los bordes y que es de 1,422 (1536x1080). Para generar las imágenes, se utiliza un *zoom-augmentation* aleatorio empleando con niveles de distorsión mucho menores que en los anteriores casos (limitados entre $f_{inf} = 200$ y $f_{sup} = 800$) consiguiendo así imágenes de mayor escala sin procesos adicionales. Además, al ampliar el rango de distorsiones utilizadas en la generación de imágenes, se consigue mejorar la capacidad de generalización de la red, mejorando también el problema presentado al no disponer de un modelo de distorsión exacto de la cámara utilizada [23]. La figura 7.12 muestra algunos ejemplos de imágenes con niveles de distorsión incluidos en el rango utilizado para el entrenamiento.

Para afrontar los problemas presentados por el cambio de entorno en las imágenes se utiliza el *data-augmentation* de adaptación de dominio presentado en el capítulo 5, abogando por cambios más agresivos en el color de las imágenes para intentar afrontar también el problema de la iluminación.

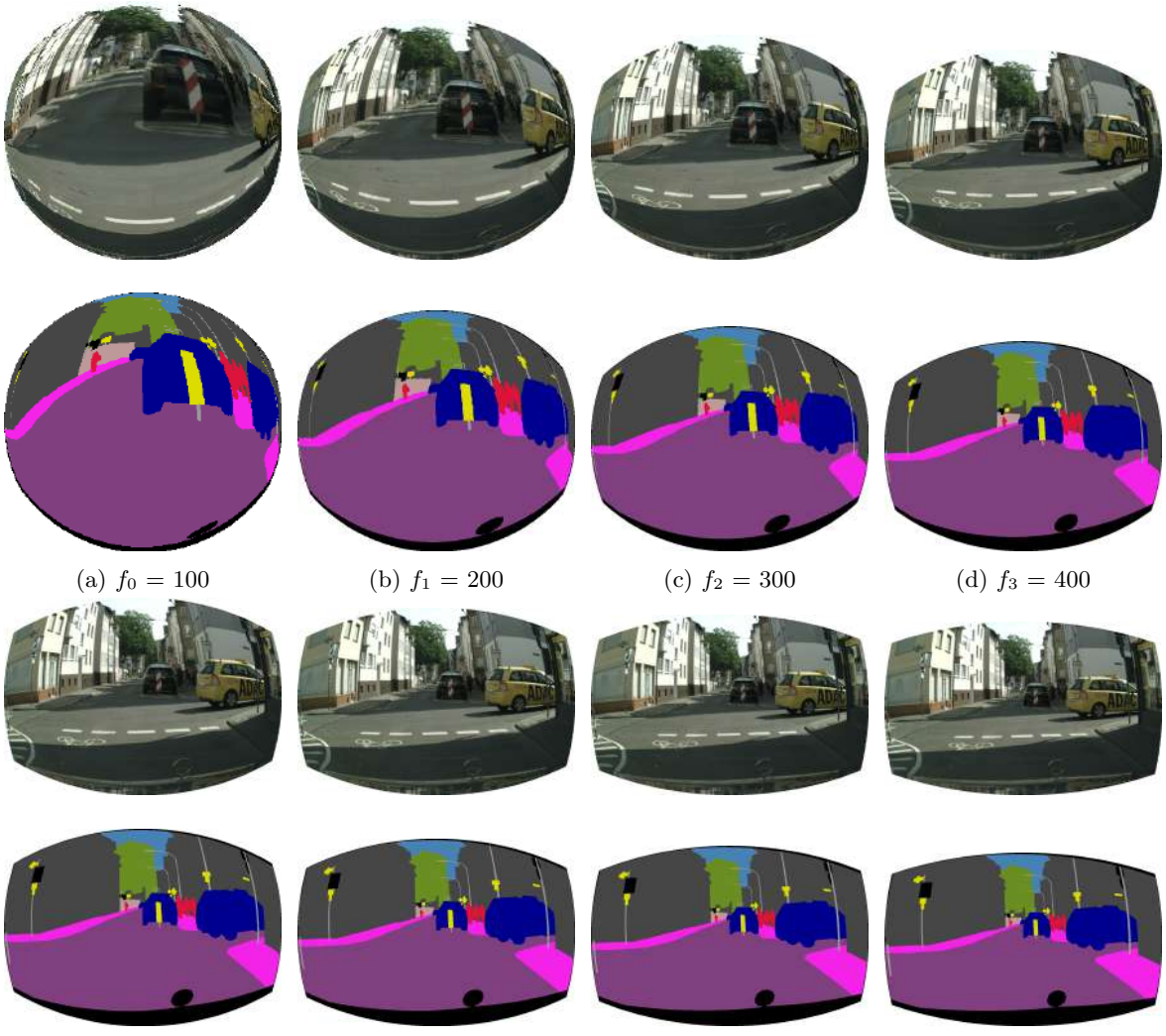
Se crean un total de 5 imágenes por cada imagen presente en CityScapes para el conjunto de entrenamiento, teniendo 4 de ellas distorsiones aleatorias (generadas según una distribución uniforme) y una de ellas con distorsión fija $f_{central} = 500$. Para el conjunto de validación se generan imágenes con distorsiones fijas empleando también la distorsión central $f_{central}$. Los entrenamientos para ambos modelos utilizan los parámetros seleccionados en el capítulo 5 a excepción del número de épocas, que se fija en 150 para el entrenamiento del encoder y en otras 150 para la etapa del modelo completo, persiguiendo obtener formas mejor definidas en la segmentación final. Para ambas arquitecturas se parte de un modelo pre-entrenado en Imagenet ante los buenos resultados previos.

Dados los malos resultados del experimento con 20 clases y las diferencias existentes tanto en forma



(a) Imagen RGB. (b) Ground Truth.

Figura 7.11: Imágenes Originales



(a) $f_0 = 100$ (b) $f_1 = 200$ (c) $f_2 = 300$ (d) $f_3 = 400$

(e) $f_4 = 500$ (f) $f_5 = 600$ (g) $f_6 = 700$ (h) $f_7 = 800$

Figura 7.12: Imágenes con diferentes distorsiones

como en apariencia entre las imágenes sintéticas y las reales se aboga por el uso de 19 clases para evitar confundir a las redes.

Las gráficas mostradas en las imágenes 7.14 y 7.13 muestran la evolución de la pérdida propagada durante el entrenamiento y del IoU obtenido sobre el conjunto de validación para los dos modelos entrenados. Como se puede observar en la gráfica 7.13, el modelo con decoder piramidal supera a la arquitectura original de la red obteniendo mejor % de IoU de clase con mayor claridad que en el

experimento anterior, siendo la diferencia ahora de un 1,32 % (**68,28** % frente a **69,60** %).

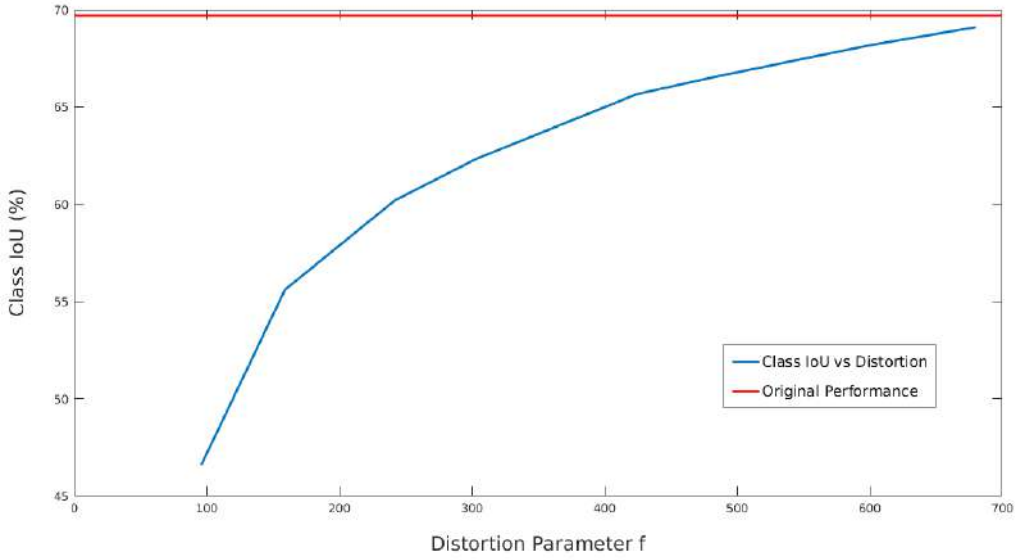


Figura 7.13: Evolución del IoU de clase durante el entrenamiento

En cuanto a la propagación de la pérdida (reflejada en la figura 7.14) se observa que el modelo original ERFNet parte claramente de una pérdida más baja (0,56 frente a 0,42 tras la primera época) pero es superado por el modelo alternativo ERFNetPSP en torno a la época 50, a partir de la cual los valores de ambas pérdidas tienden a separarse en favor del modelo con decoder piramidal.

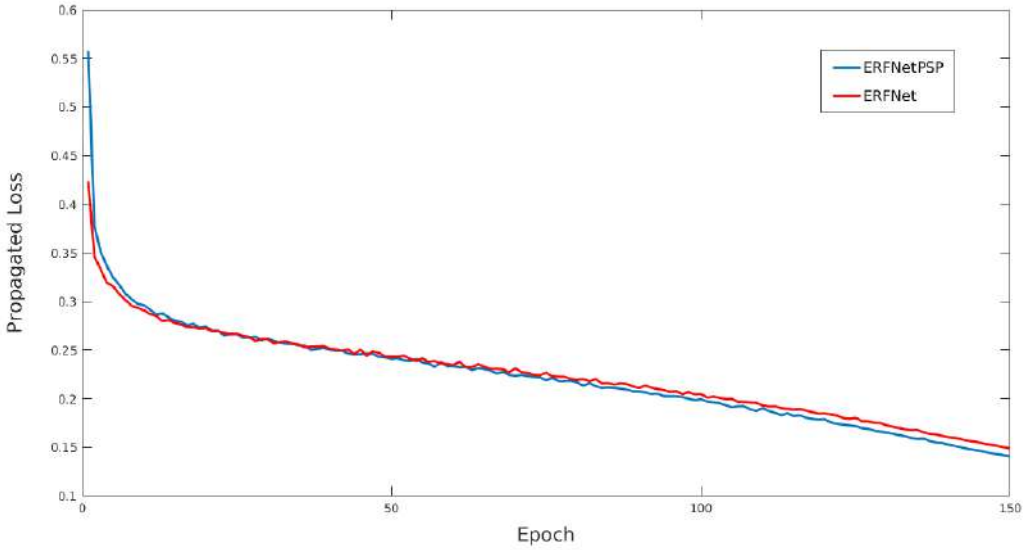


Figura 7.14: Evolución de la pérdida propagada durante el entrenamiento

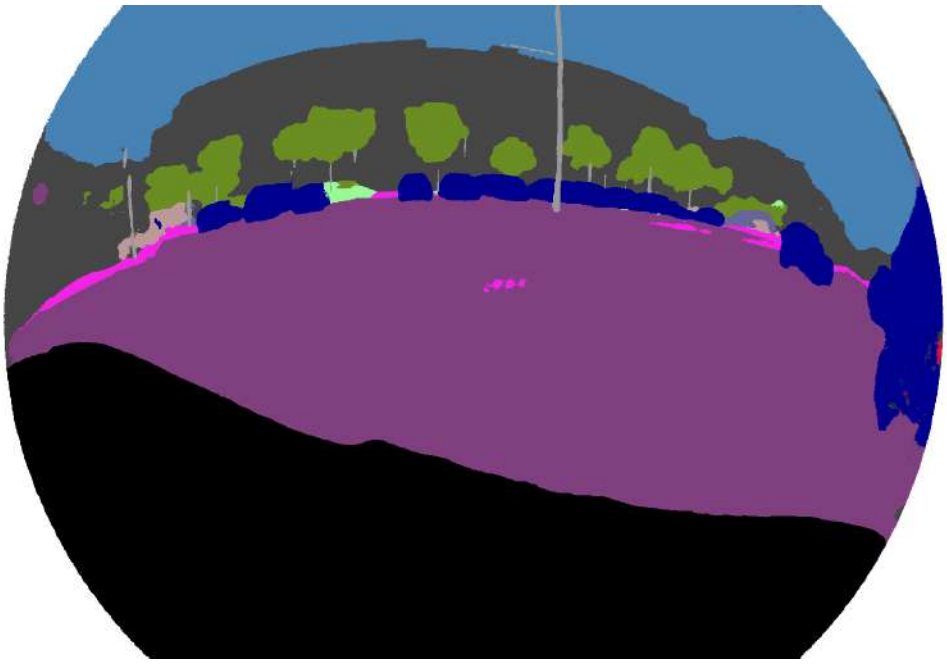
7.6 Resultados Cualitativos

En las siguientes figuras se exponen los resultados obtenidos sobre las secuencias anteriormente grabadas para las dos arquitecturas entrenadas. Como observaciones generales, cabe destacar que la región que se corresponde con el capó del vehículo desde el que se graba aparece siempre mal etiquetada, dado

que es ignorada durante el entrenamiento por lo que las redes no son capaces de interpretarla, por lo que se ha eliminado mediante una máscara. Las primeras imágenes presentadas se corresponden con la segmentación obtenida por el la arquitectura de ERFNet original.

La primera figura 7.15 muestra un ejemplo de segmentación en un entorno urbano. Los resultados expuestos son de buena calidad: los bordes de los elementos presentes en la escena se detectan con mucha precisión. No obstante, los resultados se ven degradados en las regiones laterales en las que el cielo es identificado como edificio debido a su aspecto heterogéneo dentro de la imagen.

(a) Imagen de cámara de ojo de pez de entrada



(b) Imagen segmentada con ERFNet

Figura 7.15: Ejemplos de segmentación semántica con cámara de ojo de pez

La segunda figura 7.16 muestra una segmentación para otra secuencia en un entorno más abierto. En esta imagen el cielo presenta un aspecto más homogéneo, por lo que los resultados en las zonas superiores son de mejor calidad. Cabe destacar también que las formas de los objetos detectados están muy bien definidas y que la red es capaz de identificar correctamente el peatón presente en la imagen. Sin embargo, los bordes obtienen resultados pobres y la zona asociada al capó del coche tiene una segmentación prácticamente aleatoria que influye ligeramente en las zonas adyacentes a la máscara aplicada.

(a) Imagen de cámara de ojo de pez de entrada

(b) Imagen segmentada con ERFNet

Figura 7.16: Ejemplos de segmentación semántica con cámara de ojo de pez

La figura 7.17 presenta más resultados de segmentación obtenidos por la arquitectura ERFNet original. De este conjunto de imágenes es destacable la buena detección de los bordes y formas de los objetos presentes en la escena obtenida por el modelo. Además, la red es capaz de clasificar adecuadamente las clases menos representadas en el dataset como las señales, los postes, los semáforos o las bicicletas. Como elementos negativos destacar los malos resultados obtenidos en la región asociada al cielo debido a su aspecto cambiante que induce a confusión en ciertas imágenes, la dependencia de el rendimiento de la red de la iluminación de la escena, que se reduce cuando existe mucha luz y la mala segmentación obtenida en la región del capó del coche y en las zonas adyacentes a él, que cualitativamente empeora en gran medida los resultados finales.

Abordar el problema de la detección de la parte frontal del vehículo es especialmente complejo en el caso de las cámaras con campo de visión ultra-amplio. Los campos de visión horizontal y vertical de la cámara adquirida están en torno a los 180º por lo que la cámara capta los elementos que están inmediatamente debajo o encima de ella. Este hecho hace muy difícil evitar que aparezca en las imágenes el vehículo desde el que se realiza la grabación. La solución tradicional para las cámaras convencionales es modificar la posición y/u orientación de la cámara para que éste no entre en el campo de visión de los dispositivos, pero para el nuevo tipo de cámara esta solución es inviable dado que el vehículo será visible prácticamente desde cualquier punto de montaje. La mejor solución es la definición de una región de interés tras el establecimiento de un punto de montaje fijo para mejorar la segmentación y evitar la extracción de conclusiones erróneas, cómo se ha llevado a cabo en este trabajo.

El segundo conjunto de imágenes ilustra los resultados de la segmentación asociada al modelo de ERFNet que incorpora el decoder piramidal.

La primera figura de este conjunto 7.18 presenta una imagen asociada a una secuencia de campus en la que se observa que el segundo modelo alcanza resultados de muy buena calidad, detectando con mucha precisión objetos relevantes en la escena como el peatón o los vehículos de la parte trasera de la imagen. El resultado en los bordes sigue siendo pobre, especialmente en el lado derecho aunque el izquierdo es clasificado bien.

La segunda imagen asociada a la segmentación del modelo con decoder piramidal parece representada en la figura 7.19 y pertenece a una secuencia urbana en la que se puede observar cómo el modelo entrenado es capaz de identificar clases poco representadas como la bicicleta. Nuevamente, la red falla al clasificar la región asociada al vehículo desde el que se realiza la grabación, aunque este modelo devuelve una segmentación mucho más homogénea para ella por lo que cualitativamente el resultado es mejor.

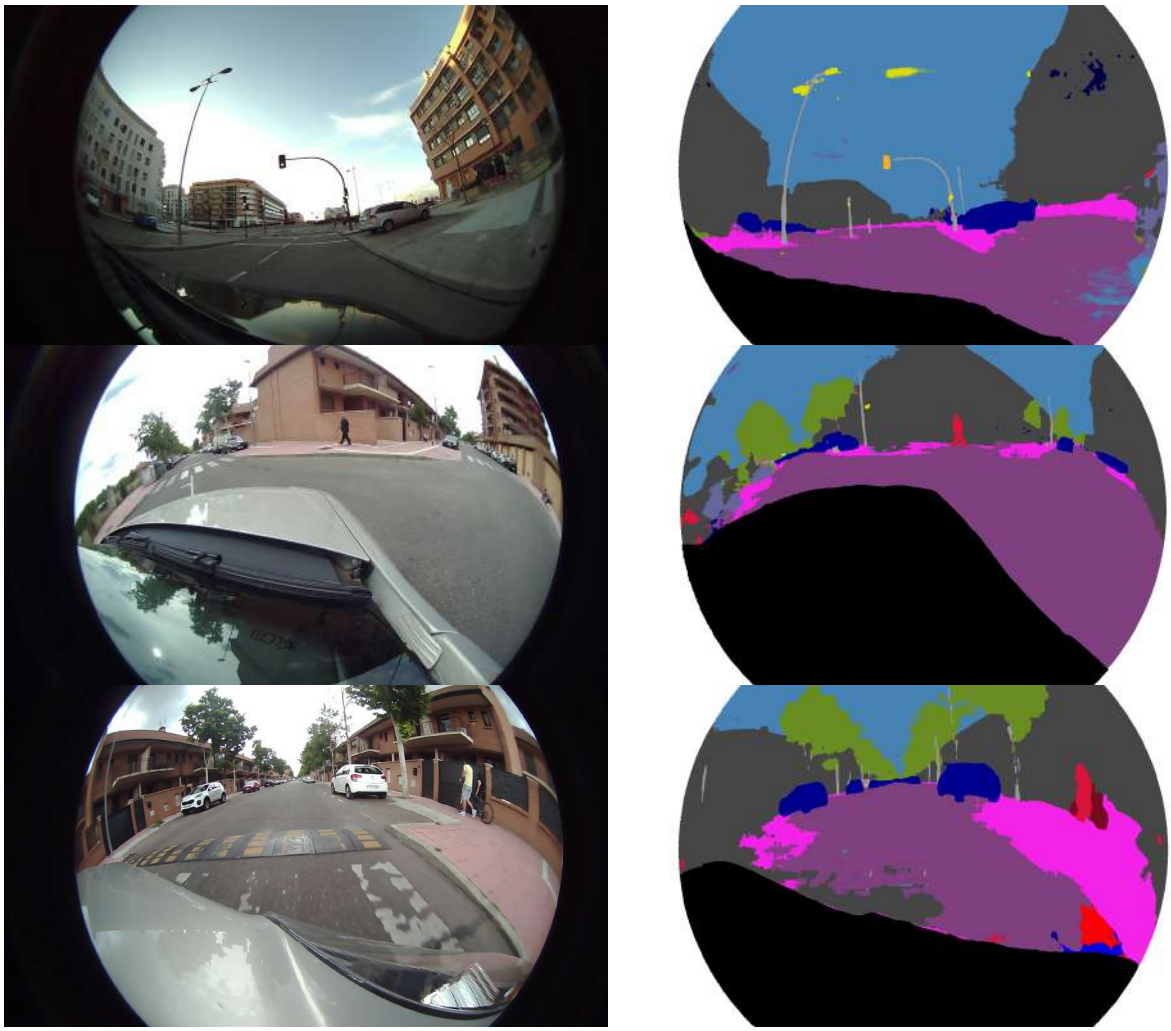


Figura 7.17: Ejemplos de segmentación semántica con ERFNet

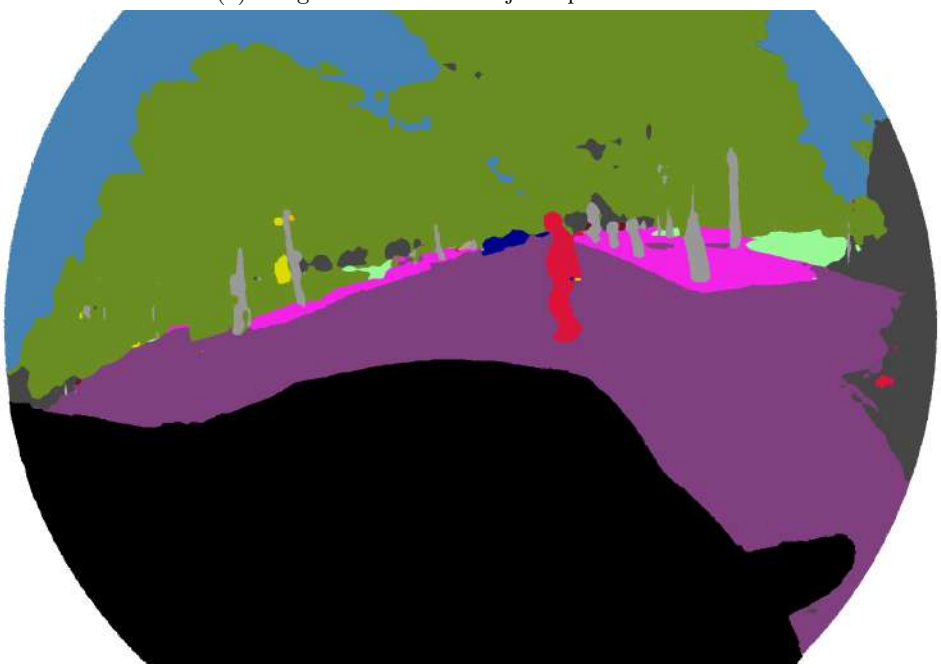
La figura 7.20 expone más resultados obtenidos por la arquitectura con el decoder piramidal. Como conclusión principal para este segundo modelo, destacar que proporciona resultados mucho más homogéneos que el modelo de partida lo que desemboca en una mejor segmentación de regiones que anteriormente resultaban conflictivas como el cielo o la región asociada al vehículo desde el que se graba y las zonas adyacentes a ésta. Cualitativamente, ambas parecen mejorar al ser más uniformes. Además la segmentación mantiene la calidad del modelo original, siendo capaz de identificar clases complicadas cómo se refleja en la figura 7.19b. Sin embargo, esta mejora tiene como punto negativo que la detección de los contornos de los objetos es menos precisa que en el caso anterior.

7.7 SmartElderlyCar

Como experimento final del trabajo la cámara se incorpora a bordo del prototipo eléctrico de SmarElderlyCar. Gracias a su estructura, el vehículo permite una integración sencilla del dispositivo sin ocluir significativamente el campo de visión de la cámara. La figura 7.21 muestra la cámara incorporada en el vehículo mediante la pletina del dispositivo:

La figura 7.22 muestra un ejemplo de imagen tomada desde el vehículo. En ella se observa claramente como el área de la imagen ocluida por la estructura se limita a la zona inferior de la imagen, en la que

(a) Imagen de cámara de ojo de pez de entrada



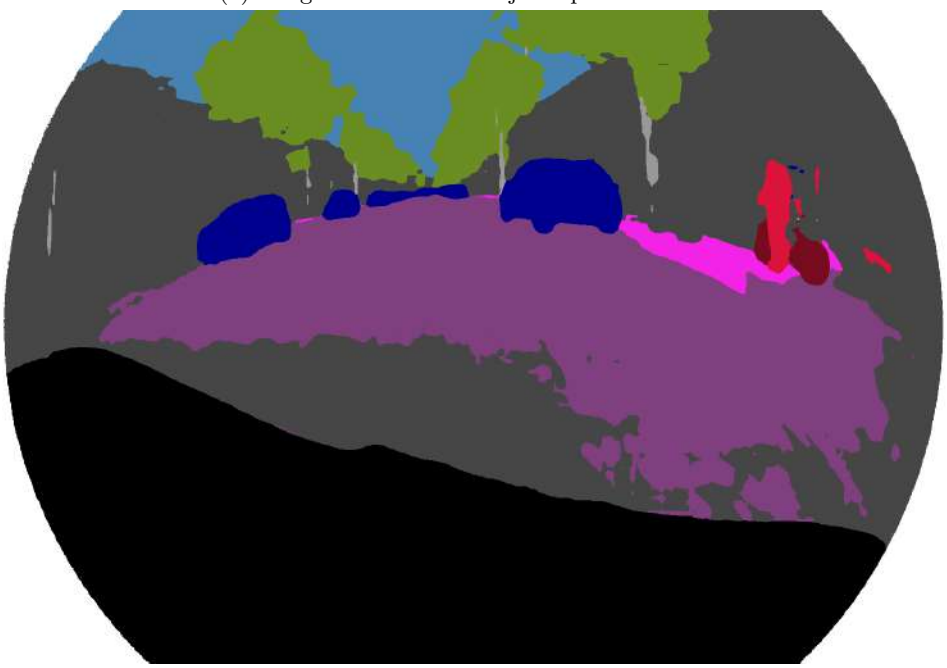
(b) Imagen segmentada con ERFNet con decoder piramidal

Figura 7.18: Ejemplos de segmentación semántica con cámara de ojo de pez

aparecen las ruedas y parte del motor del vehículo, y es mucho más pequeña que en los experimentos anteriores.

En las siguientes figuras se adjuntan ejemplos de la secuencia grabada en el campus segmentada mediante los dos modelos entrenados. La primera imagen 7.23 se corresponde con la segmentación devuelta por el modelo que incorpora el decoder piramidal. La segmentación es de mejor calidad que en los experimentos anteriores dado que ésta no se ve tan perjudicada por la zona ocupada por el vehículo. El modelo demuestra ser capaz de identificar elementos complicados debido a su escasa representación durante el entrenamiento como vallas, a su parecido a otras clases como camiones o señales de tráfico

(a) Imagen de cámara de ojo de pez de entrada



(b) Imagen segmentada con ERFNet con decoder piramidal

Figura 7.19: Ejemplos de segmentación semántica con cámara de ojo de pez

de pequeño tamaño, parcialmente ocluidas por otros objetos y situadas en zonas con niveles elevados de distorsión, pero muestra carencias a la hora de segmentar las objetos situados en las regiones laterales.

La figura 7.24 muestra un ejemplo de segmentación al detalle para el modelo ERFNet original donde, nuevamente, los resultados son de mejor calidad que en los experimentos anteriores. Al igual que en las secuencias previas, el modelo proporciona una segmentación con una mejor detección de formas, bordes y contornos a cambio de ser más heterogénea que la arquitectura del decoder piramidal.

Las dos figuras posteriores muestran múltiples ejemplos de segmentación para los modelos ERFNet

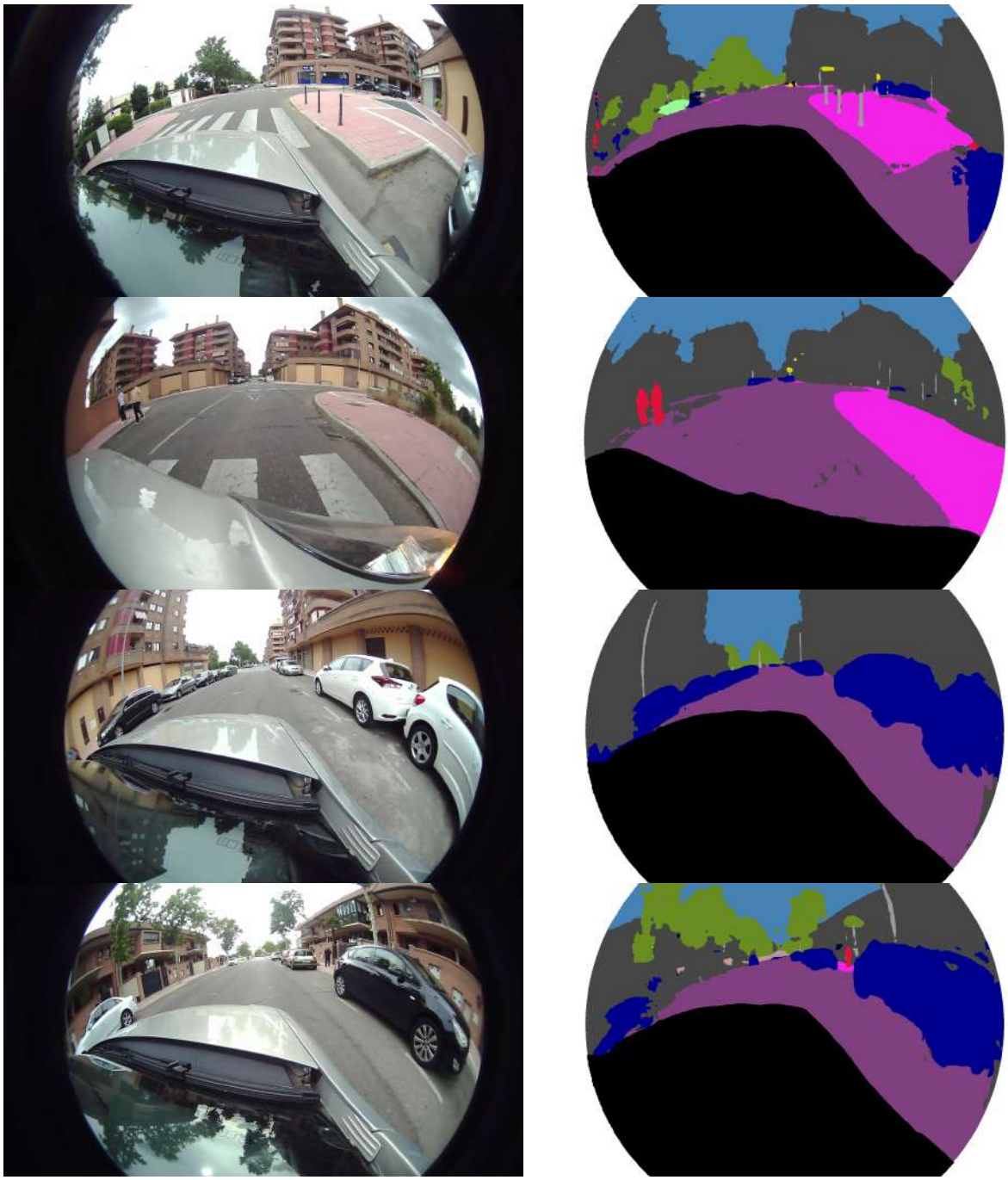


Figura 7.20: Ejemplos de segmentación semántica con ERFNet con decoder piramidal

(figura 7.26) y ERFNet con decoder piramidal (figura 7.25) en los que se confirman los buenos resultados obtenidos por las dos redes entrenadas.

Figura 7.21: Cámara integrada en el prototipo de SmartElderlyCar

Figura 7.22: Imagen tomada desde el prototipo de SmartElderlyCar

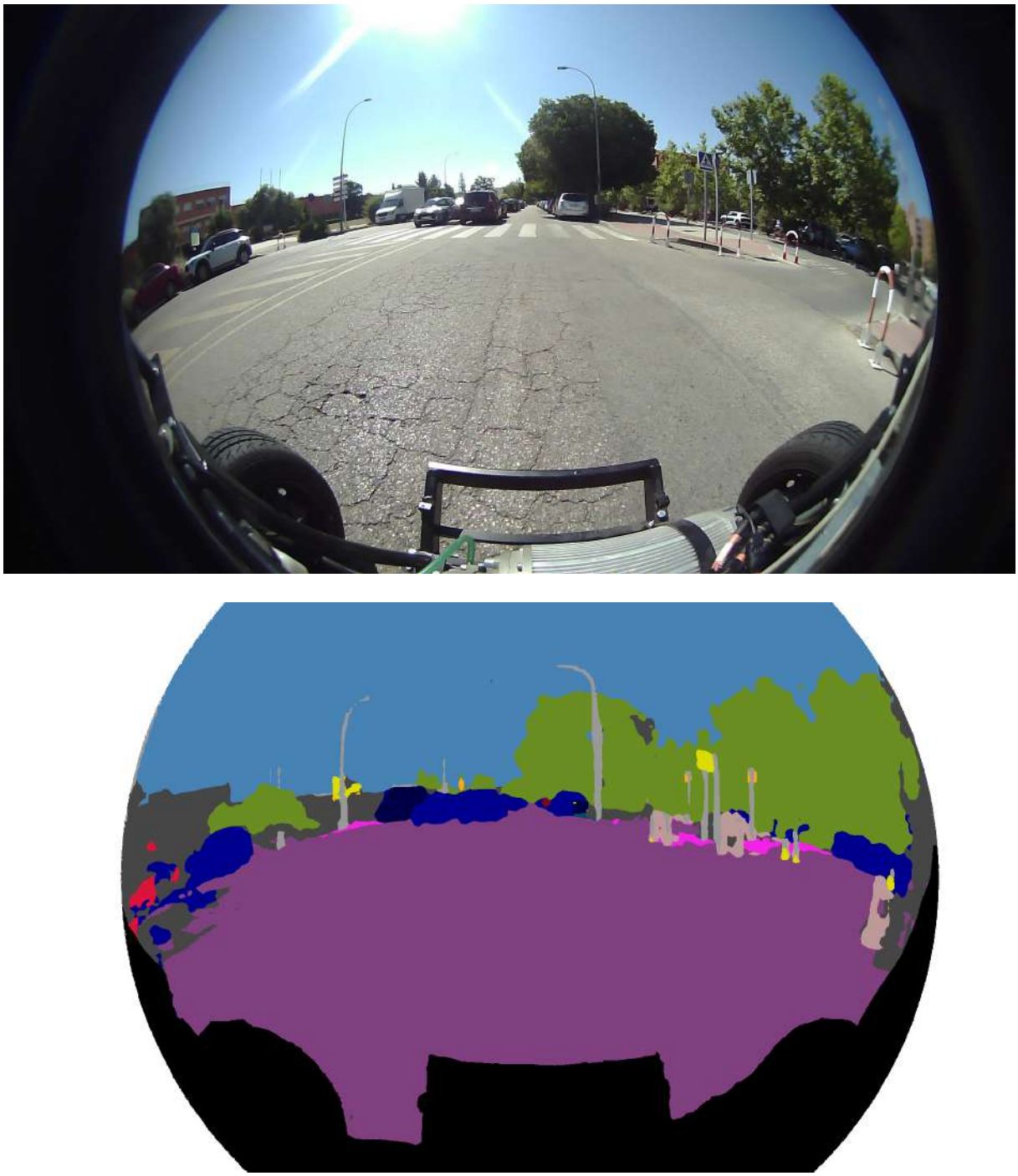


Figura 7.23: Ejemplo de segmentación con el modelo ERFNet-PSP

Figura 7.24: Ejemplo de segmentación con el modelo ERFNet

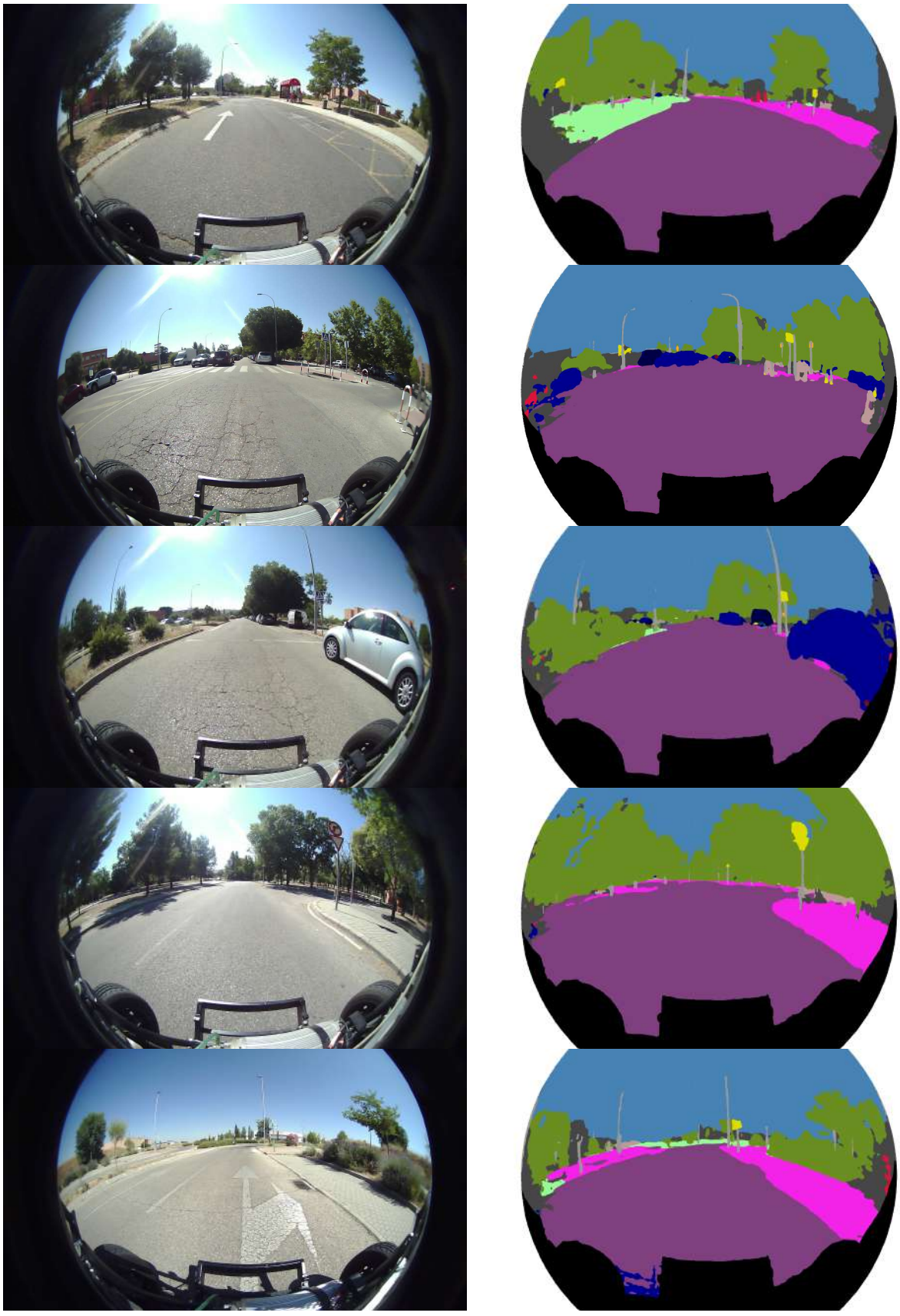


Figura 7.25: Segmentación semántica desde SmartElderlyCar con ERFNetPSP

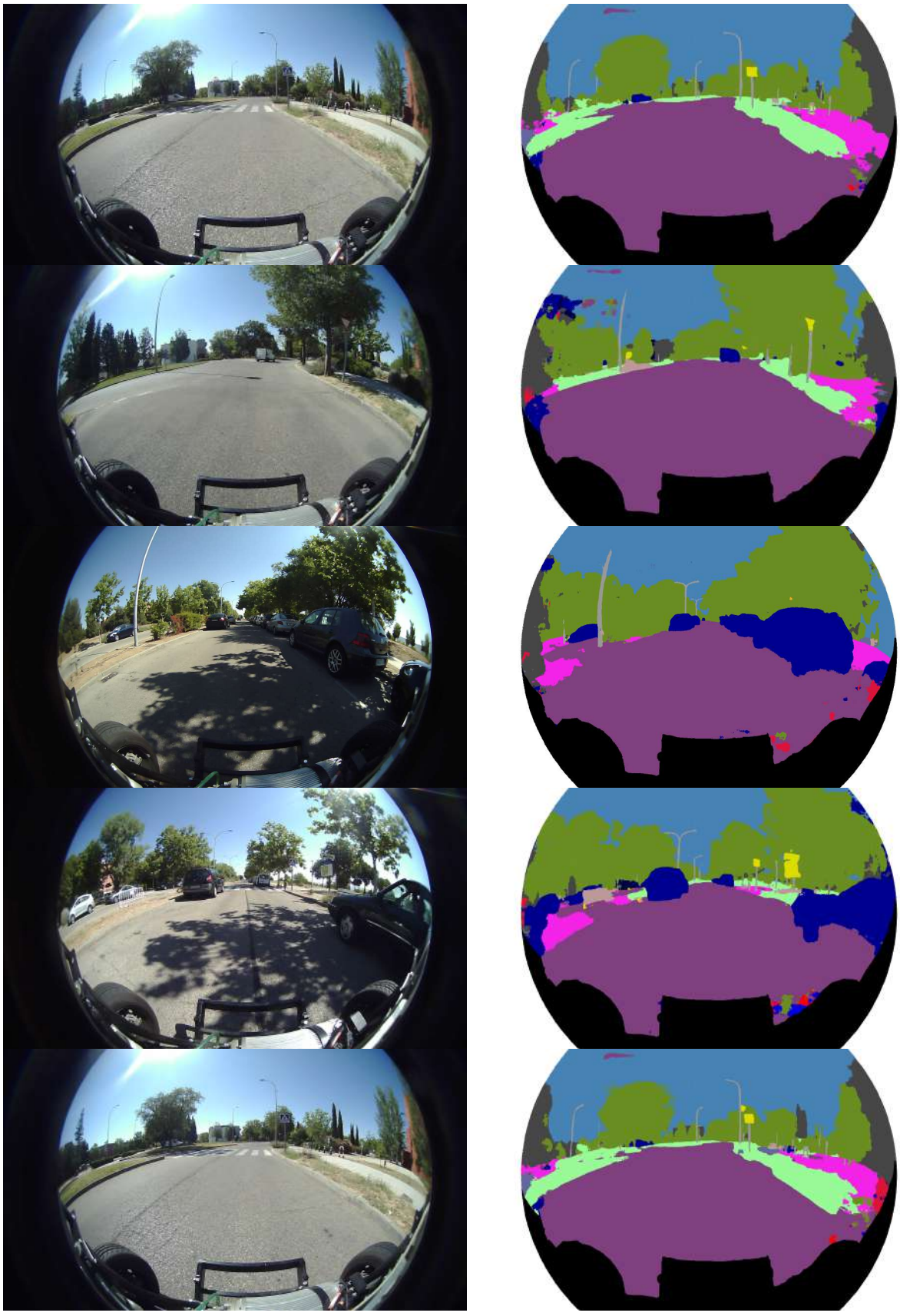


Figura 7.26: Segmentación semántica desde SmartElderlyCar con ERFNet

Capítulo 8

Conclusiones y trabajos futuros

Este capítulo expone el conjunto de conclusiones obtenidas tras el desarrollo del proyecto, así como una serie de propuestas para futuras líneas de investigación que deriven de este trabajo. Además, el capítulo presenta una breve comparativa entre los resultados obtenidos mediante las técnicas propuestas y mediante la combinación de varios métodos tradicionales.

8.1 Conclusiones

Este proyecto se ha centrado en el desarrollo de entrenamientos y arquitecturas CNN específicos para realizar segmentación semántica en imágenes de cámaras con óptica de ojo de pez. Para ello se ha modelado la óptica de una cámara de ojo de pez genérica mediante un modelo matemático utilizado por un gran número de fabricantes. Se descartó la idea inicial de caracterizar la cámara adquirida mediante un proceso de calibración, dado que éste debería implicar dos tareas: la determinación de un modelo proyectivo adecuado a la cámara utilizada (dado que este no es único para las cámaras de ojo de pez) y el cálculo de los parámetros específicos asociados al dispositivo. Sin embargo, las toolboxes existentes no incorporan el primer proceso, sino que se limitan a utilizar modelos proyectivos muy específicos (generalmente basados en geometría esférica) que son matemáticamente sencillos, pero difieren significativamente de las cámaras reales, por lo que los resultados finales de la calibración no son buenos como se mostró en el capítulo 7.

A partir del modelo matemático creado y de las librerías de OpenCV se desarrolló un programa en C++ para crear datasets modificados que permitieran llevar a cabo entrenamientos específicos para cámaras *fisheye*. La transformación se realiza en base a un parámetro que simula una distancia focal y que permite regular el nivel de distorsión incluido en las imágenes. Con esta herramienta, en el capítulo 3 se crearon diferentes datasets sintéticos con distintas distorsiones basados en CityScapes, siguiendo líneas de trabajo previas.

Como arquitectura de red básica se utilizó ERFNet, buscando dar lugar a una segmentación semántica funcional en tiempo real y de buena calidad. Como arquitectura adicional, se propuso una modificación de la red básica que incorporaba el decoder piramidal de la PSPNet, persiguiendo primar la información contextual sobre el resto de items relevantes en el entrenamiento de una CNN, dado que el resto se ven perjudicados por la distorsión de las imágenes. También se probaron redes con arquitecturas distintas a ERFNet en el capítulo 6. Además, se implementaron técnicas de *data-augmentation* específicas para cámaras de ojo de pez propuestas en [22] y se realizó una nueva propuesta basada en el uso de distorsiones

Figura 8.1: Ejemplos de imágenes de los datasets sintéticos

aleatorias, que alcanzó resultados similares a los del estado del arte con la ventaja de no requerir un estudio teórico previo para ser óptimas.

Los entrenamientos iniciales con este *data-augmentation* y las dos arquitecturas planteadas en el capítulo 5 superaron con claridad las anteriores propuestas del estado del arte en varios datasets con distorsiones distintas. Además, las diferentes técnicas de *data-augmentation* utilizadas demostraron su eficiencia, suponiendo una mejora tanto a nivel cualitativo como cuantitativo.

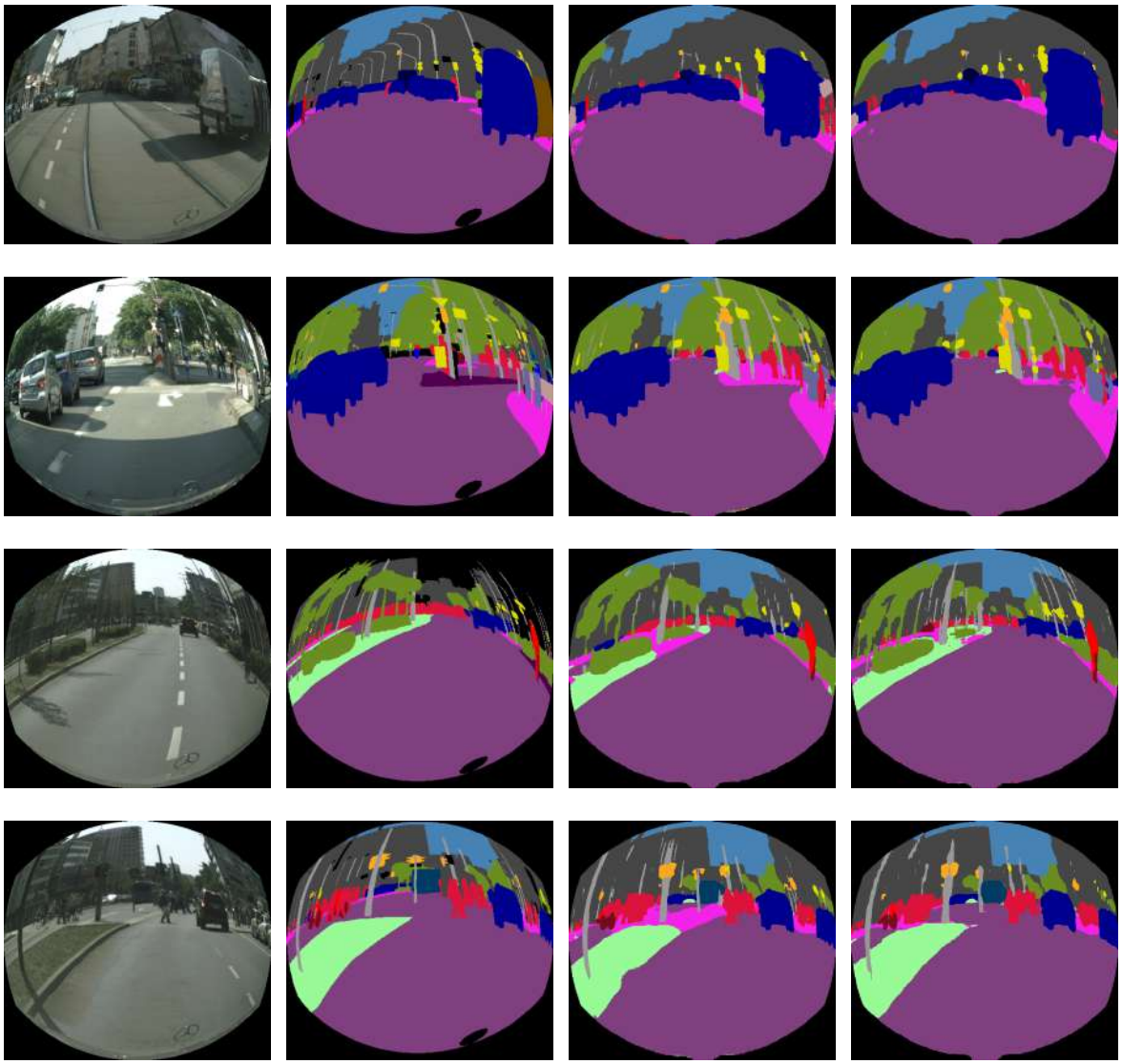
El modelo original de ERFNet obtuvo mejores resultados en los primeros experimentos, en los que se entrenaba sobre datasets con niveles de distorsión altos. No obstante, a medida que la distorsión se reduce y que se utilizan imágenes de mayores escala y resolución en las etapas de entrenamiento y test los resultados para ambos modelos se vuelven parejos a nivel cuantitativo, resultando los del modelo con decoder piramidal superiores en los últimos dos entrenamientos. A nivel cualitativo el modelo original define mejor las formas de los elementos presentes en la escena pero a cambio da lugar a más errores de píxeles aislados en la imagen. La segmentación del modelo con decoder piramidal es más homogénea y comete menos errores aislados, pero da lugar a formas muy regulares que no se adaptan del todo bien a los objetos reales de la imagen. La figura 8.2 muestra ejemplos de segmentación para ambas redes: la columna fila de imágenes contiene las imágenes de entrada, la segunda el ground-truth, la tercera la segmentación del modelo de ERFNet y la cuarta del modelo de ERFNet con decoder piramidal.

El capítulo 7 adaptó los métodos desarrollados en los primeros capítulos a las imágenes adquiridas por una cámara de ojo de pez propia, dado que los entrenamientos iniciales utilizaban imágenes con resoluciones muy bajas que no aportaban una segmentación buena en el dispositivo adquirido. El entrenamiento final se realizó para las dos arquitecturas escogidas junto con la nueva técnica de *data-augmentation* propuesta y un rango de distorsiones adecuado para las imágenes capturadas por la cámara.

Para el test final de resultados se grabaron un conjunto de imágenes repartidas en varias secuencias que incluían entornos de campus y urbanos, tanto a bordo de vehículos tradicionales como a bordo del prototipo eléctrico de SmartElderlyCar. La segmentación obtenida en las secuencias es de buena calidad para ambos modelos exceptuando para el área asociada a los bordes de las imágenes y a la región asociada al cielo. Los malos resultados en estas regiones tienen 3 motivos principales:

- **Modelo de distorsión**: el modelado matemático de la cámara no es exacto. A pesar de la buena capacidad de generalización que la red obtiene mediante el *data-augmentation* del entrenamiento, el aspecto de los elementos en los bordes es diferente en las imágenes de entrenamiento y en las capturadas mediante la cámara.

- **Apariencia de objetos**: el aspecto presentado por un objeto visto desde un ángulo de 180 grados difiere notablemente del que presenta al ser visto de frente desde una cámara tradicional. Así, para que la clasificación en los bordes fuese de calidad, el entrenamiento debería incluir imágenes en las



(a) Imágenes de entrada RGB (b) Imágenes de Ground-Truth (c) Segmentación ERFNet (d) Segmentación ERFNetPSP

Figura 8.2: Ejemplos de segmentación semántica para los modelos entrenados

que las distintas clases aparecieran representadas desde diferentes perspectivas para conseguir que la red identifique bien múltiples apariencias.

- **Brillo y reflejos**: la mala performance de la red en la zona asociada al cielo sigue siendo debida a los brillos producidos por la lente en la imagen final. Éstos confunden a la red al ocluir los objetos que quedan tras ellos y, además, degradan la información contextual de los mapas de características.

La figura 8.3 muestra un ejemplo de segmentación errónea en las dos regiones mencionadas anteriormente debido a los problemas expuestos.

Las segmentaciones obtenidas por los dos modelos en las secuencias reales presentan características análogas a las obtenidas en los datasets sintéticos. El modelo de ERFNet original es capaz de definir con mucha precisión las formas y contornos de los objetos, pero la segmentación que obtiene es muy heterogénea y en ciertas regiones como el cielo es, habitualmente, errónea. Por otro lado, el modelo con decoder piramidal, que ya obtenía mejores resultados a nivel cuantitativo para estos rangos de distorsión,

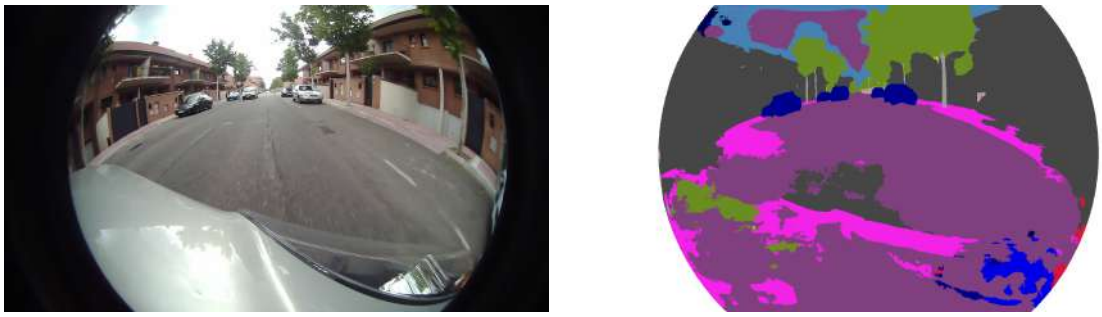


Figura 8.3: Problemas de la segmentación final

da lugar a una segmentación mucho más homogénea sin errores a nivel de píxeles aislados. Además, es muy robusta a la hora de detectar las clases que para el otro modelo resultaban conflictivas. El defecto más destacable frente a la arquitectura original continua siendo la detección de los contornos, que es claramente inferior. Sin embargo, evaluando los resultados de los dos modelos a nivel global **el rendimiento del modelo con decoder piramidal (ERFNetPSP) es mejor para la cámara real.**

Con la segmentación semántica obtenida (correcta en toda la imagen exceptuando las zonas de los bordes) y dado que para combinar las imágenes obtenidas por varias cámaras conviene que exista una región de sus campos de visión solapada para facilitar la fusión de información de los distintos dispositivos, para crear un sistema de percepción que abarque el entorno completo del prototipo eléctrico con los dispositivos y modelos entrenados hasta el momento serían necesarias 3 cámaras de ojo de pez como la utilizada en este trabajo. La estructura del vehículo facilita la integración de cámaras de este tipo 7.21 dado que apenas hay elementos del coche que puedan ocluir zonas del campo de visión de las cámaras.

Las figuras 8.4 y 8.5 muestran ejemplos de imágenes ya segmentadas en las que se ha eliminado la distorsión siguiendo el proceso expuesto en el capítulo previo. Esto permite la integración inmediata en el el sistema de percepción global del vehículo mediante la fusión con la información de profundidad proporcionada por el lidar.

8.2 Aportaciones

Este trabajo avanza en una línea de investigación poco desarrollada hasta el momento. Durante su desarrollo, se ha conseguido superar los resultados actuales del estado del arte en múltiples campos y, además, se han propuesto un conjunto de ideas novedosas que han sido implementadas y testeadas. Entre las principales aportaciones de este proyecto cabe destacar:

- **La generación de un conjunto de datasets sintéticos a partir de CityScapes** utilizando el modelo proyectivo de una cámara de ojo de pez genérica, utilizando distorsiones variadas para evaluar su influencia sobre el rendimiento de las redes y aplicar diferentes técnicas de *data-augmentation*.
- **La demostración a nivel práctico los beneficios de utilizar datasets sintéticos** con distorsiones añadidas sobre las líneas alternativas que buscan eliminar la distorsión de las imágenes para después aplicar los métodos tradicionales.
- **La presentación de un estudio comparativo de un conjunto amplio de CNNs** con diferentes virtudes sobre datasets sintéticos con distorsión de ojo de pez. Se han estudiado

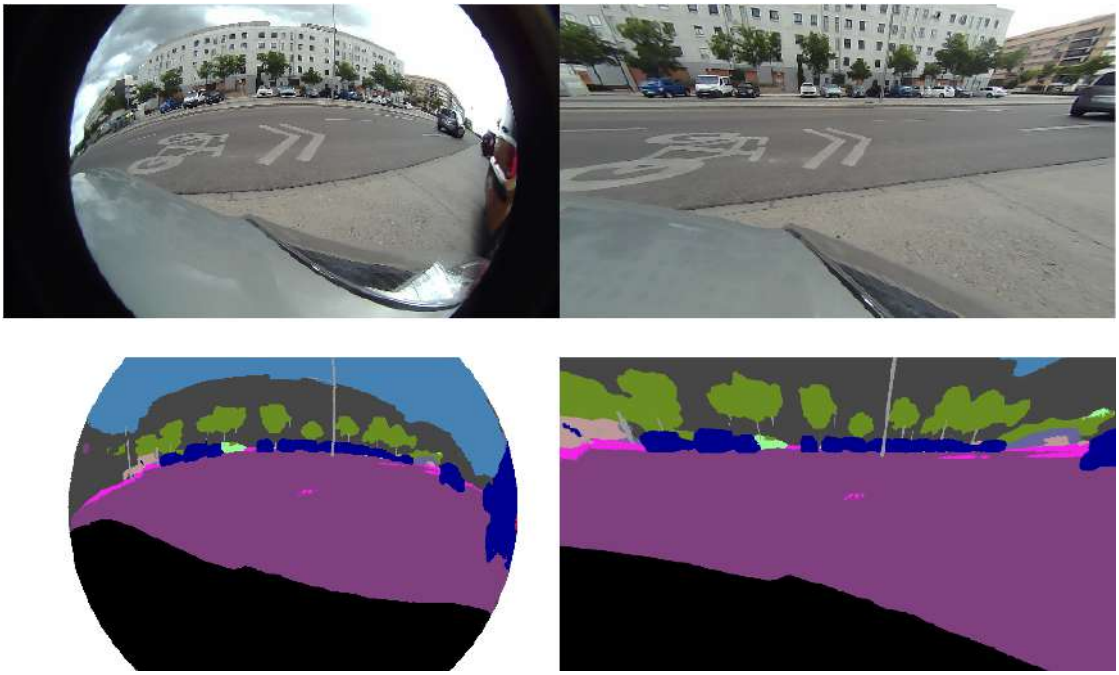


Figura 8.4: Ejemplos de segmentación semántica tras el proceso de corrección de la imagen

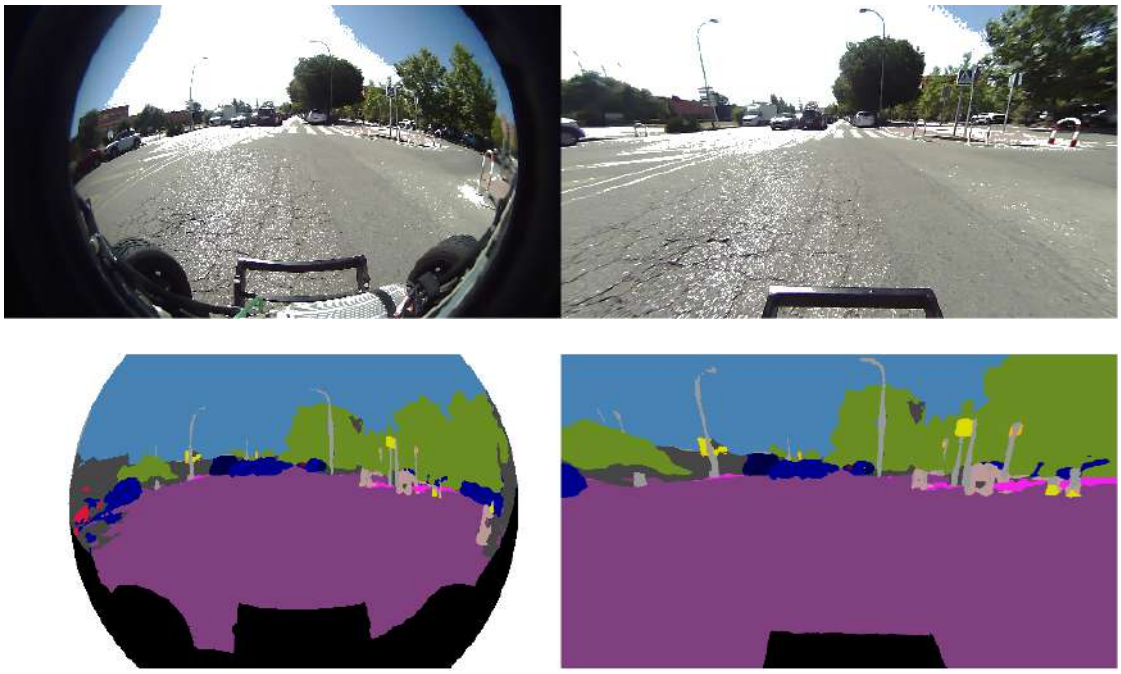


Figura 8.5: Ejemplos de segmentación semántica tras el proceso de corrección de la imagen

redes centradas en obtener la mejor performance (PSPNet y DRNs), redes enfocadas a ser eficientes computacionalmente (SegNet, FRDCNet y RDCNet) y redes que buscaban mantener un compromiso entre los dos puntos anteriores (ERFNet y derivados). De entre todas ellas, la red que constituía la línea base del trabajo (ERFNet) ha sido la que mejores resultados ha ofrecido, por lo que ha sido la utilizada finalmente.

- **La propuesta de una arquitectura modificada para la red ERFNet original (ERFNetPSP)** que utiliza el modelo de decoder piramidal de PSPNet con objetivo de mejorar la extracción de información contextual que, como se ha demostrado durante el trabajo, es especialmente relevante en las imágenes con distorsión de ojo de pez. El modelo ha demostrado ser claramente mejor a la arquitectura original en imágenes de alta resolución, confirmando la hipótesis empleada como punto de partida.

- **El estudio de múltiples tipos de *data-augmentation***, de manera aislada y combinada hasta llegar a la solución que ha sido considerada como la más valida: adaptación de dominio junto con el uso de una nueva propuesta derivada de la anterior idea de utilizar diferentes distorsiones durante el entrenamiento. Ambas técnicas han sido desarrolladas por el grupo de investigación Robesafe.

- **La propuesta, implementación y test de una nueva técnica de *data-augmentation*** basada en el uso de distorsiones seleccionadas de manera aleatoria durante el proceso de entrenamiento. Los resultados obtenidos demuestran que ésta es capaz de mejorar notablemente la capacidad de generalización de la red mejorando su respuesta ante diferentes distorsiones con lo que se consigue una performance similar a la anterior alternativa del estado del arte (*zoom-augmentation*).

- La combinación de las dos nuevas redes entrenadas en este tipo de datasets junto con el *data-augmentation* desarrollado supera claramente las anteriores propuestas existentes, mejorando en un **4,8 %** el IoU de clase obtenido para el dataset de $f_0 = 159$ por la mejor propuesta del estado del arte previa y en un **2,5 %** el del dataset con $f_0' = 240$.

- **Las redes entrenadas son funcionales en tiempo real**, alcanzando el modelo original de ERFNet los 50 fps y el que incorpora un decoder piramidal los 45 fps sobre una única Titan X. Este apartado supone una mejora importante, ya que se trata de la primera propuesta de segmentación semántica funcional en tiempo real sobre imágenes de ojo de pez.

- **La adaptación de las técnicas desarrolladas a una cámara de ojo de pez real** adquirida durante el proyecto, mediante un entrenamiento específico para las redes, con el objetivo de aplicarlas sobre un coche autónomo real.

- **La publicación de resultados de segmentación semántica de buena calidad sobre imágenes adquiridas mediante una cámara de ojo de pez real y, por primera vez, obtenidas a partir redes entrenadas únicamente con imágenes sintéticas generadas mediante una base de datos con Ground-Truth disponible como es CityScapes**. Así se demuestra la capacidad de generalización de contexto ya que se entrena sobre imágenes previamente no vistas y minimiza la ardua tarea de generación de un GT específico para el dominio sobre el que el sistema será probado.

8.3 Trabajos futuros

La línea de investigación de este trabajo queda abierta al no haberse completado el objetivo final de implementar el sistema de percepción en un vehículo autónomo. El trabajo futuro derivado de este proyecto abarca tareas asociadas a la implementación final del sistema sobre el vehículo autónomo entre las cuales se incluyen:

- La mejora de los entrenamientos realizados mediante la inclusión de imágenes de ojo de pez etiquetadas a nivel de píxel que, previsiblemente, pasarán a estar disponibles en los próximos meses.

Sin ellas no se puede abordar el problema presentado a la hora de clasificar objetos vistos desde perspectivas diferentes introducido en el apartado anterior.

- El diseño de un sistema de percepción que abarque el entorno completo del vehículo autónomo (360º) basado en 2 o 3 cámaras similares a la utilizada. Este sistema requerirá una calibración extrínseca entre las distintas cámaras y con el resto de sensores del vehículo (lidar y cámara estéreo).
- La realización de entrenamientos alternativos para las redes utilizando imágenes capturadas desde otros puntos del vehículo que proporcionen un mejor rendimiento al situar cámaras en los laterales o en la parte trasera del coche. Para este punto se debe valorar el uso de simuladores que permiten customizar el punto de anclaje de las cámaras como [67].
- La integración del nuevo sistema de percepción en el proyecto SmartElderlyCar que, además de las calibraciones extrínsecas, debe incluir una sincronización con el resto de sensores y la publicación de las imágenes mediante las librerías de ROS.
- La definición de regiones de interés que delimiten las zonas que estarán ocluidas tras la integración con objetivo de evitar la extracción de conclusiones erróneas.

Capítulo 9

Pliego de condiciones

9.1 Introducción

En este capítulo se adjunta todo el presupuesto necesario para la ejecución del proyecto. Se adjuntan tanto los presupuestos para materiales como para mano de obra, todos ellos sin impuestos incluidos.

9.2 Requisitos Software

A nivel de software, para el desarrollo del proyecto, se ha recurrido mayoritariamente a recursos de código abierto o a programas para los que la propia universidad disponía de licencias de naturaleza académica como MATLAB. La tabla 9.1 recoge el conjunto de sistemas operativos, librerías y programas utilizados durante el desarrollo del trabajo:

Tabla 9.1: Requisitos Software para el proyecto

Concepto	**Precio**	**Periodo Amortización**	**Periodo de Uso**	**Coste**
S.O Ubuntu 16.04 LTS	Open-Source	N/A	N/A	0
MATLAB 2017b	Gratuito (Licencia Académica)	N/A	N/A	0
OpenCV 3.3	Open-Source	N/A	N/A	0
CUDA	Open-Source	N/A	N/A	0
Pytorch	Open-Source	N/A	N/A	0
			Total	**0€**

9.3 Requisitos Hardware

En el apartado de recursos hardware, se incluye el coste de la cámara adquirida en el proyecto (descrita en el capítulo 7), el coste de las 2 GPUs utilizadas durante el desarrollo del trabajo, el del disco duro

utilizado para el almacenamiento de datasets y los costes de los componentes del ordenador utilizado en él.

Tabla 9.2: Requisitos Hardware para el proyecto

Concepto	**Precio**	**Periodo Amortización**	**Periodo de Uso**	**Coste**
Cámara Fisheye	34,36 $	3 años	12 meses	11,45 €
Gigabyte GeForce GTX Titan X 12GB	1070,80€	5 años	12 meses	214,16€
Gigabyte GeForce GTX 1080 Aorus-11GD	845,90€	5 años	12 meses	169,18€
Toshiba Canvio Basics 2 TB	69,90€	5 años	12 meses	13,98€
			Total	**408,77€**

9.4 Costes Adicionales

En la tabla 9.3 se recogen el conjunto de costes asociados a gastos adicionales del proyecto, entre los que se incluyen gastos de reprografía, materiales de oficina y transporte:

Tabla 9.3: Conjunto de costes adicionales del proyecto

Concepto	**Coste**
Reprografía	40€
Materiales de Oficina	30€
Transporte	240€
Total	**310€**

9.5 Mano de obra

La tabla 9.4 recoge el desglose de todos los costes asociados a la mano de obra involucrada en el desarrollo del trabajo en la que se incluyen las horas invertidas en las etapa de desarrollo, test y documentación de los resultados obtenidos.

Tabla 9.4: Requisitos Hardware para el proyecto

Concepto	**Precio**	**Horas**	**Coste**
Ingeniería/programación	20€h	700	14000€
Test	20€h	30	600€
Reprografía	20€h	60	1200€
		Total	**15800€**

9.6 Presupuesto total

La tabla 9.5 recoge los costes totales del proyecto.

Tabla 9.5: Costes totales del proyecto

Concepto	**Coste**
Costes Software	0€
Costes Hardware	408,77€
Costes Adicionales	310€
Mano de Obra	15800€
Total	**16518,77€**

Los costes totales de este proyecto ascienden a dieciséis mil quinientos dieciocho euros con setenta y siete céntimos, impuestos no incluidos.

En Alcalá de Henares, a día 1 de septiembre de 2018.

Álvaro Sáez Contreras
Graduado en Ingeniería en Tecnologías de la Telecomunicación

Bibliografía

[1] M. A. Sotelo, L. M. Bergasa, J. Nuevo, and M. Ocaña, "A monocular solution to vision based acc in road vehicles," in *Lecture Notes Compute. Sci. 3643*, 2015, pp. 507–512.

[2] I. Parra, D. Fernandez, M. A. Sotelo, P. Revenga, L. M. Bergasa, M. Ocaña, R. Flores, and M. García, "A combination of feature extraction methods for svm pedestrian detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 292–307, 2007.

[3] I. G. Daza, L. M. Bergasa, S. Bronte, J. J. Yebes, J. Almazán, and R. Arroyo, "Fusion of optimized indicators from advanced driver assistance systems (adas) for driver drowsiness detection," *Sensors*, vol. 14, no. 1, pp. 1106–1131, 2014.

[4] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 Year, 1000km: The Oxford RobotCar Dataset," *The International Journal of Robotics Research (IJRR)*, vol. 36, no. 1, pp. 3–15, 2017. [Online]. Available: http://dx.doi.org/10.1177/0278364916679498

[5] E. Romera, L. M. Bergasa, and R. Arroyo, "Can we unify monocular detectors for autonomous driving by using the pixel-wise semantic segmentation of cnns?" *arXiv preprint arXiv:1607.00971*, 2016.

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[8] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3213–3223.

[10] N. H. T. S. Administration *et al.*, "Nhtsa (2013)," *Traffic Safety Fact Sheet*, 2012.

[11] P. Sturgess, K. Alahari, L. Ladicky, and P. H. Torr, "Combining appearance and structure from motion features for road scene understanding," in *BMVC 2012-23rd British Machine Vision Conference*. BMVA, 2009.

[12] S. Sengupta, J. Valentin, J. Warrell, A. Shahrokni, and P. Torr, "Mesh based semantic modelling for indoor and outdoor scenes," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, vol. 6, no. 6.2, 2013, pp. 6–2.

[13] G. Ros, S. Ramos, M. Granados, A. Bakhtiary, D. Vazquez, and A. M. Lopez, "Vision-based offline-online perception paradigm for autonomous driving," in *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on.* IEEE, 2015, pp. 231–238.

[14] R. Arroyo, P. F. Alcantarilla, L. M. Bergasa, and E. Romera, "Towards life-long visual localization using an efficient matching of binary sequences from images," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on.* IEEE, 2015, pp. 6328–6335.

[15] M. Passani, J. J. Yebes, and L. M. Bergasa, "Fast pixelwise road inference based on uniformly reweighted belief propagation," in *Intelligent Vehicles Symposium (IV), 2015 IEEE.* IEEE, 2015, pp. 519–524.

[16] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[17] E. Hsiao, P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," 2009.

[18] S. Silberstein, D. Levi, V. Kogan, and R. Gazit, "Vision-based pedestrian detection for rear-view cameras," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE.* IEEE, 2014, pp. 853–860.

[19] A. Broggi, E. Cardarelli, S. Cattani, P. Medici, and M. Sabbatelli, "Vehicle detection for autonomous parking using a soft-cascade adaboost classifier," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE.* IEEE, 2014, pp. 912–917.

[20] Y. Qian, M. Yang, C. Wang, and B. Wang, "Self-adapting part-based pedestrian detection using a fish-eye camera," in *Intelligent Vehicles Symposium (IV), 2017 IEEE.* IEEE, 2017, pp. 33–38.

[21] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* IEEE, 2009, pp. 304–311.

[22] L. Deng, M. Yang, Y. Qian, C. Wang, and B. Wang, "Cnn based semantic segmentation for urban traffic scenes using fisheye camera," in *Intelligent Vehicles Symposium (IV), 2017 IEEE.* IEEE, 2017, pp. 231–236.

[23] L. Deng, M. Yang, H. Li, T. Li, B. Hu, and C. Wang, "Restricted deformable convolution based road scene semantic segmentation using surround view cameras," *arXiv preprint arXiv:1801.00708*, 2018.

[24] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[25] D. Scaramuzza, "Omnidirectional camera," in *Computer Vision.* Springer, 2014, pp. 552–560.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision.* Springer, 2014, pp. 740–755.

[27] 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.

[28] R. Real and J. M. Vargas, "The probabilistic basis of jaccard's index of similarity," *Systematic biology*, vol. 45, no. 3, pp. 380–385, 1996.

[29] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[30] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "Carla: An open urban driving simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[31] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[32] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for scene segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[33] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv preprint arXiv:1606.00915*, 2016.

[34] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr, "Conditional random fields as recurrent neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1529–1537.

[35] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[37] A. Chaurasia and E. Culurciello, "Linknet: Exploiting encoder representations for efficient semantic segmentation," *arXiv preprint arXiv:1707.03718*, 2017.

[38] M. Treml, J. Arjona-Medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich *et al.*, "Speeding up semantic segmentation for autonomous driving," in *MLITS, NIPS Workshop*, 2016.

[39] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2881–2890.

[40] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[41] G. Lin, A. Milan, C. Shen, and I. Reid, "Refinenet: Multi-path refinement networks with identity mappings for high-resolution semantic segmentation," *arXiv preprint arXiv:1611.06612*, 2016.

[42] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe, "Full-resolution residual networks for semantic segmentation in street scenes," *arXiv preprint*, 2017.

[43] G. Ghiasi and C. C. Fowlkes, "Laplacian pyramid reconstruction and refinement for semantic segmentation," in *European Conference on Computer Vision*. Springer, 2016, pp. 519–534.

[44] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "Efficient convnet for real-time semantic segmentation," in *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE, 2017, pp. 1789–1794.

[45] G. Lin, C. Shen, A. Van Den Hengel, and I. Reid, "Efficient piecewise training of deep structured models for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3194–3203.

[46] C. Liang-Chieh, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in *International Conference on Learning Representations*, 2015.

[47] J. Alvarez and L. Petersson, "Decomposeme: Simplifying convnets for end-to-end learning," *arXiv preprint arXiv:1606.05426*, 2016.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[49] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[50] J. M. A. Eduardo Romera, Luis M. Bergasa and M. Trivedi, "Train here, deploy there: Robust segmentation in unseen domains," in *Proceedings of the IEEE conference on Intelligent Vehicles Symposium*. IEEE ITS, 2018, p. to appear.

[51] A. Sáez, L. M. Bergasa, E. Romera, E. Lopez Guillén, R. Barea, and R. Sanz, "Cnn-based fisheye image real-time semantic segmentation," in *Intelligent Vehicles Symposium (IV), 2018 IEEE*. IEEE, 2018, pp. 1039–1044.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *European conference on computer vision*. Springer, 2014, pp. 346–361.

[53] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *null*. IEEE, 2006, pp. 2169–2178.

[54] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[55] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014.

[56] M. P. Shah, "Semantic segmentation architectures implemented in pytorch." *https://github.com/meetshah1995/pytorch-semseg*, 2017.

[57] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[58] F. Yu, V. Koltun, and T. Funkhouser, "Dilated residual networks," in *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[59] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *International Conference on Learning Representations (ICLR)*, 2016.

[60] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," *arXiv preprint arXiv:1702.08502*, 2017.

[61] V. Badrinarayanan, A. Handa, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling," *arXiv preprint arXiv:1505.07293*, 2015.

[62] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[63] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[64] V. Badrinarayanan, B. Mishra, and R. Cipolla, "Understanding symmetries in deep networks," *arXiv preprint arXiv:1511.01029*, 2015.

[65] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[66] S. Urban, J. Leitloff, and S. Hinz, "Improved wide-angle, fisheye and omnidirectional camera calibration," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 108, pp. 72–79, 2015.

[67] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3234–3243.

[68] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*. IEEE, 2014, pp. 420–425.

[69] S. Greaves and A. Somers, "Insights on driver behaviour: what can global positioning system (gps) data tell us?" *Publication of: ARRB Transport Research, Limited*, 2003.

[70] H. Ren, T. Xu, and X. Li, "Driving behavior analysis based on trajectory data collected with vehicle-mounted gps receivers," *Wuhan Daxue Xuebao (Xinxi Kexue Ban)/Geomatics and Information Science of Wuhan University*, vol. 39, no. 6, pp. 739–744, 2014.

[71] I. Y. Wong, "Using gps and accelerometry to assess older adults' driving behaviours and performance: challenges and future directions," 2013.

[72] Q. C. Sun, R. Odolinski, J. C. Xia, J. Foster, T. Falkmer, and H. Lee, "Validating the efficacy of gps tracking vehicle movement for driving behaviour assessment," *Travel Behaviour and Society*, vol. 6, pp. 32–43, 2017.

[73] Y. Gao, Z. Li, and J. McLellan, "Carrier phase based regional area differential gps for decimeter-level positioning and navigation," in *Proceedings of the 10th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 1997)*, 1997, pp. 1305–1313.

[74] A. E. Ragheb and A. F. Ragab, "Enhancement of gps single point positioning accuracy using referenced network stations," *World Applied Sciences Journal*, vol. 18, no. 10, pp. 1463–1474, 2012.

[75] R. Thitipatanapong, P. Wuttimanop, S. Chantranuwathana, S. Klongnaivai, P. Boonporm, and N. Noomwongs, "Vehicle safety monitoring system with next generation satellite navigation: Part 1 lateral acceleration estimation," SAE Technical Paper, Tech. Rep., 2015.

[76] R. M. Alkan, V. İlçi, I. M. Ozulu, and M. H. Saka, "A comparative study for accuracy assessment of ppp technique using gps and glonass in urban areas," *Measurement*, vol. 69, pp. 1–8, 2015.

[77] A. Angrisano, S. Gaglione, and C. Gioia, "Performance assessment of gps/glonass single point positioning in an urban environment," *Acta Geodaetica et Geophysica*, vol. 48, no. 2, pp. 149–161, 2013.

[78] S. Verhagen, D. Odijk, P. J. Teunissen, and L. Huisman, "Performance improvement with low-cost multi-gnss receivers," in *Satellite Navigation Technologies and European Workshop on GNSS Signals and Signal Processing (NAVITEC), 2010 5th ESA Workshop on.* IEEE, 2010, pp. 1–8.

[79] R. Odolinski, P. J. Teunissen, and D. Odijk, "Combined bds, galileo, qzss and gps single-frequency rtk," *GPS solutions*, vol. 19, no. 1, pp. 151–163, 2015.

[80] J. E. Naranjo, C. González, R. García, T. De Pedro, and R. E. Haber, "Power-steering control architecture for automatic driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 4, pp. 406–415, 2005.

[81] S. Rezaei and R. Sengupta, "Kalman filter-based integration of dgps and vehicle sensors for localization," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 6, pp. 1080–1088, 2007.

[82] A. J. Weinstein and K. L. Moore, "Pose estimation of ackerman steering vehicles for outdoors autonomous navigation," in *Industrial Technology (ICIT), 2010 IEEE International Conference on.* IEEE, 2010, pp. 579–584.

[83] S. M. LaValle, *Planning algorithms.* Cambridge university press, 2006.

[84] G. Welch and G. Bishop, "An introduction to the kalman filter. university of north carolina, department of computer science," TR 95-041, Tech. Rep., 1995.

Apéndice A

Datos de Calibración de la Cámara

A.1 Introducción

Este apéndice proporciona los datos de calibración obtenidos para la cámara real adquirida en base al modelo de Scaramuzza mediante la toolbox para gestión de imágenes *Fisheye* proporcionada por MATLAB.

A.2 Matrices de Calibración

Vector de coeficientes de Scaramuzza modificados:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 540{,}75 \\ 0 \\ -8{,}55\text{e}{-4} \\ 8{,}24\text{e}{-7} \\ -1{,}31\text{e}{-9} \end{bmatrix} \tag{A.1}$$

Dimensiones de la imagen:

$$\begin{bmatrix} w \\ h \end{bmatrix} = \begin{bmatrix} 1920 \\ 1080 \end{bmatrix} \tag{A.2}$$

Centro de la distorsión:

$$\begin{bmatrix} c_x \\ c_y \end{bmatrix} = \begin{bmatrix} 996{,}3 \\ 525{,}8 \end{bmatrix} \tag{A.3}$$

Stretch Matrix:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.4}$$

Apéndice B

SmartElderlyCar: Sistema de geoposicionamiento

B.1 Introducción

En paralelo con el desarrollo de la segmentación semántica para las cámaras de ojo de pez, se desarrolló el sistema de geoposicionamiento del vehículo autónomo eléctrico de SmartElderlyCar, el cual resulta crítico para el funcionamiento del coche dado que su navegación está basada en lanelets [68] y a que el receptor GNSS que incorpora constituye la base de la sincronización temporal de todos los sensores, entre las que se incluyen las cámaras.

B.2 Estado del arte

El posicionamiento y seguimiento de vehículos tiene múltiples aplicaciones relacionadas con el ámbito de los sistemas de transporte inteligente, entre las que se incluyen la monitorización de flotas de vehículos mediante sistemas de ayuda a la explotación (SAE), control y estudio del tráfico, localización de vehículos, etc. En la última década, múltiples trabajos han estado centrados en el análisis del comportamiento en la conducción mediante el estudio de la trayectoria de los vehículos aprovechando las señales de los sistemas de geoposicionamiento satelitales globales (GNSS), mayoritariamente GPS [69] [70] [71]. Este conjunto de métodos proporcionan información temporal y de geolocalización del vehículo en tiempo real aprovechando las señales recibidas desde una constelación de satélites de una manera simple, barata y generalmente precisa. Sin embargo, el rendimiento de estos sistemas tiene una dependencia muy fuerte de múltiples factores ajenos al sistema entre los que destacan el número de satélites visibles y su posición en el cielo, las condiciones climatológicas o la calidad de la señal recibida, que se propaga a través de un amplio conjunto de capas de la atmósfera (entre las que destaca la ionosfera) y de un gran conjunto de fuentes de imprecisiones y errores para el sistema. Este es el motivo de que la precisión alcanzada por los sistemas únicamente basados en señales satelitales alcancen precisiones pobres (generalmente entre 1 y 10 metros), requieran un tiempo elevado para comenzar a funcionar (por encima de 30 segundos) y no garanticen un servicio robusto en condiciones de señal pobre como las que se dan en entornos urbanos densos en los que se enfoca este trabajo.

El desarrollo de vehículos inteligentes aspira a solventar problemas complejos con requisitos de funcionamiento especialmente estrictos entre los que se incluye alcanzar un posicionamiento con precisión

sub-métrica. En el caso de los vehículos completamente autónomos este requisito es incluso más restrictivo, dado que las tareas de manutención del vehículo en el carril demandan una precisión centimétrica [72]. Además de precisión, los vehículos autónomos tienen otras necesidades como latencias muy pequeñas a la hora de obtener la posición, alta disponibilidad y buena robustez del servicio con independencia del entorno. Por este motivo, los sistemas únicamente basados en las señales GNSS no son adecuados para aplicaciones con necesidades exigentes.

Varias soluciones han sido propuestas para alcanzar una mejor calidad del servicio: par lidiar con el problema de la pobre precisión se ha recurrido a soluciones que incorporan correcciones diferenciales basadas en la codificación de las señales como DGPS (Differential GPS). En esta solución, una estación base fija y de posición conocida proporciona correcciones diferenciales al receptor para conseguir una mejora de la precisión obtenida [73] [74]. Otras soluciones más elaboradas como Real-Time Kinematic o RTK generan correcciones más complejas basadas en la fase de la portadora para obtener mejores precisiones que alcanzan valores inferiores a los 10 cm [75]. Para afrontar el problema de la robustez se ha recurrido al uso de los denominados sistemas satelitales múltiples de navegación global (Multi-GNSS), que se han visto fuertemente impulsadas por la aparición de constelaciones de satélites alternativas como la rusa (GLONASS), la china (Beidou), la europea (Galileo) o la japonesa (QZSS). Multi-GNSS permite aumentar de manera simple el número de satélites visibles a más de 10 con facilidad superando los 5 en prácticamente cualquier situación, al aprovechar las señales de varias constelaciones y no de una única [72]. Múltiples estudios han demostrado los beneficios del uso de este tipo de métodos combinando varios sistemas satelitales como GPS y GLONASS [76] [77], GPS y Galileo [78] o incluso los 4 sistemas disponibles (GPS+Galileo+BeiDou+QZSS) [79].

Aún incluso ambos conjuntos de soluciones combinados pueden no resultar suficientes para satisfacer las necesidades de los vehículos autónomos en ciertos entornos como los urbanos densos o en situaciones específicas como túneles o zonas cerradas. Para afrontar estos desafíos los sistemas requieren ser complementados con sistemas de distinta naturaleza que sirvan de apoyo cuando la señal satelital se degrada. En [80] un sistema RTK-GPS fue fusionado con los sensores de velocidad y giro de un vehículo para mejorar el seguimiento. Otros trabajos como [81] aprovechaban un filtro de Kalman para integrar los sensores de navegación de un coche con un sistema simple de GPS diferencial con el objetivo de obtener la precisión necesaria para permitir una navegación que evitara las colisiones con otros vehículos sin necesidad de utilizar dispositivos de naturaleza reactiva como lidar o sensores de proximidad.

Inspirados por este conjunto de ideas, para desarrollar el sistema de posicionamiento de SmartElderlyCar se apostó por la integración de un sistema de navegación multi-satelital con solución RTK con un sistema auxiliar de odometría instalado en la parte trasera del coche mediante el uso de un filtro de Kalman extendido (EKF) que, como novedad, utiliza las medidas de calidad proporcionadas por los sensores para ajustar las matrices de covarianza de empleadas en la fusión mediante una función lineal. Los sistemas utilizados han sido testeados de manera independiente y combinada para demostrar la robustez del sistema desarrollado ante diferentes situaciones desafiantes.

B.3 Arquitectura del sistema

El sistema de posicionamiento se integra en el prototipo de vehículo del proyecto (un coche eléctrico modificado y automatizado totalmente a partir de código abierto por la Universidad de Alcalá) de modelo TABBY EVO en su versión de 4 asientos.

La arquitectura del sistema engloba el módulo de navegación multi-satelital con solución RTK, basado en las constelaciones GPS y GLONASS (que utiliza una estación base dedicada para generar correcciones

diferenciales, un receptor GNSS estándar y un módem GPRS para la obtención de las correcciones) y el sistema de odometría basado en en dos encoders incrementales. Estos dos módulos son fusionados mediante las librerías de ROS (Robotic Operating System) utilizando un filtro de Kalman Extendido. La figura B.1 muestra un esquema de la estructura básica del diseño realizado.

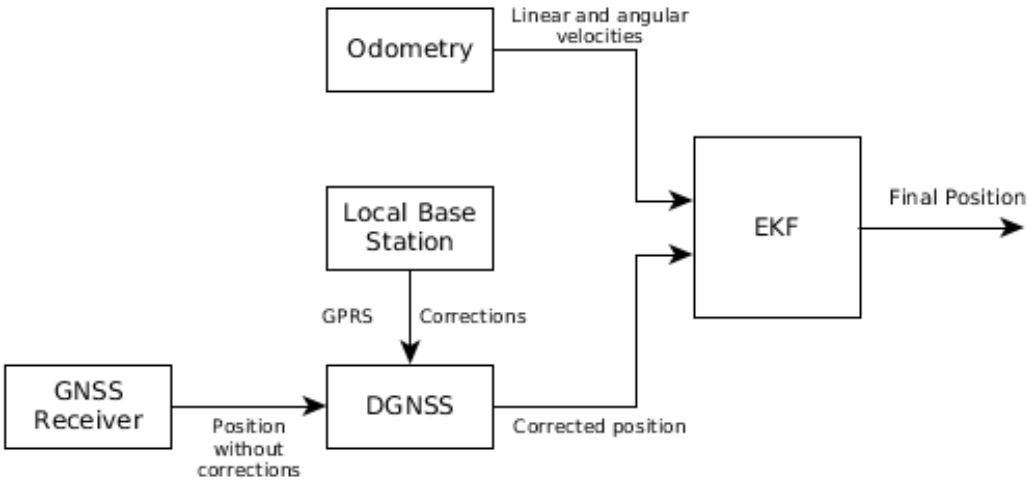


Figura B.1: Arquitectura esquemática del sistema

En cuanto a la implementación real del sistema, el receptor GNSS se sitúa en la parte superior del vehículo para obtener la mejor cobertura y evitar posibles interferencias con el resto de elementos del vehículo. La estación base utilizada se posicionó en el tejado de la Escuela Politécnica, y está basada en una antena de anillos concéntricos (Choke-Ring Antenna), específicamente escogida para lidiar con problemas derivados del multitrayecto, y en otro receptor idéntico al utilizado en el coche que se encarga de generar las correcciones diferenciales. La odometría se encuentra desplegada en ambas ruedas traseras del coche acopladas con l ayuda de dos piezas 3D impresas. ROS corre sobre dos GPUs embebidas (Nvidia Jetson TX2) y en una Raspberry Pi 3 encargada de gestionar el sistema de odometría. La figura B.2 muestra el prototipo del vehículo con los sensores incorporados.

Figura B.2: Vehículo de SmartElderlyCar

B.3.1 Multi-GNSS RTK

El módulo principal del sistema de localización es el GNSS multi-constelación con solución de posicionamiento RTK. Está compuesto por dos elementos principales: un receptor Hiper Pro GPS+ alojado en el vehículo y configurado como *rover* y una estación base fija posicionada en una zona del tejado de la Escuela Politécnica con buena visibilidad del Campus.

El rover está configurado para aprovechar la información de las constelaciones GLONASS y GPS con objetivo de mejorar la robustez y la precisión del sistema, mediante el incremento del número de satélites visibles aprovechables. Proporciona información sobre el posicionamiento a 10 Hz, dado que los vehículos autónomos requieren información en tiempo real. Además, el receptor utiliza el modo diferencial RTK, con objetivo de mejorar la precisión final obtenida. El dispositivo se comunica con el ordenador central del vehículo mediante 2 puertos serie configurados a 115200 baudios: uno para recibir las correcciones diferenciales proporcionadas por el propio ordenador y otro para devolver las medidas generadas utilizando cadenas NMEA que son interpretadas mediante un programa implementado en C++. Una vez extraída la información de las cadenas, ésta es publicada en ROS para ser fusionada con la información de la odometría.

La estación base utilizada en el sistema fue diseñada con objetivo de solventar las carencias presentadas por el despliegue de estaciones base para sistemas de posicionamiento del Instituto Geográfico Nacional (IGN):

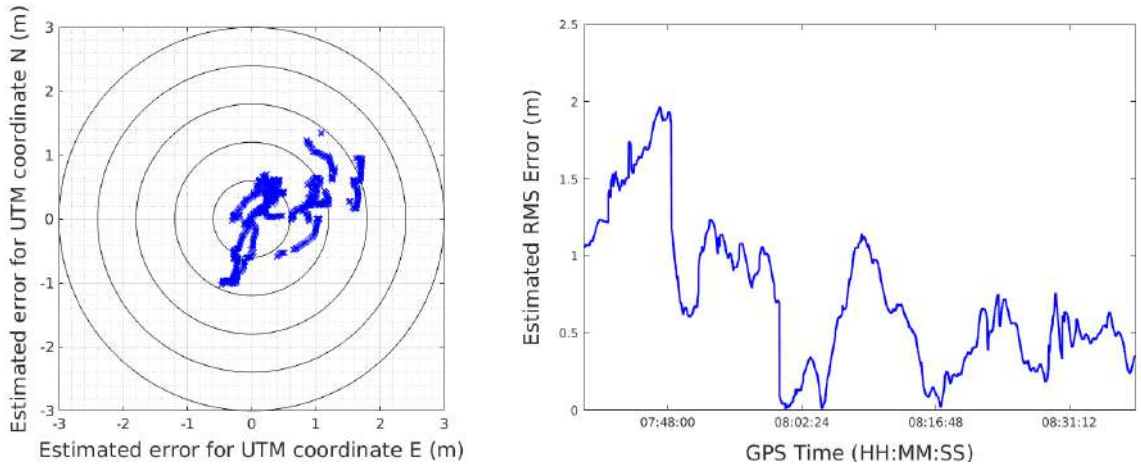
- **Frecuencia insuficiente**: las estaciones públicas de referencia de este organismo proporcionan correcciones a una frecuencia de 1 Hz, lo que resulta insuficiente para aplicaciones de navegación. Además, las estaciones presentan habitualmente periodos de mal funcionamiento en los que la publicación de las correcciones se ve retrasada debido a la saturación del servicio.
- **Escaso número de estaciones**: las correcciones proporcionadas por IGN sólo aseguran precisiones centimétricas cuando el rover se encuentra a una distancia cercana a la estación (inferior a unos 20 km). Dada la baja densidad de estaciones desplegadas, esta condición se da en pocas regiones del territorio nacional y en el entorno de pruebas del vehículo (el campus de la UAH) la condición no se satisface dado que no existe ninguna estación cercana (siendo la más cercana la de Yebes situada a 80 km).
- **Compatibilidad**: para obtener la mejor precisión posible es necesario que la estación base utilizada presente una configuración similar a la del rover, para que las correcciones diferenciales recibidas sean compatibles. Esto implica que la estación base debe utilizar GPS y GLONASS y el protocolo RTCM en su versión 3.0.

La estación base emplea una antena de anillos concéntricos, capaz de eliminar la mayoría de señales recibidas producto del multitrayecto. La antena se conecta con otro receptor Hiper Pro GPS+ situado en el interior del edificio mediante un cable coaxial de cobre. Este segundo receptor genera las correcciones diferenciales que son transmitidas a un servidor, aprovechando un puerto serie, donde son publicadas en Internet utilizando software de código abierto. Para obtener las correcciones diferenciales en el ordenador del vehículo se utiliza un módem GPRS. La figura B.3 muestra una imagen del receptor GNSS junto con el lidar del vehículo.

El modulo principal del sistema de localización (rover y estación base) fue testeado por separado para evaluar la precisión que es capaz de proporcionar en condiciones idóneas. En un primer experimento, se probó la precisión que el sistema es capaz de obtener sin el uso de correcciones diferenciales aprovechando una antena auxiliar de la Escuela Politécnica de posición conocida. Para ello, se grabaron datos durante

Figura B.3: Receptor GNSS y lidar

una hora (36000 mediciones en total) que, posteriormente, fueron comparados con la posición real de la antena. La figura B.4 muestra los resultados del experimento. El primer gráfico B.4a ilustra la desviación de las coordenadas UTM de las mediciones con respecto a la posición real en metros, mientras que la segunda figura B.4b muestra el error cuadrático medio cometido en las medidas como función del tiempo GPS en el que fueron realizadas.



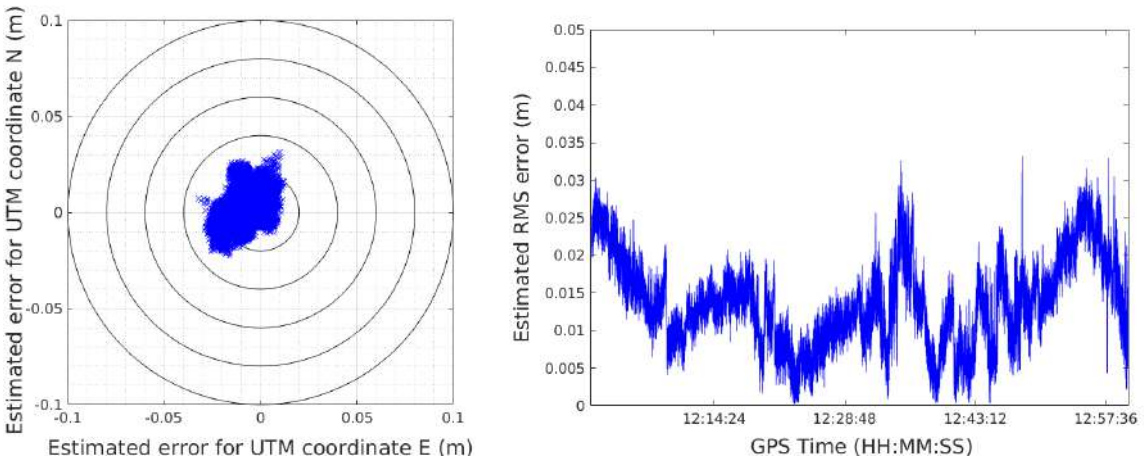
(a) Error estimado en coordenadas UTM (m) (b) Error RMS para las mediciones realizadas (m)

Figura B.4: Desempeño del sistema con configuración *standalone*

Los resultado mostrados claramente reflejan la incapacidad del sistema para satisfacer las necesidades de un vehículo autónomo con una configuración de tipo *standalone* (sin correcciones externas). La precisión media obtenida es de 71 cm, siendo el valor alcanzado inferior a un metro sólo en el 72 % de los casos. Con esta configuración, la precisión que el sistema es capaz de garantizar durante el 95 % del tiempo (estándar GPS) es de 1,7 metros. Adicionalmente, el sistema muestra una clara falta de repetibilidad al variar los resultados obtenidos según el momento de la evaluación.

En un segundo experimento, se añadieron correcciones diferenciales al receptor (esperando a que éste

completara el proceso de resolución de la ambigüedad que garantiza una solución óptima) y se repitió el experimento inicial para evaluar la precisión de una solución RTK-fija.



(a) Error estimado en coordenadas UTM (m) (b) Error RMS para las mediciones realizadas (m)

Figura B.5: Desempeño del sistema con correcciones diferenciales

La figura B.5 muestra la clara mejora que supone la incorporación de correcciones diferenciales adecuadas: la precisión media alcanzada se ha reducido hasta 1,3 cm con una desviación máxima de 3,4 cm, garantizando precisión centimétrica el 100 % del tiempo y obteniendo repetibilidad.

El proceso de resolución de la ambigüedad es otro elemento crítico en el rendimiento del sistema, ya que la precisión obtenida se ve degradada sin una solución RTK fija. Alcanzar dicha solución requiere un análisis complejo y costoso que puede, incluso, no resultar resoluble debido a mala calidad de las señales satelitales o a la visibilidad de un número insuficiente de satélites. La figura B.6 presenta el tiempo requerido para alcanzar la solución óptima basado en un conjunto de mediciones realizadas a diario durante un periodo de dos meses en condiciones favorables (de cielo abierto).

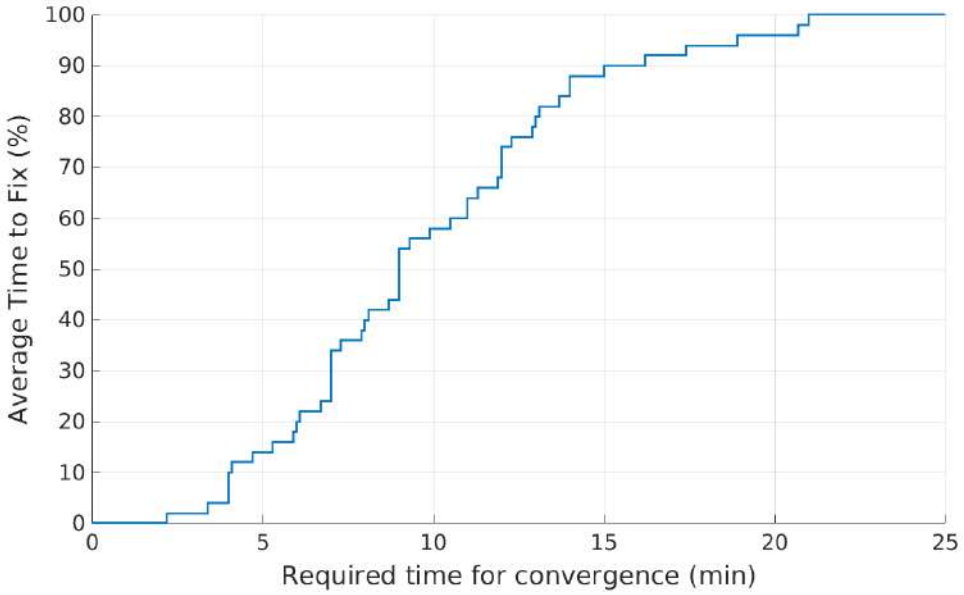


Figura B.6: Tiempo de convergencia para una solución RTK-fija

Los resultados presentados muestran que el tiempo medio (50 %) requerido para converger a una

solución por el sistema supera los 9 minutos. El 10 % de las mediciones necesitaron incluso más de 15 minutos para alcanzar precisión centimétrica aún con buena visibilidad de satélites. Los datos de los tiempos de convergencia remarcan la necesidad de utilizar sistemas complementarios que mejoren el funcionamiento del sistema principal cuando la solución óptima no se encuentra disponible.

B.3.2 Sistema de odometría incremental

El sistema de odometría implementado está basado en encoders incrementales que registran la rotación de las ruedas. Cada rueda tiene su propio eje, lo que permite que el ángulo de rotación o giro del vehículo sea medido. El sistema se encarga de registrar las velocidades linear y angular para el filtro de Kalman extendido. Sin embargo, la posición relativa calculable mediante la odometría no es proporcionada al filtro debido a los malos resultados que inducen ciertas causas externas como las irregularidades de la carretera que desembocan en un error acumulativo grande. Este error se puede evitar proporcionando únicamente las velocidades instantáneas de las ruedas. La figura B.7 muestra un detalle del sistema de odometría incorporado en las ruedas del vehículo.

Figura B.7: Encoders incorporados en la rueda mediante pieza 3D impresa

El sistema está compuesto por dos módulos: un contador de pulsos en tiempo real y un procesador del algoritmo. La captura de pulsos es una tarea crítica, dado que su pérdida implica también la del control del la medida y la introducción de un error sistemático. El contador se ha implementado mediante un Olimexino-STM32, que proporciona los datos adquiridos a una Raspberry Pi 3 que realiza los cálculos de las velocidades lineares y angulares a partir de la diferencia entre los tiempos de llegada de los pulsos. La comunicación entre ambos módulos se lleva a cabo mediante un puerto serie.

Los cálculos de las dos velocidades se referencian al punto central entre los ejes de ambas ruedas, siguiento un modelo de Ackerman [82] como se muestra en la figura B.8.

La velocidad lineal V_{avrg} es calculada mediante la ecuación B.1 como la velocidad media de las velocidades lineales de ambas ruedas donde V_R y V_L representan las velocidades lineales de las ruedas derecha e izquierda respectivamente.

$$V_{avrg} = \frac{V_R + V_L}{2} \tag{B.1}$$

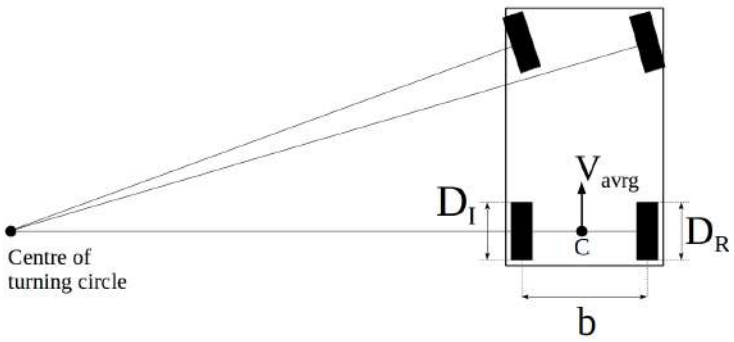


Figura B.8: Diagrama del vehículo y modelo geométrico Ackerman

Por otro lado las velocidades lineales de ambas ruedas son calculadas mediante las ecuaciones B.2 y B.3.

$$V_R = \frac{N_R D_R \pi}{PT} \tag{B.2}$$

$$V_L = \frac{N_L D_L \pi}{PT} \tag{B.3}$$

donde:

N_R número de pulsos registrados en la rueda derecha

N_L número de pulsos registrados en la rueda izquierda

D_R diámetro de la rueda derecha

D_L diámetro de la rueda izquierda

P resolución del encoder (en pulsos por revolución de la rueda)

T tiempo entre iteraciones en el cálculo

La velocidad angular ω_{avrg} se obtiene por medio de la ecuación B.4 como una derivada lineal del ángulo de rotación θ en cada iteración de los cálculos. Este ángulo se calcula según la ecuación B.5.

$$\omega_{avrg} = \frac{\theta}{T} \tag{B.4}$$

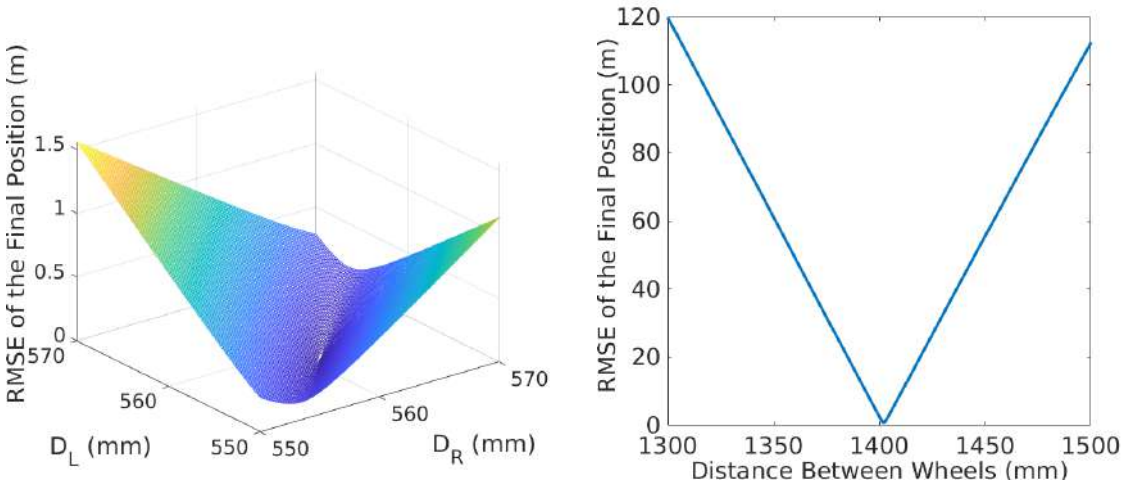
$$\theta = \frac{(N_R D_R - N_L D_L)\pi}{bP} \tag{B.5}$$

donde b representa la distancia entre los ejes de ambas ruedas.

El ángulo es calculado de este modo como consecuencia de la manera de modelar el vehículo mediante la geometría de Ackerman que convierte cualquier movimiento en un una curva. A pesar de presentar carencias, este modelo es apropiado para velocidades reducidas [83].

Para la calibración del sistema se implementó un proceso automatizado que analiza diferentes rutas predeterminadas realizadas por el vehículo (1 recta de 10 o 100 metros y otra ruta más compleja que incluye curvas) para eliminar errores sistemáticos en las trayectorias calculadas. Estas incertidumbres nacen de pequeñas variaciones en el diámetro de las ruedas (que no es fijo debido a la contracción y dilatación de las ruedas como causa de la variación de la temperatura) y de la separación de los ejes. En las rutas realizadas, la evolución de la posición es analizada y, dado que la trayectoria real es conocida, los parámetros reales D_R, D_L y b pueden ser calculados mediante un proceso de ajuste de mínimos cuadrados que minimice el error cometido al estimar la posición al final de las rutas.

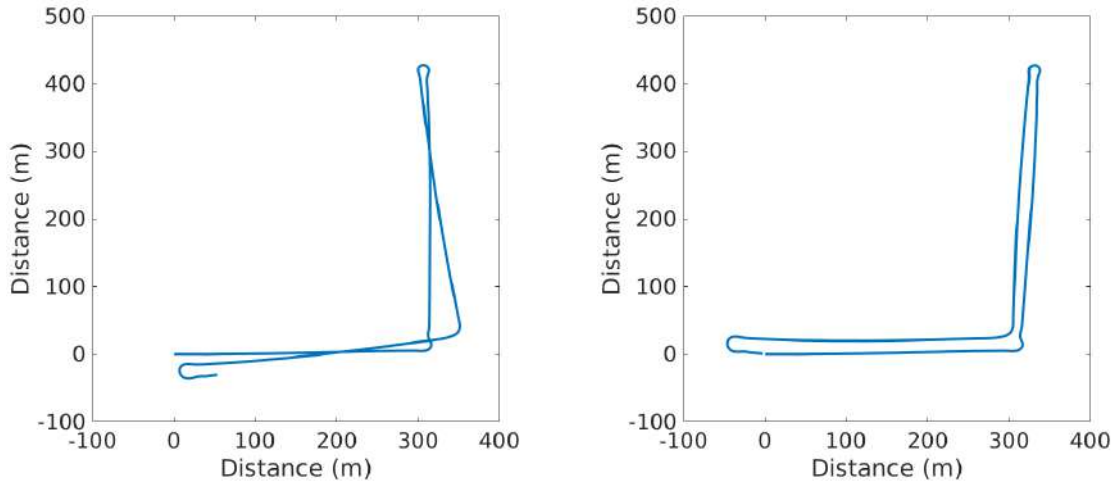
Para la determinación de los parámetros se comienza realizando el cálculo de las dimensiones reales de las ruedas mediante el análisis de los valores obtenidos para la ruta corta. A partir de ellos se realiza una evaluación del error cometido para los posibles valores de diámetros de las ruedas. Estos valores permiten generar una superficie a través de la cual se pueden determinar los valores de las dimensiones reales de las ruedas mediante la localización del mínimo absoluto de la curva. Esta curva aparece en la figura B.9a, en la que los valores D_R=560.4 mm and D_L=558 mm determinan el mínimo absoluto. El uso de una recta corta asegura que no se produzca ningún tipo de giro, lo que garantiza que el parámetro b no influya en el recorrido y que, por tanto, no importe que éste no se encuentre todavía calibrado.



(a) Superficie de calibración del diámetro (b) Calibración del parámetro b

Figura B.9: Calibración del sistema de odometría

Una vez calculados los diámetros de la rueda, se analiza una ruta más compleja que comienza y finaliza en la misma posición y que incluye curvas para calcular la distancia exacta entre las ruedas. La figura B.10a muestra los resultados de la ruta compleja con odometría sin calibrar y la figura B.10b muestra la ruta para el valor de b que minimiza el error final cometido en la estimación de la posición.



(a) Ruta sin calibración (b) Ruta con odometría calibrada

Figura B.10: Calibración del sistema de odometría

B.4 Filtro de Kalman extendido

El algoritmo y fórmula estándar de los filtros de Kalman extendidos son ampliamente conocidos [84]. La salida de estos filtros se obtiene en base a las matrices de covarianza proporcionadas por cada una de las medidas de los sensores utilizados. En una primera aproximación, las matrices básicas proporcionadas por los sensores fueron utilizadas. Sin embargo, esta solución no resultó adecuada dado que ciertos indicadores de calidad son ignorados por las matrices proporcionadas por los sensores. Por ejemplo, el receptor GNSS incorpora elementos como la dilución de la precisión horizontal o HDOP de las medidas que supone un buen indicador de calidad, pero ignora otros más relevantes como la presencia de correcciones diferenciales y el tipo de solución alcanzada.

Por este motivo, se realizó el diseño de un filtro de Kalman extendido adaptativo que aprovecha todos los parámetros de calidad proporcionados por los sensores con el objetivo de proporcionar una salida robusta con buena capacidad de adaptación a las condiciones del entorno.

B.4.1 Algoritmo

El filtro diseñado persigue obtener la posición y velocidades 3D completas en el tiempo a partir de la información provista por los sensores. El vector de estado del vehículo es un vector de 6 elementos que agrupa la posición en 3D del vehículo y la velocidad. Éste es calculado mediante la ecuación B.6 donde f representa una función de transición no lineal y w_{k-1} es el ruido asociado al proceso.

$$x_k = f(x_{k-1}) + w_{k-1} \tag{B.6}$$

Adicionalmente, cada sensor empleado proporciona mediciones que pueden ser modeladas mediante B.7:

$$z_k = h(x_k) + v_k \tag{B.7}$$

donde h es una función de transición de estado no lineal asociada al sensor y v_k es el ruido asociado a la medida que se asume está distribuido según una distribución gaussiana.

La etapa de predicción se describe mediante las ecuaciones B.8 y B.9, donde un modelo cinemático 3D estándar basado en la mecánica newtoniana es utilizado como f. F es calculado como el jacobiano de f y es utilizado para proyectar el error de la covarianza P y, finalmente, Q es el ruido asociado a la covarianza procesada.

$$\hat{x} = f(x_{k-1}) \tag{B.8}$$

$$\hat{P_k} = FP_{k-1}P^T + Q \tag{B.9}$$

La etapa de corrección se lleva a cabo mediante las ecuaciones B.10, B.11 y B.12. La primera de ellas calcula la ganancia de Kalman utilizando las matrices de medida proporcionadas por los sensores H, la covarianza de las mediciones y las covarianzas de error estimadas $\hat{P_k}$. Esta ganancia es utilizada para actualizar el vector de estado final y la matriz de covarianza.

$$K = \hat{P_k}H^T(H\hat{P_k}H^T + R)^{-1} \tag{B.10}$$

$$x_k = \hat{x_k} + K(z - H\hat{x_k}) \tag{B.11}$$

$$P_k = (I - KH)\hat{P_k}(I - KH)^T + KRK^T \tag{B.12}$$

Las matrices de covarianza asociadas a las mediciones proporcionadas a los sensores son reajustadas aprovechando los indicadores de calidas del receptor GNSS: el tipo de medida realizada en función de la solución disponible, la dilución horizontal de la precisión (HDOP) y el número de satélites visibles. En un primer paso, las matrices son actualizadas en base al tipo de medida. Para la existencia de correcciones diferenciales y de una solución óptima (fix 4) el módulo principal es capaz de alcanzar precisión centimétrica por si solo, por lo que su matriz de covarianza se ajusta para ser fuertemente priorizada frente a la de la odometría. Para la existencia de correcciones diferenciales pero sin la resolución completa del proceso de ambigüedad (fix 5) la precisión alcanzada es sub-métrica pero no resulta suficiente por lo que las matrices de covarianza es actualizada mediante una función lineal dependiente del HDOP. Finalmente, para la ausencia de correcciones diferenciales (fix 1), el receptor GNSS no asegura una precisión inferior a 1 metro, por lo que su matriz de covarianza es penalizada en beneficio de la odometría, mediante una función lineal similar a la del caso anterior.

B.5 Resultados

La etapa de test del sistema de posicionamiento desarrollado se situó en el campus externo de la UAH, en un área de en torno a 4 km de radio desde la estación base utilizada para generar las correcciones con lo que se garantiza que éstas sean óptimas. La ruta de pruebas principal presentó una longitud de en torno a 5,5 km con zonas de 2 y de 4 carriles que incluían situaciones desafiantes para los vehículos autónomos como rotondas, pasos de cebra, situaciones de STOP y de ceda el paso. Además, el recorrido presentaba condiciones desafiantes para las señales del receptor GNSS (árboles, edificios altos, tendido eléctrico, señales de tráfico, etc).

A continuación, se presentan un conjunto de datos recogidos en la ruta utilizando diferentes configuraciones que muestran la respuesta del sistema ante situaciones reales desafiantes. En todas las figuras la estela azul representa la señal del receptor GNSS, la amarilla el camino trazado por la odometría y la roja la salida final generada por el filtro de Kalman extendido.

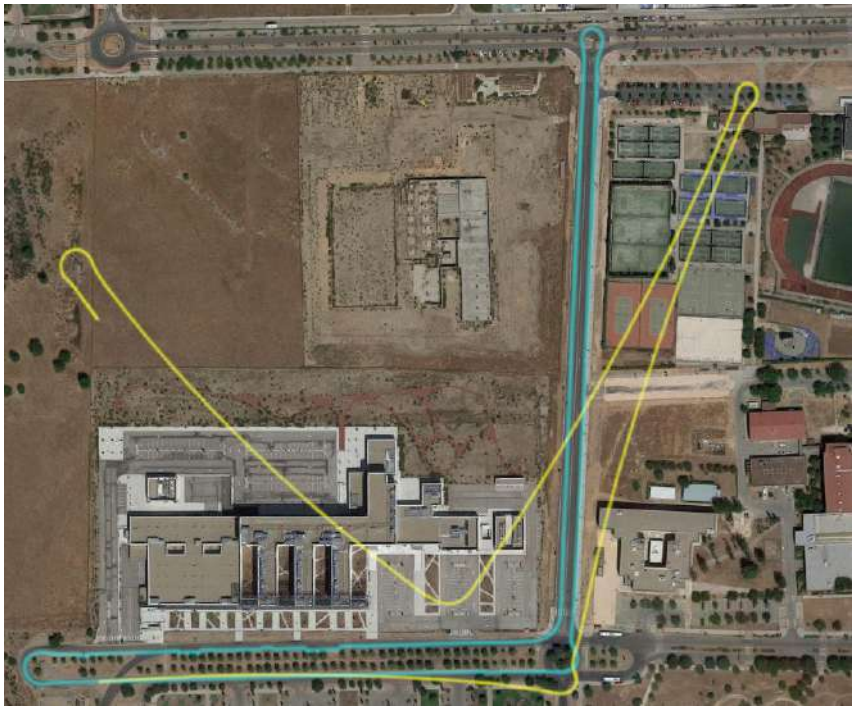
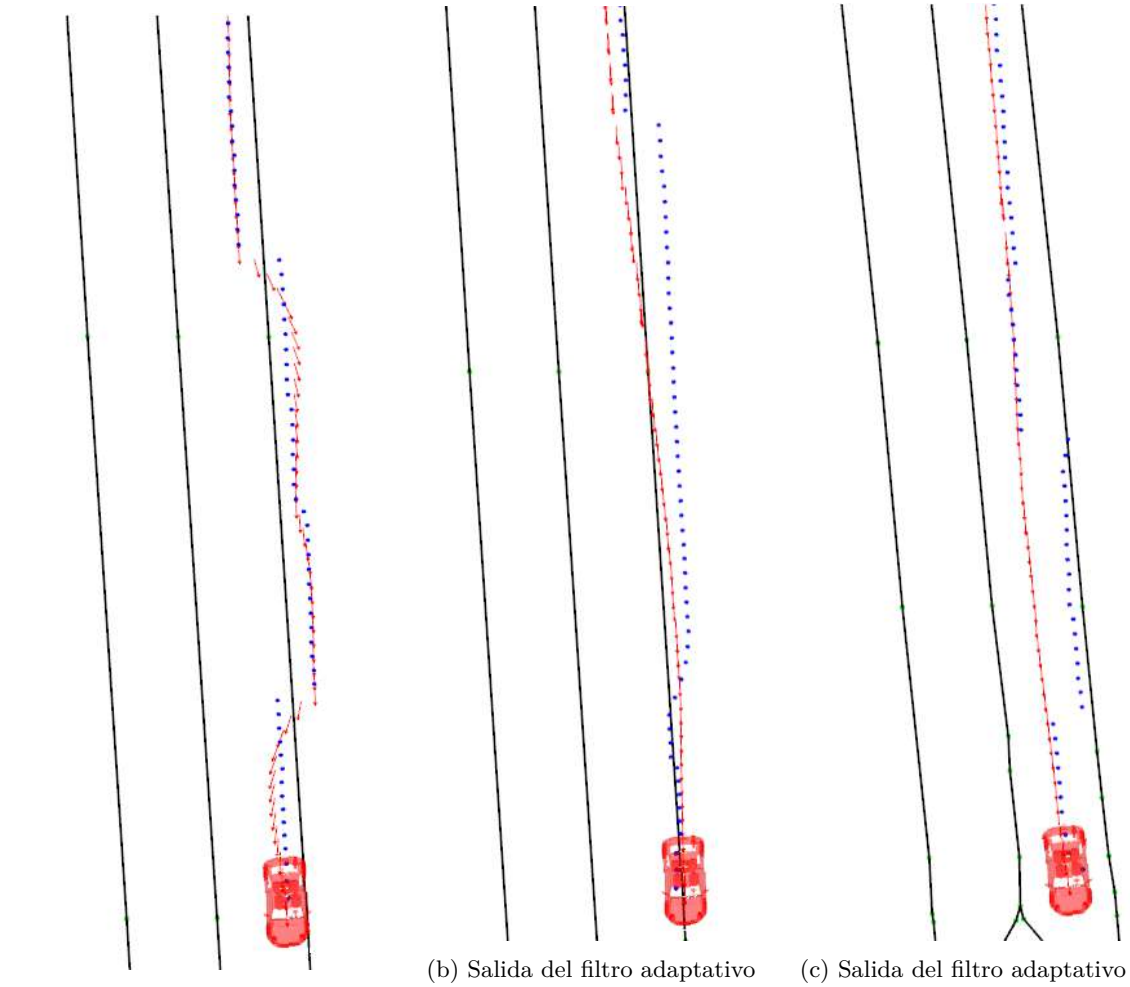


Figura B.11: Estela GNSS (azul) y trazado de la odometría (amarillo)

La primera figura B.11 presenta los datos recolectados por los sensores a través de la sección principal de la ruta. Durante dicha ruta, el receptor GNSS alcanzó una solución óptima durante la mayor parte del tiempo (RTK-Fixed), pero las correcciones diferenciales se perdieron durante varios tramos del recorrido debido a la propagación multi-trayecto de las señales. A pesar de el uso de una odometría sin calibración, se puede observar como claramente la salida del filtro mostrada en la figura B.12 se ve mejorada por la fusión de los sensores mediante el filtro, actuando de forma robusta ante la pérdida de las correcciones.

Figura B.12: Adapted EKF output

La posterior figura B.13 presenta diferentes detalles de secciones del recorrido anterior para distintas configuraciones del sistema, en las que se puede ver cómo la fusión de los sensores es beneficiosa para la posición final estimada. En las figuras los puntos azules representan la traza GNSS y la línea roja la salida del EKF. La traza de la odometría no aparece pero se encuentra calibrada en todos los recorridos. La primera de las figuras incluidas B.13a muestra la respuesta del filtro cuando éste se aplica sin la etapa adaptativa. Los resultados mostrados prueban que la configuración básica es insuficiente para afrontar situaciones reales complejas, dado que el sistema falla y la salida del filtro sale del carril cuando las señales GNSS se pierden o se degradan. La figura B.13b muestra una situación más favorable en la que el receptor GNSS se configura para funcionar en modo *standalone* pero el filtro utilizado pasa a ser adaptativo. Los resultados se ven claramente mejorados, incluso con peores condiciones de señal GNSS. A pesar de ello, el sistema todavía no es capaz de mantener el vehículo dentro del carril dado que la influencia de la señal GNSS sigue siendo excesiva. La última figura B.13c presenta la salida del filtro para su versión adaptativa con solución de posicionamiento RTK fija, que responde correctamente a la pérdida de la señal GNSS y mantiene la trayectoria real del vehículo en la salida del filtro.



(a) Salida del filtro no adaptativo

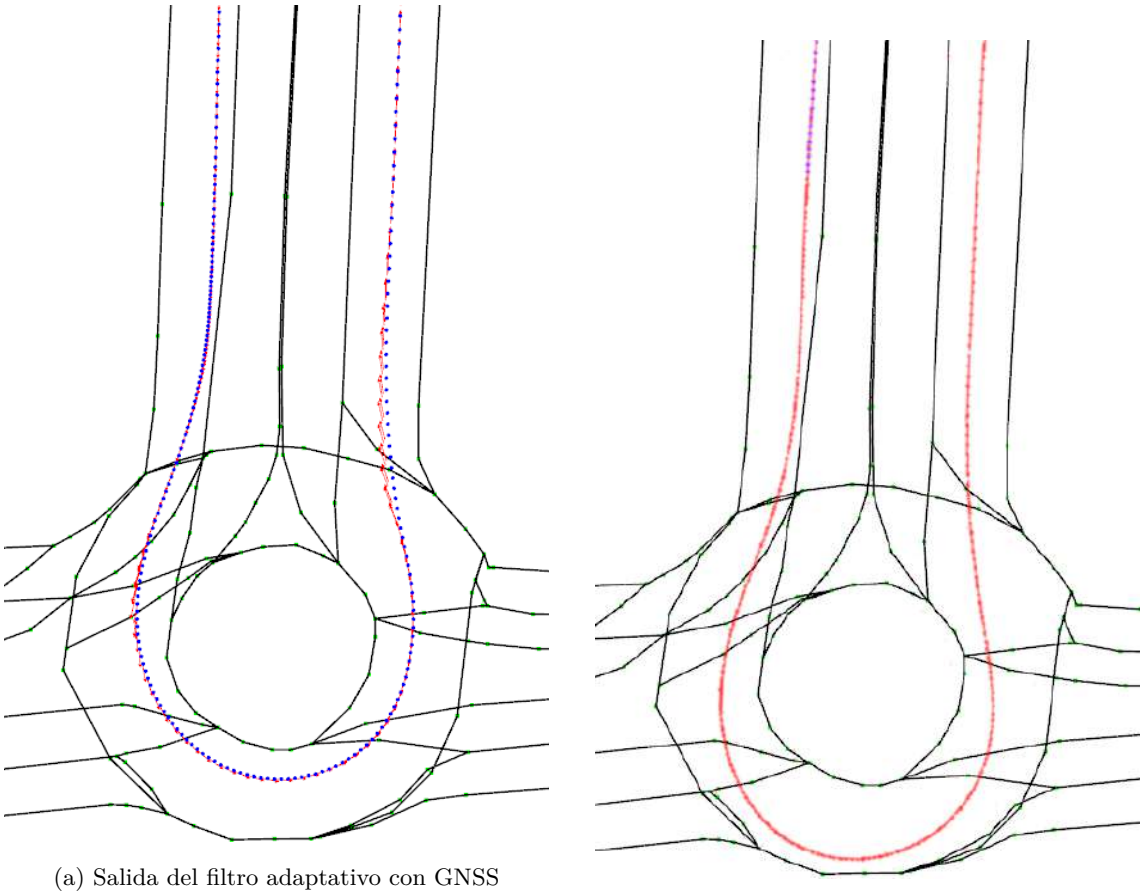
(b) Salida del filtro adaptativo con GNSS *standalone*

(c) Salida del filtro adaptativo con GNSS óptimo

Figura B.13: Ejemplo de robustez del EKF

La siguiente figura B.14a muestra la respuesta del sistema para una rotonda, que supone una situación desafiante para la odometría dado que implica un giro agresivo. La primera sub-figura B.14a muestra la salida del sistema para la fusión de sensores completa, mientras que la segunda sub-figura B.14b muestra únicamente la ruta determinada por la odometría, que se encuentra claramente desviada debido al giro brusco realizado. La fusión de sensores, aún sin una solución GNSS óptima, mejora notablemente la salida final del filtro y ubica correctamente al vehículo en la zona central de la rotonda.

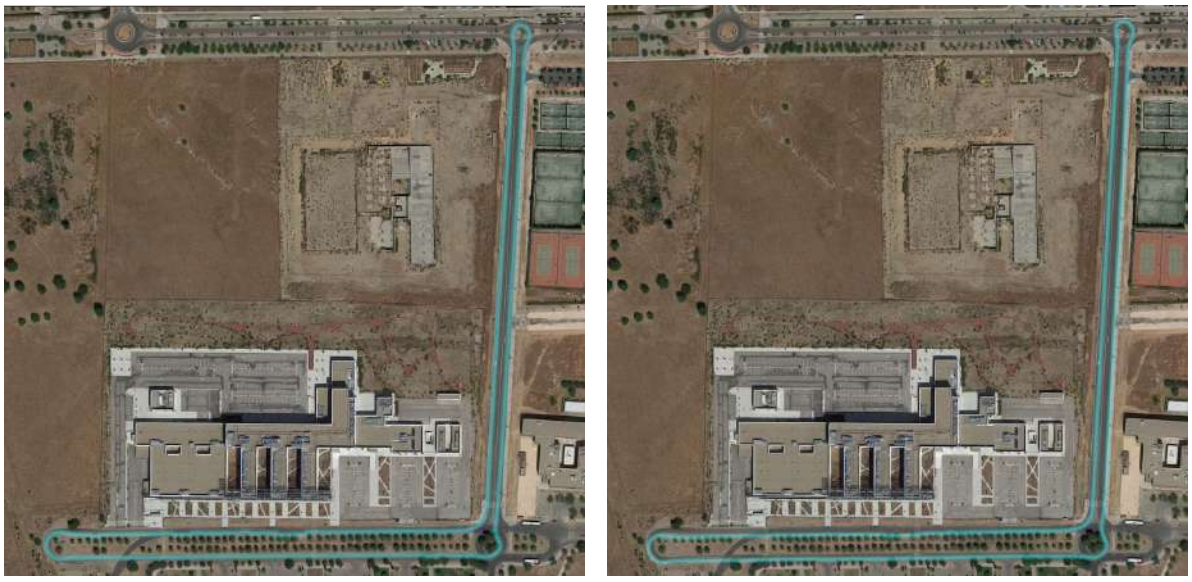
Las figuras B.15a y B.15b exponen una comparativa del recorrido trazado por el módulo principal del sistema configurado para funcionar con y sin correcciones diferenciales respectivamente. Como se puede observar en las figuras, la precisión alcanzada en el primer caso es mucho mayor. En ella el recorrido trazado sigue exactamente el centro de la carretera, mientras que para el caso en el que no existen correcciones diferenciales la estela de la ruta sale de la carretera en múltiples ocasiones y, en la mayoría del camino, no se ubica en el centro de la carretera.



(a) Salida del filtro adaptativo con GNSS *standalone*

(b) Salida determinada por la odometría

Figura B.14: Mejora sobre los resultados de la odometría



(a) Ruta con calidad Fix 4 (óptima)

(b) Ruta con calidad Fix 1 (sin correcciones)

Figura B.15: Comparativa performance GNSS