

UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA, CIENCIA Y COMPUTACIÓN (STEAM)

GRADO EN INGENIERIA EN SISTEMAS INDUSTRIALES

TRABAJO FIN DE GRADO

DISEÑO DE UN SISTEMA DE PERCEPCIÓN INTELIGENTE PARA LA GESTIÓN EFICIENTE DE APARCAMIENTOS AL AIRE LIBRE (CASO DE ESTUDIO: APARCAMIENTO DE LA UEM)

Alumno: RAFAEL DEL CASTILLO SHEVELEV

Director: GUSTAVO BERMEJO MARTÍN

JULIO 2026

TÍTULO: DISEÑO DE UN SISTEMA DE PERCEPCIÓN INTELIGENTE PARA LA GESTIÓN EFICIENTE DE APARCAMIENTOS AL AIRE LIBRE (CASO DE ESTUDIO: APARCAMIENTO DE LA UEM)

AUTOR: RAFAEL DEL CASTILLO SHEVELEV

DIRECTOR DEL PROYECTO: GUSTAVO BERMEJO MARTÍN

FECHA: 05 de julio de 2026

RESUMEN

En este proyecto se diseña un sistema de percepción inteligente para mejorar la gestión de aparcamientos al aire libre, tomando como caso de estudio el aparcamiento del campus de la Universidad Europea de Madrid. El sistema busca resolver la ineficiencia en la búsqueda de plazas, que genera pérdida de tiempo y mayores emisiones. Para ello, se propone una solución basada en visión artificial capaz de estimar la ocupación de las plazas en tiempo real y detectar incidencias.

Se estudian las soluciones existentes, se realiza trabajo de campo para entender las necesidades del caso de estudio y establecer los requisitos, y se valida el funcionamiento con vídeos reales del aparcamiento. El sistema desarrollado se basa en un detector YOLO entrenado con imágenes seleccionadas, junto con un método de asociación entre vehículos y plazas, considerando los retos que tiene un entorno exterior para un sistema de visión. Se plantea una arquitectura modular y escalable y se propone el despliegue con sus planos, orientándose a reutilizar infraestructura existente y reducir el coste frente a otras soluciones.

Palabras clave: Aparcamiento inteligente, Visión artificial, Detección de ocupación, YOLO, Ajuste fino, Despliegue del sistema.

ABSTRACT

This project consists on the design of an intelligent perception system to improve the management of outdoor *parking* lots, using the Universidad Europea de Madrid campus *parking* as a case study. The system is intended to resolve the inefficiency involved in searching for available *parking* spaces, which leads to wasted time, unnecessary vehicle circulation and increased emissions. To achieve this, a computer vision based solution is proposed, capable of estimating *parking* space occupancy in real time and detecting relevant *parking* incidents.

Existing solutions are analysed, fieldwork is carried out to understand the needs of the case study and define the system requirements, and the proposed approach is validated using real video recordings from the *parking* lot. The developed system is based on a YOLO detector finetuned with selected images, combined with a method to associate vehicles with *parking* spaces, designed to handle the challenges of outdoor environments, such as perspective, occlusions, lighting variations and distant vehicles. Finally, a modular and scalable architecture is proposed, together with a deployment plan for the campus, focusing on the reuse of existing infrastructure and on reducing implementation costs compared with other available solutions.

Key words: Smart parking, Computer vision, Occupancy detection, YOLO, Fine tuning, System deployment.

Índice

RESUMEN 3
ABSTRACT 3
Capítulo 1. Introducción 14
1.1 Contexto y motivación 14
1.2 Planteamiento del problema y justificación 16
1.3 Objetivos 17
1.3.1 Objetivo general 17
1.3.2 Objetivos específicos 17
1.4 Alcance 19
1.5 Metodología 19
1.6 Planificación 20
Capítulo 2. Marco teórico 22
2.1 Sistemas de Smart *parking* 22
2.1.1 Conceptos y funciones de los sistemas de *Smart parking* 22
2.1.2 Componentes principales de un sistema de *smart parking* 24
2.1.3 Métricas de evaluación de sistemas de Smart *Parking* 26
2.2 Visión artificial aplicada a la detección de ocupación en aparcamientos 28
2.2.1 Enfoques para detectar la ocupación de aparcamientos 28
2.2.2 Modelos de detección, entrenamiento y evaluación 29
2.2.3 Retos de la visión en aparcamientos al aire libre 31
2.2.4 Geometria de captación 32
2.3 Arquitectura de procesamiento del sistema 34
2.3.1 Procesamiento en la nube 34
2.3.2 Procesamiento en el borde (*edge computing*) 35
2.3.3 Arquitecturas híbridas 35
2.4 Bases hardware y tecnológicas del sistema 35
2.4.1 Cámaras y captación de imagen 36

2.4.2 Plataformas embebidas de procesamiento ... 38
2.4.3 Comunicación e integración del sistema ... 40
2.4.4 Consideraciones energéticas y de despliegue ... 42
2.5 Herramientas y software empleados ... 43
2.5.1 Lenguaje y entorno de desarrollo ... 43
2.5.2 Librería para visión clásica ... 43
2.5.3 Librerías de aprendizaje profundo ... 44
2.5.4 Modelos que se van a probar y comparar ... 44
2.5.5 Herramientas de apoyo ... 45
2.6 Privacidad, normativa y ética ... 45
2.6.1 Marco normativo aplicable ... 45
2.6.2 Tratamiento de imágenes y datos personales ... 46
2.6.3 Principios aplicables al diseño ... 46
2.6.4 Medidas técnicas y organizativas ... 47
2.6.5 Consideraciones éticas ... 48
Capítulo 3. Estado del arte y análisis de soluciones existentes ... 49
3.1 Soluciones comerciales actuales ... 49
3.1.1 Cleverciti ... 49
3.1.2 Parquery ... 50
3.1.3 Milesight (cámaras AI específicas para *parking*) ... 50
3.1.4 Urbiotica ... 51
3.1.5 Aspectos a tener en cuenta sobre los datos comerciales y costes ... 52
3.2 Propuestas académicas relevantes ... 53
3.2.1 Almeida et al. (2015) y Amato et al. (2017), *datasets* de referencia ... 53
3.2.2 Liu et al. (2023): transformación de perspectiva aplicada al *parking* ... 53
3.2.3 Encío et al. (2023), información contextual y red multi-rama ... 53
3.2.4 Jabbar et al. (2021): smart *parking* en campus universitario con Raspberry Pi ... 54
3.2.5 Kim et al. (2025): YOLO en *edge* con asociación plaza-vehículo ... 55
3.3 Comparación de enfoques tecnológicos ... 55
3.3.1 Limitaciones detectadas ... 56
3.3.2 Oportunidad de mejora para este trabajo ... 57
Capítulo 4. Análisis de alternativas, trabajo de campo y desarrollo del sistema de visión ... 58

4.1 Trabajo de campo, necesidades y requisitos para el *parking* de la UEM 58
4.2 Desarrollo del sistema de Visión 59
4.2.1 Definición de plazas y regiónes de interés 59
4.2.2 Visión clásica como primera aproximación 61
4.2.3 Construcción del *ground truth* 63
4.2.4 Clasificación de plazas según calidad 65
4.2.5 Detección de vehículos, primera aproximación para análisis 66
4.2.6 Método de asociación vehículo-plaza 67
4.2.7 Selección de configuración 77
4.2.8 Selección de modelo 84
4.2.9 Mejora del detector mediante *fine tuning* 90
4.2.10 Integración y validación del sistema completo 106
4.2.11 Frecuancia de funcionamiento y coste computacional 110
4.2.12 Estabilización temporal de la ocupación 113
4.2.13 Detección de incidencias de estacionamiento 116
4.3 Conclusiones del desarrollo de visión 116
Capítulo 5. Diseño e implementación del sistema y propuesta de despliegue 118
5.1 Funcionalidades del sistema 118
5.1.1 Gestión de la ocupación 118
5.1.2 Detección y registro de incidencias 119
5.1.3 Control del nivel de ocupación y de la barrera 119
5.1.4 Información para seguridad y para los conductores 120
5.2 Diseño de interfaz de usuario 121
5.2.1 Objetivo y herramienta utilizada 121
5.2.2 Pantalla principal y visualización 121
5.2.3 Control de barrera y de incidencias 122
5.3 Integración y funcionamiento del programa 123
5.4 Demostración y validación funcional del prototipo 125
5.5 Despliegue y cobertura de las cámaras sobre el aparcamiento 126
5.5.1 Análisis de reutilización de cámaras 126
5.5.2 Despliegue completo 127
5.6 Procesamiento y hardware 132

5.7 Comunicaciones, alimentación, señal y elementos del sistema 134
5.8 Tratamiento de datos y ciberseguridad 136
5.9 Arquitectura final y resumen de decisiones 137
5.10 Implantación y pruebas de campo antes del despliegue 137
Capítulo 6. Estudios económicos y de viabilidad 139
6.1 Alcance económico y criterio de cálculo 139
6.2 Referencias de mercado y fuentes utilizadas 140
6.3 Modelo económico de la solución propuesta 141
6.3.1 Coste de captación 141
6.3.2 Coste de procesamiento *edge* 142
6.3.3 Coste de comunicaciones 143
6.3.4 Coste de un módulo mínimo de percepción 144
6.3.5 Coste de software, licenciamiento e integración digital 144
6.3.6 Servicios de ingeniería, instalación y puesta en marcha 145
6.3.7 Coste de actuación e información al usuario 146
6.4 Presupuesto del sector CAD de 1.211 plazas 147
6.5 Ampliación mediante cámaras reutilizadas 148
6.6 Comparativa económica por alcance equivalente 149
6.6.1 Comparativa de software y modelo de licenciamiento 150
6.6.2 Comparativa de hardware de percepción 151
6.6.3 Comparativa de cámara con inteligencia artificial 152
6.7 Impacto ambiental y contribución a los ODS 152
6.8 Consideraciones sobre mantenimiento y operación 153
6.9 Conclusión de viabilidad 154
Capítulo 7. Conclusiones 155
7.1 Conclusiones generales 155
7.2 Líneas futuras 156
7.2.1 Continuación directa del proyecto 156
7.2.2 Evolución de mayor alcance 157
7.3 Competencias y conocimientos del grado aplicados 157
BIBLIOGRAFÍA 159
ANEXOS 164

ANEXO 1. Presupuesto desglosado para la instalación completa en la UEM 165
ANEXO 2. USO DE LA IA 167
ANEXO 3. ENTRENAMIENTO DE MODELOS 168
Anexo 3.1 Definición de parámetros de entrenamiento 168
Anexo 3.2 Primer modelo 171
Anexo 3.3 Segundo modelo 174
Anexo 3.4 Tercer modelo 177
Anexo3.5 Cuarto modelo 180
Anexo 3.6 Quinto modelo 183
Anexo 3.7 Sexto modelo 186
Anexo 3.8 Séptimo modelo 189
Anexo 3.9 Octavo modelo 192
Anexo 3.10 Noveno modelo 195
ANEXO 4. CÓDIGO DE LA APLICACIÓN 198
Anexo 4.1 Código de la interfaz de supervisión 198
Anexo 4.2 Código del sistema de visión y lógica interna 209

Índice de Figuras

Figura 1.Arquitectura conceptual de un Smart *Parking* System (SPS). Elaboración propia. 23

Figura 2.Comparación de distintos ángulos de cámara sobre un aparcamiento: vista horizontal, vista lateral y vista cenital. De Chen et al. 34

Figura 3. Comparación entre el procesamiento centralizado y el procesamiento en el borde por sectores. Elaboración propia 42

Figura 4. Monitorización de plazas mediante sensor óptico elevado Cleverciti. De Cleverciti .. 49

Figura 5. Panel de control de Parquery. De parquery 50

Figura 6. Cámara AI de Milesight aplicada a la gestión de aparcamientos. De Milesight..... 51

Figura 7. Arquitectura del sistema PakLoc/PakSta para localización automática de plazas y detección de ocupación. De Nguyen y Sartipi 54

Figura 8. Plazas del aparcamiento de la UEM definidas como regiones de interés. Elaboración propia..... 60

Figura 9. Plazas marcadas con identificador. Elaboración propia..... 60

Figura 10. Etapas de procesado aplicadas a la imagen en el método clásico sobre los vídeos de ufpark. Elaboración propia. 62

Figura 11. Método clásico aplicado al aparcamiento de la UEM. Elaboración propia. 63

Figura 12. Clasificación de las plazas del aparcamiento UEM según su calidad de captación. Elaboración propia. 65

Figura 13. Detección de vehículos sobre el aparcamiento de la UEMl con baja ocupación (derecha) y alta ocupación (izquierda). Elaboración propia..... 66

Figura 14. Método de asociación de *Parking*Management contra punto bajo sobre UFPark. Elaboración Propia 67

Figura 15. Método asociación del punto central sobre la UEM. Elaboración propia..... 69

Figura 16. Método asociación del punto bajo sobre la UEM. Elaboración propia..... 70

Figura 17. Método de asociación multipunto sobre la UEM. Elaboración propia 71

Figura 18. Limitación en vista oblicua del método multipunto sobre la UEM. Elabpración propia. 71

Figura 19. Método de asociación por solape sobre la UEM. Elaboración propia 72

Figura 20.Asignación de la plaza principal por solape de la huella con la ROI. Elaboración propia 73

Figura 21. Recuperación de una detección mediante el respaldo de puntos. Elaboración propia 74

Figura 22. Confirmación de una segunda plaza ocupada mediante la rejilla de veinte puntos. Elaboración propia 75

Figura 23. Comparación entre la rejilla normal en un coche cercano y la rejilla desplazada hacia abajo en un coche lejano. Elaboración propia 75

Figura 25. Esquema del flujo seguido para comparar configuraciones. Elaboración propia 77

Figura 26. Comparación visual entre C1_1280_025 y C2_1280_015 sobre el mismo *frame*. Elaboración propia. 79

Figura 27. Comparación visual entre C1_1280_025 y C5_1920_025 sobre el mismo *frame*. Elaborción propia 80

Figura 28. Sensibilidad en plazas válidas y problemáticas según la resolución, con confianza 0,25. Elaboración propia.. 81

Figura 29. Esquema del flujo seguido para comparar de modelos. Elaboración propiade..... 85

Figura 30. F1 sobre todas las plazas frente al tiempo de inferencia por *frame* de cada modelo. Elaboración propia. 87

Figura 31. Ejemplos de vistas de distintos aparcamientos del conjunto de entrenamiento. Elaboración propia a partir de las fuentes utilizadas. 92

Figura 32. Distribución del *dataset* utilizado en el *fine tuning*. Elaboración propia 93

Figura 34. Efecto de la resolución de entrenamiento en la detección de coches lejanos, 512 frente a 960 píxeles. Elaboración propia 96

Figura 35. Evolución de las pérdidas y métricas de validación del modelo YOLO11s seleccionado durante el ajuste fino. Elaboración propia. 98

Figura 36. Zona de estudio sobre la que se evalúa la detección, basada en plazas de estudio. Elaboración propia 100

Figura 37. Solución a problema de detección incorrecta en el *parking* piloto UEM. Elaboración propia..... 100

Figura 38. Método métricos detección vehiculos comparación YOLO11s básico vs entrenado. Elaboración propia 101

Figura 39. Comparación global de las métricas de detección de los seis modelos. Elaboración propia..... 103

Figura 40. Falsos positivos y falsos negativos de cada modelo. Elaboración propia. 103

Figura 41. Recall por tamaño aparente del coche para los seis modelos. Elaboración propia. 104

Figura 42. Detección sobre misma escena de aparcamiento. Marcando errores modelo básico a la izquierda y mostrando solución de modelo afinado a la derecha. Elaboración propia 106

Figura 43. Comparación del sistema de ocupación en la zona piloto con el YOLO11s base y el afinado, usando la misma lógica de asociación. Elaboración propia 107

Figura 44. Detalle del efecto del modelo afinado sobre la ocupación de las plazas más alejadas de la zona piloto. Elaboración propia..... 108

Figura 45. Funcionamiento del sistema final de estimación de ocupación sobre otra vista del aparcamiento de Veterinaria. Elaboración propia 109

Figura 46. Coste computacional del sistema completo en los tres escenarios, separando detección y asociación. Elaboración propia 111

Figura 47. Cambios de ocupación antes y después del suavizado en los tres escenarios. Elaboración propia 114

Figura 48. Evolución de la ocupación con y sin suavizado en la zona piloto poco ocupada. Elaboración propia 115

Figura 49. Pantalla final de la interfaz con la visión activa. Elaboración propia 122

Figura 50. Interfaz con el módulo de visión detenido y el vídeo reproduciéndose sin superposiciones. Elaboración propia..... 122

Figura 51. Control de la barrera y tabla de avisos de incidencias. Elaboración propia..... 123

Figura 52. Arquitectura del software desarollado. Elaboración propia..... 124

Figura 53. Camara Hikvision DS-2CD1043G2-I seleccionada para despliegue. De VISIOTECH 128

Figura 54. Propuesta base de despliegue sobre el plano del aparcamiento del Santander. Elaboración propia 130

Figura 55. Propuesta base de despliegue sobre el plano del aparcamiento edificio E. Elaboración propia..... 131

Figura 56. NVIDIA Jetson Orin *Nano* Super, plataforma de procesamiento seleccionada para los nodos. De Amazon.com 133

Figura 57. Resumen integración de arquitectura entre camaras, nodos y enlaces de comunicación. Elaboración propia 135

Figura 58. FAAC B614, barrera automática de exterior seleccionada. De FAAC..... 136

Figura 59. Arquitectura completa del sistema propuesto. Elaboración propia..... 137

Índice de Tablas

Tabla 1. Cronograma de planificación del Trabajo de Fin de Grado. Elaboración propia 21

Tabla 2. Comparativa entre captación por cámaras y captación por sensores individuales. Elaboración propia a partir de Paidi et al. (2018) 25

Tabla 3. Métricas de calidad de la detección habituales en SPS basados en visión. Elaboración propia 27

Tabla 4. Comparación de formatos principales de cámaras IP para monitorización exterior. Elaboración propia 36

Tabla 5. Comparación de plataformas edge AI ejecutando un modelo YOLOv5 39

Tabla 6. Comparación de soluciones comerciales y propuestas académicas relevantes 56

Tabla 7. Frames representativos usados para la evaluación. Elaboración propia 64

Tabla 8. Parámetros de la asociación vehículo-plaza. Elaboración propia 76

Tabla 9. Configuraciones evaluadas para YOLO11n. Elaboración propia 78

Tabla 10. Detecciones medias y tiempo de inferencia por configuración. Elaboracion propia. . 78

Tabla 11. Métricas de ocupación en plazas válidas para las configuraciones de YOLO11n. Elaboración Propia 80

Tabla 12. Métricas globales incluyendo todas las plazas para las configuraciones YOLO11n. Elaboración propia 82

Tabla 13. Estimación de carga temporal según la frecuencia de análisis. Elaboración Propia. .. 83

Tabla 14. Configuración elegida y referencia. Elaboración propia 84

Tabla 15. Modelos YOLO evaluados. Elaboración propia 85

Tabla 16. Métricas de ocupación en plazas válidas para los modelos. Elaboración Propia 86

Tabla 17. Métricas globales incluyendo todas las plazas para los modelos. Elaboración propia 87

Tabla 18. Carga temporal por modelo según la frecuencia de análisis. Elaboración propia. 88

Tabla 19. Decisión final de modelo. Elaboración propia. 89

Tabla 20. Resumen de las configuraciones evaluadas durante el ajuste fino. Elaboración propia 98

Tabla 21. Metricas globales de deteccion de cada modelo, base y afinado. Elaboracion propia. 102

Tabla 22. Recall por tamaño aparente del coche, base y afinado. Elaboración propia 104

Tabla 23. Carga de cómputo según el intervalo de análisis, en el escenario más exigente. Elaboración propia 112

Tabla 24. Requisitos de partida para la selección del hardware. Elaboración propia 113

Tabla 25. Casos de prueba de la validación funcional de la aplicación. Elaboración propia 126

Tabla 26. Referencias económicas utilizadas para el análisis comparativo. Elaboración propia 141

Tabla 27. Desglose del coste de captación para el sector CAD. Elaboración propia 142

Tabla 28. Desglose del coste de procesamiento edge. Elaboración propia 143

Tabla 29. Desglose corregido del coste de comunicaciones. Elaboración propia. 143

Tabla 30. Coste estimado de un módulo mínimo de percepción. Elaboración propia. 144

Tabla 31. Desglose del coste de software e integración digital. Elaboración propia 145

Tabla 32. Desglose de servicios de ingeniería, instalación y puesta en marcha. Elaboración propia 146

Tabla 33. Desglose del coste de actuación e información. Elaboración propia 147

Tabla 34. Presupuesto de la capa de percepción para las 1.211 plazas contabilizadas. Elaboración propia..... 147

Tabla 35. Presupuesto del sistema completo para las 1.211 plazas contabilizadas. Elaboración propia..... 148

Tabla 36. Coste de ampliación mediante cámaras reutilizadas. Elaboración propia..... 149

Tabla 37. Comparativa de software con alcance equivalente. Elaboración propia 151

Tabla 38. Comparativa de hardware de percepción con sensores y APMS. Elaboración propia. 151

Tabla 39. Comparativa de cámara con inteligencia artificial. Elaboración propia..... 152

Capítulo 1. Introducción

Este Trabajo de fin de Grado tiene como finalidad el diseño de un sistema de percepción inteligente para la gestión eficiente de aparcamientos al aire libre en tiempo real. Para ello, se han estudiado distintas alternativas tecnológicas de monitorización, procesamiento y despliegue, considerando especialmente su aplicación en entornos universitarios como el de la UEM. En este capítulo, se establecen las bases del proyecto donde, se expone el contexto que motiva la investigación, se formula el problema que se pretende resolver, se justifica la necesidad del proyecto y se delimitan los objetivos que guían el desarrollo del trabajo.

1.1 Contexto y motivación

La gestión del aparcamiento urbano al aire libre es uno de los problemas más persistentes y peor resueltos de la movilidad en las ciudades y en entornos con alta afluencia. A medida que la población mundial aumenta y el volumen de vehículos crece (en la Unión Europea se registra una media de 0,57 turismos por habitante (Eurostat, 2024), y solo en España circulan más de 26,47 millones de turismos, un 1,2% más respecto al año anterior (ANFAC/DGT, 2025)), la brecha entre la demanda de plazas de aparcamiento y la oferta disponible se amplía de manera progresiva. Esto genera en una serie de impactos negativos que a escala individual pueden parecer pequeños, pero que afectan tanto a la calidad de vida de las personas como al medio ambiente.

En estos entornos con alta demanda, una parte relevante del tráfico no responde a un desplazamiento estrictamente necesario, sino a la búsqueda de una plaza libre. El fenómeno de buscar aparcamiento, que en la literatura se denomina *cruising for parking*, ha sido objeto de estudio durante décadas, vinculándose de forma directa con las ineficiencias del sistema de aparcamiento. El conductor llega al destino, pero carece de información suficientemente precisa (o mecanismos que realicen guiado de manera efectiva) para localizar una plaza disponible sin recorrer trayectos adicionales. Entre la literatura disponible, se presentan los datos más representativos por la metodología utilizada y por su relevancia en patrones de movilidad urbana europea, que son más comparables a los de las ciudades españolas.

El informe de INRIX (2017), una de las referencias más citadas, estima a partir de encuestas a 18.000 conductores que la búsqueda de aparcamiento supone de media 44 horas anuales por conductor en el Reino Unido y 41 en Alemania, con un coste individual cercano a los 900 € al año que, mirándose a escala de los países completos, alcanza miles de millones de euros.

Estas cifras provienen de encuestas autorreportadas, que suelen sobreestimar el tiempo percibido, y los estudios con mediciones GPS reales dan valores muy inferiores, como los 1,5 minutos de media por búsqueda que registran Saki y Hagen (2024) en ciudades alemanas. Lo que puede parecer una contradicción, en realidad está indicando que el cruising es un fenómeno localizado. A escala de ciudad completa representa menos del 5-6 % de los viajes (Weinberger, Millard-Ball y Hampshire, 2020), pero en las zonas de aparcamiento gratuito y saturado se concentra mucho, porque Hampshire y Shoup (2018) midieron con cámaras en Stuttgart que el 15 % del tráfico buscaba plaza, y la literatura clásica sitúa el porcentaje hasta en el 30 % en los focos de máxima demanda (Shoup, 2006). El campus de la universidad, es justamente ese tipo de sitio, donde el

foco esta localizado, es gratuito y se satura en horas punta, por lo que aplicar un sistema de información en tiempo real puede ser muy eficaz. Además, la misma encuesta de INRIX añade que el 63 % de los conductores ha evitado destinos por la previsión de dificultades de aparcamiento, lo que muestra que el problema afecta también a la actividad de las zonas.

Las consecuencias ambientales y operativas son otro factor que se tiene que tener en cuenta. Sui et al. (2022), mediante simulaciones con el software VISSIM (para simulación del tráfico) calibradas con datos GPS reales, demostraron que incluso una ratio de cruising tan bajo como el 5 % puede reducir la velocidad media de una sección de la vía hasta un 74,58 %, aumentando la congestión y empeorando el funcionamiento del tráfico. Este tráfico innecesario, no solo consume tiempo y combustible, sino que añade emisiones y agrava las retenciones. El problema, además, no es solo operativo o ambiental. En el plano institucional, la Ley 7/2021, de 20 de mayo, alineada con los objetivos del Pacto Verde Europeo, obliga a implantar Zonas de Bajas Emisiones en municipios de más de 50.000 habitantes, territorios insulares y municipios de más de 20.000 que superen los límites de contaminación. Por ello, reducir este tráfico mejora la eficiencia y contribuye a la descarbonización y a la calidad del aire.

Uno de los puntos clave de este TFG es que la mayoría de las investigaciones y soluciones comerciales de *Smart parking*, se han centrado mayoritariamente en aparcamientos subterráneos o cubiertos, con condiciones climáticas y de iluminación controladas. Aunque es cierto que, con el desarrollo de las nuevas tecnologías y sistemas de visión, las investigaciones están aumentando, la atención investigadora sigue siendo significativamente menor teniendo en cuenta el problema que suponen, tal y como confirman autores como Paidi et al. (2022). Según detallan en su propio estudio de caracterización de emisiones en aparcamientos al aire libre, ante la falta de señalización, los conductores acaban recorriendo de media casi el triple de distancia (2,7 veces) de la que sería estrictamente necesaria para llegar a una plaza libre. Este dato evidencia el amplio margen de mejora que tendrían los sistemas de guiado en tiempo real si se desplegaran en estos aparcamientos, donde ya se ha demostrado ampliamente su efectividad. Por ejemplo, el programa SFpark de San Francisco, donde se implementaron una red masiva de sensores inalámbricos y siendo una de las implementaciones más estudiadas a nivel mundial, evaluadas por Krishnamurthy y Ngo (2020), logró una reducción del 30 % en los kilómetros recorridos buscando aparcamiento y del 43 % en el tiempo medio de búsqueda. Más recientemente, Dalla Chiara et al. (2022), demostraron mediante un experimento controlado que proporcionar esta información de disponibilidad reduce la fracción de cruising para los conductores de un 46 % a un 22 %, mientras que el tiempo de espera cae fuertemente de los 3,1 minutos a 1,1 minutos. El reto, por tanto, no está en su eficacia porque que ya está demostrada, sino en desplegar estos sistemas en los aparcamientos.

En el caso de España, la literatura sobre *cruising for parking* todavía es más limitada que en otros contextos europeos ya que se apoya mucho en encuestas y en simulaciones con datos reales que en campañas de medición directa. En Santander, por ejemplo, Rodríguez et al. (2022) han desarrollado estudios sobre cómo los conductores eligen dónde aparcar y cómo buscan plaza en la vía pública, utilizándose datos del entorno urbano. Posteriormente, de nuevo Rodríguez et al. (2024) estiman que una mayor disponibilidad de información puede reducir en torno a un 55 % los intentos fallidos antes de encontrar plaza y disminuir en un 37 % las emisiones de partículas.

La motivación de este TFG surge del propio entorno donde se desarrolla. El campus de la UEM es un sitio donde el problema del aparcamiento se ve claramente, dado que resulta habitual ver a conductores dando vueltas o experimentar directamente la dificultad de encontrar aparcamiento rápido. Esto sucede porque los campus universitarios tienen un elevado número de usuarios con patronos de movilidad muy parecidos, generando situaciones de saturación predecibles pero difíciles de gestionar sin información en tiempo real.

El orden de magnitud del problema en el propio campus, y el ahorro que un sistema como el propuesto podría suponer, se estiman en el apartado 6.7.

Esta saturación hace del campus un buen sitio para plantear y evaluar una solución basada en la infraestructura existente. Las farolas del campus ofrecen una posición elevada de entre 6 y 10 metros con un amplio campo de visión panorámica sobre las plazas de su entorno. Además, disponen de conexión eléctrica y evitan realizar obras civiles en el pavimento. Por ello, analizar y trabajar con el entorno de nuestra propia universidad es un escenario accesible y representativo.

1.2 Planteamiento del problema y justificación

En este contexto, el reto no es reconocer la relevancia del problema del aparcamiento al aire libre, que ya ha quedado clara, sino encontrar la mejor forma de resolverlo. Es decir, determinar cómo monitorizar de forma eficiente la ocupación de las plazas y otras incidencias relevantes, de manera que la información que da el sistema sea de verdad útil para gestionar el espacio y con ello poder guiar a los vehículos. Aparte de saber si una plaza esta libre u ocupada de forma precisa, la solución se tiene que desarrollar para ser viable tanto técnicamente como económicamente.

Los aparcamientos al aire libre tienen una complejidad añadida respecto a los interiores, porque presentan mucha variabilidad ambiental, con cambios de iluminación, sombras, condiciones meteorológicas, oclusiones y formas más distintas formas de distribuir las plazas que las interiores. Esto complica detectar con precisión el estado del aparcamiento y obliga a elegir con más cuidado tanto la tecnología de visión como la arquitectura de procesamiento, para que el sistema sea robusto ante esas condiciones.

Aunque ya existen soluciones comerciales de monitorización inteligente, la mayoría se basan en sensores individuales instalados en cada plaza, lo que supone un coste de infraestructura alto y dificulta su adopción. La mejora de los sistemas de visión artificial y de las plataformas de procesamiento abre la puerta a alternativas más flexibles y baratas, ya que una sola cámara puede cubrir muchas plazas a la vez, pero estas soluciones todavía no se ven de forma habitual en aparcamientos exteriores de uso cotidiano, lo que indica que existen barreras de coste, instalación, mantenimiento y privacidad que la tecnología por sí sola no resuelve o que no está habiendo inversión para ello. En particular, trabajar con flujos de vídeo obliga a decidir si el procesamiento se hace en la nube, en el propio nodo o con una arquitectura híbrida, una elección que afecta a el ancho de banda, la latencia, la dependencia de la conectividad y la exposición de las imágenes. Teniendo en cuenta esto, lo que interesa en el trabajo es analizar la combinación de tecnologías y decisiones de diseño necesarias para montar una arquitectura de percepción inteligente que cumpla a la vez con las necesidades reales y con la viabilidad técnica.

La UEM se plantea como caso de estudio por constituir un entorno real, accesible y representativo en el que se puede analizar la viabilidad de la solución que se escoja. De esta forma el trabajo aborda el problema técnico, pero también lo hace en un espacio donde es necesario resolver las ineficiencias que se describen en el apartado 1.1.

Para la UEM el proyecto tiene además un interés institucional ligado a sus compromisos de sostenibilidad, que se desarrolla junto con el impacto ambiental estimado en el apartado 6.7

El trabajo se justifica tanto por el interés práctico como por el valor técnico. Desde la perspectiva aplicada, se aborda la necesidad de mejorar la gestión del aparcamiento exterior mediante soluciones que ayuden a reducir recorridos innecesarios, tiempos de búsqueda y usos ineficientes del espacio. Desde una perspectiva de ingeniería, permite estudiar de forma integrada distintas decisiones de diseño con el objetivo de definir una arquitectura de percepción inteligente que no solo sea funcional, sino también viable, escalable y adaptada a un caso de uso real como es el *parking* de la UEM.

1.3 Objetivos

En este apartado se definen los objetivos que orientan el desarrollo del trabajo. De esta forma, se delimita el alcance del proyecto, se organizan las tareas a realizar y se establece una referencia clara para al final del proyecto poder evaluar el grado de cumplimiento con los resultados obtenidos.

1.3.1 Objetivo general

Diseñar y evaluar una arquitectura de percepción inteligente para monitorizar aparcamientos al aire libre, capaz de estimar la ocupación de las plazas y detectar incidencias relevantes mediante visión artificial. La solución tendrá que integrar la captación, el procesamiento y el hardware de forma que sea técnicamente viable, económicamente asequible y aplicable en un entorno real como caso de estudio de la UEM.

1.3.2 Objetivos específicos

Los objetivos específicos para este proyecto son los siguientes:

1. **Analizar el estado actual de las soluciones de monitorización inteligente aplicadas a aparcamientos al aire libre, para identificar sus limitaciones y justificar correctamente la propuesta de sistema**
    - Estudiar las soluciones comerciales y académicas más representativas.
    - Comparar enfoques basados en sensores individuales frente a sistemas basados en visión artificial.
    - Identificar las limitaciones en coste, escalabilidad, facilidad de despliegue, funcionalidades y privacidad

2. **Definir los requisitos del sistema para la UEM, para que la solución responda a una necesidad real.**

- Determinar qué información tiene proporcionar el sistema sobre el aparcamiento y las incidencias que puedan ser útiles de detectar
- Adaptar estos requisitos a las condiciones concretas del *parking* y a las necesidades reales que soliciten las personas encargadas en el trabajo de campo en la UEM

3. **Desarrollar y evaluar la estrategia de visión artificial más adecuada para estimar la ocupación.**
    - Analizar modelos o enfoques de detección aplicables al problema.
    - Considerar los retos de iluminación, sombras, meteorología adversa y oclusiones.
    - Justificar la elección del enfoque final en función del equilibrio entre precisión, coste computacional y viabilidad de integración.

4. **Diseñar y validar el método de asociación entre vehículos detectados y plazas de aparcamiento.**
    - Analizar distintos criterios geométricos para transformar las detecciones de vehículos en ocupación de plazas
    - Conseguir que las detecciones se asocien en estados de ocupación de plazas
    - Estudiar el efecto de la perspectiva, la distancia, las oclusiones y los vehículos parcialmente visibles,
    - Evaluar la capacidad del método para detectar ocupaciones secundarias o incidencias básicas de estacionamiento.

5. **Estudiar y seleccionar la arquitectura para el sistema, incluyendo la de procesamiento y la de hardware, definiéndose los componentes necesarios y su integración en la infraestructura.**
    - Comparar alternativas en procesamiento en el borde, en la nube y enfoques híbridos
    - Seleccionar la cámara o sistema de captación más adecuado para el escenario de aplicación.
    - Relacionar la arquitectura elegida con en latencia, consumo, ancho de banda, el consumo, coste computacional y privacidad.
    - Plantear la integración del sistema sobre soportes elevados preexistentes, como farolas u otros elementos del entorno.

6. **Hacer una validación representativa del sistema diseñado, simulando el prototipo software del sistema.**
    - Integrar la detección de vehículos, la asociación vehículo-plaza, la estabilización temporal y la detección de incidencias básicas.
    - Desarrollar una interfaz de supervisión que permita visualizar el estado del aparcamiento y los avisos generados por el sistema.
    - Comprobar el funcionamiento del prototipo sobre vídeos reales del aparcamiento de la UEM.
    - Identificar las limitaciones que quedarían pendientes para una implantación real, como la instalación física de cámaras, las comunicaciones, la alimentación y la validación prolongada en campo.

7. **Evaluar la viabilidad global de la propuesta desde el punto de vista técnico, económico y normativo.**
    - Analizar el comportamiento del sistema y del entorno de validación desarrollado para la implantación real.
    - Estimar el coste aproximado de implantación y compararlo con soluciones existentes.
    - Valorar su adecuación al marco normativo y de protección de datos aplicable.
    - Extraer conclusiones sobre las decisiones de diseño adoptadas, las limitaciones encontradas y las mejoras futuras.

1.4 Alcance

El trabajo abarca el estudio de las soluciones existentes, la definición de requisitos, el análisis de alternativas de captación, procesamiento, comunicaciones y hardware, y el diseño de una propuesta adaptada al aparcamiento exterior de la UEM. Incluye además el desarrollo y la validación de una estrategia de visión artificial sobre vídeos reales del campus, la estimación de ocupación plaza por plaza, la detección de incidencias básicas, la integración en un prototipo software y la propuesta de despliegue sobre la infraestructura disponible.

Quedan fuera el desarrollo de un producto comercial terminado, la instalación física a escala completa y la validación prolongada en todas las condiciones ambientales posibles. Se excluyen también las funcionalidades secundarias que no hagan falta para comprobar la arquitectura principal, y cualquier conclusión que dependa de una prueba piloto y de despliegue real.

1.5 Metodología

El trabajo sigue una metodología de diseño de ingeniería enfocada en ser experimental, en la que la solución no se fija de antemano, sino que se construye probando alternativas sobre datos reales y justificando cada decisión con criterios medibles. El método de trabajo es iterativo y se repite en cada bloque, partiendo de una aproximación, se prueba sobre el aparcamiento, se detecta dónde falla y se decide el siguiente paso a partir de ese fallo, cerrando cada etapa con la opción que mejor equilibra fiabilidad, coste y facilidad de despliegue.

El punto de partida es una revisión del estado del arte y de las bases técnicas de los sistemas de aparcamiento inteligente, la visión artificial, el procesamiento en el borde, las comunicaciones y la privacidad, que sirve para situar las limitaciones de las soluciones actuales y fijar los criterios sobre los que se decide después. A su vez, se realiza un trabajo de campo en el aparcamiento de la UEM, apoyado en los planos, las grabaciones disponibles y las reuniones con los responsables del campus, concreta las necesidades reales y los condicionantes del diseño, como la ausencia de red en el propio aparcamiento, la reutilización de la infraestructura existente, las oclusiones del arbolado y la exigencia de que la solución sea escalable.

A partir de ahí el trabajo se aborda desde dentro hacia fuera, porque la visión es el núcleo del sistema y de ella dependen el resto de decisiones. Por eso se resuelve primero qué y cómo se percibe la ocupación, y solo después se ajustan el hardware, el procesamiento y la comunicación a lo que esa parte necesita. La fase experimental se desarrolla sobre vídeos reales del aparcamiento, comparando cada alternativa sobre una misma referencia de evaluación con

métricas de detección y de ocupación y con el tiempo de inferencia, pero sin quedarse solo con los número, porque cada resultado se contrasta con la revisión visual de los casos de fallo, porque con una muestra limitada el comportamiento del sistema con la perspectiva, la distancia, las oclusiones y los vehículos pequeños puede dar más información que una diferencia de decimales en muchas ocasiones.

El análisis no termina en detectar vehículos, sino en convertir esas detecciones en información fiable plaza por plaza, por eso se evalúan por igual el detector, el criterio de asociación vehículo-plaza, la estabilidad de la decisión en el tiempo y la detección de incidencias. Por último, cada decisión técnica se contrasta con las condiciones de un despliegue real, valorando el coste computacional, el consumo, la privacidad, la facilidad de instalación, la reutilización de cámaras y la comunicación entre nodos, de modo que el objetivo no es solo un prototipo que funcione sobre un vídeo, sino comprobar si la solución puede trasladarse de forma realista a un entorno como el campus de la UEM.

1.6 Planificación

Para garantizar la viabilidad del proyecto dentro de los plazos establecidos para la entrega, se diseñó un plan de trabajo que organizara las tareas de forma progresiva, desde la investigación inicial hasta la revisión final de la memoria, cubriendo tanto la parte técnica como la documentación.

Inicialmente, el trabajo se planteó siguiendo una secuencia general, de análisis del problema, revisión bibliográfica, definición de requisitos, desarrollo del sistema de visión, validación y propuesta de despliegue. Pero, al tener el proyecto una parte experimental grande, la planificación se fue ajustando durante el desarrollo. Varias tareas se solaparon , especialmente la preparación de datos, las pruebas con modelos, el ajuste del sistema y la redacción de la memoria, ya que los resultados obtenidos en cada fase condicionaban las siguientes decisiones.

El cronograma final es semanal, va desde febrero hasta la entrega final del 5 de julio y supone unas 400 horas de dedicación, dada la cantidad de datos que hay que manejar.

TAREAS	2 feb	9 feb	16 feb	23 feb	2 mar	9 mar	16 mar	23 mar	30 mar	6 abr	13 abr
Elección tema y problema	10 h										
Objetivos, alcance y planificación		7 h									
Marco teórico			17 h								
Estado del arte				17 h							
Entrega intermedia y tutorías					10 h						
Trabajo de campo y requisitos UEM						12 h					
Planos, cámaras y caso UEM							8 h				
Preparación vídeos y frames									8 h		
ROIs y ground truth									20 h		
Pruebas iniciales visión/YOLO									30 h		
Clasificación plazas y métricas base										12 h	

TAREAS	20 abr	27 abr	4 may	11 may	18 may	25 may	1 jun	8 jun	15 jun	22 jun	29 jun
Asociación vehículo-plaza	30 h										
Comparación configuraciones		23 h									
Comparación modelos y métricas			22 h								
Dataset fine-tuning				52 h							
Interfaz y programa							11 h				
Hardware y procesamiento edge							14 h				
Entrenamiento fine-tuning								24 h			
Integración y validación final								16 h			
Frecuencia, estabilización e incidencias									8 h		
Comunicaciones y privacidad									10 h		
Despliegue, cobertura y CAD									16 h		
Conclusiones y cierre técnico										10 h	
Revisión final, bibliografía y entrega											13 h

Tabla 1. Cronograma de planificación del Trabajo de Fin de Grado. Elaboración propia

Como se observa en el cronograma de la Tabla 1, las primeras semanas fueron para elegir y acotar el tema, analizar el problema, formular objetivos y redactar el marco teórico y el estado del arte, sobre todo para preparar todos los contenidos antes de la entrega intermedia.

Después el foco pasó al caso de la UEM, con el trabajo de campo, la revisión de planos y cámaras, la preparación de vídeos, la definición de regiones de interés y la construcción del *ground truth*, que son importantes y requieren de planificación porque requieren desplazamientos y sin imprescindibles para desarrollar los siguientes pasos.

La mayor carga experimental se concentró entre finales de marzo y abril aprovechando la semana santa, con la preparación de datos, las primeras pruebas de visión y las métricas base. En mayo y junio entraron la asociación vehículo-plaza, la comparación de configuraciones y de modelos, el *dataset* de fine tuning, el entrenamiento, la integración y la validación final. El tramo final cubrió la interfaz, el hardware, el procesamiento en el borde, las comunicaciones, la privacidad y la propuesta de despliegue, y la última semana quedó para las conclusiones, la bibliografía, el formato y el cierre de la entrega.

Capítulo 2. Marco teórico

En este capítulo, se explican las bases de conocimiento necesarias para poder comprender el diseño del sistema propuesto. Se abarca desde los fundamentos de los sistemas de aparcamiento inteligente y su relación con los sistemas de visión y los enfoques de visión artificial aplicados a la detección de ocupación hasta los retos propios de los aparcamientos al aire libre y la geometría de captación de las cámaras.

Además de la visión, se analizan las distintas arquitecturas de procesamiento disponibles, así como las bases de hardware necesarias para entender el sistema completo. Finalmente, como los sistemas de visión pueden captar datos de personas, se analiza el marco normativo de privacidad y protección de datos, imprescindible para un diseño responsable.

2.1 Sistemas de Smart *parking*

2.1.1 Conceptos y funciones de los sistemas de *Smart parking*

Un sistema de aparcamiento inteligente, denominado habitualmente Smart *Parking* System (SPS), término que se utilizará también a lo largo de este trabajo, consiste en el uso de distintas tecnologías de sonorización. Estas pueden ser de ultrasonidos, infrarrojos, radar o cámaras, entre otras (Paidi et al., 2018), pudiendo implementarlas individualmente o de forma combinada con el objetivo de mejorar la eficiencia en la gestión de las plazas de estacionamiento a través de dispositivos de captación, procesamiento y comunicación de información en tiempo real.

A diferencia de métodos tradicionales, basados en observación manual o en señalización convencional que no cuenta con actualización de información en tiempo real, un SPS permite conocer el estado de ocupación de cada plaza de forma automática y continua. Además, puede incorporar otras funcionalidades ya mencionadas como la detección de vehículos mal aparcados u otros eventos de interés, trasladando toda es información tanto a los gestores del aparcamiento para que puedan hacer un seguimiento, como a los propios conductores para que puedan recibir información relevante. La Figura 1 representa de forma esquemática la estructura general de un SPS, donde el estado real del aparcamiento se capta, se procesa internamente y se distribuye a sus dos destinatarios principales.

Aunque para cada entorno se use un sistema u otro, según sus requisitos, para comprobar y vigilar la ocupación de plazas, el objetivo es siempre el mismo: mejorar la eficiencia en la gestión del aparcamiento a través de la automatización de la supervisión, que no es viable de hacerlo manualmente de forma continua sobre cientos de plazas.

Las funciones que puede cubrir un SPS varían según su grado de complejidad, al disponer de distintas necesidades y requisitos en cada entorno. Por lo general pueden incluir:

- Detección de disponibilidad de las plazas.
- Guiado de conductores hacia plazas libres.
- Gestión de acceso con barreras u otros sistemas de control de acceso.

- Monitorización del uso para análisis de patrones u otros.
- Detección de incidencias como estacionamientos indebidos o el uso indebido de las plazas, como plazas reservadas para personas con movilidad reducida o para vehículos eléctricos.

Estas son las funciones que se pueden considerar como las esenciales de este tipo de sistemas, y su concreción para el caso de estudio, con las funcionalidades que finalmente implementa la solución, se desarrollan durante el desarrollo del sistema.

La posibilidad de monitorizar en tiempo real es uno de los puntos clave de estos sistemas. Como se vio en el apartado 1.1, sin información actualizada del estado del aparcamiento aparece el fenómeno del *cruising for parking,* con todos sus impactos asociados. La aportación principal de un SPS no es únicamente la detección automática del estado de las plazas sino la generación de información actualizada que cierra la brecha entre la oferta real disponible y la demanda en cada momento.

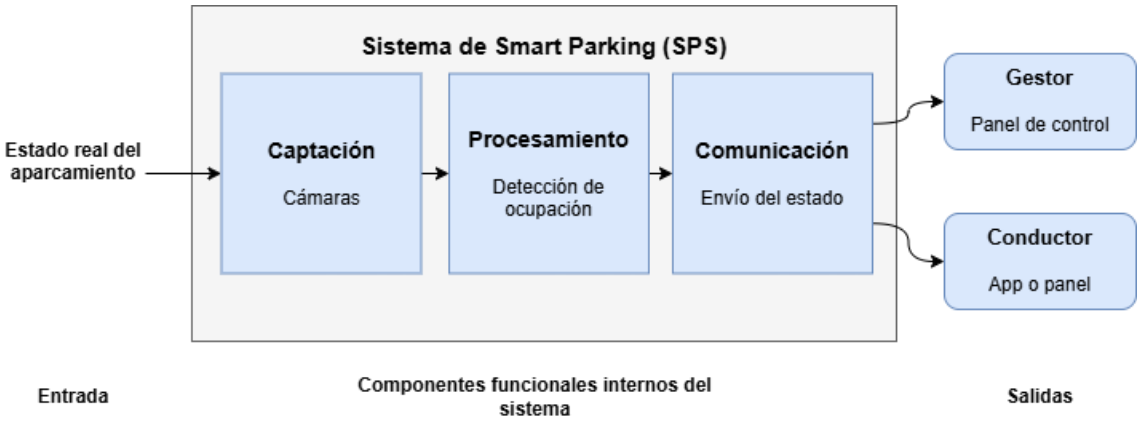


Figura 1.Arquitectura conceptual de un Smart Parking System (SPS). Elaboración propia.

Después de realizar una investigación comparando los modelos y teniendo en cuenta el rápido avance que están teniendo estas tecnologías, se ha centrado el desarrollo de un sistema en uno con cámaras con visión por computador. Se ha optado por este tipo de sistemas para los aparcamientos de exterior dado que una sola cámara puede cubrir decenas de plazas (siempre y cuando se solventen los posibles retos a los que se enfrentan estos sistemas, como se comenta en el apartado 2.2.3).

A continuación, se analizan las distintas partes compone esta arquitectura para la gestión de los aparcamientos, profundizando en los componentes propios de un sistema basado en visión

2.1.2 Componentes principales de un sistema de *smart parking*

Un SPS basado en visión puede descomponerse conceptualmente en cuatro unidades funcionales que interactúan entre sí para ofrecer el servicio completo y conforman la arquitectura del sistema. Estas son: la de captación, la de procesamiento , la de comunicación y la de aplicación. Esta estructura, que se representa en la Figura 1, permite ordenar el análisis del sistema desde la adquisición de la información hasta su conversión en un servicio útil, ya sea para gestores o conductores.

- **Unidad de captación:** es el elemento encargado de obtener la información del entorno. En los sistemas basados en sensores individuales esta función la cubren dispositivos como ultrasonidos, infrarrojos o magnetómetros, instalados de forma independiente plaza a plaza. Mientras que los basados en visión, la unidad de captación es una cámara, a la que hay que colocar a una posición elevada con un ángulo lo más adecuado posible, de manera que se pueden observar múltiples plazas al mismo tiempo.

De estas cuatro unidades, la de captación se puede considerar la parte más crítica del sistema, ya que lo que las cámaras captan es la entrada con la que va a trabajar el resto del sistema, y partiendo de eso se ajustan los demás componentes (capacidad de cálculo, ancho de banda, tipo de comunicación, etc.) para que todo quede bien dimensionado. Es la unidad que se enfrenta directamente a los retos propios del aparcamiento al aire libre. Como las oclusiones, la variabilidad de la iluminación o las condiciones meteorológicas adversas, retos que se analizan en mayor detalle en el apartado 2.2.3. Por ello, es interesante comenzar a construir la arquitectura desde aquí, ajustando el resto del sistema en consecuencia de sus necesidades.

Entre los parámetros que se deben definir en esta unidad, la altura y el ángulo de la cámara son posiblemente los más determinantes en aparcamientos al aire libre. Estos dos parámetros condicionan directamente el número de plazas que se pueden cubrir con una sola cámara, la resolución que se consigue sobre cada vehículo y la cantidad de oclusiones entre coches contiguos, y por tanto definen en gran medida la calidad de la imagen con la que va a trabajar el algoritmo de procesamiento. El compromiso técnico entre estos factores y lo que implican desde un punto de vista teórico se analiza en el apartado correspondiente de este capítulo y su aplicación al caso concreto del campus de la UEM se trata en el capítulo 5.

Paidi et al. (2018) realizan una revisión de las tecnologías de sonorización aplicables a aparcamientos al aire libre, comparando magnetómetros, ultrasonidos y soluciones basadas en visión artificial. Concluyen que una combinación de visión por computador con redes neuronales convolucionales es la opción más adecuada en este tipo de entornos, sobre todo por su menor coste de despliegue y por aguantar mejor las condiciones cambiantes del exterior. Esta combinación coincide con el enfoque del trabajo, que se ha basado en cámaras y modelos de detección de objetos mediante redes neuronales convolucionales, cuyas distintas alternativas se analizan y comparan en los apartados siguientes. La Tabla 2 resume la comparativa entre los dos enfoques (cámaras frente a sensores individuales) desde los principales criterios de diseño.

Criterio de diseño	Cámara	Sensores individuales
Cobertura por dispositivo	Múltiples plazas	Una plaza
Coste de despliegue	Reducido (pocos puntos de instalación)	Elevado (un dispositivo por plaza)
Instalación	Se puede aprovechar puntos elevados ya existentes como farolas	Requiere de obra civil o instalación en superficie
Mantenimiento	Centralizado en pocos puntos	Distribuido a lo largo de todo el aparcamiento
Información extraíble	Ocupación, tipo de vehículo, incidencias, eventos	Solo señal binaria (libre/ocupado)
Flexibilidad ante cambios de distribución	Alta (basta con reconfigurar zonas en software o reubicar las cámaras)	Baja (hay que reubicar múltiples sensores físicos)
Adecuación a aparcamiento exterior	Buena, aunque hay que abordar los retos de visión adaptando el sistema	Aceptable, aunque al estar a nivel de suelo quedan expuestos a suciedad y desgaste por uso

Tabla 2. Comparativa entre captación por cámaras y captación por sensores individuales. Elaboración propia a partir de Paidi et al. (2018)

- **Unidad de procesamiento:** Es la encargada de interpretar la información que llega de las cámaras y de generar la salida principal del sistema, es decir, el estado de ocupación de cada plaza y la detección de incidencias relevantes.

Para realizar esta interpretación existen distintos enfoques dentro del campo de la visión artificial, que van desde técnicas más clásicas basadas en el tratamiento directo de la imagen hasta planteamientos más modernos basados en el aprendizaje automático. Cada uno de estos enfoques tiene ventajas y limitaciones según el tipo de entorno y la complejidad de la escena, y en muchos casos se pueden incluso combinar para sacar partido de ambos.

Otra cuestión importante en esta unidad es dónde se ejecuta físicamente el procesamiento, ya que puede hacerse en un servidor remoto, en el propio nodo junto a la cámara o adoptando un enfoque híbrido entre ambos. Cada opción tiene consecuencias directas en latencia, ancho de banda, privacidad y consumo energético, y todos estos factores afectan la viabilidad del despliegue.

- **Unidad de comunicación:** Es la encargada de transmitir los resultados del procesamiento hacia el sistema central de gestión o directamente hacia los usuarios finales.

Las tecnologías que se pueden emplear son muy variadas e incluyen tanto comunicaciones cableadas como inalámbricas, cada una con sus propias características en cuanto a ancho de banda disponible, alcance, consumo energético y coste de despliegue.

La elección de una u otra depende sobre todo del tipo de información que se quiera enviar y del entorno donde se vaya a instalar el sistema, ya que no es lo mismo transmitir un flujo de vídeo continuo que enviar simplemente el estado de cada plaza en formato de texto, ni es lo mismo desplegar la red en un *parking* cubierto con infraestructura cableada disponible que hacerlo en

uno al aire libre sin acceso fácil a ella. Las distintas alternativas y sus implicaciones se tratan con mayor detalle más adelante.

- **Unidad de gestión/aplicación:** Es donde la información que generan las unidades anteriores se convierte en un servicio útil, ya sea para que los gestores del aparcamiento tomen decisiones o para que los conductores encuentren plaza más fácilmente.

Esta unidad puede tener distintas formas según el destinatario y el caso de uso. Para los gestores del aparcamiento, lo habitual es un panel de control web con visualización del estado en tiempo real, históricos de ocupación, estadísticas y notificación de incidencias detectadas. Para los conductores, la información puede llegar a través de una aplicación móvil que indique la disponibilidad antes de entrar al aparcamiento, o mediante paneles informativos dinámicos colocados en puntos estratégicos del propio aparcamiento o de sus accesos.

Además, esta unidad puede integrarse con otros sistemas externos, como sistemas de control de acceso por barrera o plataformas de gestión urbana más generales, lo que permite extender el alcance del servicio más allá del aparcamiento en sí.

2.1.3 Métricas de evaluación de sistemas de Smart *Parking*

Para evaluar el rendimiento de un SPS hay que disponer de criterios de medida que permitan saber si el sistema funciona bien, identificar en qué condiciones falla más y compararlo con otras soluciones.

Las métricas para evaluar el desarrollo de este trabajo se pueden agrupar en dos bloques. Por un lado, están las que miden la calidad de la detección, es decir, si el sistema acierta o no a la hora de saber el estado de cada plaza. Por otro lado, están las métricas operacionales, que miden el comportamiento desde un punto de vista más práctico, como la estabilidad en el tiempo o la velocidad de respuesta.

Para las métricas de detección, se parte de la matriz de confusión, que es una tabla que clasifica cada predicción del sistema en cuatro tipos según los aciertos o fallos:

- **Verdadero positivo (VP):** el sistema detecta una plaza como ocupada y efectivamente lo está.
- **Verdadero negativo (VN):** el sistema detecta una plaza como libre y efectivamente lo está.
- **Falso positivo (FP):** el sistema marca como ocupada una plaza que en realidad está libre.
- **Falso negativo (FN):** el sistema marca como libre una plaza que en realidad está ocupada.

Con estos cuatro valores ya se pueden calcular las métricas que se usan habitualmente para medir la calidad de la detección, que se resumen en la Tabla 3.

Métrica	Definición	Qué mide
Exactitud	(VP + VN) / Total	Porcentaje global de aciertos sobre el total de plazas evaluadas
Precisión	VP / (VP + FP)	De todas las plazas marcadas como ocupadas, cuántas lo están realmente
Sensibilidad	VP / (VP + FN)	De todas las plazas que están ocupadas, cuántas detecta el sistema
F1-Score	2 · (Precisión · Sensibilidad) / (Precisión + Sensibilidad)	Equilibrio entre precisión y sensibilidad

Tabla 3. Métricas de calidad de la detección habituales en SPS basados en visión. Elaboración propia

Cada una de estas métricas aporta información distinta. La exactitud es la más intuitiva y da una idea general del rendimiento, pero conviene tener cuidado con ella cuando una clase predomina sobre otra, ya que un aparcamiento mayoritariamente vacío puede dar valores altos sin que el sistema esté realmente acertando con las plazas ocupadas. La precisión y la sensibilidad, que también se llaman *recall,* permiten ver con más detalle qué tipo de errores comete el sistema, y esto es importante porque en un SPS no todos los errores tienen la misma consecuencia. Un falso positivo hace que el conductor se desvíe hacia un sitio donde en realidad sí podría aparcar, mientras que un falso negativo haría que el conductor se mueva hasta una plaza que en realidad está ocupada, empeorando el problema del *cruising.* El F1-score se utiliza cuando se quiere un único valor que equilibre los dos aspectos, por ejemplo, para comparar varios modelos entre sí de forma rápida sin tener que mirar dos columnas a la vez.

Como referencia, los trabajos basados en *datasets* como PKLot (Almeida et al., 2015) o CNRPark-EXT (Amato et al., 2017) suelen alcanzar valores de exactitud por encima del 95 %, aunque hay que tener en cuenta que estos resultados se obtienen en condiciones controladas y no siempre se trasladan tal cual a despliegues reales.

Métricas operacionales. Estas métricas evalúan aspectos prácticos del sistema más allá de si acierta o no con la detección, y en aparcamientos al aire libre son tan importantes como las anteriores. Las más relevantes son las siguientes:

- **Estabilidad temporal:** mide si el estado detectado para cada plaza se mantiene constante entre *frames* consecutivos. Un sistema que oscila entre libre y ocupada de un *frame* al siguiente, aunque tenga buena exactitud global, resulta poco fiable de cara al usuario

final. Por eso muchas veces se evalúa el porcentaje de cambios espurios o se aplican técnicas de suavizado temporal para reducirlos, siendo lo normal que se tenga que confirmar durante varias lecturas para dar algo por ocupado o libre.

- **Latencia:** tiempo que pasa desde que la cámara capta una imagen hasta que el sistema entrega el resultado. Es especialmente importante cuando el procesamiento se hace en el borde, donde la capacidad de cómputo es limitada, y cuando la información se utiliza para guiar a conductores en tiempo real.
- **Robustez frente a condiciones ambientales**: evalúa cómo se mantiene el rendimiento del sistema en distintas situaciones como cambios de iluminación, sombras, lluvia, niebla u oclusiones parciales. En aparcamientos al aire libre estas condiciones se dan con frecuencia, así que un sistema que funcione muy bien con buen tiempo pero falle con sombras o reflejos tiene poca utilidad práctica.

A diferencia de las métricas de detección, las operacionales no siempre se reducen a un único número, sino que se suelen analizar de forma combinada y, en muchos casos, de forma cualitativa a partir de pruebas en distintos escenarios.

2.2 Visión artificial aplicada a la detección de ocupación en aparcamientos

En este apartado, la visión artificial se estudia como una herramienta para estimar la ocupación de plazas en un aparcamiento exterior a partir de imágenes captadas por cámaras fijas. Centrándose en los aspectos que condicionan las decisiones del sistema, como: cómo se puede detectar la ocupación, cómo se ajusta y evalúa un modelo de detección, qué problemas aparecen en exteriores y cómo influye la geometría de captación.

La idea principal que debe tenerse en cuenta es que detectar vehículos no equivale automáticamente a conocer la ocupación del aparcamiento. Para obtener información útil plaza por plaza es necesario transformar la imagen en detecciones, asociar cada vehículo con una plaza concreta y estabilizar esa decisión en el tiempo. Esta diferencia entre la detección visual y la decisión de ocupación es la base que se busca para luego entender el desarrollo del trabajo.

2.2.1 Enfoques para detectar la ocupación de aparcamientos

Al aplicar estas técnicas en concreto a la detección de ocupación en aparcamientos aparecen dos formas principales de plantear el problema.

- **Enfoque orientado a la plaza.** Se definen previamente las zonas que ocupa cada plaza en la imagen y el sistema analiza cada una por separado para decidir si está libre u ocupada. El análisis por ROI encaja en este planteamiento, ya que observa directamente la región de cada plaza y decide a partir de sus características visuales. Su ventaja es que es sencillo y directo, y permite estimar la ocupación sin necesidad de detectar explícitamente vehículos. Como inconvenientes, exige definir la geometría de las plazas, a mano o de forma automática, y puede confundirse con cualquier elemento que altere la apariencia de la plaza, como sombras, peatones, hojas o reflejos. Es decir, detecta cambios dentro de la plaza, pero no entiende qué los produce.

- **Enfoque orientado al vehículo.** En vez de analizar cada plaza, se detectan primero los vehículos presentes en la escena mediante un modelo de detección, y después se decide qué plazas están ocupadas relacionando cada vehículo con su plaza. Esa asociación puede hacerse mediante criterios geométricos, como el solapamiento entre la caja y la región de la plaza, la posición del centro del vehículo o algún punto más representativo según la perspectiva. Aporta una interpretación más fiable, porque comprueba si lo que ocupa una zona es realmente un vehículo, pero depende de dos factores: que el detector localice bien los coches y que la lógica de asociación sea adecuada. En vistas oblicuas, por ejemplo, el centro de la caja puede no ser el punto más representativo para decidir qué plaza ocupa un coche, por lo que pueden requerirse reglas adaptadas a la perspectiva.

Los dos enfoques no son excluyentes y pueden combinarse. Un sistema puede partir de plazas definidas y, a la vez, usar detección de vehículos para no confundirse con elementos que no son coches. También puede apoyarse en información temporal o en el seguimiento para estabilizar la decisión y evitar cambios bruscos entre libre y ocupado. En este TFG se utiliza una combinación con un enfoque orientado a la plaza, que sirve como referencia inicial mediante regiones de interés, mientras que la línea principal se basa en detectar vehículos y asociarlos con las plazas. Con ello se consigue que el sistema completo no se reduzca a un detector, sino que combine captación, detección, asociación geométrica y decisión final sobre la ocupación.

2.2.2 Modelos de detección, entrenamiento y evaluación

Para detectar vehículos en la imagen se emplean modelos de detección de objetos. A diferencia de una clasificación simple, estos modelos no solo indican si aparece un vehículo, sino también su posición aproximada mediante una caja delimitadora. Esta información resulta especialmente útil en un aparcamiento, ya que permite relacionar cada vehículo detectado con las regiones que representan las plazas.

Dentro del aprendizaje profundo, o *deep learning*, los modelos aprenden a partir de datos en lugar de depender únicamente de reglas definidas manualmente. IBM lo describe como un subconjunto del aprendizaje automático basado en redes neuronales multicapa, inspiradas en la estructura del cerebro humano (IBM, 2024). Esta idea es relevante en el proyecto porque el sistema no se programa indicando todas las posibles apariencias de un coche, sino que el modelo aprende patrones visuales a partir de imágenes anotadas.

De forma general, los detectores de objetos pueden dividirse en dos familias:

- Por un lado, los detectores de dos etapas, como Faster R-CNN, primero proponen regiones de la imagen donde podría haber un objeto y después clasifican y ajustan esas regiones. Este planteamiento suele ofrecer buenos resultados, pero implica un mayor coste computacional, ya que el proceso se divide en varias fases. Faster R-CNN, por ejemplo, incorpora una red de propuesta de regiones para generar candidatos antes de la detección final (Ren et al., 2015).
- Los detectores de una etapa, como YOLO, predicen directamente las cajas y las clases en una sola pasada por la red. Por ello suelen ser más adecuados cuando se busca un

> equilibrio entre precisión y velocidad, especialmente en aplicaciones que deben actualizar información de forma periódica o trabajar sobre hardware limitado. El planteamiento original de YOLO formulaba la detección como un único problema de regresión desde la imagen completa hasta las cajas y probabilidades de clase, lo que permitió obtener detección en tiempo real (Redmon et al., 2016).

En este proyecto se opta por centrar el desarrollo en la familia YOLO, ya que encaja mejor con una arquitectura de procesamiento en el borde y con la necesidad de analizar imágenes de forma periódica sin transmitir vídeo continuo. Los modelos de dos etapas no se descartan como alternativa general, pero se consideran menos adecuados para la solución principal por su mayor carga computacional. En una evolución futura del sistema podrían valorarse de nuevo si se buscara maximizar la precisión en zonas concretas o si se dispusiera de mayor capacidad de cálculo.

El ajuste de un modelo de detección de este tipo requiere imágenes anotadas, en las que cada vehículo aparece marcado con una caja delimitadora y una clase asociada. A partir de estas anotaciones, el modelo compara sus predicciones con la referencia real y modifica sus parámetros durante el entrenamiento. Para comprobar que el modelo no aprende únicamente las imágenes utilizadas para entrenar, el conjunto de datos se divide normalmente en imágenes de entrenamiento, validación y prueba, lo que permite evaluar su comportamiento sobre imágenes no vistas.

En este tipo de proyectos es habitual partir de un modelo previamente entrenado sobre grandes conjuntos de datos y adaptarlo a un caso concreto mediante aprendizaje por transferencia o *fine-tuning*. Este enfoque reduce la cantidad de imágenes necesarias respecto a un entrenamiento desde cero y permite especializar el detector en condiciones concretas, como las cámaras, perspectivas, distancias y oclusiones del aparcamiento estudiado. Para comprobar la transferencia, se realiza una evaluación con métricas específicas de detección. Además de las métricas generales vistas en el apartado 2.1.3, como precisión o sensibilidad, en la detección de objetos se emplea la precisión media promedio, conocida como mAP. Esta métrica resume la calidad de las detecciones teniendo en cuenta si el modelo encuentra los objetos y en qué medida las cajas predichas se solapan con las reales. En el contexto de este proyecto también interesa analizar de forma separada los falsos positivos, los falsos negativos y el comportamiento sobre vehículos pequeños o lejanos, ya que estos casos suelen ser los más problemáticos cuando una cámara cubre varias filas de plazas.

En la práctica, el comportamiento de un detector ya entrenado depende de tres parámetros de inferencia. El primero es el tamaño de entrada (imgsz), la resolución a la que la red procesa cada imagen, que determina con cuántos píxeles se representa cada objeto y resulta decisivo para los vehículos pequeños o lejanos. El segundo es el umbral de confianza (conf), la seguridad mínima que debe tener una detección para aceptarse, donde un valor bajo recupera objetos difíciles a cambio de admitir detecciones menos fiables. El tercero interviene en la supresión de no máximos (NMS, Non-Maximum Suppression, el filtrado de detecciones repetidas sobre un mismo objeto), y se apoya en la intersección sobre la unión (IoU, Intersection over Union), que mide el grado de solape entre dos cajas como el cociente entre el área que comparten y el área que ocupan en conjunto. Un umbral de IoU bajo en el NMS fusiona con facilidad cajas próximas, lo que puede

unir dos vehículos pegados en una sola detección, mientras que uno alto las mantiene separadas. La IoU se utiliza además como criterio para decidir si una detección coincide con una caja real durante la evaluación, y es la base del cálculo del mAP. El ajuste de estos tres parámetros para el caso de estudio se desarrolla en el capítulo 4.

2.2.3 Retos de la visión en aparcamientos al aire libre

Aunque las técnicas actuales han avanzado mucho, aplicarlas en un aparcamiento al aire libre sigue planteando retos importantes que no aparecen en entornos cerrados y controlados. Los principales retos que más afectan al rendimiento real del sistema son los siguientes:

- **Variabilidad de la iluminación:** A lo largo del día, la luz cambia mucho, desde el amanecer hasta la noche. Pasando por el sol fuerte del mediodía o contraluces. Estos cambios afectan al aspecto de los coches y del suelo, y pueden hacer que un sistema que funciona bien a una hora falle a otra. De noche, además, el contraste baja y hay que depender de la iluminación artificial, que no siempre llega de forma uniforme.

- **Sombras:** En exterior pueden ser muy marcadas, y una sombra fuerte sobre una plaza vacía puede confundir al sistema y hacer que la detecte como ocupada. Afecta sobre todo a los métodos clásicos basados en intensidad, mientras que los de aprendizaje profundo aguantan mejor si han visto situaciones parecidas durante el entrenamiento.

- **Condiciones meteorológicas:** La lluvia deja gotas y reflejos sobre el suelo, la niebla reduce la visibilidad a distancia y la nieve puede cubrir parcialmente los coches o las marcas de las plazas.

- **Oclusiones:** En una vista oblicua, los coches más cercanos pueden tapar parcialmente a los más lejanos, y en aparcamientos exteriores pueden intervenir otros elementos como farolas, señales, papeleras o vegetación. En el caso de la UEM, los árboles son una limitación especialmente importante, ya que pueden ocultar parte de las plazas y reducir la información disponible para el sistema.

- **Perspectiva y resolución sobre objetos lejanos:** Es un reto propio de las cámaras elevadas que cubren muchas plazas. Las plazas cercanas se ven grandes y con detalle, mientras que las lejanas se ven pequeñas y borrosas, lo que hace que el sistema funcione mejor en unas zonas que otras.

- **Factores que afectan a largo plazo.** La suciedad sobre la lente puede ir degradando la imagen con el tiempo, lo que obliga a un mantenimiento periódico, y el viento o las vibraciones del poste pueden producir pequeños movimientos que afectan a la estabilidad de las detecciones.

Tal y como se ha visto, los métodos basados en aprendizaje profundo combinados con redes convolucionales tienden a aguantar mejor toda esta variabilidad que los métodos clásicos, que son más sensibles a cualquier cambio del entorno. En cualquier caso, ningún enfoque es

completamente inmune a estos retos, y abordarlos con criterio forma parte del diseño y la validación del sistema.

2.2.4 Geometria de captación

La geometría de captación es uno de los factores que más condicionan el rendimiento de un sistema de visión en aparcamientos. Aunque el modelo de detección sea adecuado, si la cámara no está colocada correctamente, la imagen puede no contener suficiente información para decidir la ocupación de las plazas. Siendo necesario considerar la altura de instalación, el ángulo de inclinación, el campo de visión de la lente y la resolución efectiva sobre los vehículos antes de definir el despliegue.

La altura influye directamente en la cobertura y en las oclusiones. Una cámara colocada a poca altura puede ofrecer mucho detalle en las plazas cercanas, pero los vehículos de las primeras filas pueden ocultar parcialmente los situados detrás. Al aumentar la altura, la escena se observa con una perspectiva más favorable y se reduce parte de ese solapamiento visual, lo que permite cubrir más plazas con una sola cámara. Sin embargo, este aumento de cobertura tiene una consecuencia: los vehículos aparecen más pequeños en la imagen, especialmente en las zonas alejadas.

El ángulo de inclinación también afecta al resultado. Una vista demasiado horizontal permite observar zonas lejanas, pero aumenta las oclusiones y hace que los vehículos cercanos tapen a los del fondo. Una vista más vertical reduce este problema y facilita la asociación entre vehículo y plaza, aunque puede limitar la superficie cubierta si la cámara no está suficientemente alta. Por ello, en un aparcamiento exterior suele buscarse una perspectiva intermedia, que permita cubrir varias filas de plazas sin que los vehículos lejanos pierdan demasiado detalle.

Lo que abarca la cámara depende también de la lente, que en concreto afecta su campo de visión, conocido como FOV (Field of View), que es el ángulo de escena que la cámara es capaz de captar. El campo de visión va ligado a la distancia focal de la lente, de modo que una focal corta da un campo de visión amplio, como el de una lente gran angular, que abarca mucha escena pero hace que cada objeto se vea más pequeño, mientras que una focal larga da un campo de visión estrecho, que abarca menos pero acerca y agranda los objetos. En un aparcamiento esto significa que con una lente gran angular se cubren más plazas con una sola cámara, a costa de que cada coche ocupe menos píxeles, mientras que una lente más cerrada cubre menos plazas pero las ve con más detalle. Elegir la lente es, por tanto, otro equilibrio entre cuánto se abarca y con cuánto detalle.

La anchura de escena cubierta a una distancia determinada puede aproximarse mediante una relación trigonométrica. A una distancia D, cada mitad del campo horizontal de visión dibuja un triángulo rectángulo cuyo lado opuesto es la mitad del ancho de escena. Por ello, el ancho total visible puede expresarse como:

$$W = 2 \cdot D \cdot tan(\frac{\theta}{2})$$

donde W es la anchura visible de la escena, D es la distancia desde la cámara hasta la zona observada y θ es el campo horizontal de visión. Esta expresión permite entender que, a medida que aumenta la distancia, también aumenta la anchura de escena incluida en la imagen, pero cada objeto ocupa una proporción menor de los píxeles disponibles.

Para estimar la profundidad visible en el plano vertical puede usarse una relación equivalente entre la altura de la cámara y el ángulo con el que el rayo de visión corta el suelo. La cámara y el punto del suelo observado forman un triángulo rectángulo, con la altura de instalación como lado vertical y la distancia horizontal al punto observado como lado horizontal. De esta forma:

$$d = \frac{h}{tan(\alpha)}$$

donde d es la distancia horizontal hasta el punto del suelo observado, h es la altura de la cámara y α es el ángulo del rayo respecto a la horizontal. Esta expresión es útil para estimar la primera zona visible, la posición aproximada del eje óptico y el límite geométrico del encuadre. No obstante, este límite solo indica que una zona aparece dentro de la imagen, no que los vehículos situados allí puedan detectarse con fiabilidad.

A partir de estas relaciones aparece el concepto de resolución efectiva. No basta con que una plaza aparezca dentro del encuadre, el vehículo tiene que poder ocupar suficientes píxeles para que el modelo pueda detectarlo. Si se conoce la anchura real visible de la escena y la resolución horizontal de la imagen procesada, el tamaño aproximado de un vehículo en píxeles puede estimarse mediante una proporción. Todo el ancho visible W se reparte entre N_{px} píxeles horizontales, por lo que un vehículo de anchura aproximada a_v ocupará la parte proporcional correspondiente a su anchura:

$$p_x = (\frac{a_v}{W}) \cdot N_{px}$$

donde p_x es la anchura aproximada del vehículo en píxeles, a_v es la anchura real del vehículo, W es la anchura real de la escena a esa distancia y N_{px} es la resolución horizontal de la imagen procesada. Esta relación se utiliza como criterio práctico para comprobar si una cámara mantiene suficiente detalle en las zonas más alejadas.

En el ámbito de la videovigilancia existe un estándar europeo, conocido como DORI (Detection, Observation, Recognition, Identification), recogido en la norma IEC/EN 62676-4, que fija cuántos píxeles por metro hacen falta para cada tarea, desde 25 px/m para detectar la presencia de un objeto hasta 250 px/m para identificarlo (IEC, 2015). En este proyecto no se pretende identificar vehículos ni matrículas, sino detectar la presencia de ocupación. En consecuencia, la colocación de una cámara no puede decidirse únicamente por la superficie que aparece en la imagen. Hay que comprobar también la resolución disponible sobre los vehículos, las oclusiones entre filas, la deformación en los bordes de la imagen y la relación entre cada vehículo detectado y la plaza que ocupa. Este análisis geométrico se aplica posteriormente en el apartado 5.6 para

justificar la selección de cámara, la altura de montaje, la inclinación y la distancia útil de detección en el caso del aparcamiento de la UEM.

Elegir la altura y el ángulo es, entonces, buscar un compromiso entre la cobertura, que es cuántas plazas se ven, la resolución sobre cada plaza, que marca hasta qué distancia se detecta bien, y las oclusiones, que es cuánto se tapan unos coches a otros. Esto se aprecia en la Figura 2, que compara sobre un aparcamiento real tres ángulos de cámara distintos, desde una vista casi horizontal hasta una vista cenital, donde se ve cómo al subir el punto de vista las plazas se distinguen mejor y los coches se tapan menos entre sí (Chen et al., 2020). No hay una combinación perfecta, sino la que mejor encaja en cada aparcamiento según su tamaño, la disposición de las plazas y los soportes disponibles. La configuración concreta para el caso del campus de la UEM, partiendo de estos criterios, se analiza en el Capítulo 5 durante el desarrollo del proyecto.

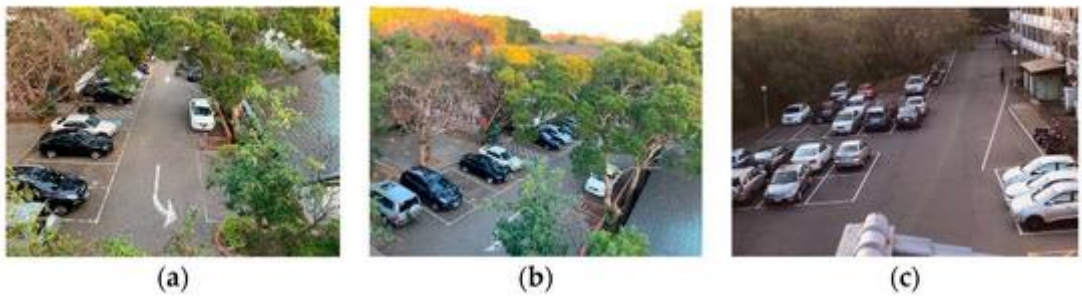


Figura 2.Comparación de distintos ángulos de cámara sobre un aparcamiento: vista horizontal, vista lateral y vista cenital. De Chen et al.

2.3 Arquitectura de procesamiento del sistema

Cuando la cámara capta las imágenes y ya se sabe con qué técnicas analizarlas, queda una decisión importante: determinar dónde se realiza ese procesamiento y cómo se transmite la información. No es lo mismo que el análisis se haga en el propio dispositivo de la cámara que en un servidor remoto, y esa elección afecta al coste, a la velocidad de respuesta, al consumo de red y a la privacidad. En este apartado se explican las tres arquitecturas de procesamiento, que son la nube, el borde y la combinación de ambas, y después las opciones para transmitir los datos.

2.3.1 Procesamiento en la nube

En el procesamiento en la nube, conocido como *cloud computing*, las imágenes que capta la cámara se envían a un servidor remoto, normalmente un centro de datos, donde se hace todo el análisis. La cámara funciona solo como un sensor que captura y transmite, mientras que el trabajo pesado lo hace una máquina potente a distancia.

Su principal ventaja es la capacidad de cálculo. Un servidor en la nube puede ejecutar modelos muy grandes y precisos que no cabrían en un dispositivo pequeño, y es fácil de escalar, ya que añadir más cámaras o más capacidad resulta sencillo. También centraliza el almacenamiento y el mantenimiento en un único punto.

El inconveniente es que enviar vídeo de forma continua sale caro. Consume mucho ancho de banda, porque las imágenes ocupan bastante y se generan sin parar, e introduce latencia, que es el retardo entre que se capta la imagen y se obtiene la respuesta, algo que puede ser un problema cuando hay que reaccionar rápido. A esto se suman las dudas de privacidad, ya que las imágenes viajan a un servidor externo, lo que es delicado cuando pueden aparecer personas o matrículas.

2.3.2 Procesamiento en el borde (*edge computing*)

En el procesamiento en el borde, conocido como *edge computing*, el análisis se hace en el propio dispositivo de captación o en un equipo cercano a la fuente de datos, sin enviar el vídeo a un servidor remoto (Shi et al., 2016). Este enfoque resuelve buena parte de los problemas del anterior. Como el vídeo no tiene que viajar, la latencia baja mucho y el sistema responde casi en tiempo real, algo útil cuando hay que actualizar el estado de las plazas de forma continua. También reduce el consumo de red, porque en lugar de imágenes solo se transmite el resultado del análisis, que ocupa muy poco, por ejemplo el número de plazas libres. Y mejora la privacidad, ya que las imágenes pueden analizarse y descartarse en el sitio sin salir nunca de él.

A cambio tiene una limitación clara, que es la capacidad de cálculo. Los dispositivos de borde suelen ser pequeños y de bajo consumo, así que no pueden ejecutar los modelos más grandes, sino versiones más ligeras, lo que puede suponer algo menos de precisión. También complica un poco el mantenimiento, porque hay que gestionar muchos dispositivos repartidos en lugar de un solo servidor central.

2.3.3 Arquitecturas híbridas

En la práctica, las dos opciones anteriores no son excluyentes, y muchos sistemas reales las combinan en lo que se conoce como un enfoque híbrido. La idea es repartir el trabajo, de modo que lo que necesita respuesta rápida y genera muchos datos se hace en el borde, y lo que requiere más potencia o una visión de conjunto se deja para la nube.

En un sistema de aparcamiento, esto podría traducirse en que cada cámara analiza sus plazas y decide su estado en el borde, y luego envía solo esa información resumida a un servidor central, donde se juntan los datos de todas las cámaras, se almacenan, se muestran al usuario y se pueden ejecutar análisis más complejos como estadísticas de ocupación a lo largo del tiempo. Así se aprovecha la rapidez y el bajo consumo de red del borde junto con la capacidad y la centralización de la nube. Por ese equilibrio, es una de las opciones más razonables para sistemas con una gran cantidad de cámaras.

2.4 Bases hardware y tecnológicas del sistema

Hasta ahora se ha visto qué se quiere hacer, que es detectar la ocupación de las plazas mediante visión artificial, y dónde se puede procesar la información, en la nube, en el borde o de forma híbrida. En este apartado se explican los componentes físicos y tecnológicos que hacen posible ese sistema, es decir, qué tipo de cámara, qué hardware de procesamiento y qué medios de comunicación se utilizan, y qué condiciones de despliegue conviene tener en cuenta.

2.4.1 Cámaras y captación de imagen

La unidad de captación es, junto con el algoritmo, el elemento más decisivo del sistema, porque marca el punto de partida. Si la imagen no es buena, ningún modelo va a corregir esa carencia más adelante. Por eso conviene conocer los tipos de cámara que existen, qué parámetros las definen y qué características son críticas cuando van a instalarse en un entorno exterior.En sistemas de monitorización se utilizan habitualmente cámaras de red, también llamadas cámaras IP, que envían el vídeo digitalmente a través de la red en lugar de mediante una señal analógica. Esto facilita conectarlas a equipos de procesamiento y manejarlas a distancia. Según Axis Communications, las cámaras IP pueden clasificarse en distintos formatos según su carcasa y aplicación, entre los que destacan los tipo *bullet*, *dome*, PTZ, panorámicas, modulares, térmicas o las tipo *box*(Axis Communications, s.f.). Las más habituales son tres:

- Las cámaras tipo *bullet.* Con forma alargada, son comunes en exterior porque permiten orientarlas con facilidad y suelen llevar carcasa robusta.
- Las cámaras tipo *dome.* Con forma semiesférica, son más discretas y resultan útiles tanto en interior como en exterior, sobre todo donde se quiere disimular hacia dónde miran.
- Las cámaras PTZ (pan-tilt-zoom, es decir, con movimiento horizontal, vertical y zoom motorizado). Permiten cambiar el encuadre a distancia y cubrir áreas grandes con una sola unidad, aunque a un coste mayor.

Los otros formatos cubren zonas muy amplias a costa de menos detalle por plaza, pero para un sistema fijo de detección de ocupación, los de tipo *bullet* o *dome* son suficientes y más económicos. La Tabla 4 resume las características principales de estos tres formatos de cámara y permite comparar de forma directa sus ventajas, limitaciones y posible aplicación dentro de un sistema de monitorización de aparcamientos.

Tipo	Forma y uso	Ventajas	Desventajas	Encaje con *parking*
Bullet	Alargada, visible y orientable.	Fácil de orientar y robusta en exterior	Más visible y menos discreta	Muy adecuada para farolas, postes o techo.
Dome	Semiesférica, discreta y protegida.	Menos impacto visual y buena contra manipulaciones	Ajuste de orientación menos cómodo según modelo.	Adecuada si se quiere que sea más discreta.
PTZ	Movimiento remoto y zoom motorizado	Permite cambio de encuadre y acercarse a zonas concretas	Más cara y compleja. Puede romper la relación cámara-plaza	Menos utilizada en ocupación fija, pero sirve como apoyo.

Tabla 4. Comparación de formatos principales de cámaras IP para monitorización exterior. Elaboración propia

Los parámetros que definen el rendimiento de una cámara para visión artificial son varios y conviene conocerlos para poder justificar después la elección de un modelo concreto. Algunos ya se introdujeron al hablar de geometría y de imagen, así que aquí se enumeran de forma agrupada y centrada en lo que importa para la elección:

- **Sensor de imagen:** Es el chip que capta la luz y la convierte en señal eléctrica, y suele ser de tipo CMOS (Complementary Metal-Oxide-Semiconductor), una tecnología de fabricación de sensores muy extendida por su bajo consumo. Determina la calidad básica de la captura, y su tamaño (por ejemplo 1/2,8 pulgadas) influye en la sensibilidad a la luz, ya que un sensor más grande recoge más luz por píxel y se comporta mejor en condiciones difíciles.
- **Resolución:** Define cuántos píxeles tiene la imagen, por ejemplo Full HD o 4K. Una mayor resolución permite ver con detalle plazas más lejanas, pero también genera más datos a procesar, así que hay que elegirla en función de la distancia a cubrir y de la capacidad del equipo que la va a analizar.
- **Campo de visión (FOV) y lente:** Como se vio en el 2.2.4, el FOV depende de la distancia focal de la lente. Conviene saber si se prefiere una lente fija (focal cerrada) o varifocal (con focal ajustable tras instalación), ya que esto último da flexibilidad para afinar el encuadre en el sitio.
- **Sensibilidad a la luz y apertura:** La apertura del diafragma se mide con un valor f (por ejemplo f/1.6), que indica cuánta luz entra a través de la lente, donde valores más bajos significan más luz. Para entornos al aire libre, donde la iluminación cambia mucho a lo largo del día, conviene una cámara con buena apertura, y a veces es necesaria iluminación infrarroja propia para funcionar de noche.
- **Velocidad de fotogramas y compresión:** Los FPS, marcan la fluidez del vídeo, aunque para detección de ocupación no hace falta una tasa alta. Los códecs de compresión, que son los algoritmos que reducen el tamaño del vídeo para que ocupe menos, son habitualmente H.264 y H.265, este último más eficiente y útil cuando interesa reducir el tráfico de red.
- **Tipo de carcasa y formato:** Es el cuerpo físico de la cámara (*bullet*, *dome*, etc., como se ha visto arriba) y determina la facilidad de instalación, la discreción y la robustez.
- **Rango dinámico amplio (WDR):** Es la capacidad de la cámara para captar a la vez las zonas muy iluminadas y las muy oscuras de una misma escena sin que unas queden quemadas ni otras negras, y se expresa en decibelios. En exteriores es un parámetro relevante, porque los contraluces del amanecer y el atardecer y las sombras duras del mediodía, descritos en el apartado 2.2.3, generan justamente ese tipo de escenas con contraste extremo.

A estos parámetros se suman los requisitos específicos del exterior, que tienen tanta importancia como las propias prestaciones de la cámara. La norma internacional de grado de protección IP (Ingress Protection, es decir, protección frente a entrada de elementos), definida por la IEC, clasifica la resistencia de la carcasa del equipo frente a la entrada de polvo y de líquidos mediante un código de dos dígitos (IEC). En entornos exteriores expuestos es habitual recurrir a cámaras con grados altos, como IP66 o superiores, en función de cuánto vaya a estar expuesto el punto de

instalación. IP66 garantiza protección total contra polvo y resistencia a chorros de agua potentes, mientras que IP67 añade resistencia a inmersión temporal. Si la cámara se monta en un lugar accesible y existe riesgo de vandalismo, conviene además que cumpla la norma IK10 de resistencia a impactos mecánicos, que verifica que la carcasa soporta golpes de hasta 20 julios (IEC, 2002). También importa el rango de temperatura de operación, sobre todo si se va a desplegar en zonas con inviernos fríos o veranos extremos.

Con esta base, la elección de una cámara concreta para un proyecto pasa por cruzar los parámetros técnicos (resolución, FOV, sensibilidad) con las condiciones del entorno (exterior con protección adecuada, alimentación disponible, distancia a las plazas) y con el presupuesto. La posición y la cobertura de la cámara, que son las decisiones más importantes después del modelo, ya se trataron en el apartado 2.2.4.

2.4.2 Plataformas embebidas de procesamiento

Cuando el procesamiento se realiza en el borde, como se vio en el apartado 2.3, hace falta un equipo cercano a la cámara que ejecute el modelo. Ese equipo es lo que se conoce como una plataforma embebida, un pequeño ordenador compacto, de bajo consumo y pensado para integrarse en un sistema más grande, capaz de ejecutar tareas concretas como la inferencia de un modelo de visión. En los últimos años han aparecido distintas plataformas pensadas específicamente para inteligencia artificial en el borde, conocidas como dispositivos de *edge AI*, y conviene conocer sus características antes de elegir una.

A la hora de comparar estas plataformas hay varios criterios que importan:

- **Capacidad de cómputo.** Se suele medir en TOPS (*Tera Operations Per Second*, billones de operaciones por segundo), un indicador de cuántas operaciones de inferencia puede hacer el dispositivo por segundo.
- **Velocidad de inferencia.** Se mide en fotogramas por segundo procesados (FPS) con un modelo de referencia. Es más representativa que los TOPS en términos prácticos, ya que refleja lo que se va a obtener en uso real.
- **Consumo eléctrico.** En vatios, marca la energía que necesita y condiciona tanto el coste de operación como la viabilidad si la alimentación es limitada o autónoma. Hay que distinguir entre el consumo del propio acelerador y el consumo total del nodo, que incluye también la placa base, la cámara, la comunicación y la refrigeración.
- **Memoria disponible.** Suele ser memoria RAM (*Random Access Memory*, la memoria de trabajo temporal del equipo) de varios gigabytes, y determina qué modelos caben en el dispositivo. Los modelos más grandes necesitan más memoria, así que este parámetro puede limitar las opciones.
- **Coste de la plataforma.** Importante cuando hay que desplegar varias unidades, una por cámara o por grupo de cámaras.
- **Compatibilidad con *frame*works.** Determina lo fácil que será portar el modelo desde el entorno de desarrollo al dispositivo. Los *frame*works son librerías de software que se usan para construir y entrenar modelos de aprendizaje profundo, siendo los más extendidos PyTorch y TensorFlow.

Entre las plataformas más utilizadas en proyectos de visión por computador en el borde están las siguientes. La familia NVIDIA Jetson, con modelos como el Jetson *Nano* (más básico), el Jetson Orin *Nano* (gama media) y el Jetson Orin NX o AGX Orin (gama alta), ofrece una GPU integrada (Graphics Processing Unit, una unidad de procesamiento gráfico capaz de hacer muchos cálculos en paralelo, lo que la hace muy eficiente para redes neuronales) que permite ejecutar modelos de detección como los de tipo YOLO con bastante soltura, a costa de un consumo que va de unos pocos vatios en los modelos pequeños hasta más de 25 W en los más potentes. En el lado más económico y eficiente está la combinación de una Raspberry Pi con un acelerador de inferencia, como el Google Coral USB o el módulo Hailo, que añaden capacidad de cálculo específica para redes neuronales. Estos aceleradores pueden ofrecer inferencia con consumos de pocos vatios en el propio acelerador, aunque el consumo total del nodo dependerá también del resto de componentes. Suelen estar limitados a ejecutar modelos previamente cuantizados, lo que los hace más rígidos en cuanto a qué modelos pueden correr.

Además del hardware, el rendimiento en el borde depende mucho de la optimización del modelo. En plataformas embebidas suele ser necesario convertir el modelo a formatos específicos, aplicar cuantización (reducir la precisión numérica de los pesos para que el modelo sea más ligero) o utilizar motores de inferencia optimizados, que son herramientas que adaptan y aceleran un modelo ya entrenado para que corra de forma eficiente en un hardware concreto. Los más conocidos son TensorRT en las plataformas NVIDIA y TensorFlow Lite en Coral, además de las herramientas propias del acelerador empleado. Sin esta optimización, un modelo que funciona bien en un ordenador en el que se desarrolla puede no ser viable en un dispositivo de borde. Por ello, la elección de la plataforma no depende solo de los TOPS declarados, sino también de la facilidad para exportar el modelo entrenado sin perder precisión ni modificar demasiado la cadena de desarrollo.

Para dar una idea concreta del compromiso entre potencia y consumo, un estudio reciente comparó varias de estas plataformas ejecutando un modelo YOLOv5 de detección de objetos (Minott et al., 2025). La Tabla 5 resume los resultados, donde se ve a primera vista que el Jetson Orin NX casi dobla en velocidad a la Raspberry Pi 5 con Coral, aunque a costa de algo más de consumo, y que la Raspberry Pi alcanza una temperatura bastante más alta que las dos Jetson.

Plataforma	Velocidad (FPS)	Consumo(W)	Temperatura
Jetson Orin NX	41,8	10,6	45 °C
Raspberry Pi 5 + Coral TPU	21,5	8,3	80 °C (sin refrigeración activa)
Jetson *Nano*	<20	~7	42 °C

Tabla 5. Comparación de plataformas edge AI ejecutando un modelo YOLOv5.

Estas cifras corresponden al estudio citado, en condiciones concretas de prueba, y dejan claro que ninguna plataforma es la mejor en todo a la vez. La elección depende del modelo que se quiera ejecutar, del número de cámaras que debe atender cada dispositivo y de las restricciones de coste, consumo y temperatura del despliegue. La Raspberry Pi 5 con Coral resulta muy eficiente en consumo y coste, aunque su temperatura sin refrigeración activa apunta a la necesidad de disipación adicional en exterior, mientras que el Jetson Orin NX ofrece más potencia con una temperatura estable, a cambio de un coste mayor. Con esta información se dispone de una base para comparar plataformas según los criterios anteriores.

El estudio no incluye la gama intermedia Jetson Orin *Nano*, situada entre el *Nano* y el Orin NX de la tabla, pero permite acotar el rango de rendimiento, consumo y temperatura en el que se mueve esa familia, que es lo que se necesita para la selección del capítulo 5.

2.4.3 Comunicación e integración del sistema

Un sistema de monitorización de aparcamientos no es una cámara aislada, sino una arquitectura distribuida en la que varios nodos (cámaras y plataformas de procesamiento) tienen que coordinarse entre sí y comunicarse con un sistema central de gestión que recoge los resultados y los pone a disposición del usuario. Por eso, además de captar y procesar, hay que decidir cómo se transmite la información. Esta forma de organizar el sistema, basada en sensores distribuidos y un nivel central de gestión, es la habitual en arquitecturas IoT aplicadas al ámbito del *smart parking* (Pham et al., 2015).

Como ya se vio en el apartado 2.3, cuando el procesamiento se hace en el borde basta con transmitir metadatos en lugar de vídeo. En el caso de un aparcamiento, esos metadatos pueden ser tan simples como el estado de cada plaza (libre u ocupada), una marca de tiempo y un identificador del nodo, es decir, unos pocos bytes cada cierto tiempo en lugar de un flujo continuo. Esto abre la puerta a usar tecnologías de comunicación de bajo ancho de banda, abarata los costes de red y facilita el despliegue en zonas con conectividad limitada.

Para transmitir esos metadatos existen distintos protocolos a nivel de aplicación. Uno de los más extendidos en sistemas IoT (*Internet of Things*, objetos cotidianos conectados a internet) es MQTT (*Message Queuing Telemetry Transport*), un protocolo estándar del consorcio OASIS (organización internacional que promueve estándares abiertos en el sector de las tecnologías de la información), ligero y basado en publicación y suscripción, pensado para conectar dispositivos remotos con poco ancho de banda (MQTT.org). En este esquema, los nodos publican sus datos en unos canales llamados *topics* y un servidor central, conocido como *broker,* los distribuye a quienes los necesiten, lo que encaja bien con sistemas donde muchos dispositivos envían información al mismo punto. Otras opciones comunes son HTTP/REST, que es el modelo de comunicación habitual de la web (*HyperText Transfer Protocol* con un estilo de diseño llamado REST), más adecuado cuando el sistema central se integra con aplicaciones web o móviles que consultan el estado bajo demanda, y CoAP (*Constrained Application Protocol*), una alternativa ligera pensada también para dispositivos con recursos limitados. En proyectos reales no es raro combinar varios protocolos, por ejemplo MQTT entre nodos y servidor, y HTTP/REST entre s,ervidor y aplicación de usuario.

A nivel físico existen dos familias de necesidades muy distintas en un sistema como este. Para los datos ligeros de estado bastan redes de bajo ancho de banda, desde la conectividad fija o wifi si está disponible hasta las redes móviles 4G o 5G, pasando por las redes de área amplia y baja potencia (LPWAN, Low-Power Wide-Area Network) como LoRaWAN o NB-IoT, adecuadas para enviar pocos bytes con muy poco consumo pero incapaces de transportar vídeo, por lo que en este trabajo solo serían aplicables al tramo de metadatos. Al tratarse de una red de nodos conectados, también conviene contemplar mecanismos básicos de seguridad, como la autenticación de los dispositivos y el cifrado de los mensajes, especialmente si se emplean redes públicas o móviles. Por último, además de los datos de ocupación, el sistema central debería recibir información básica de estado de cada nodo, como la conexión, la última actualización, la temperatura o posibles errores, para facilitar el mantenimiento y detectar fallos antes de que afecten al servicio.

Para el tramo que sí transporta vídeo, cuando no se puede cablear, la solución habitual en exteriores son los radioenlaces wifi de largo alcance, equipos dirigidos que operan normalmente en la banda de 5 GHz y establecen enlaces estables de decenas o cientos de Mbps entre puntos con línea de visión directa. Pueden configurarse como punto a punto (PtP, Point to Point), un enlace dedicado entre dos equipos, o como punto a multipunto (PtMP, Point to Multipoint), donde una estación base atiende a varios equipos cliente, lo que reduce el número de radios necesarias cuando varios puntos de captación deben comunicarse con un mismo destino. Su capacidad es suficiente para varios flujos de vídeo comprimido en H.264 o H.265, que suelen transmitirse mediante el protocolo de streaming RTSP (*Real Time Streaming Protocol)*, el estándar con el que las cámaras IP sirven su vídeo. Estos enlaces requieren línea de visión entre antenas, por lo que su viabilidad depende de la geometría del emplazamiento y de obstáculos como el arbolado.

Toda esta arquitectura encaja bien con un diseño modular, en el que cada nodo funciona de forma autónoma y publica sus resultados, mientras que el sistema central se encarga de agregarlos, almacenarlos y mostrarlos al usuario. Esto tiene una ventaja importante para el mantenimiento, ya que si un nodo cae el resto del sistema sigue funcionando, y se pueden añadir o quitar nodos sin necesidad de rediseñar el conjunto. La Figura 3 compara este planteamiento con el procesamiento centralizado. A la izquierda, el modelo centralizado lleva el vídeo de cada sector hasta el servidor, que hace todo el análisis. A la derecha, el procesamiento en el borde por sectores deja la inferencia en cada nodo y envía al centro solo los estados y los eventos, con el vídeo disponible bajo petición, que es la forma modular por la que opta este sistema.

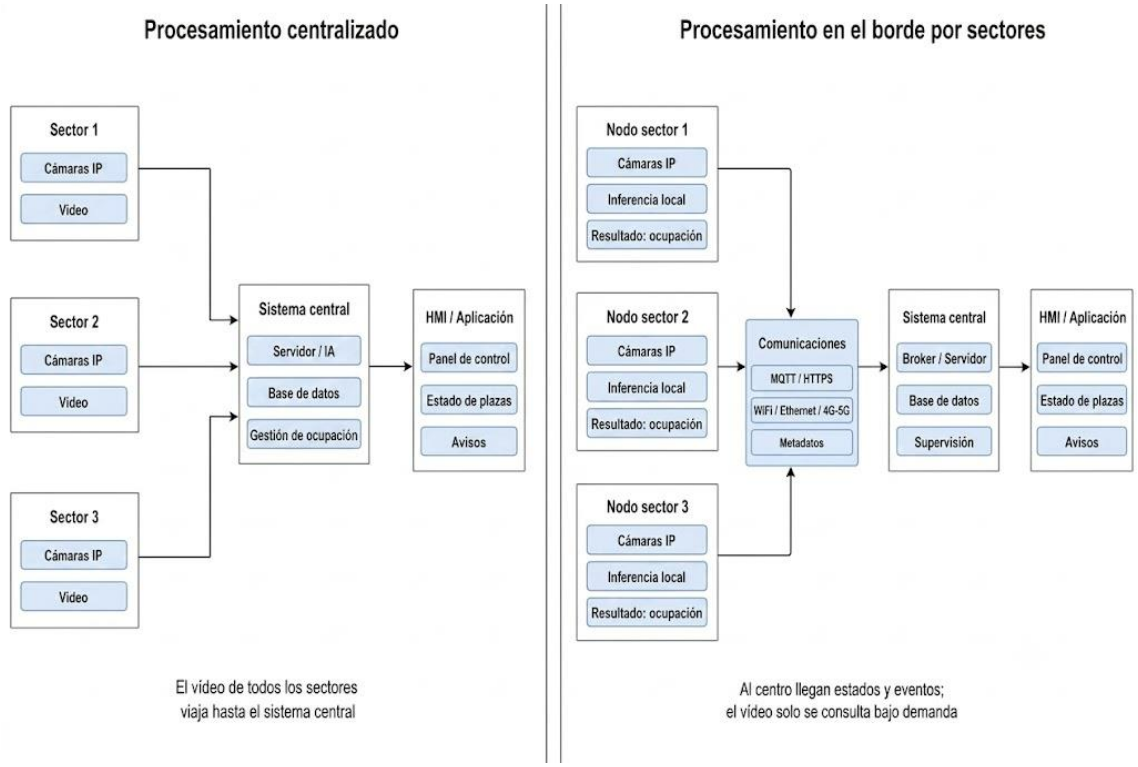


Figura 3. Comparación entre el procesamiento centralizado y el procesamiento en el borde por sectores. Elaboración propia

Esta forma modular es la que se sigue habitualmente en sistemas de monitorización distribuidos.

2.4.4 Consideraciones energéticas y de despliegue

Más allá de los componentes en sí, muchas decisiones técnicas dependen del entorno donde se va a instalar el sistema, así que conviene tener claras las condiciones reales en las que va a funcionar.

Una de las restricciones más importantes es la alimentación de los nodos. Una cámara con su plataforma de procesamiento necesita corriente continua, y la forma de llevarla hasta el punto de instalación condiciona el despliegue,y estos puntos, como una farola, un poste de iluminación, paredes de edificios o cabinas técnicas, sirven como soporte para las cámaras, un aspecto que abarata mucho el despliegue, lo que ahorra obra civil y reduce el impacto visual de la instalación. Esta idea de reutilizar lo que ya hay es uno de los pilares del enfoque *smart city* y es la base del planteamiento de este trabajo.

La opción más cómoda de alimentar el sistema es que el punto donde se monte la cámara ya disponga de red eléctrica, en cuyo caso basta con sacar una toma. Otra opción habitual en cámaras IP es la alimentación por el propio cable de red, conocida como PoE (Power over Ethernet, es decir, alimentación a través de cable Ethernet), que permite llevar al mismo tiempo datos y corriente eléctrica por un solo cable, lo que simplifica mucho la instalación al evitar tirar de cableado eléctrico independiente hasta cada cámara (Cisco). Eso sí, si el nodo incluye además de la cámara una plataforma de procesamiento, comunicación móvil o refrigeración, conviene comprobar que la potencia que entrega el estándar PoE elegido es suficiente, ya que puede ser

necesario recurrir a sus variantes más potentes, conocidas como PoE+ y PoE++ (versiones de PoE que entregan más potencia eléctrica, hasta unos 30 W y 90 W respectivamente). En el caso concreto de aprovechar farolas, conviene comprobar si disponen de alimentación permanente o solo durante el horario de encendido, ya que esto condiciona directamente la viabilidad del nodo, porque si la farola solo recibe corriente de noche, el sistema necesitaría una batería o un switch para que funcione durante el día con esa alimentación.

En lugares sin acceso a la red eléctrica se podría recurrir a una alimentación autónoma mediante paneles solares y baterías, una solución cada vez más usada en proyectos de *smart city*, aunque obliga a que el consumo total del nodo sea bajo, lo que limita las plataformas y los modelos que se pueden usar. En este sentido, el consumo eléctrico del nodo (cámara más plataforma) deja de ser solo un dato técnico y pasa a ser un criterio práctico de elección, especialmente en despliegues con una alimentación limitada.

Por último, hay que tener en cuenta las restricciones físicas y ambientales del entorno. La cámara y la plataforma de procesamiento van a estar expuestas a la lluvia, al sol, al polvo, a la humedad y a los cambios de temperatura, por lo que tanto la cámara (con sus grados IP e IK ya comentados) como el equipo de procesamiento deben ir alojados en una caja estanca adecuada. El acceso para mantenimiento es otro factor a considerar, ya que un nodo difícil de alcanzar encarece cualquier intervención posterior, igual que la seguridad física frente a vandalismo o robo. Todas estas condiciones, aunque parezcan menores frente a la parte técnica, son las que muchas veces deciden si un sistema se puede instalar realmente en el sitio previsto o no. La aplicación de todos estos criterios al caso del campus de la UEM se desarrolla en el Capítulo 5.

2.5 Herramientas y software empleados

La idea no es explicar cada herramienta a fondo, sino presentar con qué se trabaja, ya que algunas se utilizan directamente para procesar las imágenes y otras como apoyo en tareas auxiliares. La única decisión que se deja abierta a propósito es la del modelo concreto de detección, que se resuelve mediante las pruebas comparativas del capítulo 4.

2.5.1 Lenguaje y entorno de desarrollo

El lenguaje de programación elegido es Python, ya que es el más extendido en el ámbito de la visión por computador y del aprendizaje profundo, sobre todo por la cantidad de librerías especializadas que tiene disponibles. Como editor se utiliza Visual Studio Code, que permite trabajar con *scripts* y con *notebooks* de Jupyter de forma cómoda y es uno de los entornos más habituales para este tipo de proyectos.

2.5.2 Librería para visión clásica

Para la primera aproximación basada en regiones de interés se utiliza OpenCV, una librería de código abierto ampliamente empleada en visión por computador (Bradski, 2000). En este trabajo se aplica principalmente para leer vídeos, extraer fotogramas, manipular imágenes, trabajar con máscaras y calcular valores sobre las regiones asociadas a cada plaza. También se emplea como apoyo en la visualización de resultados y en la generación de salidas gráficas durante las pruebas.

Aunque la solución final se apoya en detección de vehículos mediante YOLO, OpenCV resulta útil para construir una línea clásica de comparación y para resolver tareas auxiliares del sistema. De esta forma, permite mantener una referencia sencilla frente al método basado en aprendizaje profundo.

2.5.3 Librerías de aprendizaje profundo

Para los modelos basados en deep learning, la base que se utiliza es PyTorch (Paszke et al., 2019), un *frame*work de código abierto ampliamente usado tanto en investigación como en producción, que permite definir, entrenar y ejecutar redes neuronales. Sobre PyTorch se apoyan otras librerías más específicas que facilitan trabajar con familias de modelos concretas sin tener que programarlas desde cero. La principal es Ultralytics(Jocher y Qiu, 2024), que ofrece implementaciones listas para usar de los modelos de detección de la familia YOLO, simplificando mucho el flujo de entrenamiento, evaluación e inferencia.

Para la preparación de los datos de entrenamiento se utiliza Roboflow (Roboflow, s.f.), una plataforma de gestión de *dataset*s de visión que permite anotar las imágenes marcando las cajas de cada vehículo, asistir esa anotación con detección automática, organizar las versiones del conjunto y exportarlo en el formato de etiquetas que utiliza YOLO. Los entrenamientos se ejecutan en Google Colab, un entorno de cuadernos en la nube que da acceso a GPUs (G*raphics Processing Unit*, unidades de procesamiento gráfico) sin necesidad de hardware propio, lo que resulta necesario porque el equipo local no dispone de memoria gráfica suficiente para entrenar a la resolución requerida. Por último, la interfaz del prototipo se construye con CustomTkinter (Schimansky, s.f.), una librería de interfaces gráficas para Python que permite integrar el vídeo, los indicadores y los controles en la misma aplicación que ejecuta la visión.

2.5.4 Modelos que se van a probar y comparar

La selección del modelo no se fija de antemano, sino que se realiza a partir de pruebas comparativas durante el desarrollo. El trabajo se centra en modelos de la familia YOLO, ya que ofrecen un equilibrio adecuado entre detección y coste computacional para una aplicación que debe funcionar de forma periódica y con posibilidad de procesamiento en el borde.

Durante las pruebas se comparan distintas variantes de YOLO y diferentes configuraciones de inferencia, modificando parámetros como la resolución de entrada, el umbral de confianza y el valor de IoU utilizado en la supresión de no máximos. Esta comparación permite valorar no solo la precisión de detección, sino también el comportamiento sobre vehículos pequeños, lejanos o parcialmente ocluidos, que son los casos más relevantes para el aparcamiento de la UEM.

La decisión final se apoya en las métricas de evaluación descritas en los apartados 2.1.3 y 2.2.2, junto con la observación visual de los resultados y el coste computacional de cada configuración. El desarrollo completo de esta comparación se presenta en el capítulo 4.

2.5.5 Herramientas de apoyo

Además de las librerías principales, se usan varias herramientas auxiliares que cubren tareas habituales del proyecto. NumPy (Harris et al., 2020) se utiliza para el manejo de matrices y operaciones numéricas, que es la base sobre la que trabajan tanto OpenCV como PyTorch. Pandas resulta útil para tratar tablas de datos, por ejemplo cuando se registran los resultados de las pruebas para compararlos después. Matplotlib (Hunter, 2007) se emplea para representar gráficos, como las curvas de pérdida del entrenamiento o las comparaciones entre modelos. Y Shapely (Gillies et al., s.f.) se incluye porque permite trabajar con geometrías en el plano, lo que encaja bien con la idea de asociar las cajas de detección a las regiones que delimitan cada plaza.

Con esta combinación de herramientas se cubre todo el flujo de trabajo, desde la lectura y preparación de las imágenes hasta el entrenamiento, la inferencia y el análisis de resultados.

2.6 Privacidad, normativa y ética

Cualquier sistema que capte imágenes en un sitio donde puedan aparecer personas o vehículos identificables tiene que cumplir, además de los retos técnicos, una serie de obligaciones legales y consideraciones éticas que no se pueden dejar de lado. De hecho, en sistemas basados en cámaras este suele ser uno de los puntos más delicados, ya que un mal planteamiento aquí puede invalidar todo el sistema desde el punto de vista legal por muy bien que funcione técnicamente. En este apartado se explica el marco normativo que aplica, qué se considera dato personal cuando se trabaja con imágenes, qué principios deben guiar el diseño, qué medidas suelen pedirse y qué consideraciones éticas conviene tener en cuenta, para que estas cuestiones queden claras antes de avanzar hacia el diseño concreto.

2.6.1 Marco normativo aplicable

En España, el tratamiento de datos personales se rige principalmente por dos normas. La primera es el Reglamento General de Protección de Datos (RGPD), aprobado por la Unión Europea en 2016, que se aplica directamente en todos los países miembros (Reglamento UE 2016/679). La segunda es la Ley Orgánica 3/2018, de Protección de Datos Personales y garantía de los derechos digitales, conocida como LOPDGDD, que adapta y completa el RGPD a nivel nacional. Las dos se aplican a cualquier tratamiento de datos personales, incluida la captación de imágenes con cámaras.

La autoridad encargada de velar por el cumplimiento de esta normativa en España es la Agencia Española de Protección de Datos (AEPD), que además de imponer sanciones publica guías y criterios sobre cómo aplicar el RGPD en distintos contextos, entre ellos uno específico para videovigilancia (AEPD 2025). A nivel europeo, el Comité Europeo de Protección de Datos, conocido como EDPB (European Data Protection Board), publicó en 2019 unas directrices específicas sobre el tratamiento de datos personales con dispositivos de vídeo, que sirven como referencia común en toda la Unión Europea (EDPB, 2019).

Conviene tener en cuenta que la finalidad del sistema de este trabajo no es exactamente la videovigilancia tradicional, ya que no se trata de vigilar personas con fines de seguridad, sino de

detectar la ocupación de plazas. Aun así, los criterios de la AEPD aplican igual porque se están captando imágenes en las que pueden aparecer personas y matrículas identificables, y eso basta para que el sistema cuente como un tratamiento de datos personales. Además, la AEPD señala que la videovigilancia debería usarse solo cuando no haya otros medios menos invasivos para conseguir el mismo objetivo, una idea que conviene tener presente en cualquier sistema basado en cámaras.

Aplicado al caso del campus de la UEM, la universidad actuaría como responsable del tratamiento al ser la entidad que decide para qué se usan las imágenes y cómo se manejan, igual que cualquier otra universidad española tiene su propia política de privacidad ajustada al RGPD y a la LOPDGDD.

2.6.2 Tratamiento de imágenes y datos personales

El artículo 4 del RGPD define dato personal como cualquier información sobre una persona física identificada o identificable. Bajo esa definición, una imagen en la que pueda reconocerse a una persona se considera dato personal. Y, algo importante para este trabajo, la matrícula de un vehículo también se considera dato personal, porque permite identificar a su titular. La AEPD lo ha confirmado en distintas resoluciones, donde trata las matrículas como datos sujetos a la normativa de protección de datos (AEPD, 2025).

Esto significa que, en el momento en que un sistema capta imágenes donde aparecen personas o matrículas, está tratando datos personales y le aplica el RGPD y la LOPDGDD. En un aparcamiento al aire libre la situación es clara, porque aunque el objetivo del sistema no sea identificar a nadie sino saber si una plaza está libre u ocupada, las imágenes captadas pueden incluir personas que pasan por la zona y vehículos con matrículas visibles. Las directrices del EDPB tratan justamente este tipo de situaciones aplicadas a aparcamientos como ejemplo (EDPB, 2019). Por todo esto, hay que plantearse desde el principio cómo se tratan esas imágenes, qué se guarda y durante cuánto tiempo.

2.6.3 Principios aplicables al diseño

El artículo 5 del RGPD recoge una serie de principios que cualquier tratamiento de datos personales debe respetar. Los más importantes para un sistema como este son los siguientes:

- **Licitud, lealtad y transparencia.** El tratamiento tiene que basarse en una causa legítima de las que reconoce el artículo 6 del RGPD, y los afectados deben saber que se están captando sus datos. En aparcamientos privados gestionados por entidades como una universidad, la base habitual es el interés legítimo del responsable, siempre que se respeten los derechos de los afectados.
- **Limitación de la finalidad.** Los datos solo pueden usarse para lo que se recogen. Si las imágenes se captan para gestionar la ocupación, no pueden destinarse después a otros usos como vigilancia laboral o identificación de personas, salvo que haya una nueva base legal que lo justifique.

- **Minimización de datos.** Solo se deben tratar los datos estrictamente necesarios para el objetivo. La AEPD señala que, aplicado a la videovigilancia, esto supone limitar el número de cámaras a las necesarias, ajustar la zona captada a lo imprescindible y no recoger más información de la que el objetivo justifica (AEPD, s.f.).
- **Limitación del plazo de conservación.** Las imágenes no se pueden guardar indefinidamente. El artículo 22.3 de la LOPDGDD fija la supresión de las imágenes en el plazo máximo de un mes desde su captación, salvo cuando deban conservarse para acreditar infracciones o a disposición de las autoridades.
- **Integridad y confidencialidad.** Los datos deben tratarse de forma que se garantice su seguridad mediante medidas técnicas y organizativas, como el cifrado de las comunicaciones y el control de acceso.

Estos principios no son una formalidad, sino que condicionan directamente el diseño técnico del sistema. Por ejemplo, el principio de minimización encaja muy bien con la idea de procesar las imágenes en el borde y transmitir solo metadatos, como se vio en el apartado 2.3, ya que así no salen imágenes del lugar y se reduce mucho el tratamiento de datos personales.

2.6.4 Medidas técnicas y organizativas

Más allá de los principios, la normativa pide aplicar medidas concretas que aseguren que se cumplen. Las más habituales en sistemas que captan imágenes son las siguientes:

- **Señalización informativa en la zona vigilada,** con carteles que indiquen la existencia del sistema, el responsable y cómo ejercer los derechos de acceso, rectificación, supresión y oposición , junto con los de limitación y portabilidad que añade el RGPD, La AEPD aclara que no basta con poner un cartel, sino que tiene que contener la información mínima que pide el RGPD.
- **Medidas de seguridad sobre los datos,** como el cifrado de las comunicaciones, sobre todo si se transmiten por redes públicas o móviles, y la protección de los servidores donde se almacenan o procesan las imágenes.
- **Control de acceso a las imágenes,** limitando quién puede consultarlas y dejando registro de los accesos para poder auditarlos.
- **Eliminación automática de las imágenes** una vez pasado el plazo establecido, salvo las excepciones tasadas.
- **Posibilidad de ejercer derechos por parte de los interesados,** que pueden pedir acceder a las imágenes en las que aparezcan o solicitar su supresión cuando proceda. En el contexto de aparcamientos, la AEPD ha resuelto varios casos a favor del derecho de acceso del usuario afectado.

Cuando el sistema se apoya en tecnologías nuevas, como pasa con la inteligencia artificial aplicada a la visión por computador, el artículo 35 del RGPD puede pedir además una Evaluación de Impacto relativa a la Protección de Datos, conocida como EIPD. Este análisis se hace antes de poner el sistema en marcha y sirve para identificar los riesgos para los derechos de las personas y las medidas para reducirlos (AEPD, 2025). La AEPD incluye entre los casos típicos los

tratamientos que usan tecnologías innovadoras o que implican observación sistemática de zonas accesibles al público, lo que puede aplicar a un sistema como este según su alcance.

2.6.5 Consideraciones éticas

Más allá del cumplimiento estricto de la norma, el propio RGPD recoge en su artículo 25 el concepto de privacidad desde el diseño (privacy by design), que consiste en tener en cuenta la protección de los datos personales desde las primeras fases del proyecto en lugar de añadirla como un parche al final. Aplicado a este sistema, esa idea encaja con plantearse desde el principio qué se necesita captar de verdad y qué se puede descartar, en lugar de guardar todo por defecto.

En la misma línea, cuando un sistema implique almacenar o transmitir imágenes, conviene valorar técnicas que reduzcan la cantidad de información personal que se mantiene. Entre ellas están el difuminado o pixelado de matrículas y de rostros, la anonimización de las imágenes antes de guardarlas durante mucho tiempo, o directamente la opción de no guardar las imágenes y quedarse solo con los resultados del análisis. Aunque estas técnicas no siempre son obligatorias, son una forma de reforzar el cumplimiento de los principios anteriores y de demostrar que el diseño se ha pensado teniendo en cuenta la privacidad de los afectados.

En este trabajo, todas estas cuestiones se tendrán en cuenta a la hora de tomar decisiones de diseño en los capítulos siguientes, sobre todo las relacionadas con qué se transmite, qué se guarda y durante cuánto tiempo.

Capítulo 3. Estado del arte y análisis de soluciones existentes

En este capítulo se analiza la situación actual de las soluciones disponibles para la detección de ocupación en aparcamientos al aire libre. La idea es ver qué se usa hoy en el mercado, qué se investiga a nivel académico y dónde quedan huecos por cubrir, para situar después la propuesta de este trabajo en un contexto realista. Los distintos tipos de tecnología de detección, sensores en suelo, ultrasónicos o cámaras, ya se introdujeron al hablar de componentes en el apartado 2.1.2, así que aquí no se reabren, y el capítulo se centra en soluciones reales del mercado y de la investigación.

3.1 Soluciones comerciales actuales

Existen varias empresas que han desarrollado sistemas comerciales de smart *parking*, cada una con un enfoque tecnológico distinto. A continuación se analizan las que mejor representan el estado actual del mercado para aparcamientos al aire libre, ya que son las más relevantes para este trabajo.

3.1.1 Cleverciti

Cleverciti es la referencia comercial más cercana en concepto de captación, con sensores elevados sobre infraestructura existente. Su sistema se basa en sensores ópticos montados sobre infraestructura existente, como farolas, postes o edificios, y cada sensor puede detectar hasta 100 plazas a la vez con un ángulo de visión de hasta 220 grados (Cleverciti, s.f.).

El procesamiento se hace en el propio sensor, en el borde, usando inteligencia artificial y aprendizaje profundo. Solo los metadatos (estado de cada plaza) salen del dispositivo, lo que les permite cumplir con la normativa de protección de datos y reducir el ancho de banda necesario.

Sus dispositivos integran dos cámaras 4K y están pensados para exterior, funcionando bajo lluvia, nieve o calor. Están desplegados en más de 100 ciudades, y el 3 de noviembre de 2025 ParkHelp anunció su adquisición para integrar su tecnología en una plataforma global, con la que el grupo resultante supera las 600.000 plazas monitorizadas en más de 700 ubicaciones de 50 países (ParkHelp, 2025). En la Figura 4 puede verse uno de sus sensores instalado.

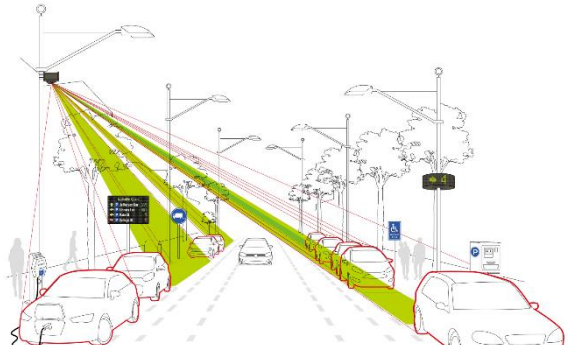


Figura 4. Monitorización de plazas mediante sensor óptico elevado Cleverciti. De Cleverciti

3.1.2 Parquery

Parquery, con sede en Zúrich (Suiza), se diferencia por ofrecer un enfoque puramente software. En lugar de vender hardware propio, su solución analiza imágenes de cámaras IP ya existentes, lo que reduce mucho el coste de despliegue al aprovechar la infraestructura instalada. El algoritmo puede ejecutarse en la nube o de forma local en las instalaciones del cliente, y según los datos de la compañía es capaz de monitorizar más de 300 plazas en una sola imagen (Parquery s.f.).

Su software se integra con cámaras comerciales como las de Axis y funciona en aparcamientos al aire libre, en calle, en áreas de servicio y en otros entornos exteriores. Cumple con el RGPD y con la ley suiza de protección de datos, Entre sus funciones está la detección de ocupación en tiempo real y los avisos cuando un vehículo supera el tiempo permitido o se estaciona en plazas reservadas, lo que la convierte en la referencia más directa en cuanto a modelo de software sobre cámaras existentes.

La Figura 5 muestra su panel de control, donde el sistema identifica las plazas libres y ocupadas sobre la imagen de la cámara.

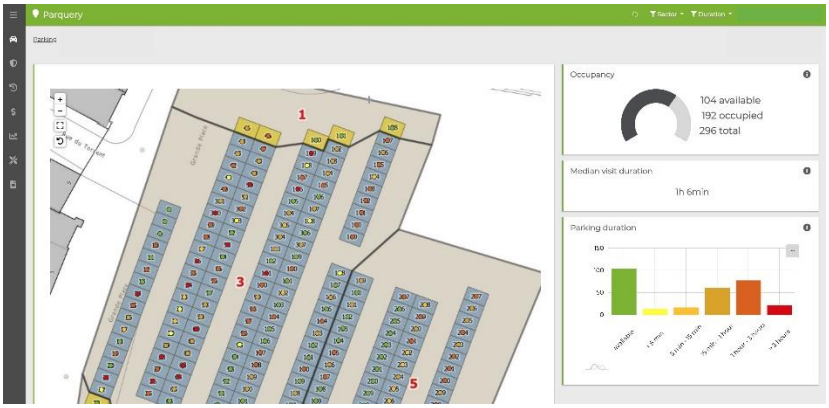


Figura 5. Panel de control de Parquery. De parquery

3.1.3 Milesight (cámaras AI específicas para *parking*)

A diferencia de Cleverciti o Parquery, que ofrecen una solución de sistema, Milesight es un fabricante de cámaras IP que ha sacado una cámara específica para gestión de *parking* con IA integrada. Su cámara de exterior gestiona hasta 100 plazas con una precisión declarada de más del 98%, incorpora un sensor STARVIS para baja luz, resolución 4K y lente motorizada, y dispone de reconocimiento de matrículas y características del vehículo (Milesight s.f.).

Lo que se compra con ellos es la cámara como producto cerrado, con su software integrado, mientras que la gestión global del aparcamiento (visualización, registros, integraciones) queda en manos del cliente. Es una opción intermedia entre montar una solución propia desde cero y contratar un servicio completo de terceros.

La Figura 6 muestra el aspecto de la cámara, que sigue el formato *bullet* típico de las cámaras de exterior, dentro del sistema de gestión de aparcamientos en exterior que utilizan.

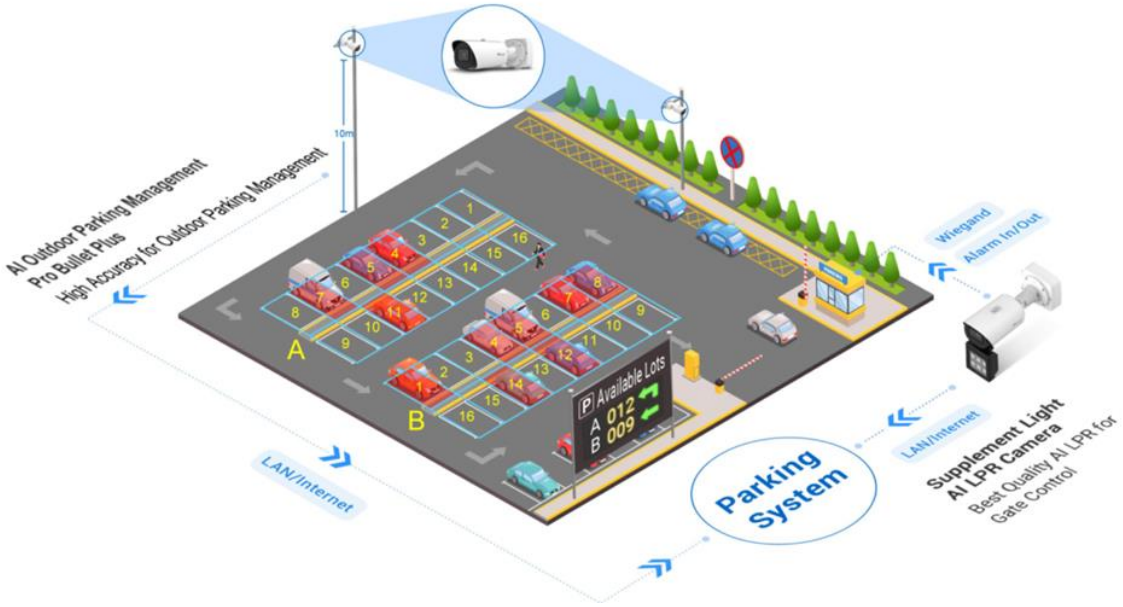


Figura 6. Cámara AI de Milesight aplicada a la gestión de aparcamientos. De Milesight

3.1.4 Urbiotica

Urbiotica es una empresa española especializada en IoT e inteligencia artificial aplicada al estacionamiento, y resulta especialmente relevante por ser referente nacional y por la línea en la que está trabajando. Su plataforma es modular y combina sensores magnéticos de superficie, un sistema basado en cámaras con IA llamado U-Spot VISIO, capaz de detectar vehículos incluso en plazas no delimitadas, y una plataforma de gestión, U-Admin. Lo más interesante para este TFG es su módulo U-Admin ParkCtrl, que detecta en tiempo real usos indebidos como sobretiempos, estacionamiento no autorizado en plazas de movilidad reducida o carga y descarga fuera de horario. Además, en junio de 2026 Urbiotica amplió su plataforma con funcionalidades de lectura de matrículas (LPR, License Plate Recognition) para automatizar el control del estacionamiento regulado (ESMARTCITY, 2026).).

Urbiotica encaja en dos cosas que hacen que se toman como ideas para el trabajo:

- Su U-Spot VISIO va en la misma línea de detección por cámara que se quiere desarollar.
- Su ParkCtrl demuestra que la detección de infracciones, como el coche mal aparcado o en plaza reservada, es una funcionalidad con demanda real en el mercado, lo que valida la necesidad de esta directamente y que se plantee una incidencia de ese tipo como funcionalidad adicional tiene ya sitio en el mercado.

3.1.5 Aspectos a tener en cuenta sobre los datos comerciales y costes

Las prestaciones que anuncian estas soluciones, precisiones del 98 o 99% y decenas o cientos de plazas por cámaras, hay que mirarlas con algo de cuidado. Suelen medirse en condiciones controladas, con cámaras bien colocadas y aparcamientos diseñados pensando en el sistema, así que en entornos reales, con cámaras existentes, perspectivas no ideales y oclusiones por vegetación o coches grandes, los resultados pueden diferir bastante de lo que indica el fabricante.

En cuanto al coste, no se dispone de tarifas públicas completas porque se suelen comercializar por presupuesto a medida, solo algunos elementos hardware, como cámaras específicas de *parking*, pueden encontrarse en distribuidores, pero no representan el coste completo de implantación. Pero sí se puede analizar el modelo de coste de cada una, que es lo que más las diferencia:

- Cleverciti se apoya en hardware propietario, un sensor por zona que hay que comprar e instalar.
- Parquery funciona por suscripción, cobrando por plaza monitorizada con independencia del número de cámaras, mediante un pago recurrente o una licencia perpetua, reutilizando las cámaras existentes.
- Milesight vende la cámara como producto, un pago único por unidad que se adquiere a través de distribuidores.

Frente a estos modelos, el enfoque de este trabajo es un software que se incorpora ya sea en la propia cámara de la localización del cliente, como la posibilidad de hacer un híbrido entre las cámaras del cliente y las propias, como un sistema completo con las propias cámaras, que podrían tener la colocación ideal. Esa diferencia de modelo es la base sobre la que después, en el capítulo 6, se cuantifica el coste real del sistema propuesto y se compara con estas referencias con los datos disponibles.

Aun así, las cuatro soluciones comparten un patrón que coincide con la línea de este trabajo, procesamiento en el borde, transmisión solo de metadatos, aprovechamiento de la infraestructura existente cuando es posible y atención al cumplimiento de la normativa de protección de datos.

Además, en España existen además primeros despliegues públicos en aparcamientos abiertos que confirman la viabilidad del enfoque por cámaras. El Parque Científico y Tecnológico Cartuja de Sevilla puso en marcha un piloto basado en 21 cámaras y 5 paneles informativos para comunicar las plazas disponibles en tiempo real (Ayuntamiento de Sevilla, 2019), y en Badajoz se implantó un sistema que combina sensores, pantallas, aplicación móvil y web para informar de la ocupación y detectar usos indebidos(Centro Demostrador TIC de Extremadura, s.f.). Son despliegues cercanos en escala y objetivo al que se propone para el campus, lo que los convierte en la referencia nacional más directa de este trabajo.

3.2 Propuestas académicas relevantes

Al margen del mercado, existe un trabajo de investigación muy activo en torno a la detección de ocupación en aparcamientos al aire libre con cámaras. Estos trabajos exploran enfoques que aún no se han trasladado al mercado y ayudan a entender qué problemas de ingeniería se están atacando y cómo. A continuación, se comentan los más representativos para este trabajo.

3.2.1 Almeida et al. (2015) y Amato et al. (2017), *dataset*s de referencia

No son propuestas de sistema en sí, pero son trabajos clave porque crearon los dos *dataset*s que se han convertido en estándar para evaluar cualquier solución de detección de ocupación: PKLot (Almeida et al., 2015) y CNRPark-EXT (Amato et al., 2017). El primero ofrece más de 700.000 plazas anotadas en condiciones de sol, nubes y lluvia desde tres puntos de vista distintos. El segundo amplía la idea a nueve cámaras, distintos días y situaciones con sombras y oclusiones. Aparecen citados en casi cualquier estudio del campo y en este trabajo se utilizan como una referencia clave a la hora de comparar resultados de los modelos que se vayan a probar, ya que permiten contrastar con la literatura sin tener que partir de cero. En el marco teórico ya se introdujeron en el apartado 2.2.4.

3.2.2 Liu et al. (2023): transformación de perspectiva aplicada al *parking*

Como ya se vio en el apartado 2.2.3, Liu et al. (2023) presentan un sistema basado en cámaras con transformación de perspectiva inversa (IPM) para convertir las vistas oblicuas de varias cámaras en una vista aérea del aparcamiento, sobre la que delimitan las plazas y detectan la ocupación con un modelo YOLOv5. Es interesante porque ataca directamente la perspectiva oblicua, el problema que aparece al montar cámaras en farolas o postes, que es justo el caso de la UEM. En este trabajo la IPM se valoró como técnica de referencia para corregir esa perspectiva, aunque finalmente la asociación vehículo-plaza se resuelve por geometría directa sobre la imagen, sin rectificar la vista, una decisión que se justifica en el capítulo de desarrollo.

3.2.3 Encío et al. (2023), información contextual y red multi-rama

Este trabajo, desarrollado en la Universidad Politécnica de Madrid (Encío et al., 2023), propone una red neuronal multi-rama basada en arquitectura ConvNeXt que aprovecha la información contextual de la imagen completa, en lugar de analizar cada plaza de forma aislada. La idea de fondo es que, al mirar solo el recorte de una plaza, se pierde información del entorno (iluminación general, plazas vecinas, sombras), lo que aumenta el riesgo de error en situaciones complicadas como cambios bruscos de luz, oclusiones entre coches o efectos de la perspectiva de la cámara.

Su aportación consiste en que cada rama de la red analiza independientemente esa información compartida del entorno para predecir la ocupación de una plaza concreta teniendo en cuenta el contexto. Los resultados sobre PKLot y CNRPark-EXT mejoran a varios métodos previos. Para este TFG es relevante porque ataca de frente las oclusiones y los cambios de luz, los retos mencionados en el apartado 2.2.4 que también aparecen en el campus, y porque deja una idea útil a tener en cuenta, que mirar el entorno de la plaza y no solo su recorte puede mejorar la fiabilidad.

Nguyen y Sartipi (2024): detección automática de plazas

Uno de los puntos débiles de la mayoría de sistemas de detección por visión es que requieren etiquetar manualmente las plazas en la imagen, lo que supone un trabajo costoso y poco escalable. Nguyen y Sartipi (2024) atacan justo ese problema. Su propuesta consta de dos módulos. PakLoc se encarga de detectar automáticamente la posición de las plazas a partir del vídeo, y PakSta evalúa simultáneamente el estado de todas esas plazas en cada fotograma.

Los autores afirman una reducción del 94,25% en el trabajo manual de etiquetado y obtienen una precisión del 93,6% sobre PKLot, superando a métodos que sí dependen del etiquetado manual. Es uno de los trabajos más recientes y prácticos que existen, ya que ataca un problema operativo importante, el de poder desplegar el sistema en aparcamientos nuevos sin tener que volver a marcar las plazas una a una. Para este TFG sirve como referencia de hasta dónde se puede llegar reduciendo trabajo manual, y abre la puerta a plantearse una forma más escalable de adaptarlo si el sistema tuviera que ampliarse a otros aparcamientos del campus. En la Figura 7 se ve el esquema general de su arquitectura, apareciendo en rojo el PaKsta (detección de estado de ocupación) y en azul el PakLoc (para la localización automática de las plazas.

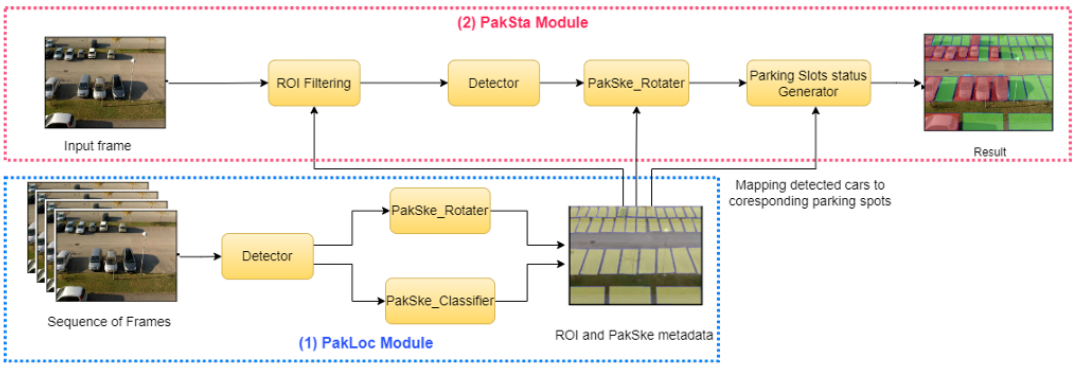


Figura 7. Arquitectura del sistema PakLoc/PakSta para localización automática de plazas y detección de ocupación. De Nguyen y Sartipi

3.2.4 Jabbar et al. (2021): smart *parking* en campus universitario con Raspberry Pi

Por su cercanía al caso de este TFG merece la pena destacar este trabajo, desarrollado para un campus universitario en Malasia.

Los autores proponen una arquitectura IoT en la que cada nodo combina una Raspberry Pi 4 con una cámara Pi, un sensor GPS y sensores ultrasónicos, encargados de detectar la ocupación de las plazas en exterior y enviar la información a una plataforma IoT en la nube (Jabbar et al., 2021). Aunque no es una solución basada solo en visión artificial, demuestra que con hardware asequible se puede llegar a un sistema funcional aplicado a un campus. Para este TFG es interesante porque enseña cómo otros han resuelto los problemas de ingeniería propios de un entorno universitario, como el reparto de los nodos, la conectividad o la integración con un sistema de gestión, que son justo los retos que aparecen en la UEM.

3.2.5 Kim et al. (2025): YOLO en *edge* con asociación plaza-vehículo

Uno de los trabajos más recientes y más alineados con la línea de este TFG. Kim et al. (2025) despliegan un detector YOLOv8 sobre una plataforma AI-BOX *edge* y, además del modelo, añaden un módulo de decisión temporal que asocia las detecciones a regiones de interés definidas para cada plaza, aplicando suavizado y reglas pensadas para mejorar la estabilidad frente a oclusiones y lluvia. Sobre un *dataset* propio de 13.691 imágenes alcanzan una latencia mediana de 195 milisegundos a 10 fotogramas por segundo, con un consumo de 3,35 vatios durante 24 horas continuas.

Lo interesante no es solo que usen YOLO en el *edge*, sino que reconocen que detectar coches no basta, y que la parte difícil es convertir esas detecciones en información fiable plaza por plaza, por lo que añaden la capa de ROIs y reglas temporales. Esa idea, asociar detección con plaza y estabilizar la decisión en el tiempo, es una de las claves que se incorporan al planteamiento de este trabajo.

3.3 Comparación de enfoques tecnológicos

Tras revisar las soluciones comerciales y las propuestas académicas, surge la pregunta por la que se comenzó este proyecto. Si existen tantos sistemas, algunos con cifras muy altas de precisión y desplegados a gran escala, por qué no se ven implantados de forma generalizada en aparcamientos cotidianos como el de un campus. La respuesta no está solo en el coste, tiene que ver con la diferencia entre tecnología disponible y despliegue real viable, que es lo que se quiere explorar en este apartado. La Tabla 6 resume las soluciones analizadas desde el punto de vista de qué aporta cada una, qué limitación práctica tiene y qué interés concreto puede tener para un escenario como el de la UEM.

Solución	Aportación principal	Limitación práctica	Interés para la UEM
Cleverciti	Sensor optimizado, procesamiento en borde y transmisión de metadatos.	Hardware propietario y coste elevado por sensor.	Referencia conceptual para sensores elevados en infraestructura existente.
Parquery	Software sobre cámaras IP existentes.	Servicio recurrente y dependiente del proveedor.	Referencia de arquitectura software basada en reutilizar cámaras.
Milesight AI	Cámara con IA integrada lista para usar.	Producto cerrado y con menor flexibilidad de adaptación.	Punto intermedio para comparar frente a una solución propia.
Urbiotica	Plataforma modular española con cámara IA y control de infracciones	Solución comercial completa, orientada a operadores	Referente nacional, su ParkCtrl respalda la incidencia que se plantea

Liu et al. (2023)	Transformación de perspectiva aplicada a un aparcamiento real.	Centrado en geometría, no tanto en despliegue completo.	Técnica aplicable al problema de vista oblicua en cámaras elevadas.
Encío et al. (2023)	Uso de información contextual con red multi-rama.	No aborda despliegue ni eficiencia en dispositivos de borde.	Enfoque útil para mejorar la fiabilidad por plaza en condiciones difíciles.
Nguyen y Sartipi (2024)	Detección automática de plazas de aparcamiento.	Validación limitada en entornos exteriores reales recientes.	Idea interesante para reducir el trabajo manual de adaptación a nuevas zonas.
Jabbar et al. (2021)	Sistema de aparcamiento inteligente en campus universitario con hardware económico.	Combina varios sensores, no solo cámaras.	Ejemplo cercano de despliegue universitario de bajo coste.
Kim et al. (2025)	YOLO en *edge* con asociación plaza-vehículo y reglas temporales.	Probado en un escenario concreto.	Referencia muy alineada con la línea práctica de este trabajo.

Tabla 6. Comparación de soluciones comerciales y propuestas académicas relevantes

3.3.1 Limitaciones detectadas

A partir de lo analizado, las limitaciones que mejor explican por qué la tecnología existente no se ha trasladado a la mayoría de aparcamientos reales son las siguientes:

- Coste y modelo de negocio, ya que las soluciones comerciales completas suponen una inversión alta en hardware propietario, del orden de cientos de euros por plaza según las referencias vistas, o un servicio recurrente por suscripción. En un aparcamiento universitario, donde el servicio no genera ingresos directos, ese tipo de modelos son difíciles de justificar.
- Distancia entre condiciones ideales y reales, porque as cifras que dan los fabricantes están medidas con cámaras en posiciones óptimas, mientras que en la práctica hay que aprovechar las farolas o postes existentes, con la altura y el ángulo que tengan, lo que introduce perspectiva oblicua, oclusiones y plazas con poca resolución. Estos retos ya se introdujeron en el apartado 2.2.4 y son los que más afectan al rendimiento real.
- Adaptación a cada aparcamiento, dado que un modelo entrenado para un escenario suele funcionar peor en otro distinto, y la mayoría de sistemas todavía dependen de un etiquetado manual de las plazas, lo que dificulta escalar el despliegue.
- Conectividad, mantenimiento y privacidad , porque un despliegue real exige resolver la conexión en zonas alejadas de la red interna, el mantenimiento de varios nodos distribuidos y la protección de los datos, problemas que rara vez aparecen en las cifras comerciales y que muchas veces son los que más condicionan que el sistema funcione.
- El salto de detección a sistema útil , ya que detectar coches con una cámara es un problema bastante resuelto, pero lo difícil es convertir esas detecciones en información fiable plaza por plaza, sobre todo cuando la cámara mira en oblicuo. La asociación plaza-

vehículo y la estabilidad de la decisión a lo largo del tiempo son justo los puntos donde aparece la mayoría de los problemas reales.

3.3.2 Oportunidad de mejora para este trabajo

De todo lo anterior sale una idea clara, y es que el problema no es de tecnología sino de aplicación.

Hay modelos de visión muy buenos y soluciones comerciales desplegadas a gran escala, pero rinden en condiciones favorables y se apoyan en hardware o servicios pensados para grandes operadores, así que lo que falta no es un detector mejor, sino ver qué hace falta para que un sistema así funcione de verdad en un sitio normal y con pocos medios, que es lo que se plantea aquí.

Para eso el trabajo se apoya en lo que ya se ha ido viendo, porque el estado del arte deja claro que lo difícil es pasar de detectar coches a dar una ocupación fiable plaza por plaza, sobre todo con la cámara mal colocada, mientras que el marco teórico da las bases sobre las que se decide después, la perspectiva, el procesamiento en el borde y la protección de datos. La UEM es el escenario donde se prueba todo esto, y como tiene una cámara ya instalada en una posición poco favorable, lo convierte en un caso real y exigente en lugar de una demo de laboratorio.

El sistema igualmente no se piensa solo para esa cámara, sino como una solución escalable que valga para otros aparcamientos, y por eso la UEM no es solo el sitio donde se prueba sino también donde se estudia qué se podría conseguir según los medios, valorando opciones de cámara, colocación y coste para ver cuál daría el mejor resultado en ese escenario. Es decir, no se intenta solo que funcione con la cámara que hay, también se plantea cómo sería el despliegue ideal y qué haría falta para llegar a él.

La forma de solucionarlo es la misma que se lleva durante el trabajo, probar cada opción, ver dónde falla y decidir el siguiente paso con criterio, buscando el equilibrio entre fiabilidad, coste y facilidad de despliegue, porque la idea no es ganar en cifras a lo que ya existe, sino llegar a un sistema realista y útil que se pueda llevar a otros aparcamientos del campus.

Capítulo 4. Análisis de alternativas, trabajo de campo y desarrollo del sistema de visión

En este capítulo se recoge todo el proceso de análisis que lleva desde el problema real hasta las decisiones de diseño del sistema. Se parte del trabajo de campo en el aparcamiento de la universidad para entender qué se necesita y con qué limitaciones se cuenta, y a partir de ahí se van estudiando y comparando las distintas alternativas, desde la forma de detectar los vehículos hasta el modelo y la configuración más adecuados, justificando cada elección con pruebas sobre datos reales. El objetivo del capítulo es dejar claro no solo qué se decide, sino por qué se decide, de modo que el sistema final del capítulo siguiente sea la continuación de de todo lo analizado aquí.

4.1 Trabajo de campo, necesidades y requisitos para el *parking* de la UEM

Antes de diseñar nada se hizo un trabajo de campo para entender qué se necesitaba de verdad en el aparcamiento, sobre todo mediante varias conversaciones con el personal de la universidad encargado del *parking* y de la seguridad del campus. La idea no era enseñarles una solución ya cerrada, sino escuchar qué problemas tienen y con qué limitaciones reales habría que contar.

En lo que más insistieron fue en que no quieren obras ni infraestructura, ni un sensor en cada plaza ni levantar el suelo, sino la solución más sencilla posible de montar, lo que descarta de entrada los sensores individuales por plaza y empuja hacia un enfoque que no dependa de infraestructura nueva, como el que se les planteó. El motivo es que confirmaron la suposición de que cada vez hay más coches y que hoy una persona da vueltas buscando sitio sin saber si queda alguno libre, de donde sale la necesidad principal, conocer la ocupación del aparcamiento y poder gestionarla.

Junto a esto pidieron dos añadidos, que cuando el *parking* se llene se cierre una barrera en la entrada, porque no se fían de que un panel que indique cero plazas libres impida que la gente siga entrandoy que les interesarua avisar de incidencias sencillas, sobre todo coches mal aparcados o en plazas reservadas para personas con movilidad reducida, lo que evita tener a un vigilante pendiente de una tarea repetitiva. Ninguno es el núcleo del sistema de visión, pero encajan y muestran em lo que se diferencia los sitenas de visión, cosas que solo se pueden detectar con una cámara.

Las reuniones sirvieron también para conocer las limitaciones del sitio y, la más importante es que en el aparcamiento no hay red disponible, lo que refuerza procesar la información en el propio lugar en vez de enviar las imágenes a un servidor. Comentaron además que las osclusiones como las de los arboles habían sido una limitación para plantear este tipo de sistemas, ya que algunas zonas están bastante cubiertas por árboles y eso dificulta ver bien los coches.

Por último se pidió el material para trabajar sobre datos reales, los planos del aparcamiento permiten medir el despliegue sobre dimensiones reales, y las grabaciones de las cámaras de seguridad, pensadas para vigilancia y no para medir ocupación, no se asumen como algo fijo, sino

como escenario de partida sobre el que ver hasta dónde llega el sistema y decidir después si compensa aprovechar esas cámaras o diseñar una disposición propia.

De todo lo anterior se ordenan los requisitos que guían el resto del diseño:

- Una solución sin obras, sencilla de desplegar.
- Conocer la ocupación del aparcamiento y gestionarla para mejorar la eficiencia.
- Funcionar sin depender de una red, procesando los datos en el propio aparcamiento.
- Avisar de incidencias relevantes para seguridad, como coches mal aparcados o en plazas reservadas.
- Permitir integrar elementos añadidos, como una barrera que se cierre cuando el aparcamiento esté lleno.
- Mantenerse adaptable y escalable a nuevas necesidades.

4.2 Desarrollo del sistema de Visión

4.2.1 Definición de plazas y regiónes de interés

Una vez fijados los requisitos, la parte de visión se aborda en primer lugar porque es la que determina hasta qué punto el sistema percibe bien la ocupación y de ella dependen el resto de decisiones. Antes de comparar las distintas formas de detectar los vehículos hay que dejar definida la base común sobre la que se mide, que son las plazas, de modo que todos los métodos se evalúen exactamente sobre lo mismo.

Como el objetivo es medir la ocupación, lo primero es decidir qué es una plaza para el sistema, y cada una se define como un polígono sobre la imagen, una región de interés con las coordenadas de sus vértices guardada en un archivo JSON, que es la zona sobre la que cada método decide después si hay un coche o no. Para marcarlas se utilizó la herramienta de selección de puntos de Ultralytics, que carga un *frame* del aparcamiento y permite ir señalando los vértices de cada plaza para exportarlas directamente al formato JSON con el que trabaja el sistema. Como se puede ver en la Figura 8, el marcado se hizo sobre un *frame* con el menor número posible de coches, para que las plazas se vean bien y se puedan delimitar con comodidad.



Figura 8. Plazas del aparcamiento de la UEM definidas como regiones de interés. Elaboración propia

A cada plaza se le asignó además un identificador, de forma que se sabe en todo momento de qué plaza se habla. Esto resulta muy útil durante el desarrollo para localizar cuál está fallando o cuál es la afectada por cualquier situación, y permite dar la ocupación plaza por plaza. En la Figura 9 se ven las mismas plazas ya con su identificador.



Figura 9. Plazas marcadas con identificador. Elaboración propia

El marcado se hizo de forma manual a propósito, para aislar lo que de verdad se quiere evaluar. Lo que se mide en este trabajo es el método de asociación y de decisión de ocupación, no la capacidad de localizar las plazas, de manera que si estas se detectaran automáticamente un fallo en esa parte se mezclaría con los resultados y ya no se sabría si un error viene de situar mal la plaza o de decidir mal su estado. Marcándolas a mano la geometría se sabe que la geométrica de las plazas esta bien representada y lo que se mida después depende solo del modelo de ocupación, que es lo que interesa comparar y por lo que se empieza porque es el grueso del sistema.

No se han marcado todas las plazas del aparcamiento porque al ser una prueba piloto no tenía sentido, y con un conjunto bien elegido ya se puede analizar todo lo que hace falta. Se marcaron 101 plazas escogidas de forma estratégica para que entren las distintas situaciones que se dan en el aparcamiento, desde las que se ven de frente y cerca hasta las más lejanas o en peor ángulo, de modo que el análisis cubra tanto los casos fáciles como los complicados y no solo la parte que mejor funcionaría.

Definir las plazas a mano encaja con un piloto, pero a mayor escala obligaría a marcarlas en cada aparcamiento nuevo, con el coste que eso supone, por lo que como mejora futura se plantea hacerlo de forma automática por dos vías que se complementan. La más sencilla es detectar las propias líneas que delimitan las plazas con técnicas de visión clásica de detección de bordes y líneas, que recoge todas las plazas aunque alguna no se ocupe nunca, a cambio de depender de que esas líneas se vean, algo que en exterior no siempre pasa. La otra, que encaja muy bien con este sistema, es reaprovechar el detector de vehículos que ya está en marcha, acumular sus detecciones durante un tiempo y deducir las plazas de las zonas donde los coches aparcan una y otra vez, sin depender de las líneas del suelo, aunque así solo se aprenderían las plazas que llegan a usarse. Las dos apenas añaden coste sobre lo que el sistema ya hace y podrían combinarse, usando las líneas donde se ven bien y la ocurrencia de coches donde no, lo que reduciría el trabajo manual de adaptarlo a nuevos sitios.

4.2.2 Visión clásica como primera aproximación

El desarrollo empezó por un método de visión clásica, sin inteligencia artificial ni entrenamiento, porque era interesante ver hasta dónde se llegaba estimando la ocupación con algo tan sencillo como aplicar filtros sobre la imagen. Es la opción más ligera y barata que hay, no necesita GPU ni datos para entrenar, así que valía como punto de partida y como referencia con la que comparar después lo que aportan los modelos más modernos.

Esta primera prueba se hizo antes de tener las grabaciones de la UEM, sobre los vídeos de ufpark, un conjunto de un aparcamiento más pequeño que permitía empezar a trabajar sin depender todavía del material del campus. El método suaviza la imagen con un filtro gaussiano y la pasa a escala de grises, y en cada plaza, recortada un poco hacia su centro para no coger los bordes ni el coche de al lado (reduciendo el Roi de cada plaza marcado original a un 85%), calcula la intensidad media y la desviación típica de los píxeles, marcándola como ocupada cuando la zona

es bastante oscura y con suficiente variación de textura. En la Figura 10 se ven las distintas transformaciones que se aplican a la imagen para conseguir el resultado.

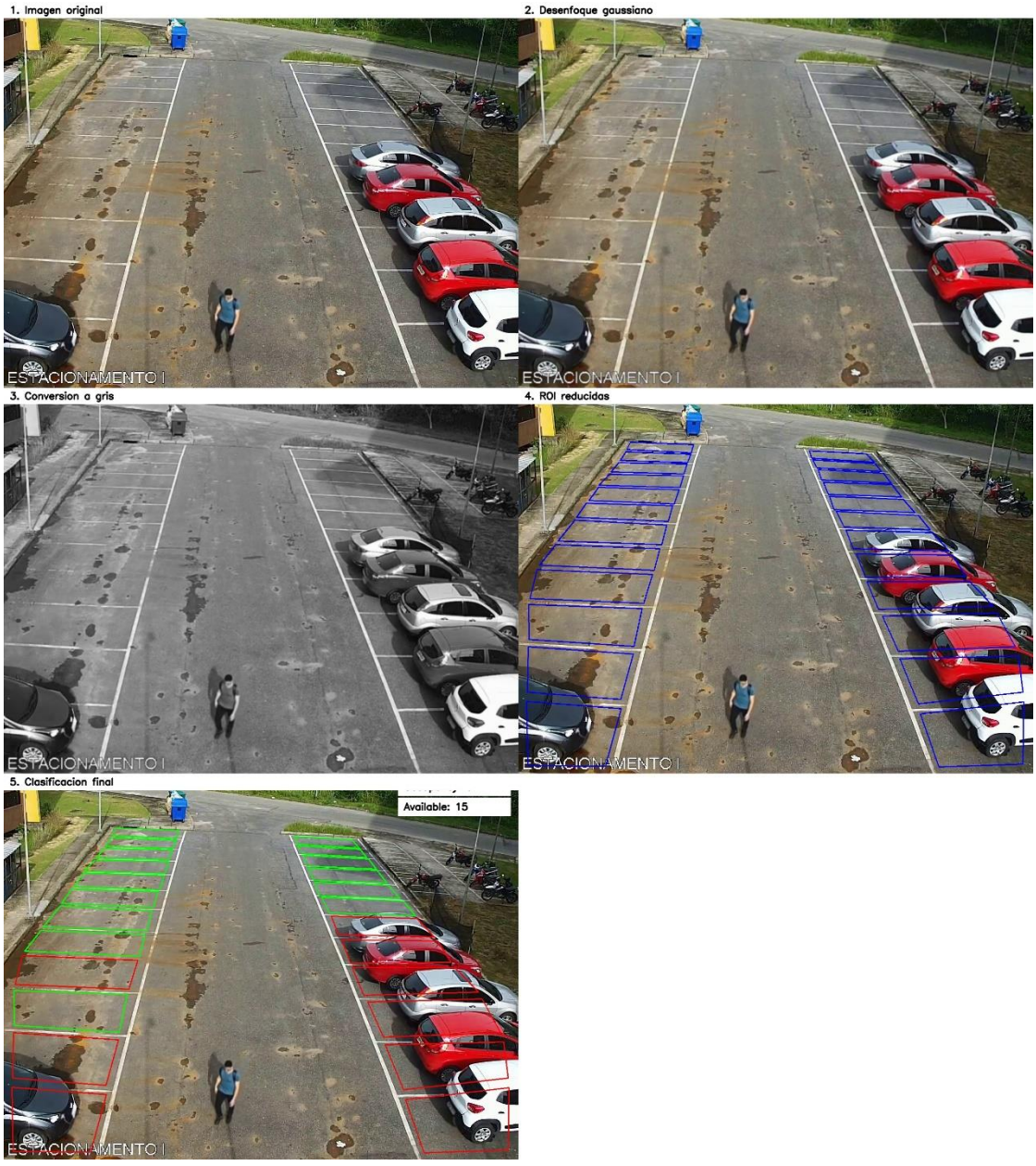


Figura 10. Etapas de procesado aplicadas a la imagen en el método clásico sobre los vídeos de ufpark. Elaboración propia.

Con los vídeos de la UEM se repitió la prueba sobre el aparcamiento real, donde se comportó de forma parecida. En la Figura 11 se ve el método funcionando sobre el *parking* de la UEM.

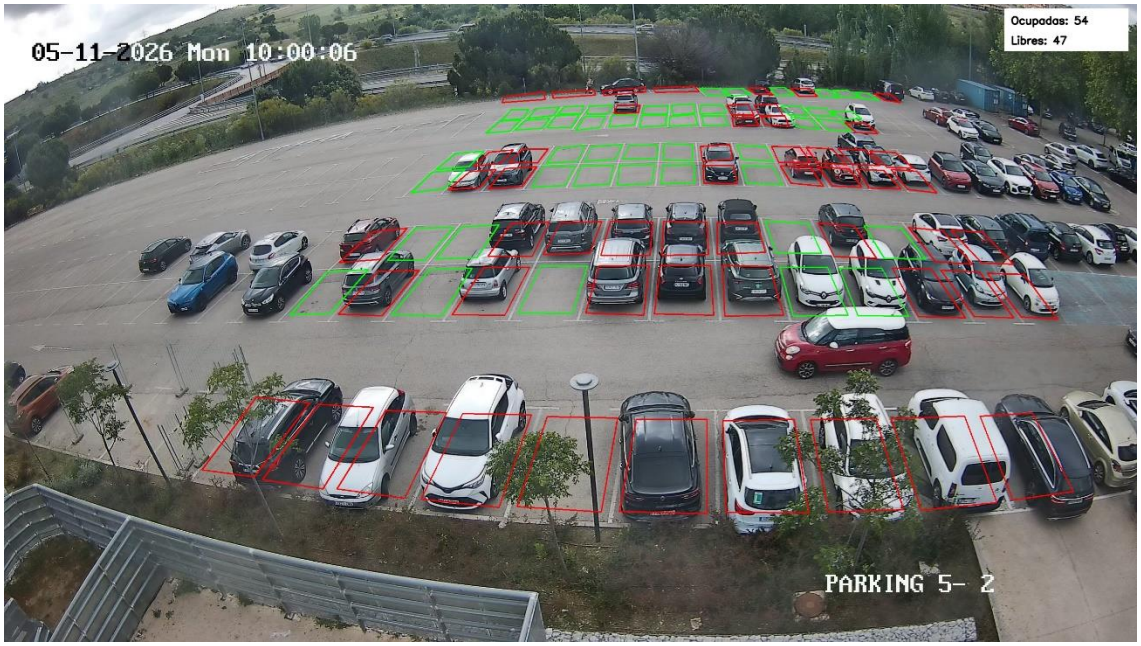


Figura 11. Método clásico aplicado al aparcamiento de la UEM. Elaboración propia.

En ambas figuras se ve que detecta bastante bien, pero también que falla. En ufpark una mancha oscura en el suelo se marca como plaza ocupada, y en la UEM pasa algo parecido con algunos obstáculos. El caso más claro se ve con un coche de techo blanco que cruza por delante de dos plazas ocupadas por coches blancos, y como la intensidad es parecida el método las da por libres. Son fallos que apuntan todos a lo mismo.

En condiciones ideales sorprende lo bien que detecta la ocupación para lo simple que es. El problema es que un aparcamiento al aire libre no es un sitio ideal, y cualquier sombra o condición no ideal altera el resultado, porque el método solo reacciona a los cambios en los píxeles y nunca llega a saber si lo que hay sobre la plaza es un coche. No es algo que se arregle ajustando los umbrales, es una limitación de cómo funciona. Por eso se descarta de las comparaciones con el resto de modelos, ya que estos sí reconocen el coche en sí y no se les puede medir con la misma vara.

Aun así no se tira el trabajo, porque deja un par de ideas para más adelante. La primera, que detectar movimiento en la imagen puede servir para activar el sistema solo cuando algo cambia, y así no procesar todos los *frame*s y ahorrar cómputo en un despliegue si fuera *Edge* al final. La segunda, que lo mucho que le afectan las sombras y la luz confirma que las condiciones no ideales van a ser el reto principal, y que hace falta un método de detección de vehículos que reconozca el coche y no solo el cambio de imagen, que es lo que se prueba en los siguientes apartados.

4.2.3 Construcción del *ground truth*

Para poder medir si el sistema acierta hace falta una referencia con la ocupación real de cada plaza, y esa referencia es el *ground truth*, que se construye a mano revisando el vídeo y anotando plaza por plaza si está libre u ocupada. Funciona como base para sacar métricas y comparar contra

ella los resultados de las distintas pruebas, así que se define una sola vez aquí y el resto de apartados se apoyan en ella.

Las métricas se sacan sobre este clip, pero el sistema no se piensa solo para estas escenas, sino para funcionar en cualquier momento de lo que capta la cámara, de modo que el *ground truth* sirve para comparar las pruebas entre sí en igualdad de condiciones y no como una garantía de que todo vaya a funcionar igual en cualquier situación.

El *ground truth* no se anota sobre todos los *frame*s del vídeo, porque son muy parecidos entre sí y solo conseguirían inflar las métricas repitiendo escenas casi idénticas. En su lugar se eligen los *frame*s en los que se produce un cambio claro de ocupación dentro del clip, de modo que cada uno aporta una situación distinta. Son pocos, pero suficientes para comparar los métodos en igualdad de condiciones, sobre todo teniendo en cuenta que en cada *frame* se clasifican las 101 plazas, así que el número de decisiones que se evalúa es alto aunque los *frame*s sean cuatro. La Tabla 7 recoge esos *frame*s y el cambio que representa cada uno.

Tiempo	*Frame*	Descripción
0.0 s	0	Estado inicial
12.0 s	300	P024 ya ocupada
31.0 s	775	P027 ya ocupada
39.0 s	975	P089 ya libre

Tabla 7. Frames representativos usados para la evaluación. Elaboración propia

Cada plaza se anota en uno de tres estados, libre, ocupada o desconocida. El estado desconocido corresponde a situaciones ambiguas, como un coche en plena maniobra de entrada o salida en el *frame* inicial, en las que no se puede decir con seguridad si la plaza cuenta como ocupada, y esas plazas se excluyen del cálculo de métricas para no penalizar ni premiar al sistema en casos que ni una persona clasificaría con claridad.

En el instante inicial, de las 101 plazas hay 43 ocupadas y 56 libres, y las 2 restantes quedan como desconocidas. A partir de ahí cada *frame* de la tabla recoge un cambio concreto, una plaza que pasa a ocupada o que se queda libre, que es lo que permite comprobar si el sistema sigue bien esas transiciones.

Toda esta información se vuelca en un archivo CSV con el estado real de cada plaza en cada *frame*, que es lo que después se cruza de forma automática con las predicciones de cada prueba para calcular las métricas. Tenerlo en un único archivo permite que cualquier método se compare

exactamente contra la misma referencia y de la misma forma, sin tener que revisar nada a mano cada vez.

Sobre esta referencia se evalúa todo lo que viene a partir de aquí, siempre sobre los mismos *frame*s y las mismas plazas, de modo que las diferencias que aparezcan entre pruebas se deban al método y no a haber mirado momentos distintos del vídeo.

4.2.4 Clasificación de plazas según calidad

Una vez marcadas las plazas y construida la referencia, salta a la vista que no todas se ven igual de bien desde la cámara, así que antes de medir nada se clasifican según su calidad de captación. Esta clasificación se hace por geometría y no por resultados, porque depende de dónde está cada plaza respecto a la cámara y no del método que se use después.

Lo que separa a unas de otras son las condiciones de captación vistas en el apartado 2.2.4, la resolución, el ángulo, la distancia a la cámara y las oclusiones. Con eso se definen tres niveles:

- Una plaza válida se ve de frente o con buen ángulo, cerca y con resolución suficiente.
- Una dudosa está en una situación intermedia, algo lejos, en ángulo o con algo de oclusión.
- Una problemática se ve lejos, muy oblicua, con poca resolución o tapada por otros coches.

En la Figura 12 se ven coloreadas según esta clasificación, las válidas en azul, las dudosas en naranja y las problemáticas en rojo.



Figura 12. Clasificación de las plazas del aparcamiento UEM según su calidad de captación. Elaboración propia.

De las 101 plazas, 48 son válidas, 14 dudosas y 39 problemáticas. Que casi cuatro de cada diez sean problemáticas no es un fallo, sino que la cámara del recinto está puesta para vigilancia y no para medir ocupación, por lo que muchas plazas quedan lejos o en mal ángulo.

El sistema se evalúa sobre las plazas válidas, porque representan lo que daría una cámara bien colocada, que es al final lo que se pretende proponer, y medir sobre ellas es lo justo para juzgar el método. Clasificarlas antes de medir evita marcar una plaza como mala solo porque luego falle, que sería circular, y así, si las problemáticas dan peores resultados, queda confirmado que el problema es de captación y no del método.

Aun así no se renuncia a las problemáticas, sino que se evalúa todo y se intenta sacar el modelo adelante también con esta cámara, tomando las decisiones que sean necesarias en cada punto. Algunas son de software, como ajustar o reentrenar el modelo para que rinda mejor desde esta vista, que se puede probar y mostrar con datos. Otras son de captación, como reforzar la zona que se ve mal con una cámara auxiliar, justificar que una cámara de más calidad no tendría ese problema, o plantear directamente la colocación ideal de las cámaras, opciones que seguramente sean la solución real pero que en un piloto se razonan más que se demuestran.

4.2.5 Detección de vehículos, primera aproximación para análisis

Con la base de plazas y la referencia ya definidas, el primer paso es ver cómo se comporta la detección de vehículos sobre la escena tal cual, sin ajustar nada, para saber de qué se parte. Para estas pruebas se usa YOLO, un detector estándar y ampliamente usado, que sirve para ver el comportamiento de partida y sacar métricas, sin que esto suponga elegirlo todavía como modelo definitivo, ya que la comparación entre modelos se hace más adelante y con otro detector el comportamiento sería parecido. Sobre los coches cercanos detecta bien y genera su caja sin problema, pero en las filas del fondo deja coches sin detectar, que coinciden en buena parte con las plazas clasificadas como problemáticas. En la Figura 13 se ve este comportamiento, con los coches de delante detectados y los del fondo sin caja.



Figura 13. Detección de vehículos sobre el aparcamiento de la UEMl con baja ocupación (derecha) y alta ocupación (izquierda). Elaboración propia.

Lo importante es de dónde nace esta limitación, porque está en la detección y no en la asociación. Si un coche del fondo no genera caja, no hay nada que asociar a su plaza, de modo que el problema es del modelo de detección y no de cómo se reparten luego las cajas entre las plazas. Esta limitación es la que condiciona los siguientes pasos, donde se prueban distintas configuraciones y modelos, y se intenta resolver más adelante.

4.2.6 Método de asociación vehículo-plaza

Una vez que el detector YOLO localiza los vehículos de la escena, queda un segundo problema, que es decidir qué plaza o plazas se consideran ocupadas a partir de cada caja de detección. Que el detector localice bien un coche no significa acertar la ocupación, porque en una vista oblicua la caja de detección queda desplazada respecto a la plaza donde el coche se apoya de verdad, así que decidir a qué plaza pertenece cada caja es un problema aparte. Por eso esta parte se trabajó como un desarrollo propio, probando criterios geométricos cada vez más completos y mirando sobre el vídeo qué resolvía cada uno y qué seguía fallando.

El desarrollo empezó antes de tener las grabaciones de la UEM, trabajando con las secuencias de UFPark, donde el aparcamiento se ve con una perspectiva muy oblicua. El primer criterio que se probó fue el de *Parking*Management, la solución de aparcamiento de Ultralytics, que coge el centro de la caja de cada coche y marca como ocupada la plaza en cuyo polígono cae ese punto. Sobre UFPark se vio enseguida su límite, porque en las plazas más inclinadas el centro caía en la plaza de delante y marcaba la equivocada, mientras que bajar ese punto a la base de la caja lo arreglaba, ya que al ser todas las plazas oblicuas el punto bajo caía siempre dentro de la plaza correcta. La Figura 14 compara los dos casos sobre UFPark, donde se puede ver como a la izquierda, el punto central se proyecta sobre una plaza distinta de la que ocupa el coche. A la derecha, el punto bajo cae dentro de la plaza correcta haciendo que se detecten las plazas correctamente.

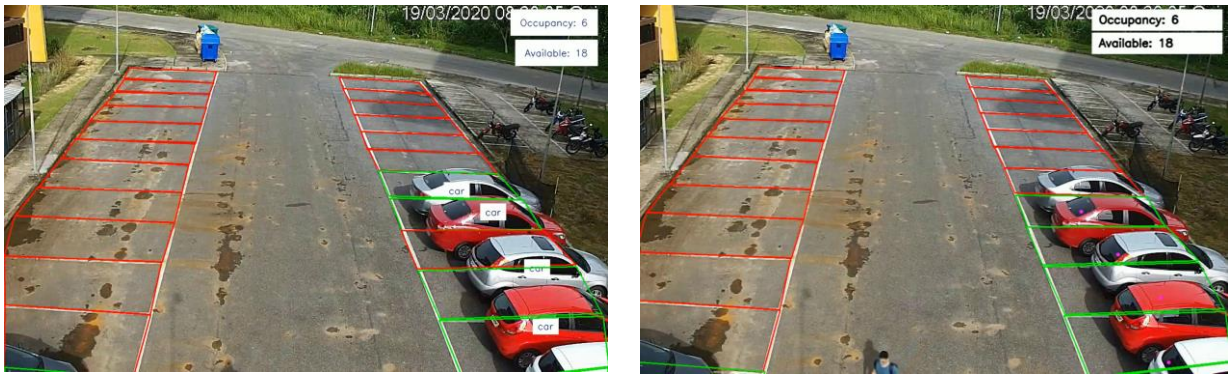


Figura 14. Método de asociación de ParkingManagement contra punto bajo sobre UFPark. Elaboración Propia

Al recibir las grabaciones del aparcamiento de la UEM el problema se volvió más variado, porque este aparcamiento no se ve de una sola forma, sino que mezcla las plazas cenitales y más cercanas en primer plano con plazas casi oblicuas al fondo, como se ha pidió observar en los anteriores apartados. Esa diferencia de perspectiva dentro de una misma escena es justo lo que complica la

asociación, porque un criterio pensado para un tipo de vista no encaja en el otro. En cuanto se probaron ahí los métodos anteriores se vio que un único punto ya no era suficiente, porque cada criterio de un punto resuelve un tipo de vista y falla en el otro. Por eso se vuelven a probar los métodos sobre la UEM, para confirmar sus límites y, a partir de ahí, buscar un criterio que funcione en los dos casos a la vez.

Antes de seguir de deja claro la decisión sobre cómo se montaron las pruebas. La asociación se eligió de forma independiente del modelo de detección, porque son dos cosas distintas. La asociación se desarrolla de forma independiente del modelo de detección, porque son dos cosas distintas, el modelo decide dónde está el coche y la asociación decide a qué plaza pertenece. Para que la calidad de la detección no influyera en esta comparación se usó un tamaño de imagen de entrada (imgsz) alto y un umbral de confianza (conf) bajo, de modo que se detectaran casi todos los vehículos de la escena y lo único que se valorara fuera el criterio de asociación. Esa configuración no será la del sistema final, ya que un imgsz alto consume más recursos. La elección del modelo, su configuración y cada cuántos fotogramas por segundo (FPS) hay que muestrear el vídeo se tratan más adelante en el trabajo, después de solucionar la asociación.

El segundo apunte es que los métodos no se puntúan uno contra otro sobre el *ground truth*, porque al irlos desarrollando se vio que no eran opciones que compitieran entre sí, sino una evolución en la que cada uno corrige al anterior, así que tiene más sentido contarlos como un desarrollo encadenado.

A continuación, se explican los métodos que se han desarrollado, como funcionan y como han evolucionado hasta llegar al final:

4.2.6.1 Método del punto central

El primero que se prueba sobre la UEM es el del punto central, el mismo criterio de *Parking*Management pero con implementación propia, para poder compararlo mas visualmente con el resto. A partir de la caja de cada coche se calcula su punto central promediando las dos esquinas, y si ese punto cae dentro del polígono de una plaza, se marca como ocupada.

En las plazas cenitales y cercanas funciona bastante bien, porque el centro de la caja queda cerca del centro real del coche y por tanto dentro de su plaza, pero en las plazas oblicuas del fondo el centro queda por encima de donde el coche se apoya y acaba cayendo en una plaza adelantada. La Figura 15 enseña ese fallo, con varios coches cuyo punto central se proyecta sobre una plaza

que no es la suya. El coche está bien detectado, así que no es un fallo de detección sino de asociación, que es justo lo que se quiere mejorar.

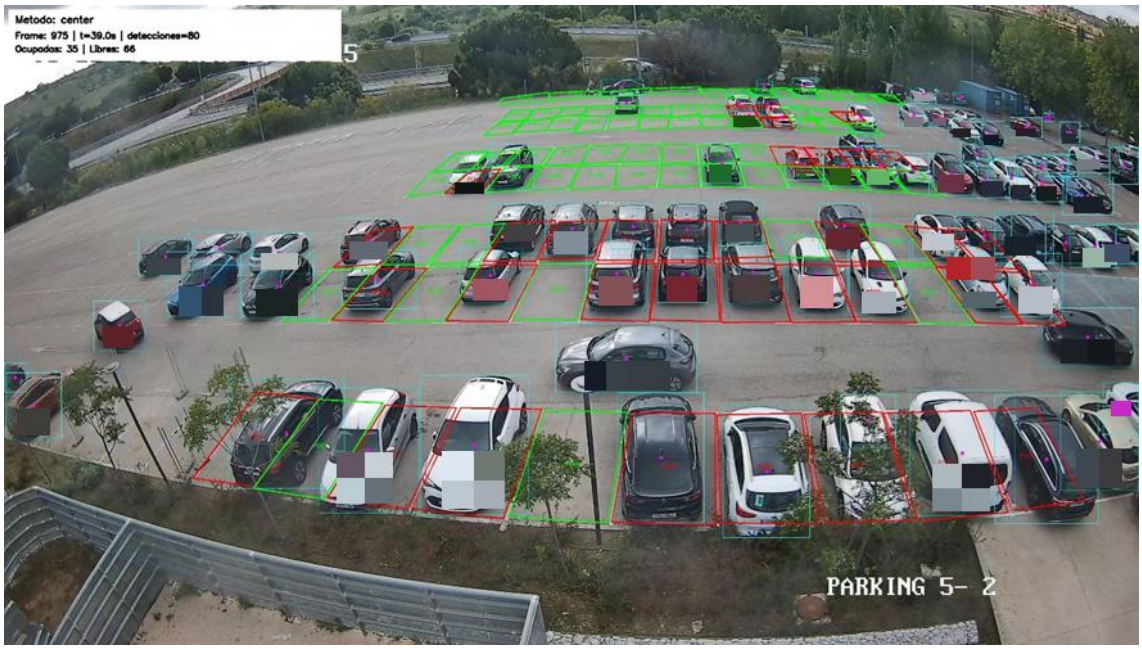


Figura 15. Método asociación del punto central sobre la UEM. Elaboración propia.

4.2.6.2 Método del punto bajo

Para corregir ese fallo en las oblicuas se vuelve al punto bajo, que ya había funcionado en UFPark. En lugar del centro se calcula un punto en la parte inferior de la caja, en torno al 90 % de su altura, que representa mejor la zona donde el coche pisa el suelo y evita que las cajas muy grandes desplacen la referencia.

En las zonas oblicuas el punto bajo mejora al centro, porque baja el punto de asociación hasta la base del coche y deja de caer en la plaza de delante. Sin embargo, al probarlo en todo el aparcamiento aparece el problema contrario. En las plazas cenitales y cercanas, o cuando un coche entra muy al fondo de su plaza, el punto bajo puede quedar fuera de la ROI y dejar la plaza como libre aunque esté ocupada. En la figura 16 se puede observar exactamente esto. Además, al seguir dependiendo de un solo punto, este método tampoco detecta bien los coches que invaden dos plazas. Se ve entonces que ni el centro ni el punto bajo resuelven el aparcamiento entero, porque cada uno acierta en un tipo de vista y falla en el otro.

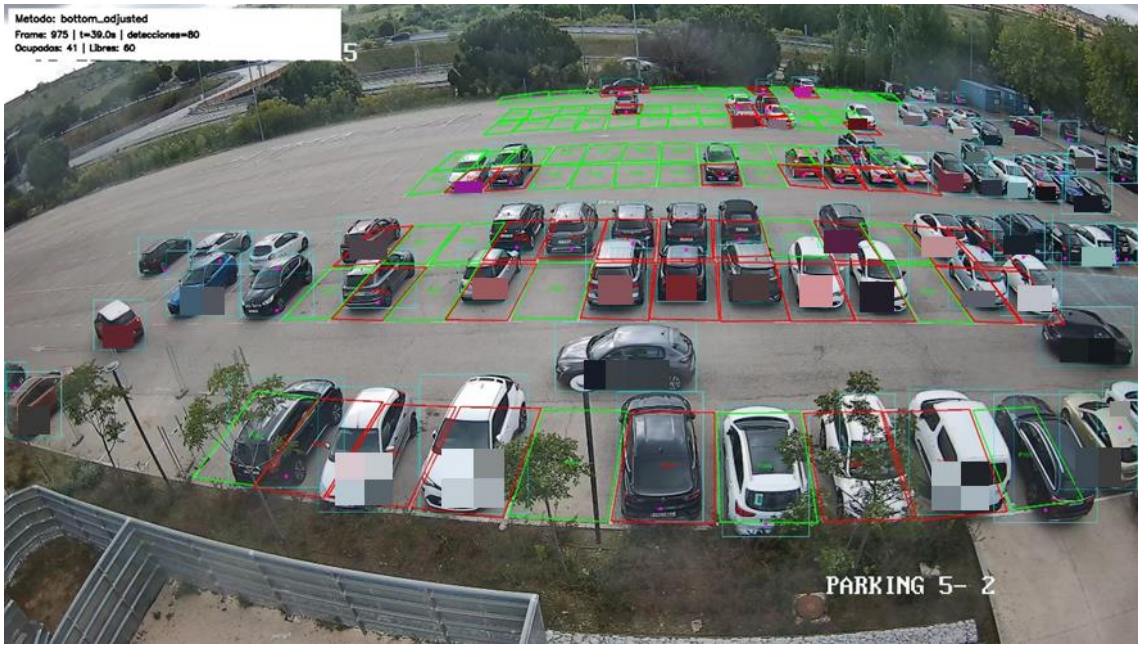


Figura 16. Método asociación del punto bajo sobre la UEM. Elaboración propia.

4.2.6.3 Método multipunto

Visto que un único punto no cubre los dos tipos de vista a la vez, se prueba un método multipunto, que en lugar de un punto utiliza varios repartidos por la caja, un punto central y varios en la base, incluidos los laterales. La idea es que entre todos esos puntos al menos uno represente bien la plaza ocupada, tanto en las plazas cenitales como en las oblicuas, y que los puntos laterales ayuden además a detectar cuando un coche invade más de una plaza. La plaza se marca como ocupada si alguno de esos puntos cae dentro de su ROI.

Este método mejora algunos casos que los anteriores no resolvían, y sobre todo aporta algo nuevo, que es marcar los coches mal aparcados que ocupan dos plazas a la vez. La figura 17 muestra ese acierto, donde en la imagen se puede ver como en los coches más cercanos a la izquierda cuyos puntos de base caen en dos plazas contiguas, haciendo que las dos se marquen como ocupadas, que es lo correcto en ese caso.

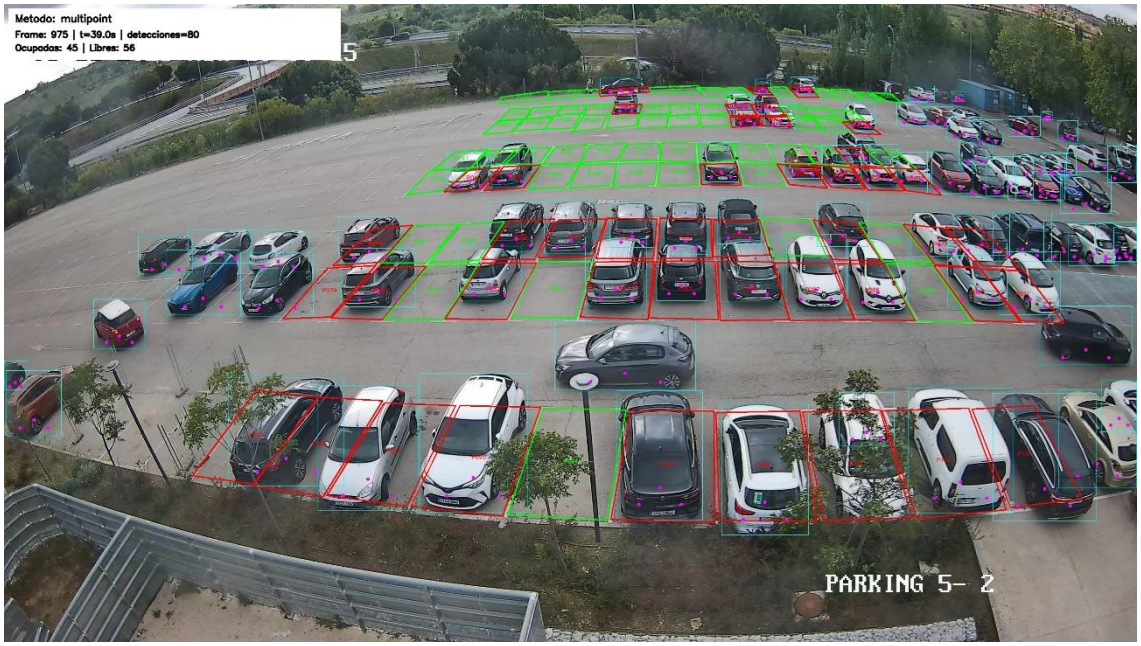


Figura 17. Método de asociación multipunto sobre la UEM. Elaboración propia

El problema es que usar varios puntos también añade errores. Si los puntos laterales se separan mucho, pueden caer en la ROI de una plaza vecina y marcarla como ocupada aunque el coche no la pise, generando un falso ocupado. Y si se juntan demasiado para evitar eso, se pierde la capacidad de detectar los coches a dos plazas, que era la ventaja del método. La figura 18 enseña este fallo, con un punto lateral cayendo en la plaza de al lado. Ajustar esa separación depende mucho de la escena, así que el multipunto resulta útil pero poco estable como criterio principal teniendo en cuenta que se quiere un modelo lo más escalabre posible. Esta prueba deja una idea clara, y es que juntar varios puntos no soluciona el problema de fondo, porque arrastra las limitaciones de los métodos por punto en vez de quitarlas.

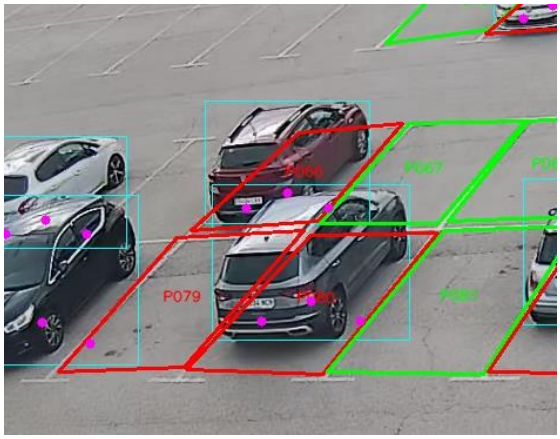


Figura 18. Limitación en vista oblicua del método multipunto sobre la UEM. Elabpración propia.

4.2.6.4 Método de solape con ROI

Después de los métodos por puntos se cambia de enfoque y se pasa a trabajar con áreas en lugar de puntos sueltos. Este método, que utiliza el *overlap* o solape, calcula qué porcentaje de la plaza queda cubierto por la caja del coche, y marca la plaza como ocupada si ese porcentaje supera un umbral. En vez de depender de un punto que puede caer justo en un borde, mira la superficie que comparten el coche y la plaza, que es una relación más completa.

Para el cálculo se usa solo la mitad inferior de la caja y no la caja entera, porque la parte de arriba recoge el techo y el cuerpo del coche, que en vista oblicua se proyectan sobre las plazas de detrás, mientras que la mitad de abajo se corresponde con la zona donde el coche se apoya en el suelo.

El umbral de solape se ajusta probando sobre las imágenes, ya que con valores bajos el método marca plazas de más, porque un roce pequeño entre la caja y la plaza ya supera el límite, así que se sube hasta un valor en torno a 0,35, que deja fuera esos roces sin perder los coches que sí están sobre la plaza. Con ese ajuste el comportamiento sobre el vídeo es más estable, como se ve en la Figura 19. Este método mejora a los de punto, porque no depende de una sola coordenada, y además recoge de forma natural los coches que pisan dos plazas, ya que una misma caja puede tener solape suficiente con dos ROIs.

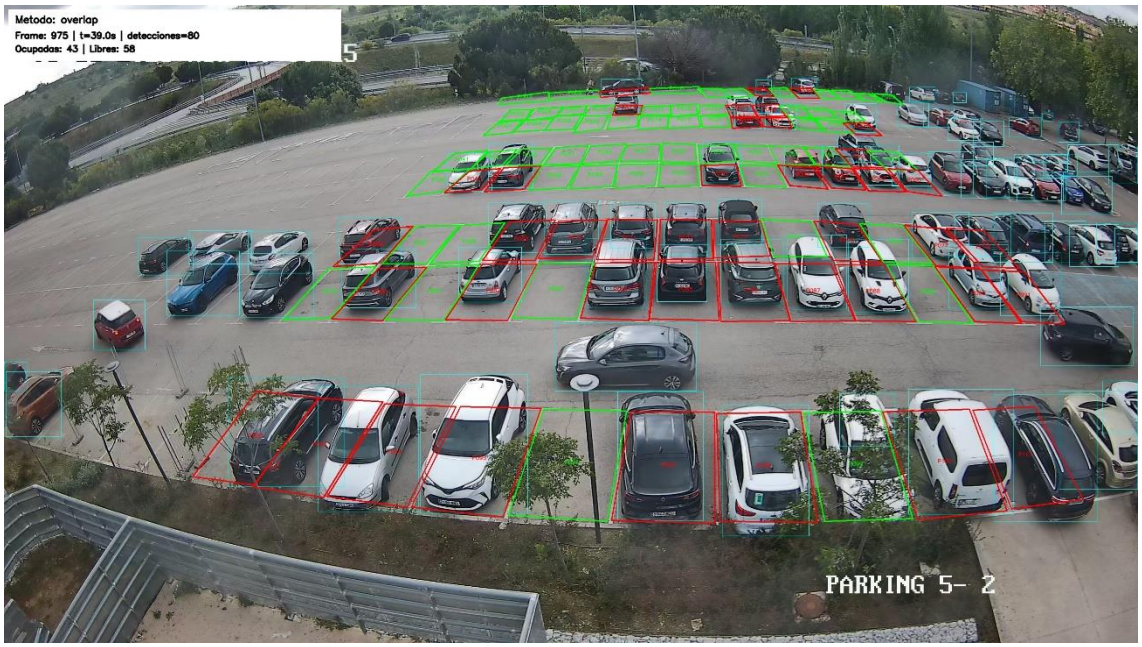


Figura 19. Método de asociación por solape sobre la UEM. Elaboración propia

Aun así, como se observa en la imagen, el método mejora a los anteriores pero todavía tiene algunas limitaciones. En la primera línea aparece un coche tapado parcialmente por un árbol, algo que también puede pasar con cualquier otro obstáculo o cuando el vehículo está muy lejos. En esos casos YOLO solo detecta la parte visible del coche, así que su caja de detección sale más pequeña, su solape con la plaza no llega al 35% y la plaza se queda marcada como libre aunque esté ocupada, que es un error critico por lo que se busca mejorarlo un poco más. Algo parecido ocurre cuando un coche está muy adelantado dentro de la plaza, donde el solape tampoco basta para detectarlo. Por eso el método de solape mejora bastante respecto a los de punto, pero por sí solo no resuelve todos los casos.

4.2.6.5 Método final, overlap con respaldo de multipunto y comprobación de plaza

El método de solape resolvía la mayoría de los casos, pero seguía fallando cuando la caja del coche salía pequeña por una oclusión, un árbol o la distancia, porque el área compartida con la

plaza no llegaba al umbral y la plaza se quedaba como libre estando ocupada. El método final parte de ese solape y le añade dos comprobaciones por puntos, una para recuperar esas detecciones que el área dejaba fuera y otra para confirmar cuándo un coche ocupa de verdad dos plazas.

Selección de la plaza principal

Respecto al *overlap* anterior se ajusta qué parte de la caja entra en el cálculo. En lugar de la mitad inferior completa se usa una franja más recortada, la huella, que va del 12 % al 88 % del ancho y del 52 % al 100 % del alto. Se quitan los extremos laterales porque al ensanchar la caja se proyectan sobre las plazas de al lado, y se quita la mitad superior porque recoge el techo y el cuerpo del coche, que en vista oblicua caen sobre las plazas de la fila de detrás. Queda solo la zona inferior, que es donde el coche se apoya en el suelo y la que de verdad indica su plaza.

El solape se mide sobre el área de la plaza, igual que en el método anterior. Se probó también normalizarlo sobre el área de la huella del coche, pero reaccionaba más a los pequeños cambios de tamaño y posición de las cajas de YOLO entre fotogramas, así que se mantuvo la normalización sobre la plaza por dar un comportamiento más estable. La plaza con mayor solape se toma como principal siempre que supere 0,30. El umbral baja un poco respecto al 0,35 del *overlap* anterior porque la huella es más pequeña que la mitad inferior y cubre menos superficie de la plaza.

La Figura 20 muestra un coche asignado a su plaza por este criterio:



Figura 20.Asignación de la plaza principal por solape de la huella con la ROI. Elaboración propia

Una vez pasado este proceso de ROI hay dos posibles procesos:

- No ha detectado nada por lo que se activa el respaldo.
- Ha detectado pero hay posibilidad de que este ocupando otra plaza adicional secundaria.

Respaldo/fallback cuando no hay solape suficiente

Cuando ninguna plaza llega a ese 0,30, el coche no se descarta directamente. Entran entonces seis puntos repartidos por la parte baja de la caja, y se asigna la plaza que contenga al menos tres de ellos, quedándose con la que más recoja. Con esto se recuperan justo los casos que el solape por área perdía, los coches con la caja pequeña por estar lejos o tapados y los que quedan muy adelantados dentro de la plaza. Este respaldo asigna siempre una sola plaza y nunca marca una segunda, porque una asociación que ya es dudosa de por sí no debe usarse para afirmar que un coche ocupa dos plazas a la vez.

En la Figura 21 se ve un coche que no alcanzaba el solape mínimo y queda asignado por el respaldo.

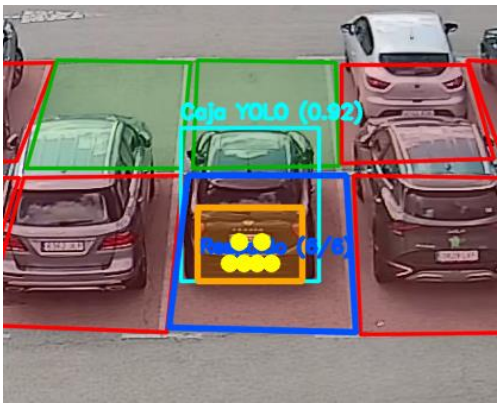


Figura 21. Recuperación de una detección mediante el respaldo de puntos. Elaboración propia

Comprobación de segunda plaza

Con la principal fijada, se comprueba si el coche está ocupando además una segunda plaza. Un coche que se queda entre dos plazas inutiliza las dos, porque en ninguna cabe otro coche, así que lo correcto es marcar ambas como ocupadas y no solo aquella en la que encaja mejor. Esta comprobación solo se lanza cuando una segunda plaza distinta de la principal supera 0,20 de solape, un valor más bajo que el principal porque una plaza invadida de lado recoge menos huella del coche.

Ese solape mínimo por sí solo no basta, porque la perspectiva puede meter algo de huella en la plaza contigua sin que el coche llegue a pisarla. Para confirmarlo se reparte sobre la caja una rejilla de 20 puntos, 5 columnas por 4 filas, y se acepta la segunda plaza solo si al menos nueve de esos puntos caen dentro de ella. Pedir nueve obliga a que el coche se extienda de verdad sobre la plaza y no a que entren uno o dos puntos sueltos por el borde, que es lo que daba falsos ocupados en el multipunto original.

La Figura 22 muestra un coche que ocupa dos plazas, con la segunda confirmada por la rejilla.

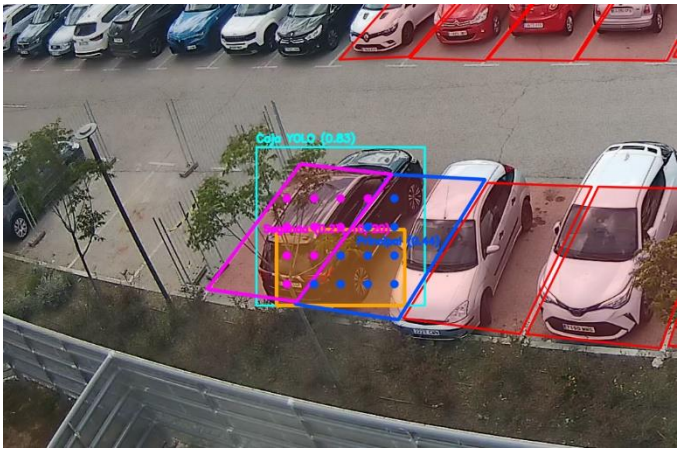


Figura 22. Confirmación de una segunda plaza ocupada mediante la rejilla de veinte puntos. Elaboración propia

Además, esta rejilla se ajusta según el tamaño de la caja de detección de YOLO.

En los coches lejanos la rejilla necesita un ajuste porque cuando la caja es pequeña sus puntos superiores caen con facilidad en la plaza de la fila de detrás por efecto de la perspectiva, lo que confirmaría una segunda plaza que en realidad no existe. Por eso, cuando la caja ocupa menos de 0,0015 del área de la imagen y se considera lejana, la rejilla se desplaza hacia la parte baja de la caja, concentrando los puntos en la zona de apoyo y dejando fuera la franja superior que provocaba el error, dejadno un modelo mas polivalente.

La Figura23 compara las dos rejillas, la normal sobre un coche cercan a la izquierda y la desplazada sobre uno lejano a la derecha.



Figura 23. Comparación entre la rejilla normal en un coche cercano y la rejilla desplazada hacia abajo en un coche lejano. Elaboración propia

Además, para prevenir que haya conflictos entre distintos coches, y que el modelo funcione correctamente una plaza que ya es principal de un coche no puede marcarse como segunda de otro, de modo que una caja desplazada no robe la plaza que corresponde a un coche distinto, y cada coche genera como mucho una segunda plaza, para que una sola detección no encienda varias por roces pequeños.

4.2.6.6 Parámetros finales del modelo

La Tabla 8 recoge los valores de la versión final, donde no se busca valer para cualquier instalación, ya que dependen de la posición de la cámara y del tamaño de las plazas en la imagen, pero se mantuvieron fijos en todas las pruebas de este caso y dieron un funcionamiento estable. De cualquier manera, también fueron pensados para ser lo más robustos y escalables posible.

Parámetro	Valor	Función
Región horizontal de la huella	12 %–88 %	Elimina los extremos laterales de la caja
Región vertical de la huella	52 %–100 %	Conserva la zona inferior relacionada con el apoyo del vehículo sobre el suelo
Solape mínimo principal	0,30	Acepta la asignación de la plaza principal
Solape mínimo secundario	0,20	Activa la comprobación de una posible segunda plaza
Puntos de la rejilla	20	Comprueban si el vehículo invade una segunda plaza
Puntos mínimos en la segunda plaza	9	Exigen una invasión suficientemente amplia
Puntos de respaldo	6	Recuperan asociaciones que no alcanzan el solape mínimo
Puntos mínimos de respaldo	3	Permiten asignar una única plaza mediante el mecanismo de respaldo
Umbral de caja pequeña	0,0015	Desplaza la rejilla hacia la parte inferior en coches pequeños o lejanos
Máximo de segundas plazas por coche	1	Evita ocupaciones múltiples producidas por pequeños solapes accidentales

Tabla 8. Parámetros de la asociación vehículo-plaza. Elaboración propia

4.2.6.7 Resultado del desarrollo

El recorrido desde un único punto hasta este método final fue cubriendo los problemas que cada criterio anterior dejaba abierto, desde el punto central que acertaba en las plazas cercanas y fallaba en las oblicuas, el punto bajo corregía las oblicuas pero salía de las cenitales, el multipunto representaba los coches a dos plazas pero metía falsos ocupados, y el solape ganaba estabilidad pero perdía las detecciones parciales.

El método final usa el solape sobre la huella como criterio principal y reserva los puntos para los casos dudosos y para validar la segunda plaza, que era el equilibrio que ningún método conseguía por su cuenta.

Una vez fijado este criterio, no se vuelve a modificar y de deja como definitivo. En las pruebas posteriores se mantiene la misma asociación y solo se cambia el detector, sustituyendo el modelo base por el afinado, para comprobar si la mejora en la detección de coches pequeños y lejanos se traduce directamente en una ocupación más correcta.

4.2.7 Selección de configuración

Una vez definido el método de asociación, el siguiente paso es ajustar la configuración de detección del modelo. Aquí todavía no se comparan modelos distintos, sino distintas configuraciones del mismo, YOLO11n, para no mezclar variables, porque si se cambiaran a la vez el modelo, la resolución, la confianza y la asociación sería imposible saber qué está causando cada cambio.

Por eso se fijan el modelo YOLO11n y el método de asociación final, y la prueba se centra solo en dos parámetros la resolución de entrada (imgsz) y el umbral de confianza (conf), definidos en el apartado 2.2.2. El parámetro de IoU del NMS se deja fijo en 0,5, porque no es lo que se estudia aquí.

La figura 25 resume el flujo completo seguido en cada prueba. Cada *frame* pasa por el detector, que devuelve las cajas de los coches, esas cajas pasan al método de asociación, que decide qué plazas quedan ocupadas, y esa predicción se compara con el *ground truth* del apartado 4.2.2 para sacar las métricas. La clase positiva es la plaza ocupada, así que un falso libre es una plaza ocupada marcada como libre, que es el error más grave porque mandaría a un conductor a una plaza que no existe, y un falso ocupado es una plaza libre marcada como ocupada, que resta capacidad al sistema. Para contar el que cuenta como un acierto (la interpretación del falso positivo o negativo) se utilizan las métricas de evaluación de sistemas de *parking* establecidas en el apartado 2.1.3.



Figura 24. Esquema del flujo seguido para comparar configuraciones. Elaboración propia

4.2.7.1 Configuraciones evaluadas

Se evaluaron seis configuraciones, combinando tres resoluciones, 1280, 1536 y 1920, con dos umbrales de confianza, 0,25 y 0,15, para ver por separado si la mejora viene de subir la resolución, de bajar la confianza o de las dos cosas.

La tabla 9 resume las seis configuraciones y el motivo de cada una:

Configuración	imgsz	conf	iou	Objetivo de la prueba
C1	1280	0.25	0.5	Configuración ligera inicial
C2	1280	0.15	0.5	Efecto de bajar la confianza sin subir resolución
C3	1536	0.25	0.5	Efecto de subir la resolución a un punto intermedio
C4	1536	0.15	0.5	Resolución intermedia con confianza más permisiva
C5	1920	0.25	0.5	Alta resolución manteniendo la confianza
C6	1920	0.15	0.5	Alta resolución con confianza baja

Tabla 9. Configuraciones evaluadas para YOLO11n. Elaboración propia

4.2.7.2 Resultados de detección e inferencia

Antes de mirar las métricas de ocupación se observa cuántos coches detecta cada configuración y cuánto tarda, porque no basta con detectar más, hay que ver si ese aumento compensa en coste. Al subir la resolución aumentan las detecciones, porque los coches pequeños o lejanos se representan con más píxeles, y al bajar la confianza también aumentan, porque el modelo conserva detecciones menos seguras.

La tabla 10 recoge las detecciones medias y el tiempo por *frame*:

Configuración	imgsz	conf	Detecciones medias	Tiempo medio/*frame*
C1	1280	0.25	57.00	0.0954 s
C2	1280	0.15	63.25	0.0962 s
C3	1536	0.25	64.75	0.1409 s
C4	1536	0.15	72.75	0.1441 s
C5	1920	0.25	73.00	0.2050 s
C6	1920	0.15	79.50	0.2064 s

Tabla 10. Detecciones medias y tiempo de inferencia por configuración. Elaboracion propia.

Todas las medidas de tiempo se han tomado sobre la CPU del equipo de desarrollo, un Intel Core i7-1280P, y sirven para comparar las configuraciones entre sí en igualdad de condiciones, no como valor absoluto del sistema final. Aunque una GPU procesaría la inferencia más rápido que una CPU al trabajar en paralelo, las medidas se toman sobre la CPU por acercarse más a un equipo *edge* de bajo consumo, que es el destino real del sistema. La comparación entre modelos es

relativa y su orden se mantiene independientemente del hardware, mientras que el rendimiento sobre el equipo de despliegue se estima en su apartado.

Al comparar primero las configuraciones con la misma confianza de 0.25, se ve claramente el efecto de la resolución. Con 1280 se obtienen 57 detecciones medias, con 1536 se sube a 64.75 y con 1920 se llega a 73 detecciones medias. Esto indica que el aumento de resolución permite recuperar coches que a menor tamaño de entrada no aparecen, especialmente vehículos lejanos o pequeños en la imagen.

Bajar la confianza a 0,15 también añade detecciones en todas las resoluciones, pero no todas útiles para las plazas que se estudian, así que las configuraciones con 0,15 se toman como prueba complementaria y la decisión se apoya en las de 0,25, más estables para evitar posibles falsos positivos en otros momentos.

Las Figuras 26 y 27 lo enseñan sobre el mismo *frame*, la primera compara las dos configuraciones de 1280 y se ve cómo al bajar la confianza se recuperan algunas detecciones en coches lejanos, y la segunda compara la configuración ligera con la de alta resolución y se aprecia cómo a 1920 aparecen coches del fondo que a 1280 no se detectaban.



Figura 25. Comparación visual entre C1_1280_025 y C2_1280_015 sobre el mismo frame. Elaboración propia.



Figura 26. Comparación visual entre C1_1280_025 y C5_1920_025 sobre el mismo frame. Elaborción propia

4.2.7.3 Resultados sobre plazas válidas

La evaluación principal se hace sobre las plazas válidas, porque son las que vería una cámara bien colocada, mientras que las dudosas y problemáticas se analizan aparte. Como se ve en la Tabla 11, las seis configuraciones aciertan casi a la perfección, todas por encima del 97 por ciento de exactitud y sin ningún falso ocupado, con diferencias de solo uno o dos falsos libres en las 48 plazas válidas a lo largo de los cuatro *frame*s. En las plazas cercanas y bien vistas la resolución apenas influye, porque el coche se detecta igual de bien a 1280 que a 1920, así que detectar plazas cercanas es un problema sencillo que el modelo ligero resuelve sin necesitar más píxeles.

Configuración	FP	FN	F1
C2(1280/0,15)	0	0	1.0000
C5(1920/0,25)	0	1	0.9963
C6(1920/0,15)	0	1	0.9963
C1(1280/0,25)	0	2	0.9925
C3(1536/0,25)	0	4	0.9850
C4(1536/0,15)	0	4	0.9850

Tabla 11. Métricas de ocupación en plazas válidas para las configuraciones de YOLO11n. Elaboración Propia

Dentro de estos resultados que son tan iguales, la mejor es la C2, que logra cero falsos de los dos tipos, y esa pequeña mejora viene de bajar la confianza, que recupera un par de detecciones sin penalizar el tiempo. Además, esos falsos libres no se deben a la asociación sino a que el modelo no detecta algún coche en ese *frame* concreto, un fallo que en vídeo continuo se piensa reducir con suavizado temporal, manteniendo el estado de la plaza durante varios *frame*s antes de

cambiarlo. Aun así el cambio entre configuraciones es mínimo, lo que confirma que las válidas no obligan a una resolución concreta.

4.2.7.4 Resultados sobre todas las plazas

Además de las válidas se analizaron los resultados con todas las plazas, incluyendo dudosas y problemáticas, apareciendo la diferencia que las válidas no mostraban. No es el criterio principal de decisión, pero permite ver el comportamiento en las zonas difíciles del encuadre y valorar si merece la pena mejorar el modelo y hacerlo más escalable en ese sentido.

La Figura 28 compara la sensibilidad en válidas y en problemáticas según la resolución, y mientras en las válidas se mantiene plana en torno al 0,97 y 0,99, en las problemáticas sube del 0,29 a 1280 al 0,50 a 1536 y hasta el 0,86 a 1920. Los coches lejanos que a baja resolución se perdían reaparecen al darles más píxeles, lo que confirma que esas plazas no fallan por una limitación del modelo, sino por la cantidad de píxeles con que se ve cada coche, que depende de la distancia y el ángulo de la cámara.

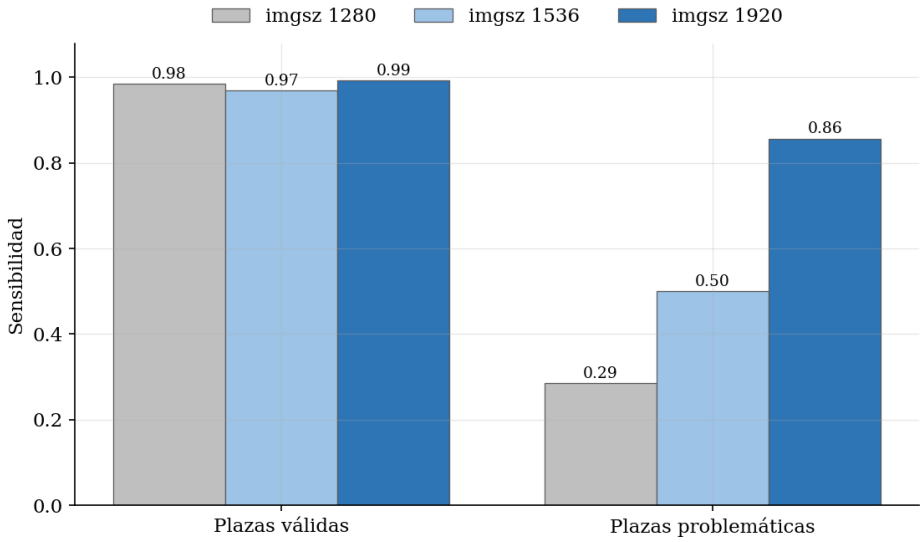


Figura 27. Sensibilidad en plazas válidas y problemáticas según la resolución, con confianza 0,25. Elaboración propia..

Esta mejora con la resolución es posible porque la cámara de la UEM graba a 2560 x 1440 píxeles, así que al procesar a 1280 se descartaba buena parte del detalle que el vídeo sí contenía, y al subir el tamaño de entrada se recupera información real de los coches lejanos, no inventada. El límite no lo pone solo el modelo sino la resolución que capta la cámara. Esto es importante porque luego afecta al estudiar su colocación y su tipo en el despliegue.

Sobre el conjunto completo, la Tabla 12 confirma que la mejor es la C5, con un F1 de 0,9709, un solo falso ocupado y nueve falsos libres. La C6, con confianza 0,15, baja de nueve a siete los falsos libres pero sube de uno a cinco los falsos ocupados, porque a confianza baja entran detecciones menos fiables que marcan plazas libres como ocupadas. Por eso a alta resolución es

mejor la 0,25, que mantiene mejor el equilibrio y no genera casi falsos ocupados que como ya se ha mencionado sería más crítico.

Configuración	FP	FN	F1
C5(1920/0,25)	1	9	0.9709
C6(1920/0,15)	5	7	0.9657
C4(1536/0,15)	4	13	0.9504
C2(1280/0,15)	1	17	0.9464
C3(1536/0,25)	1	22	0.9305
C1(1280/0,25)	1	27	0.9141

Tabla 12. Métricas globales incluyendo todas las plazas para las configuraciones YOLO11n. Elaboración propia

4.2.7.5 Coste de computo y frecuencia de muestreo

Queda también ver si el coste mayor de la alta resolución es asumible. El sistema no necesita procesar los 25 *frame*s por segundo del vídeo, porque un aparcamiento cambia despacio, la entrada o salida de un coche dura varios segundos y el estado de una plaza se mantiene estable mucho más que un *frame*, así que basta con revisar la ocupación una vez por segundo o cada varios segundos. Si que conviene no bajar mucho para que el sistema se actualice lo más cercano posible al tiempo real, y porque si se añaden por ejemplo notificaciones de incidencias estas se deberían notificar cuanto antes, entonces un margen, ni tan alto que malgaste cómputo ni tan bajo que el dato se quede desfasado.

Lo primero es saber cuánto encarece el *frame* subir la resolución, porque ese dato sirve luego para estimar el coste. Comparando la configuración ligera C1 con la de la más alta resolución C5:

$$\frac{T_{C5}}{T_{C1}} = \frac{0{,}2050}{0{,}0954} \approx 2.15$$

Es decir, la C5 tarda unas 2,15 veces más por *frame* que la C1. Pero, lo que cuenta no es el tiempo de un *frame*, sino ese tiempo por los *frame*s que se analizan en un segundo, de modo que el coste por segundo se calcula como

$$CPS = T \cdot f$$

Donde T es el tiempo que tarda el modelo en procesar un *frame* y f a la frecuencia del análisis. Esto permite compensar la resolución con la frecuencia, ya que para igualar el coste de dos configuraciones se iguala ese producto y la frecuencia de la más cara sale de:

$$f_{alta} = f_{baja} \cdot (\frac{T_{baja}}{T_{alta}})$$

Partiendo de la C1 a un *frame* por segundo, la C5 iguala su coste con:

$$f_{C5} = 1 \cdot \left(\frac{0{,}0954}{0{,}2050}\right) \approx 0{,}47\ FPS$$

Esto es un *frame* cada algo más de dos segundos y no tiene por qué ser la frecuencia que se acaba usando, sino el límite que marca el coste, y demuestra que la alta resolución no obliga a renunciar a nada, porque incluso analizando cada dos segundos el sistema funcionaria bien para la aplicación, así que la frecuencia real puede mantenerse más alta dentro del margen anterior sin que el cómputo sea un problema.

La Tabla 13 lleva ese tiempo a carga de trabajo según cada cuánto se analiza un *frame*, calculada como el tiempo de inferencia entre el periodo de muestreo. No es consumo eléctrico, para eso habría que medir la potencia del equipo, pero sirve como indicador de coste y viabilidad.

Configuración	Tiempo/*frame*	1 *frame*/s	1 *frame*/2s	1 *frame*/5s
C1(1280/0,25)	0.0954 s	9.54 %	4.77 %	1.91 %
C5(1920/0,25)	0.2050 s	20.50 %	10.25 %	4.10 %

Tabla 13. Estimación de carga temporal según la frecuencia de análisis. Elaboración Propia.

La conclusión es que la C5 es más costosa por *frame* que la C1, pero sigue siendo viable, porque ese coste se mantiene bajo aun analizando un *frame* por segundo. Es decir, ajustando los parámetros ya se ha ganado margen de cómputo antes incluso de comparar modelos más grandes.

4.2.7.6 Decisión de configuración

Juntando todo, la configuración elegida es la C5, YOLO11n a 1920 con confianza 0,25 e iou 0,5.

Como en las válidas todas quedan muy igulaadas, el criterio que desempata es el despliegue real, y esta cámara genera muchas plazas lejanas que un sistema completo debería cubrir. La 1920 las recupera, subiendo la sensibilidad en las problemáticas del 0,29 al 0,86 sin perder acierto en las válidas y con un solo falso ocupado en todo el conjunto.

En un despliegue con la cámara bien colocada no harían falta tantos píxeles y bastaría una resolución menor con el mismo acierto y menos coste, como se ve en las válidas, pero precisamente probar el sistema en estas condiciones desfavorables demuestra que es escalable, porque mantiene un buen resultado en una situación mala sin cambiar de modelo, solo subiendo la resolución y compensando el coste con la frecuencia.

La 1280 se conserva como referencia de que las plazas cercanas se resuelven sin coste añadido, y sería la opción adecuada si la cámara no dejara plazas lejanas. La confianza 0,25 se elige frente a la 0,15 porque a alta resolución la 0,15 apenas recupera coches y a cambio mete más falsos ocupados.

Aun así, 0,25 es un umbral bajo, por debajo de los valores que se usan habitualmente, y se fija así a propósito porque el modelo de fábrica detecta los coches lejanos y pequeños con poca seguridad, de modo que con un umbral más alto se perderían. Bajarlo es la forma de recuperarlos, asumiendo a cambio que el sistema es más permisivo, un riesgo que aquí queda controlado porque incluso a 0,25 apenas aparece algún falso ocupado. Que haya que bajar tanto el umbral es en sí una señal de que el detector trabaja al límite con esos coches, y es justo lo que se intenta corregir más adelante con el *fine tuning*, ya que un modelo afinado debería detectarlos con más confianza y permitir un umbral más alto y estable.

Esto se deja resumido en la tabla 14:

Papel	Modelo	imgsz	conf	iou	Motivo
Configuración del sistema	YOLO11n	1920	0.25	0.5	Cubre mejor todo el aparcamiento y mantiene pocos falsos ocupados
Referencia ligera	YOLO11n	1280	0.25	0.5	Referencia con la misma confianza para plazas cercanas y bien visibles

Tabla 14. Configuración elegida y referencia. Elaboración propia

Con esta configuración fijada, el siguiente apartado compara distintos tamaños de modelo manteniendo esta resolución y este método de asociación, para ir encadenando hacia la solución final completa.

4.2.8 Selección de modelo

Ya definidos los otros pasos anteriores, solo queda como último paso elegir el modelo que se va a utilizar. Hasta aquí todas las pruebas se hicieron con YOLO11n, así que se comprueba si un modelo con más capacidad o más reciente mejora el resultado, o si el modelo ligero ya es suficiente.

En esta prueba se mantienen fijos los parámetros de inferencia que se eligieron en el apartado anterior y se cambia una sola cosa, el modelo, de modo que cualquier diferencia se deba solo a él.

El flujo de cada prueba es idéntico al del apartado de configuración, pero con la diferencia de que ahora la pieza que se va sustituyendo es el detector y no los parámetros de entrada como en al anterior y como se muestra en la Figura 29 de ese apartado.

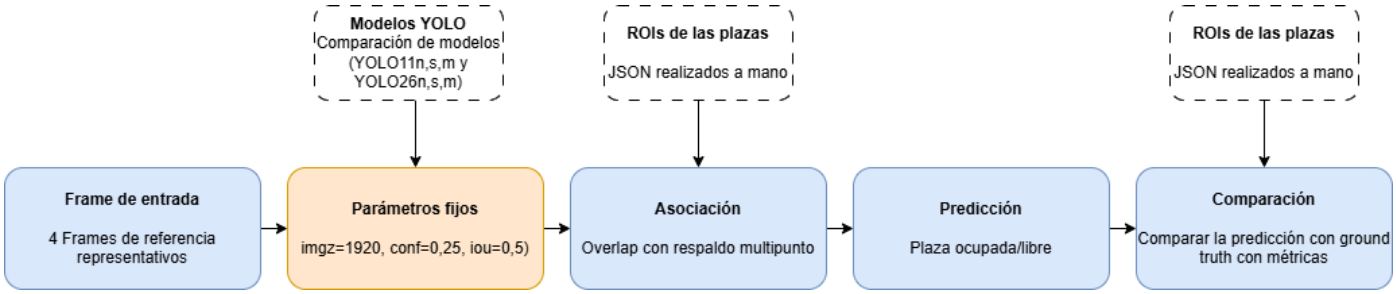


Figura 28. Esquema del flujo seguido para comparar de modelos. Elaboración propiade

4.2.8.1 Modelos evaluados

Se probaron seis modelos de dos familias, los tamaños *nano*, *small* y *medium* de YOLO11, que es la que se venía usando, y esos mismos tres de YOLO26, más reciente, para resolver dos dudas a la vez, cuánto influye el tamaño dentro de una misma familia y si la versión nueva aporta algo frente a la anterior al mismo nivel.

La Tabla 15 recoge los seis con el papel de cada uno:

Modelo	Mótivo
YOLO11n	Referencia ligera
YOLO11s	Posible equilibrio entre precisión y coste
YOLO11m	Comprobar si aumentar el tamaño mejora la ocupación
YOLO26n	Comparar la versión más reciente en tamaño ligero
YOLO26s	Comprobar si mejora a YOLO11s con coste parecido
YOLO26m	Comprobar el máximo rendimiento de los modelos probados

Tabla 15. Modelos YOLO evaluados. Elaboración propia

La evaluación se hizo sobre los mismos cuatro *frames* del *ground truth* y con la configuración ya fijada, midiendo el acierto por tipo de plaza y el tiempo de inferencia por *frame*, y dejando fuera las plazas en estado desconocido por el mismo motivo que en las pruebas anteriores de configuraciones (coches realizando maniobras en ese momento).

4.2.8.2 Resultados sobre plazas válidas

Igual que con las resoluciones, en las plazas válidas casi no hay diferencia entre modelos, como se ve en la Tabla 16. Cuatro de los seis aciertan absolutamente todo y los dos *nano* se quedan a una sola plaza, un falso libre cada uno, porque una plaza cercana y bien encuadrada es fácil y no necesita un modelo grande detrás como se puede ver. Esta tabla no sirve por sí sola para elegir, todos quedan prácticamente igualados, y lo único reseñable es que YOLO11s ya recupera ese falso libre del *nano* sin acercarse al coste de los medianos.

Modelo	FP	FN	F1
YOLO11s	0	0	1.0000
YOLO11m	0	0	1.0000
YOLO26s	0	0	1.0000
YOLO26m	0	0	1.0000
YOLO11n	0	1	0.9963
YOLO26n	0	1	0.9963

Tabla 16. Métricas de ocupación en plazas válidas para los modelos. Elaboración Propia

4.2.8.3 Resultados sobre todas las plazas

La parte que de verdad ayuda a tomar la decisión sale al mirar todas las plazas, incluidas las dudosas y problemáticas. La Figura 30 cruza el F1 con el tiempo por *frame* de cada modelo, donde el tamaño de cada punto es el peso del archivo del modelo (proporcionalmente), que va de la mano de su capacidad, más parámetros suponen un archivo más grande y más cómputo, así que un punto pequeño y arriba a la izquierda es lo ideal, ligero y acertado, y un punto grande es un modelo más potente pero más pesado de ejecutar.

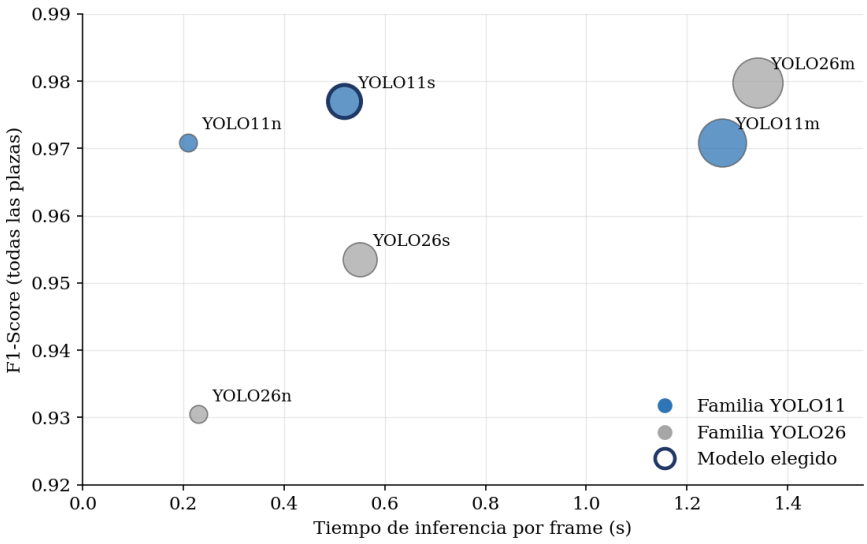


Figura 29. F1 sobre todas las plazas frente al tiempo de inferencia por frame de cada modelo. Elaboración propia.

Como se ve en la Figura 30 y se muestra con más detalle en la Tabla 17, el F1 más alto es para YOLO26m, con un falso ocupado y seis falsos libres, pero YOLO11s se queda a un paso con dos falsos ocupados y los mismos seis falsos libres, así que en acierto van casi igual y se separan en el coste.

Modelo	FP	FN	F1
YOLO26m	1	6	0.9798
YOLO11s	2	6	0.9770
YOLO11m	1	9	0.9709
YOLO11n	1	9	0.9709
YOLO26s	4	12	0.9535
YOLO26n	1	22	0.9305

Tabla 17. Métricas globales incluyendo todas las plazas para los modelos. Elaboración propia

Lo más interesante es que en estas condiciones en las que se analiza es YOLO11m no mejora en nada a YOLO11n, los dos terminan con el mismo F1 y el mismo reparto de errores pese a que el *medium* tarda unas seis veces más, y que YOLO26s, con un tiempo parecido al de YOLO11s, da más fallos de los dos tipos. Además, las plazas problemáticas no mejoran con un modelo más grande, donde el que más acierta es el propio *nano*. Igual que se vio con la resolución, el límite en esas plazas es de píxeles y no de capacidad del modelo, así que uno más potente no lo arregla.

Esto explica por qué los modelos grandes no rinden mejor aquí, porque un modelo con más parámetros distingue mejor los casos difíciles y destaca en escenas con mucha variedad, muchos tipos de vehículo, luz cambiante o escenas nocturnas, pero el problema de este trabajo es sencillo para un detector, una sola clase en una escena fija con las plazas siempre en el mismo sitio, así que un modelo pequeño llega pronto al máximo acierto posible y la capacidad extra no tiene dónde aprovecharse.

Hay que tener en cuenta, eso sí, que todo esto se mide sobre un solo vídeo en condiciones buenas, y que, en situaciones peores, con lluvia, de noche, con más oclusiones o la cámara sucia, es posible que esa capacidad extra de los modelos grandes empezara a notarse, porque es justo donde sirve. Aquí no se aprovecha, pero no se descarta que en otras condiciones pudiera hacerlo. Comprobar que en este escenario no hace falta un detector pesado es en sí un resultado también útil.

4.2.8.4 Coste de computo y tamaño

El acierto no basta para decidir, hay que mirarlo junto al coste, porque el sistema tiene que ir en un equipo ligero. Con el mismo criterio de coste por segundo del apartado anterior, lo que pesa no es el tiempo de un *frame* suelto sino ese tiempo por los *frame*s que se analizan cada segundo.

Entre YOLO26m y YOLO11s que son los más intereantes de comparar por los resultados que se han obtenido la relación de tiempos es:

$$\frac{T_{YOLO26m}}{T_{YOLO11s}} = \frac{1.3418}{0.5176} \approx 2.59$$

Unas 2.6 veces más por *frame*, para ganar solo 0.0028 de F1. Y frente a los *nano* es aún mayor, YOLO11m tarda unas seis veces más que YOLO11n sin mejorar nada para este caso y con las necesidades que se tienen. La tabla 18 lleva ese tiempo a carga de trabajo según cada cuánto se analiza un *frame*.

Modelo	Tiempo/*frame*	1 *frame*/s	1 *frame*/2 s	1 *frame*/5s
YOLO11n	0.21 s	21 %	10.5 %	4.2 %
YOLO11s	0.52 s	52 %	26 %	10.4 %
YOLO26n	0.23 s	23 %	11.5 %	4.6 %
YOLO26s	0.55 s	55 %	27.5 %	11 %
YOLO11m	1.27 s	127 %	63.5 %	25.4 %
YOLO26m	1.34 s	134 %	67 %	26.8 %

Tabla 18. Carga temporal por modelo según la frecuencia de análisis. Elaboración propia.

Como se ve en la Tabla 18, los *nano* y los *small* se ejecutan a un *frame* por segundo dejando margen, mientras que los medianos pasan del cien por cien, así que en este equipo no llegarían a procesar uno cada segundo y solo valdrían analizando más despacio.

Como segundo indicador se usa el tamaño del archivo del modelo, que no es la memoria que ocupa al ejecutarse pero sí dice si un modelo es más pesado y más difícil de meter en un dispositivo *edge*. YOLO26m pesa 42,21 MB frente a los 19,31 MB de YOLO11s, y los dos *nano* rondan los 5 MB.

No se analiza la memoria de vídeo VRAM (video random access memory), porque las pruebas se han hecho sobre CPU y ese consumo solo aparece al ejecutar sobre una tarjeta gráfica, algo que se valora en el apartado de hardware al comparar equipos con GPU.

4.2.8.5 Decisión de selección de modelo

El modelo elegido es YOLO11s, que entre los viables es el que mejor se adapta a esta cámara, porque acierta todas las plazas válidas, reduce los falsos libres del *nano* y queda casi igual que YOLO26m en acierto, pero con menos de la mitad de tiempo y un archivo más del doble de pequeño.

YOLO26m da el mejor número, pero su ventaja es mínima y cuesta más del doble en tiempo y tamaño, algo que no encaja con un sistema pensado para ir en un equipo de bajo consumo, así que se deja anotado como el mejor en cifras, pero descartado por coste. No es que el modelo grande sea peor, es que su potencia extra sirve para escenas más complicadas que esta y aquí no se aprovecha, y por eso YOLO11m, YOLO26n y YOLO26s también se quedan fuera. Esta elección es sobre el modelo base, antes de intentar mejorarlo, de modo que YOLO11s es la mejor base de partida sobre la que trabajar después.

La Tabla 19 resume la decisión sobre los modelos:

Papel	Modelo	Motivo
Modelo final para la simulación	YOLO11s	Mejor equilibrio entre acierto y coste de cómputo
Modelo ligero de referencia	YOLO11n	Muy rápido y de menor coste, suficiente en condiciones favorables
Mejor modelo numérico no elegido	YOLO26m	Mejor F1, pero coste de tiempo y tamaño demasiado alto
Modelos descartados	YOLO11m, YOLO26n, YOLO26s	No mejoran lo suficiente o peor equilibrio

Tabla 19. Decisión final de modelo. Elaboración propia.

Se puede ver en este análisis lo cerca que queda el *nano*, pero eso esta indicando algo sobre el escenario. En condiciones favorables, con plazas cercanas y el aparcamiento sin llenar, YOLO11n y YOLO11s aciertan casi lo mismo, e incluso en las plazas problemáticas el *nano* es el que más acierta de los 6, así que ahí el *nano* bastaría y sería lo más eficiente. La ventaja del *small* aparece cuando el escenario se complica, en las plazas difíciles que esta cámara genera por su mala colocación, y por eso se elige el *small* para este caso y se deja el *nano* como referencia muy válida también para despliegues bien colocados. Que la diferencia sea tan estrecha y se concentre en las plazas difíciles dice mucho, porque lo que queda por ganar no está en el escenario fácil, que ya está resuelto, sino en esas zonas problemáticas, que es justo lo que se intenta mejorar después.

En cualquier caso, esta elección responde al escenario concreto de la cámara de la UEM, y en un despliegue distinto, con la cámara mejor situada, la opción más eficiente podría ser el propio *nano*, lo que confirma que la decisión del modelo va ligada a las condiciones de captación y no es completamente definitiva si no para sacar conclusiones generales con datos reales de los que se dispone que se pueden extrapolar a otros escenarios.

La detección y la asociación no están hechas a medida de esta cámara, valen para cualquier aparcamiento con las plazas marcadas, como se vio al probarlas sobre UFPark con una vista muy distinta, y de un sitio a otro lo que cambia son los números, no la lógica.

Queda, eso sí, una limitación pendiente. En el clip de evaluación, que no llega a llenarse, el sistema recupera la mayoría de plazas, pero cuando el aparcamiento se llena los coches del fondo dejan de detectarse, como se vio en el apartado 4.2.5, y eso no lo arreglan del todo ni la resolución ni el tamaño del modelo. Para resolverlo hay dos caminos:

- Por software, reentrenando el modelo elegido con *fine tuning* para que detecte mejor esos coches del fondo.
- Por captación, añadiendo una cámara que cubra esa zona junto a la actual o planteando directamente la colocación ideal que se estudia más adelante.

Aparte de esta limitación, todo se ha probado sobre un solo vídeo, así que también quedaría por ver cómo responde el sistema con lluvia, de noche o con cambios fuertes de luz.

4.2.9 Mejora del detector mediante *fine tuning*

En el apartado anterior quedó claro que con el aparcamiento lleno los coches del fondo siguen sin detectarse, y que ni la resolución ni el tamaño del modelo lo resuelven del todo. Antes de plantear que la solución pase por la cámara, se intenta mejorar la detección por software con un *fine tuning*, que consiste en reentrenar el modelo elegido con imágenes nuevas para que aprenda a detectar mejor justo esos casos que falla. Es el primer camino a probar porque no cuesta nada intentarlo y se puede medir dentro del trabajo, y si funciona ahorraría tener que tocar la instalación, es decir la necesidad de añadir más cámaras. Aquí, solo se eta trabajando mide la detección de coches y no la ocupación de plazas, porque lo que se está mejorando es el detector, que es una pieza independiente de la asociación para el enfoque de este trabajo.

4.2.9.1 Planteamiento y estrategia de datos

El *fine tuning* parte del modelo elegido y lo sigue entrenando con imágenes parecidas a las que le cuestan, coches lejanos, pequeños, oblicuos o con oclusiones, para que mejore donde el modelo de fábrica se queda corto.

La primera decisión es que el entrenamiento no se hace con el vídeo de la UEM, por dos motivos:

- Solo se dispone de un vídeo de un día y con poca variación, así que entrenar solo con ese material llevaría a sobreajuste, especializando el modelo en esa escena concreta sin que aprendiera a generalizar.
- Ese mismo vídeo es el que se usa para evaluar, así que entrenar y evaluar con las mismas imágenes falsearía los resultados, porque el modelo las acertaría de memoria, un problema conocido como fuga de datos (data leakage).

Por eso el vídeo de la UEM se reserva entero como conjunto de prueba y para la demostración final, y el modelo se entrena con imágenes de otros aparcamientos. Entonces se parte de la limitación de no poder especializar el modelo a el *parking* por la poca cantidad de datos que se tienen, haciendo un estudio de si la generalización funcionaria, ya que el modelo se entrena con *parking*s distintos y se prueba en uno que no ha visto, que es la situación real de un despliegue en un aparcamiento nuevo. Es más exigente que entrenar y probar en el mismo sitio, pero también más realista, porque demuestra si el sistema sirve fuera del escenario en que se entrenó, que es lo que se busca como objetivo en realidad, que sea lo más escalable posible.

4.2.9.2 Construcción del dataset

El conjunto de entrenamiento se ha montado a mano partiendo de lo que fallaba. Al ver que el modelo base perdía los coches lejanos y pequeños, se buscaron aparcamientos con situaciones parecidas o más difíciles que la de la UEM, vistas elevadas y oblicuas, coches al fondo y oclusiones, para entrenar esa carencia concreta. No se trataba de juntar muchas imágenes, sino las que de verdad atacaban el problema.

El conjunto final tiene 671 imágenes de distintos aparcamientos, lo que aporta variedad de ángulos, distancias y encuadres. Para construirlo se combinaron imágenes procedentes de *dataset*s públicos de referencia, como PKLot, CNRPark-EXT y AIM *Parking Dataset*, junto con una selección propia de imágenes de Roboflow y otras fuentes, escogidas por su parecido con el caso de estudio., el AIM *parking dataset*, que incluye vistas muy variadas y que se encontró además de los más famosos para completar la variedad del *dataset* y que tienen vistas muy parecidas a la de este trabajo, y una selección propia de imágenes de Roboflow y otras fuentes, escogidas una a una por su parecido con el caso. La Figura 31 muestra ejemplos de algunos de esos aparcamientos.

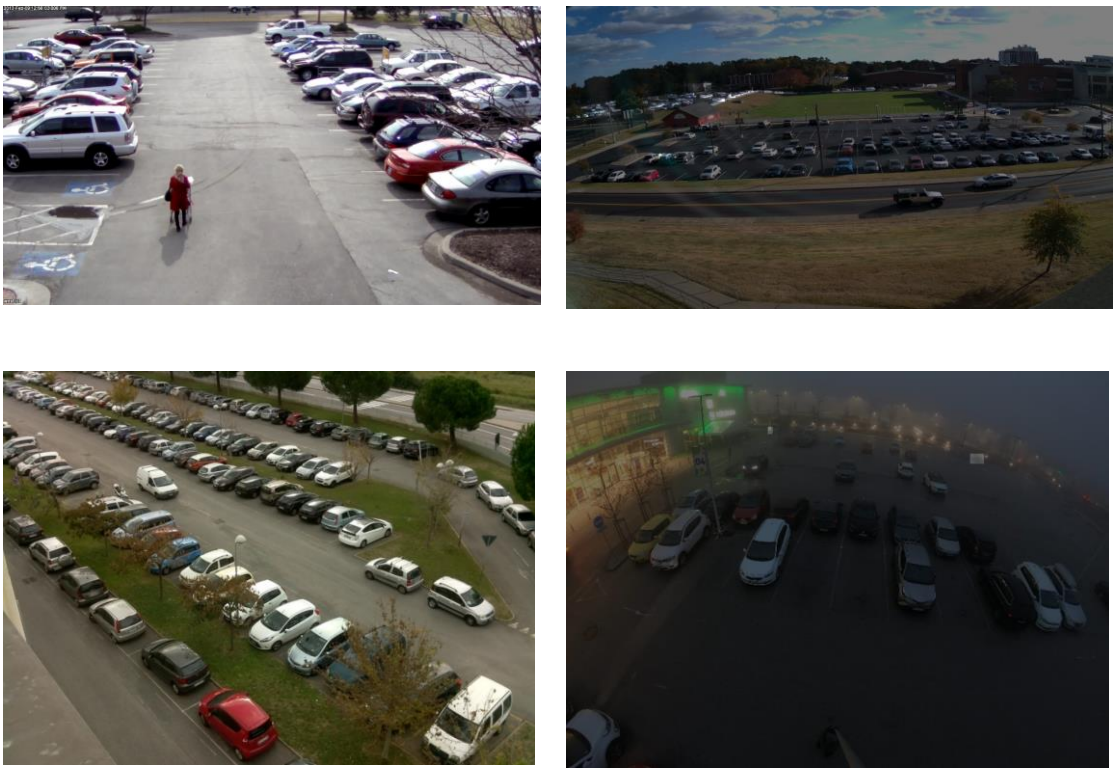


Figura 30. Ejemplos de vistas de distintos aparcamientos del conjunto de entrenamiento. Elaboración propia a partir de las fuentes utilizadas.

El *dataset* incluye también algunas imágenes en condiciones más variadas, como escenas nocturnas, lluvia, niebla o reflejos, para dar algo más de robustez al modelo. Aun así, el foco del trabajo y de la evaluación sigue siendo la detección de coches lejanos y pequeños, que es el problema detectado, por lo que el comportamiento ante esas condiciones no se evalúa aquí de forma específica, además de no disponer de videos con estas características en la zona piloto, y por ello queda como línea futura de mejora.

4.2.9.3 Distribución del dataset

Una vez reunidas y revisadas las imágenes, el *dataset* se repartió en entrenamiento, validación y prueba, cuidando que imágenes muy similares o procedentes del mismo aparcamiento no quedaran repartidas entre entrenamiento y evaluación, para no falsear los resultados por el mismo motivo de la fuga de datos. La distribución final se muestra en la Figura 32. De 489 imágenes se utilizan para entrenamiento (72,9 %), 108 para validación (16,1 %) y 74 para prueba interna (11,0 %).

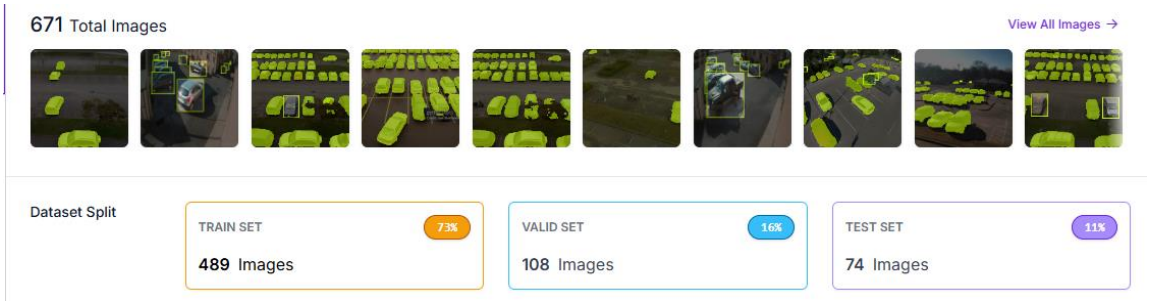


Figura 31. Distribución del dataset utilizado en el fine tuning. Elaboración propia

Esta prueba interna sirve para comprobar el comportamiento del modelo sobre imágenes externas no vistas durante el entrenamiento, pero no sustituye a la evaluación principal del trabajo. La prueba realmente importante sigue siendo la realizada sobre los *frame*s de la UEM, ya que ese es el caso real de estudio y el escenario sobre el que se quiere valorar si el sistema generaliza correctamente. De igual forma sirve para sacar las métricas correspondientes.

4.2.9.4 Etiquetado

Antes de haber dividido todo se hizo el etiquetado que fue con diferencia la parte más laboriosa y la más importante de todo el proceso, porque un modelo aprende exactamente de lo que se le marca, y un etiquetado descuidado arruinaría el entrenamiento por buenas que fueran las imágenes. No se trata de marcar un dato por imagen, sino de señalar uno a uno todos los coches de cada aparcamiento, que en las vistas llenas son decenas por imagen, encuadrando cada caja con cuidado en escenas con los vehículos pegados y en perspectiva.

Para agilizarlo se probó primero la anotación automática de Roboflow, que detecta los coches sola con su IA, pero no resultó fiable, ya que dejaba vehículos sin marcar y generaba bastantes cajas incorrectas, y fallaba sobre todo con los coches pequeños y lejanos, que son justo los que interesa detectar. Usar esas etiquetas tal cual habría metido errores en el entrenamiento, así que las 671 imágenes se revisaron una a una a mano, corrigiendo las cajas mal puestas, añadiendo los coches que faltaban y marcando desde cero los que la herramienta no resolvía. En la Figura 33 se puede ver este proceso sobre un mismo aparcamiento, donde se muestra la imagen sin anotar, anotada por Roboflow y luego corregida manualmente.



Figura 33. Proceso de etiquetado sobre un mismo aparcamiento. Elaboración propia

Se anotaron todos los coches que se distinguían a simple vista, incluso los muy lejanos ampliando la imagen para poder verlos, para enseñar al modelo a detectarlos en esas condiciones, y solo se dejaron fuera los que aparecían como una mancha irreconocible, donde ni el ojo humano distingue un coche. El caso más difícil de marcar fue el de los coches en oblicuo y muy juntos, uno detrás de otro, donde apenas se ve dónde acaba uno y empieza el siguiente, que es lento de anotar, pero es justo lo que aporta valor para detectar los coches del fondo.

4.2.9.5 Entrenamiento

El reentrenamiento parte de los modelos ya conocidos del apartado anterior se hace por aprendizaje por transferencia (*transfer learning*), partiendo de los pesos preentrenados de YOLO11s, YOLO11n y YOLO26s. De este modo, los modelos no empiezan de cero sino que conservan lo que ya saben sobre el reconocimiento de vehículos y siguen entrenándose sobre el dataset de aparcamientos del apartado anterior, que es lo que permite especializarlos con relativamente pocas imágenes.

Se afinan los tres modelos y no solo el que se eligió en la fase base, porque el *fine tuning* puede cambiar el orden. Un modelo que partía algo por detrás podría adelantar al resto al especializarse, así que dar por bueno al ganador base sin volver a comprobarlo sería un sesgo. Por eso, cada arquitectura se entrena y luego se compara contra su propia versión de fábrica sobre las mismas imágenes, y la elección final sale de esa comparación y no de una suposición previa.

En todos los entrenamientos se trabaja con una única clase, la de los coches, y las aumentaciones de datos se aplican solo sobre el conjunto de entrenamiento, dejando las imágenes de validación y prueba sin tocar para que representen escenas reales y midan de verdad la capacidad de generalizar. Como algoritmo de optimización, se usa AdamW en lugar de SGD porque converge más rápido y de forma más estable en *dataset*s pequeños como este, sin necesitar un ajuste tan delicado de la tasa de aprendizaje. Entre las aumentaciones se combinan cambios de brillo y color, escalado, traslación, volteo horizontal y la mezcla de varias imágenes con *mosaic*, cuya intensidad se fue regulando a lo largo de las pruebas, ya que una aumentación floja no aporta variedad

suficiente y una demasiado agresiva genera escenas que no se parecen a un aparcamiento real. El número de épocas de cada entrenamiento se fija como un máximo, y para no alargarlo sin necesidad ni caer en sobreajuste se usa la estrategia de la parada temprana (*early stopping*) con el parámetro patience, que corta cuando la métrica de validación deja de mejorar durante un número dado de épocas. De cada ejecución se guarda el archivo best.pt, con los pesos del mejor resultado de validación y no los de la última época, que no tiene por qué ser la mejor.

Los primeros entrenamientos se hicieron en la versión gratuita de Google Colab con una GPU NVIDIA T4, pero su memoria limitada y el corte de las sesiones al agotarse el cómputo gratuito volvían lentos e inestables los entrenamientos a 960 píxeles, así que se pasó a Google Colab Pro con una GPU NVIDIA A100, de coste contenido, que por su mayor memoria permitió subir el tamaño del lote para un gradiente más estable, entrenar y comparar los tres modelos en sesiones largas sin cortes. Los tiempos y la memoria de cada ejecución se recogen en el Anexo 3 como referencia del coste, sin usarlos para comparar entre sí entrenamientos hechos sobre hardware distinto.

La búsqueda de hiperparámetros no se planteó como un barrido de todas las combinaciones posibles, sino como un proceso en el que después de cada entrenamiento se revisaban las pérdidas, las métricas de validación, la matriz de confusión y el comportamiento sobre las escenas más difíciles, y a partir de ahí se decidía qué cambiar en el siguiente. En algunas pruebas se tocó más de un parámetro a la vez, así que el objetivo no era aislar el efecto exacto de cada uno, sino llegar a una configuración estable y buena para el problema.

Primera fase: resolución de entrenamiento

El primer entrenamiento se hizo con YOLO11s a 512 píxeles, que permitía un lote de 16 imágenes en la T4 y servía de primera referencia. Detectaba bien los coches más visibles pero seguía perdiendo los pequeños y lejanos y los que quedaban medio tapados, que son justo el problema. En el segundo se subió la entrada a 960 píxeles, múltiplo de 32 como exige la red, para que los coches pequeños perdieran menos detalle al redimensionar la imagen, un cambio que disparó la memoria y obligó a bajar el lote de 16 a 6. No se subió más, a 1280, porque a ese tamaño la memoria obligaba a un lote tan pequeño y unos tiempos tan altos que el entrenamiento se volvía inviable en Colab, así que 960 quedó como equilibrio entre detalle y recursos. La detección de objetos pequeños mejoró y las métricas de validación acompañaron, así que esa resolución se mantuvo en el resto de pruebas. Conviene separar esto de la inferencia, porque entrenar a 960 no obliga a inferir a 960, y de hecho la inferencia se hace a 1920, el tamaño fijado antes por la resolución real de la cámara, mientras que los 960 son solo un compromiso de la fase de entrenamiento.

La Figura 34 compara la misma escena lejana detectada con el modelo entrenado a 512 y a 960, donde se ve cómo a la resolución mayor se recuperan coches del fondo que antes se perdían (marcados en azul).



Figura 32. Efecto de la resolución de entrenamiento en la detección de coches lejanos, 512 frente a 960 píxeles. Elaboración propia

Segunda fase: pérdidas y aumentaciones

Con la resolución fijada, se probaron distintas combinaciones de los pesos de la función de pérdida, la tasa de aprendizaje y las aumentaciones. En el modelo 3 se subió el peso de la localización de las cajas para penalizar las detecciones demasiado grandes que aparecían sobre el fondo, pero no las quitó y además volvió al modelo más conservador con los coches lejanos y ocluidos, así que no se mantuvo. El modelo 4 cambió la configuración de forma más amplia, subiendo la tasa de aprendizaje inicial y bajando la final para explorar más al principio y afinar más al final, dando algo más de peso a las pérdidas de caja y clasificación y ampliando el *patience* para que el entrenamiento no se cortara antes de tiempo. También se moderó el *mosaic*, porque una combinación de imágenes demasiado agresiva genera escenas poco realistas que pueden enseñar al modelo a aceptar regiones grandes de fondo como si fueran un coche, que es parte del origen de esas cajas grandes falsas. Esta versión convergió de forma estable y mejoró claramente en los coches pequeños, oscuros del fondo y en los que quedaban medio tapados, y aunque seguía soltando alguna caja grande rara de vez en cuando, fue el mejor de los YOLO11s, así que quedó como candidato de esa arquitectura.

Los modelos 5 y 6 se hicieron para ver si se podía exprimir algo más. El 5 subió el lote, el peso de las cajas y el margen del *early stopping*, pero no eliminó las cajas grandes y encima perdió sensibilidad en coches agrupados y ocluidos, quedando demasiado conservador. El 6 reequilibró otra vez las pérdidas y suavizó algunas aumentaciones, y salió más equilibrado que el 5 pero sin superar al 4. De aquí salió una idea útil, y es que seguir subiendo el peso de la caja no arreglaba las detecciones grandes de forma fiable y encima podía costar *recall* en los casos difíciles, así que se conservó el modelo 4 y el problema de las cajas grandes se resolvió después con el filtro geométrico de la evaluación.

Tercera fase: comparación con otras arquitecturas

Con una configuración buena para YOLO11s, tocaba comprobar si otra arquitectura daba mejor equilibrio entre detección y coste, que es donde de verdad se ve si el ganador base sigue siéndolo después del *fine tuning*. El modelo 7 usó YOLO26s con los mismos parámetros comunes que el modelo 4 y resultó competitivo, con alguna mejora puntual en coches muy lejanos, pero menos constante que el YOLO11s en escenas con vehículos agrupados u ocluidos en agrupaciones con

vista lateral, aunque se conservó como representante de su familia para la evaluación final. El modelo 8 añadió regularización y planificación de la tasa de aprendizaje sobre el 26s buscando generalizar mejor, y mejoró en algunos coches pequeños pero perdió oclusiones, y al no superar de forma estable al 7 y por no disponer de suficientes unidades de cómputo para seguir ajustando los hiperparámetros de este modelo se optó por entrenar el modelo con YOLO11n, usando los hiperparámteros de modelo 4 y comprobar si se lograba un modelo competitivo usando menos computo. El modelo 9 entrenado a partir de YOLO11n para ver si una arquitectura más ligera se acercaba al *small* reduciendo mucho el tiempo de inferencia, y aunque sus métricas internas quedaron algo por debajo, fue lo bastante competitivo como para mantenerlo como alternativa ligera.

La Tabla 20 resume los entrenamientos y la conclusión principal de cada uno. Las configuraciones completas, las curvas de aprendizaje, las matrices de confusión y los registros de ejecución están en el Anexo 3.

Modelo	Arquitectura	Modificación principal	Épocas procesadas	Conclusión
M1	YOLO11s	Configuración inicial con imgsz=512	183/280	Referencia inicial, pierde vehículos lejanos
M2	YOLO11s	Aumento de imgsz a 960	142/280	Mejora la detección de coches pequeños
M3	YOLO11s	Mayor peso de la pérdida de caja	280/280	No resuelve las cajas anómalas y reduce sensibilidad
M4	YOLO11s	Rediseño de aprendizaje, pérdidas y aumentaciones	218/500	Mejor equilibrio, candidato YOLO11s
M5	YOLO11s	Mayor batch, peso de caja y patience	390/500	Comportamiento más conservador
M6	YOLO11s	Reequilibrado de pérdidas y aumentaciones	192/500	Resultado intermedio, no supera a M4
M7	YOLO26s	Configuración basada en M4	400/400	Candidato YOLO26s, competitivo pero menos estable
M8	YOLO26s	Regularización y planificación adicional	272/600	Mejora algunos vehículos lejanos, pero empeora las oclusiones

Modelo	Arquitectura	Modificación principal	Épocas procesadas	Conclusión
M9	YOLO11n	Configuración ligera basada en M4	419/500	Candidato YOLO11n por su menor coste

Tabla 20. Resumen de las configuraciones evaluadas durante el ajuste fino. Elaboración propia

La Figura 35 muestra la evolución del entrenamiento del modelo 4, el representante de YOLO11s. En la parte de pérdidas se ve cómo los tres componentes, el de la caja, el de clasificación y el la caja, la de clasificación y la de bordes, bajan deprisa en las primeras épocas y luego se aplanan, y lo importante es que las curvas de validación siguen a las de entrenamiento sin separarse, que es la señal de que el modelo no se está aprendiendo de memoria las imágenes sino aprendiendo a generalizar. Las métricas de precisión, *recall* y mAP suben en paralelo y se estabilizan en un *plateau* antes de llegar al máximo de épocas, momento en el que la parada temprana corta el entrenamiento al dejar de haber mejora apreciable. No aparece la separación entre entrenamiento y validación que delataría un sobreajuste, así que se interpreta como una convergencia sana.

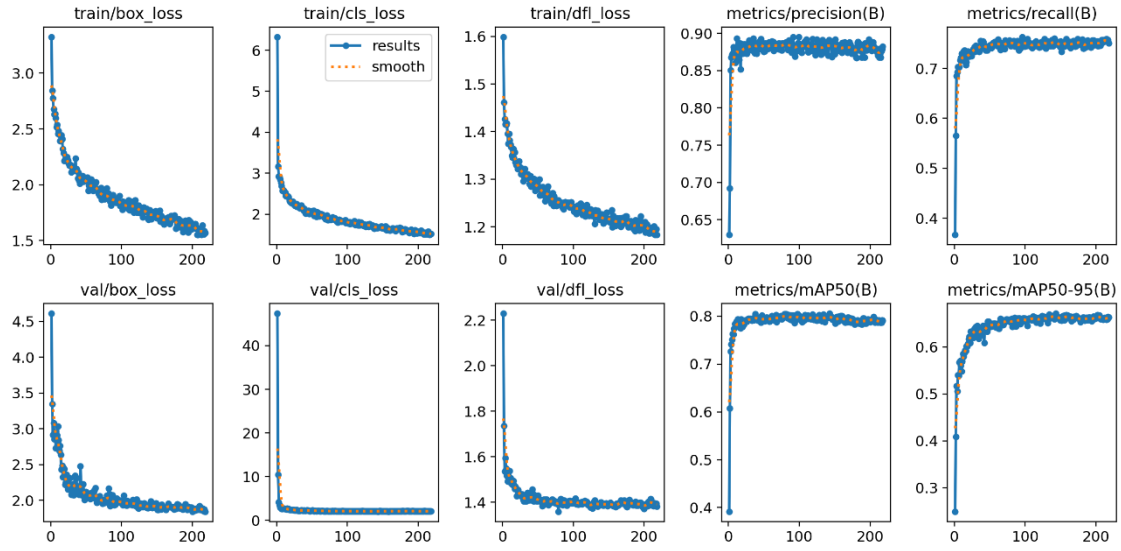


Figura 33. Evolución de las pérdidas y métricas de validación del modelo YOLO11s seleccionado durante el ajuste fino. Elaboración propia.

Como resultado de esta fase salen tres modelos afinados para la evaluación externa, el modelo 4 por YOLO11s, el modelo 7 por YOLO26s y el modelo 9 por YOLO11n. La elección final no se queda en las métricas internas de validación, sino que compara estos tres contra sus versiones base sobre las mismas imágenes reales de la UEM y con una configuración de inferencia común, como se explica en el apartado siguiente.

4.2.9.6 Planteamiento de la evaluación

Para medir si el *fine tuning* mejora la detección hay que comparar el modelo base contra el afinado en igualdad de condiciones, sobre las mismas imágenes y con los mismos parámetros, de forma que la única diferencia sea el entrenamiento.

La evaluación se hace sobre imágenes reales del aparcamiento de la UEM, que el modelo no ha visto durante el entrenamiento, así que mide de verdad la capacidad de generalizar a un escenario nuevo, teniendo en cuenta las dificultades que el anterior.

Se anotaron a mano las cajas de los coches en 9 imágenes del aparcamiento, elegidas para cubrir desde el *parking* casi vacío hasta el más lleno, buscando la mayor variedad de ocupación posible sin repetir la misma escena. La anotación se hizo en Roboflow y se exportó en formato YOLO, el mismo del entrenamiento, generando un fichero con las cajas reales de cada imagen. Esas cajas son el *ground truth* de detección, lo que de verdad hay en la imagen, es otro *ground truth* diferente al de ocupación, y muy eficaz para comparar la detección real.

La comparación entre lo que detecta cada modelo y esas cajas reales la hace un script de forma automática, emparejando cada detección con su caja real por solape y contando aciertos, fallos y falsas detecciones. Frente a la comprobación más manual del apartado de ocupación, aquí el recuento es directo sobre las cajas, lo que permite evaluar muchos coches por imagen de forma objetiva y repetible para los seis modelos.

La evaluación no se hace sobre toda la imagen, sino sobre una zona de estudio que rodea las plazas con un margen alrededor, generada a partir del mismo fichero de plazas que se usa en la ocupación. El sistema solo necesita detectar los coches del aparcamiento de la zona piloto, no todos los vehículos de la escena, como los de calles o zonas laterales que no forman parte del piloto. Si se evaluara toda la imagen, esas detecciones laterales afectarían a las métricas sin ser relevantes, así que se aplica el mismo filtro espacial a las anotaciones y a las predicciones, y solo se evalúan los vehículos dentro del área de estudio.

La Figura 36 muestra esa zona, con lo que queda fuera oscurecido. De las 555 cajas anotadas en total entre las 9 imágenes entran 520 en la zona y 35 quedan fuera. Esto limita la evaluación al alcance real del sistema, y las detecciones de fuera se registran en archivos auxiliares y se tienen en cuenta pero no entran en las métricas principales.

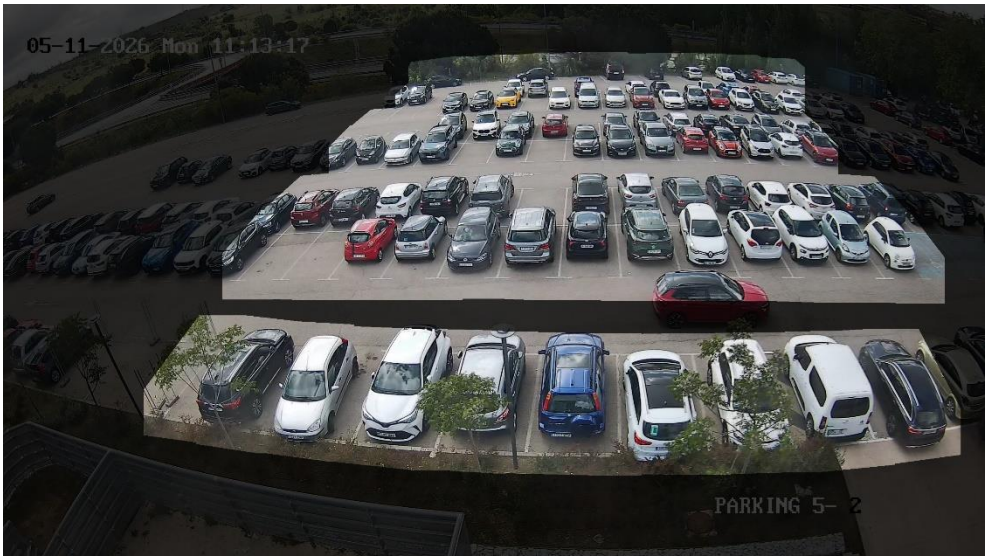


Figura 34. Zona de estudio sobre la que se evalúa la detección, basada en plazas de estudio. Elaboración propia

Al revisar las detecciones del afinado apareció un efecto secundario que había que solucionar antes de cerrar la comparación. Los modelos afinados generaban de vez en cuando alguna caja muy grande sobre el fondo, detecciones falsas que un coche real nunca produciría por su tamaño.

La causa de fondo es el *dataset*, que tenía pocas imágenes de *parking*s vacíos dado su dificultad para encontrarlos, así que la solución correcta habría sido completarlo con más imágenes de ese tipo para que el modelo aprendiera a no generarlas. Al no disponer de ellas, se aplica un filtro geométrico como postprocesado que descarta las cajas demasiado grandes para ser un coche en la escena, exige más confianza a las grandes y mantiene las pequeñas para no perder coches lejanos. Es una solución viable y aplicable de forma general, dado que se basa el tamaño físico posible de un vehículo y no por el resultado buscado, y se aplica con el mismo criterio a todos los modelos, aunque en la práctica solo actúa sobre los afinados porque el base no genera esas cajas. En la figura 37, se puede ver el problema que había con las cajas y como se solucionó después de aplicar el filtro geométrico.



Figura 35. Solución a problema de detección incorrecta en el parking piloto UEM. Elaboración propia

Una caja detectada se cuenta como acierto cuando se solapa lo suficiente con una caja real, medido con la IoU definida en el apartado 2.2.2, con un mínimo de 0,50, un valor que es estándar y suficiente para evaluar. El funcionamiento del modelo para sacar las métricas se observa en la Figura 38, donde se puede ver en verde las cajas que coinciden, en rojo los FP y en amarillo los FN, comparando los resultados del modelo YOLO11s en este caso.



Figura 36. Método métricos detección vehiculos comparación YOLO11s básico vs entrenado. Elaboración propia

Aparte de observarse el funcionamiento, ya se puede observar la mejora entre el modelo básico y el entrenado.

En cuentoa a la inferencia se hace a 1920 píxeles, con confianza 0,25, los mismos parámetros para el modelo base y el afinado, comos se habían fijado antes del fine tunning.

El umbral de solape interno del detector, que decide cuándo dos cajas muy juntas son el mismo coche, se subió a 0,60 porque con el valor anterior dos coches pegados uno detrás de otro se juntaba en una sola detección en algunas situaciones, y subirlo los separa en dos.

Las métricas que se utilizan para evaluar cual es el modelo, aparte de las ya utilizadas precisiones, sensibilidad y F1, son:

- **AP50**: AP50 media considerando como correctas las detecciones con IoU mayor o igual que 0,50.
- **mAP50-95**: media de AP considerando distintos umbrales de IoU entre 0,50 y 0,95.

Aunque AP50 y mAP50-95 se incluyen por ser métricas estándar en detección de objetos, en este caso la métrica más importante es la sensibilidad, especialmente en vehículos pequeños y lejanos, que ya se utilizo para la ocupación de plazas en el método de asociación. La sensibilidad es interesante en este caso sobre todo en coches pequeños y lejanos, porque el fallo que se quiere corregir son coches del fondo que no se detectan. Para verlo, las cajas se separan por tamaño aparente en pequeñas, medianas y grandes, donde las pequeñas son los lejanos.

La evaluación se hace imagen a imagen, sin suavizado temporal, para comparar los detectores de forma limpia. En vídeo real, algún fallo puntual de un *frame* podría corregirse manteniendo el

estado entre *frame*s con suavizado o seguimiento, pero esa capa no se incluye aquí para no inflar los resultados y se deja como mejora posterior.

4.2.9.7 Resultados del fine tunning

Se comparan los tres modelos elegidos, YOLO11s, YOLO11n y YOLO26s, cada uno en su versión de fábrica y en su versión afinada, con los mismos parámetros y sobre las mismas imágenes.

En la Tabla 21 se pueden verlos resultados globales:

Modelo	Precision	Sensibilidad	F1	AP50	mAP50-95	FP	FN
YOLO11n base	0,984	0,723	0,834	0,852	0,659	6	144
YOLO11n afinado	0,979	0,969	0,974	0,985	0,805	11	16
YOLO11s base	0,992	0,729	0,840	0,949	0,774	3	141
YOLO11s afinado	0,996	0,987	0,991	0,998	0,814	2	7
YOLO26s base	0,988	0,648	0,783	0,911	0,749	4	183
YOLO26s afinado	0,962	0,969	0,966	0,974	0,798	20	16

Tabla 21. Metricas globales de deteccion de cada modelo, base y afinado. Elaboracion propia.

Además, para representarse de manera más visual también se muestra de manera gráfica en la figura 39:

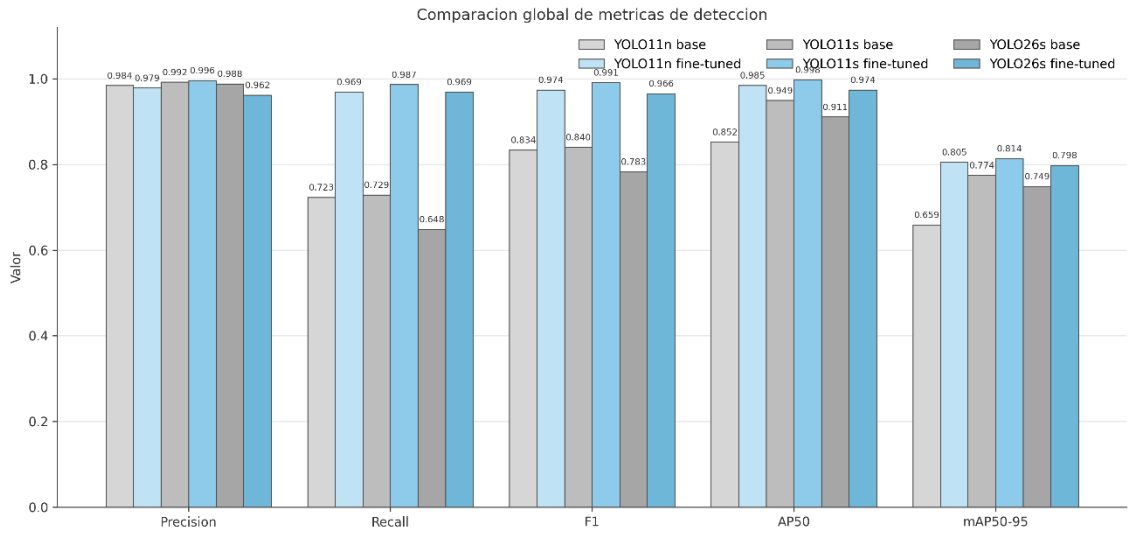


Figura 37. Comparación global de las métricas de detección de los seis modelos. Elaboración propia.

Lo primero que se ve en la Tabla 21 y en la Figura 39 es que los tres modelos base fallan en lo mismo, una precisión muy alta, por encima del 0,98, pero una sensibilidad baja, entre 0,65 y 0,73. Detectan bien cuando detectan, pero se dejan muchos coches sin detectar, justo los del fondo, lo que da muchos falsos negativos, 141 en el 11s, 144 en el 11n y 183 en el 26s, como muestra la Figura 40. Es el problema que se venía arrastrando, el detector de fábrica no ve los coches lejanos.

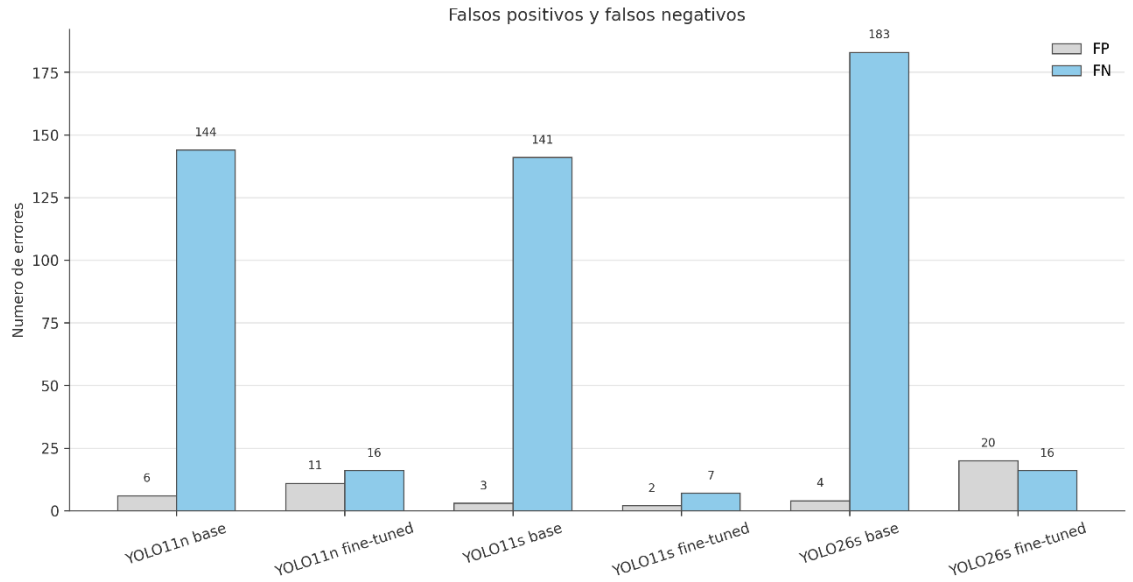


Figura 38. Falsos positivos y falsos negativos de cada modelo. Elaboración propia.

El *fine tuning* corrige eso de forma clara en los tres modelos. La sensibilidad sube por encima de 0,96 y los falsos negativos bajan muy fuerte, manteniendo la precisión alta, con reducciones del 95% en YOLO11s y también muy altas en el resto de modelos.. La mejora está en recuperar coches que antes se perdían, no en detectar de más.

Donde de verdad se puede observar esta mejora tan grande es en el análisis de sensibilidad por tamaño de coche, que se muestra tanto en la Tabla 22 como más visualmente en la Figura 41.

Modelo	Pequenos / lejanos	Medianos	Grandes / cercanos
YOLO11n base	0,593	0,996	0,244
YOLO11n afinado	0,960	1,000	0,902
YOLO11s base	0,347	0,996	0,878
YOLO11s afinado	0,975	1,000	0,976
YOLO26s base	0,236	0,950	0,768
YOLO26s afinado	0,980	0,996	0,866

Tabla 22. Recall por tamaño aparente del coche, base y afinado. Elaboración propia

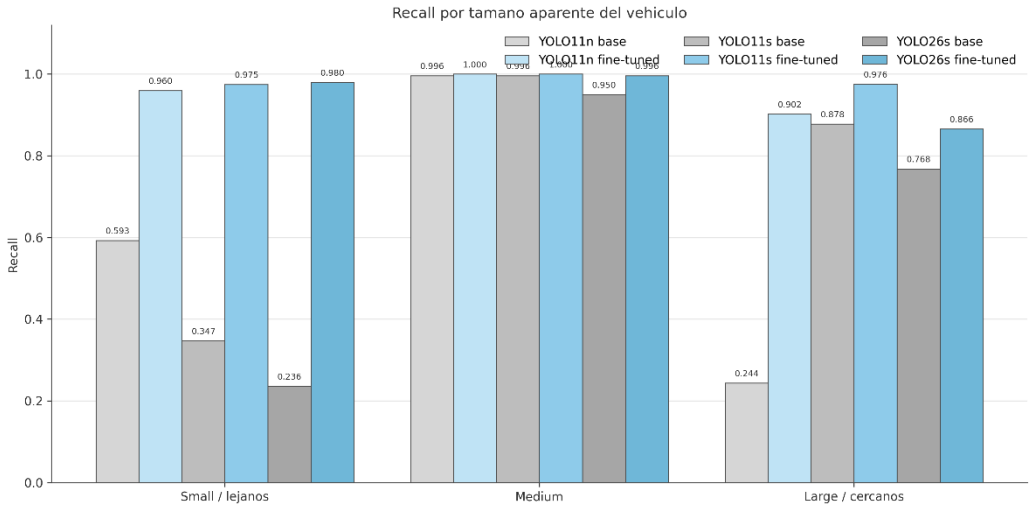


Figura 39. Recall por tamaño aparente del coche para los seis modelos. Elaboración propia.

En los coches pequeños y lejanos, que son el problema, el modelo base falla mucho, el 11s solo detecta el 0,347 de ellos, y el afinado sube al 0,975. Casi todos los coches del fondo que antes se perdían ahora se detectan. En los medianos y cercanos apenas había margen y se mantienen altos. Esto confirma que la mejora va dirigida justo donde estaba la carencia, los coches lejanos, y no es una subida genérica.

Entre los tres modelos afinados, la elección parte del YOLO11s, que era el elegido en el apartado anterior. El 11s afinado es el mejor en casi todo, recall 0,987, F1 0,991, solo 2 falsos positivos y 7 falsos negativos (teniendo en cuenta lo que se emencionó sobre los *frame*s), y sensiblidad en lejanos 0,975.

Viendo este resultado tan bueno, se decide comprobar los otros dos modelos referentes para ver si mejoraban al 11s. El YOLO26s afinado, incluido como alternativa adicional, no mejora al 11s en esta evaluación, da más falsos positivos, 20 frente a 2, así que se descarta, ya que aunque en principio es un modelo pensado para detectar coches lejanos de forma más efectiva, aquí los resultados no han sido los mejores. El YOLO11n afinado se queda muy cerca del 11s, con recall 0,969 y recall en lejanos 0,960, pero con la ventaja de ser una arquitectura más ligera, que tarda del orden de la mitad por imagen en inferencia. Como el ajuste fino solo reajusta los pesos y no cambia la arquitectura, el tiempo de cada modelo afinado es el mismo que el de su versión base, por lo que el detalle de velocidad y su efecto sobre la frecuencia de funcionamiento se tratan más adelante en su apartado.

La decisión como antes se decide buscando el equilibrio.

El 11s afinado se elige como modelo final porque da el mejor acierto en los coches lejanos manteniendo muy pocos falsos de los dos tipos, y porque en pruebas sobre otras vistas se mostró más estable que el *nano*. El 11n afinado queda como alternativa ligera, ya que acierta casi lo mismo a la mitad de coste, una opción a tener en cuenta en un despliegue con cómputo más ajustado, aunque con algo menos de margen en las situaciones difíciles. El *fine tuning* no cambia la familia elegida, sigue siendo YOLO11, pero confirma que el *nano* afinado se acerca mucho al *small*, reforzando que para este problema no hace falta un modelo grande.

Estos resultados son de la evaluación imagen a imagen, pensada para comparar los detectores de forma exigente. Sobre el vídeo completo el comportamiento se veía más estable y fluido, y los pocos fallos que quedan se concentran en algún coche del fondo en un *frame* suelto, que en vídeo continuo se recuperaría en los *frame*s vecinos. Esos fallos puntuales son los que mitigaría una capa de suavizado temporal, que se deja como mejora posterior y no se mezcla en esta comparación.

Otro efecto útil del *fine tuning* es que el modelo afinado detecta con más confianza que el de fábrica. El umbral de 0,25 se había fijado bajo porque el modelo base detectaba los coches lejanos con poca seguridad. Como ajuste posterior podría evaluarse un umbral más alto, por ejemplo 0,40 que es un valor típico, para ver si se reducen los pocos falsos positivos que quedan sin perder coches lejanos. No se aplica aquí para mantener los mismos parámetros entre todos los modelos, pero en el modelo final es una opción muy destacada, dado que esto podría evitar falsos positivos en otros entornos.

La Figura 42 muestra el efecto sobre una misma escena, con el base perdiendo los coches del fondo y el afinado detectándolos. Además de afinando la detección en coches muy pegados o con oclusiones. Se marcan en rojo los problemas en la imagen sin entrenar a la izquierda, que luego se ven solucionados en la entrenada a la derecha.



Figura 40. Detección sobre misma escena de aparcamiento. Marcando errores modelo básico a la izquierda y mostrando solución de modelo afinado a la derecha. Elaboración propia

En conclusión, los resultados muestran que el *finetuning* reduce de forma muy significativa el problema de los vehículos pequeños y lejanos en las escenas UEM evaluadas.

En conjunto, el ajuste fino corrige el problema que arrastraba el sistema, los coches pequeños y lejanos, sin estropear lo que ya iba bien, y lo consigue sobre un aparcamiento que el modelo no había visto y con un *dataset* no muy grande, así que la mejora no depende de haber entrenado con el propio *parking,* lo que demuestra que ha generalizado correctamente. En la práctica esto reduce la necesidad de compensar los fallos con cambios físicos en la instalación, como añadir otra cámara o mover la cobertura, aunque la colocación de la cámara sigue siendo un factor de diseño importante en el despliegue real. El YOLO11n con ajuste se queda como alternativa ligera de nuevo.

Con el detector resuelto, y dado que la asociación coche-plaza se había validado por separado, el último paso es volver a integrar el detector elegido en la parte de ocupación para tener el sistema completo en funcionamiento.

4.2.10 Integración y validación del sistema completo

4.2.10.1 Comprobación en la zona piloto

Elegido el YOLO11s afinado, se vuelve a montar en el sistema completo de ocupación para comprobar que sigue funcionando junto con el método de asociación vehículo-plaza desarrollado antes. Aquí no se repite una evaluación con métricas del detector, porque su mejora frente al base ya quedó medida en detección en el apartado anterior, sino que se comprueba si esa mejora en las cajas se traslada al estado final de las plazas, que es lo que de verdad usa el sistema.

Para verlo se comparan el YOLO11s base y el afinado sobre el mismo vídeo de la zona piloto, con las mismas regiones de plaza, los mismos parámetros de inferencia y exactamente la misma lógica de asociación, de modo que lo único que cambia entre las dos salidas sea el modelo que detecta los coches. En los dos casos se infiere a 1920 píxeles, con confianza 0,25 e IoU 0,60, y se mantienen el filtro de cajas anómalas, la huella inferior, el solape principal, el respaldo y la comprobación de segunda plaza.

Como el fallo que se arrastraba estaba en la detección y no en la asociación, cualquier diferencia entre las dos salidas se debe únicamente al modelo.



Figura 41. Comparación del sistema de ocupación en la zona piloto con el YOLO11s base y el afinado, usando la misma lógica de asociación. Elaboración propia

En la Figura 43 se observa que el sistema sigue funcionando bien después del ajuste fino. Las plazas de coches que el base ya detectaba se mantienen igual, lo que indica que el entrenamiento no ha estropeado lo que antes iba bien, y la diferencia aparece en las filas del fondo, donde el base perdía coches pequeños y el afinado los recupera, todas esats se encuentran marcadas en amarillo. Al aparecer esas cajas que antes faltaban, las plazas correspondientes dejan de salir como libres y pasan a marcarse como ocupadas.

La Figura 44 amplía la zona más alejada, donde se ve con más claridad cómo esas plazas pasan de aparecer libres con el base a marcarse ocupadas con el afinado.





Figura 42. Detalle del efecto del modelo afinado sobre la ocupación de las plazas más alejadas de la zona piloto. Elaboración propia

La comprobación sobre el vídeo completo confirma además que el modelo afinado se integra en el flujo sin cambiar nada del resto del sistema. La salida sigue marcando las plazas libres en verde y las ocupadas en rojo, y los cambios de estado del aparcamiento se reflejan a lo largo de la secuencia. Como material de apoyo se genera un vídeo comparativo del base y el afinado, que sirve para ver la estabilidad del resultado y analizarla.

Eso sí, esta comprobación es cualitativa como se ha mencionado, sobre la salida del sistema, no una medición de ocupación con métricas, así que lo que se afirma es lo que se observa en el vídeo, que las plazas del fondo que el base dejaba libres por no ver el coche pasan a marcarse ocupadas con el afinado. Con eso, la mejora del ajuste fino no se queda en producir mejores cajas de forma aislada, sino que se traslada a la limitación que afectaba al sistema completo en la zona piloto, ya que recuperadas las detecciones que faltaban, la asociación desarrollada antes las usa sin cambios para dar una imagen más completa del estado del aparcamiento.

4.2.10.2 Aplicación en otra vista del aparcamiento del Veterinario

Después de comprobar el funcionamiento del sistema completo en la zona piloto, se realiza una segunda prueba sobre otra vista del aparcamiento del Veterinario. Este encuadre presenta una perspectiva y una distribución diferentes, por lo que permite verificar si la solución desarrollada puede trasladarse a una nueva cámara sin depender exclusivamente de la escena utilizada durante el desarrollo principal.

Para esta comprobación se mantiene el sistema final sin realizar un nuevo entrenamiento ni modificar la lógica de funcionamiento. Se utiliza el mismo modelo YOLO11s afinado, el mismo filtro geométrico, la misma asociación vehículo-plaza y la configuración de inferencia establecida en las pruebas anteriores. La única adaptación necesaria es utilizar el fichero con las regiones de las plazas correspondientes a la nueva vista, ya que su posición y geometría aparente cambian con la perspectiva de la cámara.



Figura 43. Funcionamiento del sistema final de estimación de ocupación sobre otra vista del aparcamiento de Veterinaria. Elaboración propia

Como se observa en la Figura 45, el sistema consigue representar de forma coherente el estado de las plazas visibles en el nuevo encuadre, marcando en rojo las ocupadas y en verde las libres. En la escena mostrada, las plazas se corresponden correctamente con la presencia o ausencia de vehículos, lo que permite comprobar que el detector afinado y el método de asociación continúan funcionando correctamente fuera de la zona piloto.

Este resultado es especialmente relevante porque el modelo no se entrenó con imágenes procedentes de los aparcamientos de la UEM. El conjunto utilizado para el entrenamiento estaba formado por imágenes de otros aparcamientos, con diferentes perspectivas, distancias y condiciones visuales. A pesar de ello, el modelo es capaz de aplicar lo aprendido sobre una vista nueva y obtener un resultado correcto, lo que muestra que no se ha limitado a memorizar un escenario concreto.

No obstante, la comprobación realizada se limita a dos vistas del entorno de la UEM. Para confirmar completamente la capacidad de generalización sería necesario ampliar las pruebas a más aparcamientos, cámaras, condiciones de iluminación y épocas del año. Además, en una versión futura sería conveniente incorporar una identificación automática de las plazas, de manera que la adaptación a una nueva cámara no dependiera de una definición manual de sus regiones.

Aun con estas limitaciones, la prueba muestra que el sistema desarrollado no está restringido a la zona piloto y que puede reutilizarse en otros encuadres manteniendo el mismo modelo y la misma lógica de funcionamiento. Esto permite considerar la solución como una base escalable y ampliable para cubrir nuevas zonas mediante la incorporación de cámaras adicionales.

4.2.11 Frecuancia de funcionamiento y coste computacional

Con el modelo de detección elegido y el método de asociación cerrado, se midió cuánto cuesta ejecutar el sistema completo y cada cuánto tiempo tiene sentido hacerlo. Aunque el vídeo está grabado a 25 fotogramas por segundo, no hace falta procesarlos todos, porque el estado de un aparcamiento cambia mucho más despacio que en otras aplicaciones de visión en tiempo real como ya se vió anteriormente. La idea es quedarse con un intervalo que conserve la información que importa y que deje margen para el resto de tareas del sistema, como las comunicaciones o la interfaz.

Coste computacional del sistema completo

La medición se hizo con el modelo y la configuración de inferencia ya elegidos, sobre el sistema completo, y en tres secuencias de aproximadamente un minuto:

Se utilizaron tres secuencias de aproximadamente un minuto, procesadas inicialmente a una imagen por segundo:

- La zona piloto en una situación menos ocupada.
- La misma zona piloto prácticamente llena.
- Otra vista del aparcamiento de Veterinaria.

Lass dos primeras con 101 plazas y la tercera con 54, todas a 2560 × 1440 de resolución original de la cámara.

Hay un detalle de cómo se hizo la prueba que cambia cómo se lee. Se ejecutó en la CPU del equipo de desarrollo, sin tarjeta gráfica. No es el hardware con el que se desplegaría el sistema, sino al revés, el caso más desfavorable, un procesador corriente sin ninguna aceleración. La idea es que si el sistema cumple incluso ahí, cualquier equipo de despliegue con algo de aceleración irá igual o mejor, nunca peor. Lo que se mide no es el rendimiento final, sino el suelo por debajo del cual no va a bajar.

La Figura 46 muestra el tiempo medio por imagen en cada escena, separando detección y asociación.

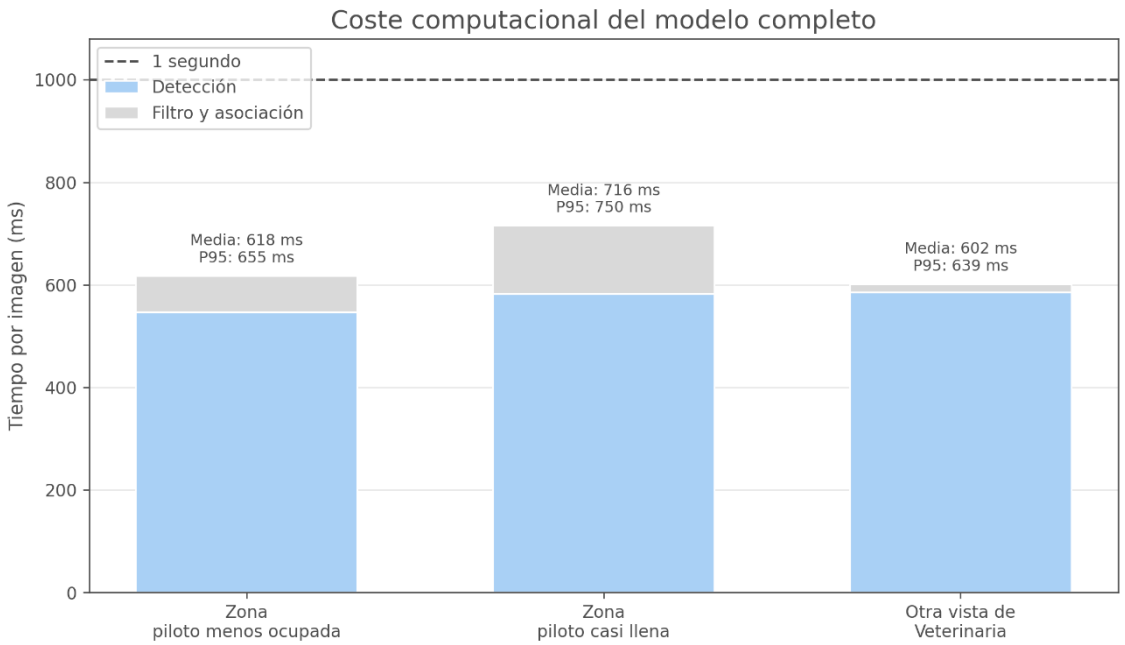


Figura 44. Coste computacional del sistema completo en los tres escenarios, separando detección y asociación. Elaboración propia

La figura enseña dos cosas. La detección se lleva casi todo el coste, sobre 550 a 585 ms, y apenas cambia entre escenas, mientras que la asociación, el método propio de este trabajo, cuesta muy poco, desde 15 ms en la escena con menos coches hasta 133 ms en la más llena. Que la asociación suba con la ocupación tiene sentido, porque a más coches detectados más cajas hay que comprobar contra las plazas, pero aun así es una fracción pequeña frente a la detección. Sumando las dos, el peor caso, la escena casi llena, se queda en 750 ms en el percentil 95, por debajo de un segundo. El percentil 95 es el tiempo por debajo del cual está el 95 % de las imágenes, y se usa en lugar de la media por ser más conservador, ya que se acerca al peor caso en vez de quedarse en el promedio.

De aquí sale la primera conclusión, procesar una imagen cuesta menos de un segundo incluso en el equipo más flojo y sin GPU. Eso fija el techo, el sistema podría analizar hasta una vez por segundo aun en el peor hardware, y deja claro que no necesita nada especializado para funcionar.

Cada cuánto conviene analizar

Que el sistema pueda ir hasta una vez por segundo no significa que deba, dado que cada cuánto analizar no lo decide el ordenador, que aguanta de sobra, sino el aparcamiento, porque lo que importa es cada cuánto ocurre algo que valga la pena ver. Y aquí el problema juega a favor. Un coche no aparece ni desaparece de golpe, tarda en maniobrar para entrar o salir de una plaza, del orden de varios segundos como poco. Eso quiere decir que un cambio de ocupación no es un instante, sino un evento que dura lo suficiente como para aparecer en varias imágenes seguidas aunque no se mire de forma continua.

Con esa idea, analizar una imagen cada dos segundos capta cualquier entrada o salida con holgura, porque la maniobra dura bastante más que ese intervalo y se ve en varias lecturas. Mirar más a menudo, por ejemplo cada segundo, no descubre ningún cambio nuevo, solo repite la misma lectura más veces y gasta el doble. Y como procesar es el grueso del trabajo del sistema, ese gasto de más no es menor. La Tabla 23 compara los intervalos por lo que de verdad los diferencia, la carga de cómputo que suponen.

Intervalo	Frecuencia	Análisis por hora	Fracción del ciclo ocupada	Valoración
1 s	1,0 FPS	3600	75 %	Capta lo mismo que 2 s gastando el doble
2 s	0,5 FPS	1800	37,5 %	Suficiente para el problema y la mitad de carga
5 s	0,2 FPS	720	15 %	Reducción grande de carga a coste de retardo

Tabla 23. Carga de cómputo según el intervalo de análisis, en el escenario más exigente. Elaboración propia

La comparación deja la decisión clara. LA segunda gasta el doble que dos para captar exactamente lo mismo, porque el evento que se quiere ver dura mucho más que cualquiera de los dos intervalos, así que ese gasto extra no compra nada. Cinco segundos sí ahorraría más, pero empieza a ser arriesgado, porque una maniobra rápida podría quedar justo entre dos análisis y verse a medias o no verse, y en un sistema que avisa de plazas libres no compensa apurar tanto. Dos segundos es el punto equilibrado, suficiente de sobra para el ritmo al que cambia un aparcamiento y con la carga reducida a la mitad respecto a ir cada segundo.

Hay además una razón que conecta con la etapa siguiente. El estado de cada plaza no se decide con una sola imagen, sino exigiendo que un cambio se repita en varias lecturas seguidas, que es la estabilización temporal del apartado próximo. Para que eso funcione interesa que el intervalo sea pequeño frente a lo que dura un evento, de modo que un cambio real genere varias lecturas antes de confirmarse. Dos segundos cumple esa condición, lo bastante frecuente para alimentar esa confirmación y lo bastante espaciado para no gastar de más, así que frecuencia y estabilización se diseñan juntas y no por separado.

Con todo esto se elige analizar una imagen cada dos segundos, equivalente a 0,5 fotogramas por segundo, manteniendo el último estado conocido entre análisis. El único efecto es un retraso de como mucho dos segundos en reflejar un cambio, imperceptible en un aparcamiento, y a cambio el sistema gasta la mitad y deja sitio para el resto de tareas.

Qué se le pedirá al hardware

Esta medición no elige aquí el equipo de despliegue, pero sí deja la lista de lo que tendrá que cumplir, que se concreta en el capítulo de hardware. Son tres condiciones. Ejecutar el detector elegido procesando cada imagen en menos de dos segundos, que es el tiempo entre análisis, algo que ya se cumple hasta en CPU. Disponer de memoria suficiente, ya que el proceso completo usó como máximo un poco más de 1 GB de RAM. Y dejar margen para las comunicaciones, la interfaz y el guardado de resultados, que es justo lo que libera el intervalo de dos segundos al no tener el procesador ocupado todo el rato.

La Tabla 24 reúne los valores medidos que sirven de referencia para esa elección.

Parámetro	Valor medido	Implicación
Tiempo total por imagen, P95	750 ms en el escenario más exigente	Debe permanecer por debajo del intervalo de 2 s
Intervalo de análisis	2 s	Tiempo disponible para completar cada ciclo
Memoria máxima del proceso	Aproximadamente 1,15 GB	El equipo debe disponer de esta memoria más un margen adicional
Modelo utilizado	YOLO11s afinado, 19,31 MB	Condiciona el entorno de inferencia y el almacenamiento
Frecuencia final	0,5 FPS	1800 inferencias por hora y cámara

Tabla 24. Requisitos de partida para la selección del hardware. Elaboración propia

No se da un consumo en vatios, porque la prueba fue en CPU y no se midió su potencia directamente, y porque el coste económico y energético se trata en su propio capítulo. Lo que sí se puede adelantar es que analizar cada dos segundos en lugar de cada uno reduce a la mitad el número de inferencias, por lo que todo esto se tiene en cuenta a la hora de buscar e dispositivo definitivo

4.2.12 Estabilización temporal de la ocupación

Con la frecuencia fijada en una imagen cada dos segundos, se añadió una última etapa de estabilización temporal. Hasta aquí el estado de cada plaza se decidía imagen a imagen de forma independiente, lo que dejaba pasar cambios breves cuando el detector perdía un instante la caja de un coche o cuando un vehículo maniobrando se asociaba de forma puntual a una plaza. La idea

del suavizado no es arreglar fallos permanentes del detector, eso no puede, sino evitar que una sola lectura aislada cambie el estado que se le muestra al usuario.

La forma de conseguirlo es no fiarse de una única imagen, sino exigir que un cambio se repita varias veces seguidas antes de darlo por bueno, y pedir más confirmaciones para liberar una plaza que para ocuparla. En concreto, las reglas que se siguen son:

- Una plaza libre solo pasa a ocupada cuando se detecta coche en 3 lecturas seguidas, que a dos segundos por muestra son unos cuatro segundos desde la primera detección, y con eso se evita que una asociación suelta de una maniobra o una caja mal puesta enciendan la plaza.
- Una plaza ya ocupada, en cambio, solo pasa a libre tras cinco lecturas seguidas sin coche, así que aguanta hasta cuatro pérdidas puntuales de la caja sin soltarse, y si el coche vuelve a aparecer antes de la quinta, la cuenta se reinicia y la plaza sigue ocupada.

Que cueste más liberar que ocupar esta hecho así queriendo, porque para el sistema es peor decir que una plaza está libre cuando el coche sigue aparcado, porque eso manda a un conductor a una plaza que no existe, que era justo el error que se quería evitar desde el principio. Por eso prefiere tardar unos segundos de más en avisar de una plaza disponible antes que ponerla libre por equivocación. El precio es un pequeño retraso, unos cuatro segundos para confirmar una ocupación y unos diez para una liberación, que en un aparcamiento no se nota, porque un coche tarda bastante más que eso en entrar o salir y lo que importa es que el estado sea estable, no instantáneo.

La estabilización se probó sobre las tres secuencias usadas antes. La

Figura 47 compara cuántos cambios de estado hay con y sin suavizado en cada una.

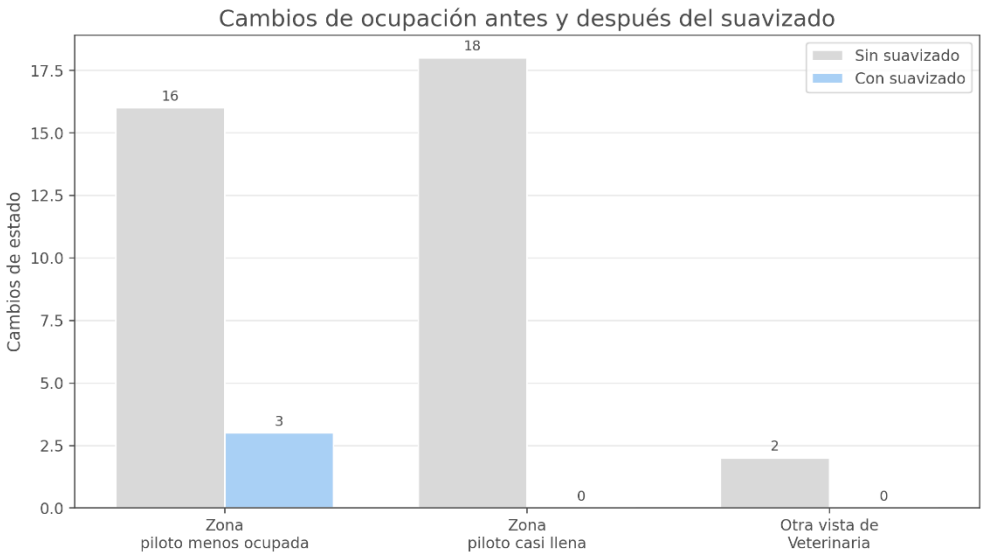


Figura 45. Cambios de ocupación antes y después del suavizado en los tres escenarios. Elaboración propia

Se ve que el suavizado baja mucho el número de cambios en los tres casos, lo que quiere decir que la salida sin suavizar tenía bastantes saltos cortos que no se correspondían con coches entrando o saliendo de verdad. La prueba más clara está en la zona poco ocupada, porque en esa secuencia se sabe que ocurren exactamente tres cambios de ocupación reales, y tras el suavizado el sistema deja registrados justo esos tres. No se cuela ningún cambio por los coches que maniobran ni se libera ninguna plaza por una pérdida puntual de la caja, así que ahí queda claro que la regla de tres lecturas frena el ruido pero deja pasar los cambios que de verdad pasan. La Figura 48 muestra esa misma secuencia a lo largo del tiempo, con la ocupación sin suavizar y suavizada.

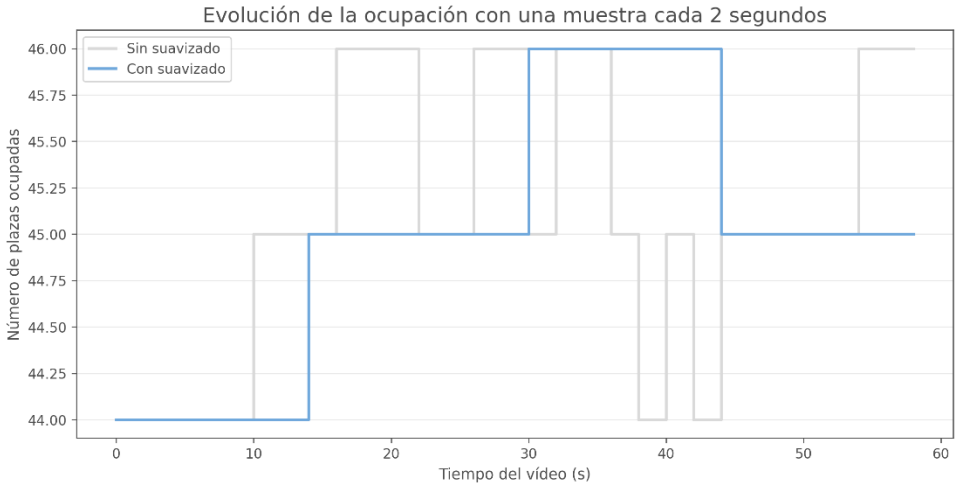


Figura 46. Evolución de la ocupación con y sin suavizado en la zona piloto poco ocupada. Elaboración propia

En esa figura se aprecia cómo la línea sin suavizar sube y baja en saltos cortos que son las pérdidas momentáneas de detección, mientras que la suavizada se queda firme y solo se mueve en los cambios reales.

En los otros dos casos el suavizado deja la ocupación completamente estable durante toda la secuencia, que es justo lo correcto, porque en esas escenas no entra ni sale ningún coche y el número de plazas ocupadas no cambia. Sin suavizado aparecían saltos cortos por pérdidas momentáneas de detección, y al aplicarlo desaparecen del todo, manteniendo el estado fijo como debe ser. Esto enseña bien para qué sirve la estabilización, ya que cuando no hay cambios reales el sistema no se inventa ninguno, y cuando los hay, como en la zona poco ocupada, los conserva.

Aun así, conviene dejar claro lo que el suavizado no hace, que es arreglar errores mantenidos en el tiempo. Si un coche deja de detectarse cinco lecturas seguidas, su plaza acabará marcándose libre aunque siga ahí, y un falso positivo muy insistente podría llegar a confirmarse, aunque con el margen de tres lecturas que se ha fijado no se ha dado el caso en las pruebas. A cambio introduce un retraso pequeño, mayor al liberar que al ocupar, atado a la frecuencia de dos segundos. Son límites asumibles frente a lo que aporta, que es una salida mucho más estable y fiel a lo que de verdad pasa en el aparcamiento.

Igualmente, con la estabilización se consigue que la salida del sistema represente mejor la evolución real de las plazas y no las variaciones puntuales del detector.

4.2.13 Detección de incidencias de estacionamiento

Además de estimar la ocupación, el sistema implementa una primera incidencia, la del coche que ocupa dos plazas a la vez. No hace falta nada nuevo para detectarla, porque la información ya la calcula el método de asociación. La comprobación de segunda plaza, que se añadió para que un vehículo a caballo entre dos plazas marcara ambas como ocupadas, confirma justo eso, que una única detección se extiende de verdad sobre dos plazas, y esa es la señal de que el coche está mal aparcado invadiendo la de al lado.

El funcionamiento es directo, básicamente cuando el método confirma que un coche ocupa una segunda plaza, además de marcar las dos como ocupadas, se genera un aviso indicando las plazas afectadas, para que de esta forma el equipo de seguridad lo pueda detectar con facilidad. La ocupación sigue funcionando igual, y encima de ella se añade esta capa de alerta que se apoya en un resultado que el sistema ya producía. En la demostración final esto se ve completo, con el aviso saltando sobre el coche que invade dos plazas. Este aviso se puede ver en funcionamiento en el capítulo 5 de desarrollo del sistema final.

El interés va más allá de este caso concreto, porque demuestra con algo ya implementado que el sistema admite alertas automáticas. A partir de aquí, otras incidencias se construirían igual, definiendo qué condición las dispara sobre la información disponible y lanzando el aviso correspondiente.

Conviene acotar qué cubre y qué no. Esta alerta detecta un tipo concreto de mal aparcamiento, el del coche que pisa dos plazas, que es el que el método identifica de forma fiable. Otros casos, como un coche torcido dentro de su propia plaza, mal encajado sin llegar a invadir, o fuera del área marcada, no los cubre esta incidencia y quedarían como líneas futuras, cada una con su propia condición de disparo. Por eso esta se desarrolla como demostración del mecanismo, mientras que el catálogo completo de incidencias se plantea como continuación natural del trabajo, ya que el sistema ha mostrado que puede soportarlas.

4.3 Conclusiones del desarrollo de visión

El desarrollo de este capítulo deja montado un módulo de visión que estima la ocupación del aparcamiento a partir del vídeo de una cámara fija.

La configuración que queda fijada es el detector YOLO11s afinado a una resolución de inferencia de 1920 píxeles, con confianza 0,25 e IoU 0,60, la asociación vehículo-plaza por huella inferior con solape y respaldo de puntos, un análisis cada dos segundos y una estabilización temporal que pide tres detecciones para ocupar una plaza y cinco ausencias para liberarla. Sobre esa base se añade además el aviso de un coche que ocupa dos plazas, como primera incidencia que demuestra que el sistema admite alertas.

Más allá de la configuración, el desarrollo deja una idea que condiciona el diseño físico del sistema. El problema de fondo no era solo de software, porque buena parte de los fallos venían de la distancia a las plazas del fondo, las oclusiones y la perspectiva oblicua, y eso no se arregla del todo entrenando mejor el modelo. El *fine tuning* recupera la mayoría de los coches lejanos, pero la colocación de la cámara sigue marcando el límite de lo que se puede ver, así que el diseño del sistema trata la cámara y su posición como una decisión de ingeniería, no como algo ya dado.

Como salida, el módulo entrega el estado confirmado de cada plaza, el número de plazas libres y ocupadas, las marcas de tiempo y las incidencias detectadas. Esa información es la entrada del sistema completo que se aborda a continuación, donde se diseña sobre qué hardware se ejecuta, cómo se capta y se comunica, y cómo se presenta al usuario y se integra con la entrada del aparcamiento.

Capítulo 5. Diseño e implementación del sistema y propuesta de despliegue

Una vez desarrollado y validado el sistema de visión, el siguiente paso consiste en integrarlo dentro de una solución que pueda utilizarse para gestionar un aparcamiento real. Hasta este punto se ha trabajado principalmente sobre la capacidad del sistema para detectar vehículos, asociarlos con las plazas y mantener una salida estable. En este capítulo se completa esa parte mediante una aplicación sencilla para el personal de seguridad y se define cómo se llevaría el sistema al entorno físico.

Primero se presentan las funciones que debe cumplir la solución y la forma en la que se han unido los distintos módulos dentro de una misma aplicación. Después se muestra el funcionamiento completo mediante la interfaz desarrollada. Por último, se estudian las cámaras que podrían reutilizarse en la UEM, las posiciones que requerirían nuevas cámaras, el hardware de procesamiento, las comunicaciones, la alimentación y la integración con la señal de aparcamiento completo y la barrera de acceso.

El diseño se plantea para que pueda trasladarse a otros aparcamientos, aunque se utiliza el campus de la UEM como caso de referencia. Por esta razón, se separan las decisiones generales del sistema de los datos concretos que dependen del plano, las cámaras ya instaladas y la infraestructura disponible en cada emplazamiento.

5.1 Funcionalidades del sistema

El objetivo de la solución no es únicamente mostrar las detecciones sobre un vídeo. El resultado debe convertirse en información útil para los usuarios y para el personal encargado de gestionar el aparcamiento. Para ello se han definido las siguientes funciones:

5.1.1 Gestión de la ocupación

La función principal es determinar el estado de cada plaza. El sistema debe identificar cuándo una plaza está libre u ocupada y mantener ese estado hasta que se confirme un cambio real. Como se ha explicado en el capítulo anterior, la salida no se modifica con una única detección, sino que se aplica una estabilización temporal para evitar que una pérdida momentánea del vehículo haga aparecer una plaza libre que en realidad sigue ocupada.

A partir del estado individual de las plazas se calculan también:

- El número total de plazas libres y ocupadas.
- La ocupación de cada zona del aparcamiento.
- El porcentaje de ocupación total.
- El estado visual de cada plaza sobre la imagen

La aplicación mantiene el último estado confirmado entre análisis, con la frecuencia de una imagen cada dos segundos fijada en el capítulo anterior.

5.1.2 Detección y registro de incidencias

La primera incidencia implementada es la detección de un vehículo que invade dos plazas. Cuando la asociación confirma que una misma detección ocupa de forma suficiente dos regiones vecinas, las dos plazas se marcan como ocupadas y se genera un aviso para el personal de seguridad.

El aviso registra como mínimo:

- el tipo de incidencia.
- la cámara y la zona en la que se produce.
- las plazas afectadas.
- la fecha y hora de inicio.
- el estado de la incidencia.
- la fecha y hora de cierre.

La incidencia no se genera por una única lectura. En la configuración final se confirma después de 6 detecciones consecutivas y se cierra cuando la condición desaparece durante cinco lecturas seguidas, aunque el operario también puede cerrarla manualmente. De esta forma, un coche que atraviesa momentáneamente dos plazas no se considera automáticamente mal aparcado, mientras que un vehículo que permanece en esa posición sí acaba generando el aviso.

En el prototipo se utiliza una única vista, por lo que la cámara y la zona están identificadas directamente en la cabecera de la aplicación. En una ampliación con varias cámaras, cada aviso debería incorporar también el identificador del nodo, la cámara y la zona correspondiente.

La estructura permite añadir otras incidencias más adelante sin cambiar el detector principal. Por ejemplo, podrían incluirse vehículos en zonas prohibidas, obstáculos en el carril, plazas reservadas utilizadas incorrectamente o pérdidas prolongadas de una cámara. En este trabajo se mantiene un catálogo reducido para demostrar el funcionamiento sin convertir el sistema en una plataforma de vigilancia demasiado compleja.

5.1.3 Control del nivel de ocupación y de la barrera

La aplicación incorpora una representación del control de acceso mediante una barrera virtual. El operario puede establecer con un control deslizante el número de plazas ocupadas a partir del cual el aparcamiento debe considerarse completo. Este valor puede coincidir con la capacidad total o fijarse por debajo si se desea limitar el acceso antes de alcanzar la ocupación máxima.

El control de la barrera dispone de dos modos:

- Modo automático: La barrera se abre o se cierra a partir de la ocupación estimada por el sistema. Para ordenar el cierre, el número de plazas ocupadas debe alcanzar el límite configurado durante dos análisis consecutivos. Esta comprobación se realiza sobre los estados ya estabilizados de las plazas y no directamente sobre las detecciones instantáneas.
- Modo manual: El personal responsable puede abrir o cerrar la barrera utilizando los botones de la interfaz, independientemente del porcentaje de ocupación detectado.

Cuando la ocupación se encuentra por debajo del límite establecido, el sistema muestra el aparcamiento como disponible y mantiene la barrera abierta en modo automático. Cuando se confirma que se ha alcanzado el límite, el estado cambia a completo y la barrera pasa a cerrada.

Si se detiene el módulo de visión, no se generan nuevos resultados de ocupación, aunque el vídeo continúa disponible. El control manual de la barrera permanece accesible para que el operador pueda actuar sin depender de la estimación automática.

En esta demostración, la barrera se representa únicamente mediante su estado dentro de la HMI. En una instalación real, la orden debería transmitirse al controlador físico de acceso. Los elementos de seguridad propios de la barrera, como fotocélulas, sensores de presencia y sistemas que evitan el cierre sobre un vehículo, deberán funcionar de manera independiente al sistema de visión. Además, lo correcto es cerrar la barrera cuando ha entrado la capacidad máxima de vehículos, no cuando todos han terminado de aparcar, porque durante las maniobras podrían seguir entrando coches. Por ello, la solución final tiene que incorporar un sensor de paso junto a la barrera que contabilice las entradas.

5.1.4 Información para seguridad y para los conductores

La solución diferencia entre la información necesaria para el personal de seguridad y la información que recibirían los conductores.

La interfaz interna desarrollada permite consultar:

- El vídeo de la cámara.
- El estado visual de las plazas.
- El número de plazas libres y ocupadas.
- El porcentaje de ocupación.
- El estado de la cámara y del módulo de visión.
- El estado y el modo de funcionamiento de la barrera.
- El límite de ocupación configurado.
- Las incidencias activas y finalizadas durante la sesión.

La tabla de avisos permite desplazarse por los registros generados, consultar las plazas afectadas y finalizar manualmente una incidencia activa.

La información destinada a los conductores es más sencilla. En el prototipo se representa mediante el indicador virtual situado en la parte superior de la aplicación, que muestra si el aparcamiento está disponible o completo. En un despliegue real, esta información se colocaría en un panel luminoso preparado para exterior y situado antes de la entrada, incluyendo opcionalmente el número de plazas libres.

El histórico implementado se limita a las incidencias registradas durante la ejecución de la aplicación. El almacenamiento permanente de datos, las estadísticas por día o franja horaria y el análisis de la duración de las ocupaciones se plantean como posibles ampliaciones posteriores.

5.2 Diseño de interfaz de usuario

Aquí se muestra el sistema con el que se logra poder interactuar con las funcionalidades establecidas.

5.2.1 Objetivo y herramienta utilizada

La interfaz se ha desarrollado como una aplicación de escritorio con CustomTkinter, porque el objetivo no era una plataforma web sino integrar el módulo de visión y los controles dentro del mismo entorno Python del desarrollo, lo que permite reunir indicadores, botones, tabla y vídeo en una sola ventana sin un servidor adicional. No es una interfaz hombre-máquina (HMI, human-machine interface) industrial certificada, sino un demostrador funcional, y en una implantación a mayor escala podrían mantenerse la misma lógica y los mismos datos aunque la interfaz se trasladase a una aplicación web, a un sistema SCADA o unificarse en una plataforma central.

El código se apoya en la separación en dos archivos, núcleo y aplicación, que se describe en el apartado de 5.3 de integración.

5.2.2 Pantalla principal y visualización

La interfaz se organiza en una cabecera con el estado general, un área central de vídeo y controles, y una tabla de avisos. Sobre el vídeo, las plazas se superponen en verde si están libres, en rojo si están ocupadas y en amarillo si están afectadas por una incidencia, con su número dibujado dentro de la región. En la parte derecha se agrupan el porcentaje de ocupación con su barra, los contadores de plazas, los avisos activos, el estado de la visión y de la barrera, el selector entre modo automático y manual, el límite de ocupación y los botones de apertura y cierre. El botón de detener la visión suspende la inferencia pero no la reproducción, de modo que el operador sigue viendo la fuente limpia mientras el estado superior pasa a visión detenida, lo que sirve para comprobar que la cámara sigue operativa y para distinguir un fallo de la fuente de una parada del algoritmo. La Figura 49 muestra la pantalla principal con la visión activa y la Figura 50 con la visión detenida.

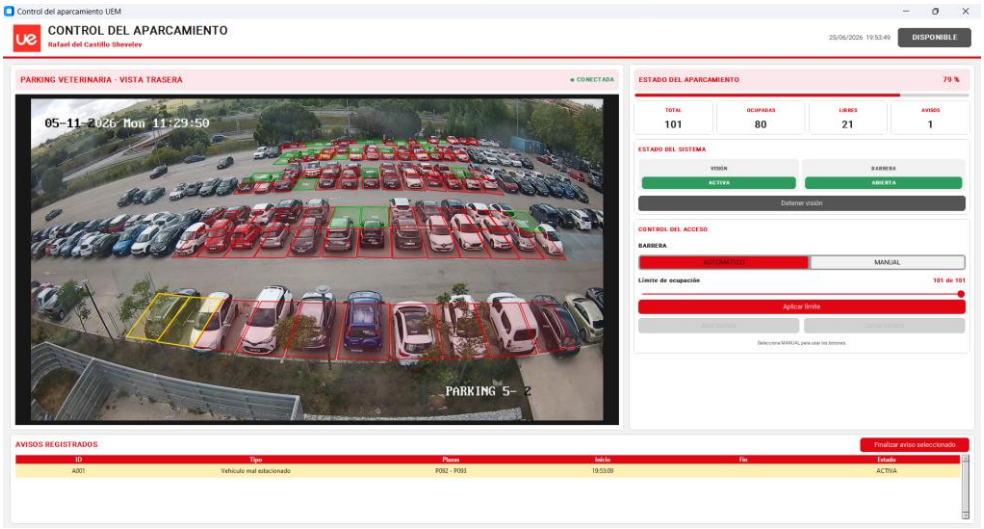


Figura 47. Pantalla final de la interfaz con la visión activa. Elaboración propia

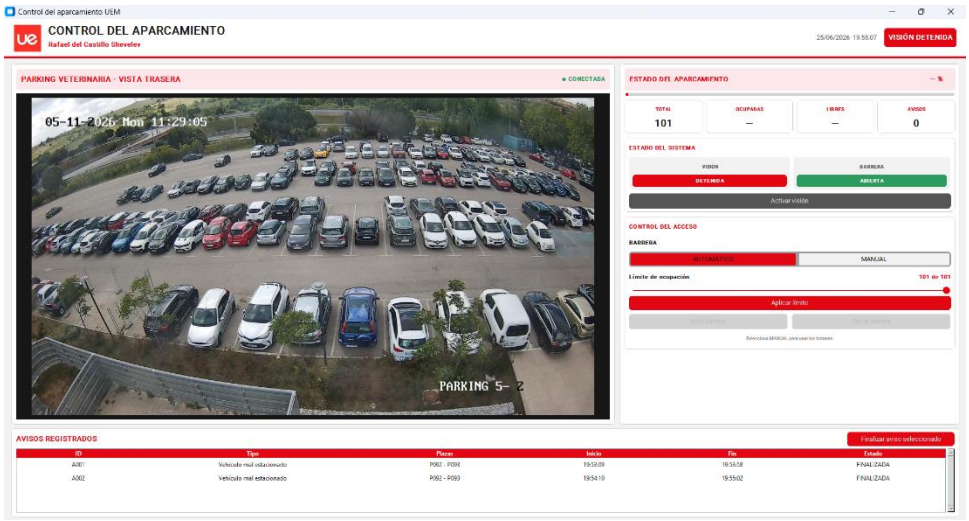


Figura 48. Interfaz con el módulo de visión detenido y el vídeo reproduciéndose sin superposiciones. Elaboración propia

5.2.3 Control de barrera y de incidencias

El control del acceso reúne el selector entre modo automático y manual, los botones de apertura y cierre, en verde y rojo, y el deslizante del límite de ocupación descrito en 5.1.3, cuyo valor debe aplicarse después de moverlo para evitar cambios accidentales.

Por otro lado, la tabla de avisos, en la parte inferior, recoge para cada incidencia un identificador, el tipo, las plazas afectadas, la hora de inicio y de cierre y el estado activo o finalizado, resalta las filas activas y permite cerrar un aviso a mano sin eliminar la fila. El histórico se conserva solo durante la ejecución y se pierde al cerrar la aplicación, porque todavía no hay una base de datos para ello, pero lo ideal seria guardar todo como un registro y poder consultarlo cuando se quiera.

La Figura 51 muestra el cierre automático de la barrera al alcanzarse el límite configurado, los botones equivalentes del modo manual y la tabla de avisos, con una incidencia finalizada mediante el botón correspondiente.

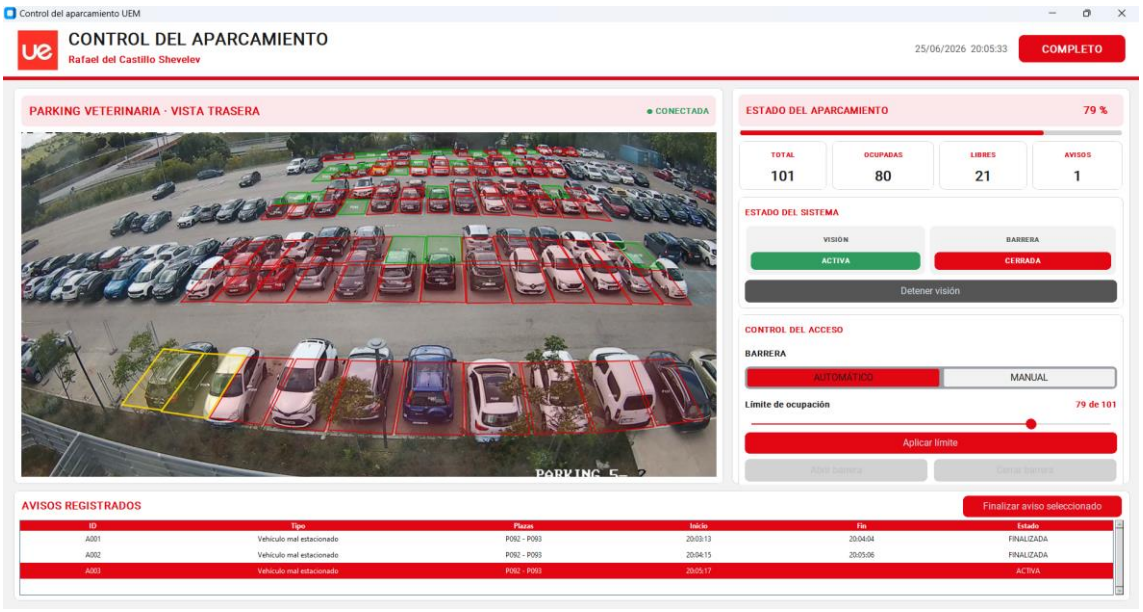


Figura 49. Control de la barrera y tabla de avisos de incidencias. Elaboración propia

5.3 Integración y funcionamiento del programa

El punto de partida fue el script de visión validado en el capítulo anterior, que permitía procesar una imagen o un vídeo, detectar los vehículos y calcular el estado de las plazas, pero que todavía estaba planteado como una herramienta de prueba, así que para convertirlo en una aplicación utilizable hubo que integrar la captura, la inferencia, la gestión de estados, las incidencias, el control de acceso y la interfaz dentro de un mismo flujo de funcionamiento.

Una de las primeras decisiones fue separar la reproducción del vídeo del análisis del modelo, porque el vídeo debía seguir mostrándose con fluidez, aunque YOLO necesitase cierto tiempo para procesar una imagen. Por ello el programa mantiene una función que lee y muestra continuamente los fotogramas y otra que analiza la muestra más reciente cuando se cumple el intervalo de dos segundos, y cuando termina la inferencia sus resultados se incorporan al estado del sistema y se mantienen visibles hasta el siguiente análisis.

Sobre esta base, el código se reorganizó en dos archivos principales. El primero, *parking*_core.py, contiene el núcleo del sistema y reúne la captura, la ejecución del detector, la asociación vehículo-plaza, la estabilización temporal, la gestión de incidencias y la lógica de la barrera, mientras que el segundo, app_*parking*.py, construye la interfaz gráfica., representa los resultados y recoge las

acciones del operador, una separación que evita mezclar la lógica de visión con el diseño de la HMI y permite modificar la presentación sin alterar el método de ocupación.

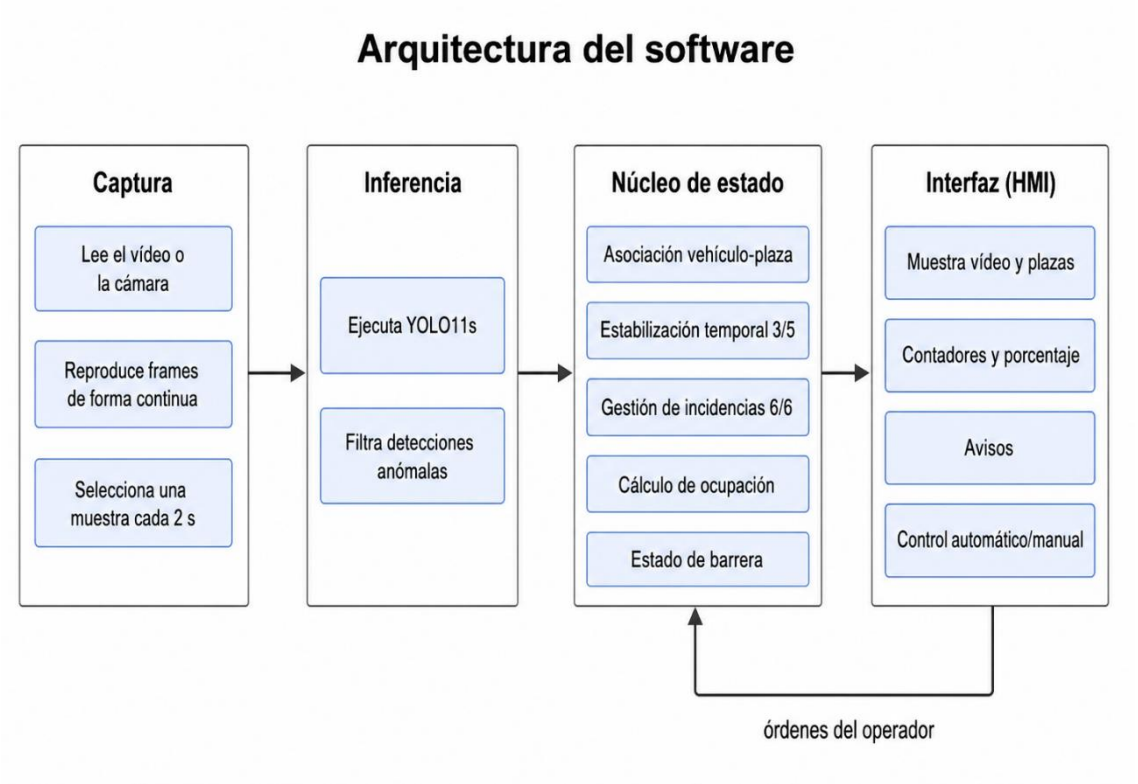


Figura 50. Arquitectura del software desarollado. Elaboración propia

La Figura 52 resume el flujo general, en el que la captura recibe el vídeo y selecciona una imagen cada dos segundos, la inferencia aplica el modelo YOLO11s afinado y genera las detecciones, la lógica de ocupación y control relaciona esas detecciones con las plazas, aplica la estabilización temporal, gestiona las incidencias y actualiza la barrera.

El núcleo mantiene en todo momento una representación del estado del aparcamiento, con el estado de cada plaza, los contadores, las incidencias, el límite configurado y la posición virtual de la barrera, y la interfaz consulta periódicamente esa información para actualizar sus indicadores. La lógica de incidencias y el control de acceso descritos en el apartado 5.1 se incorporaron reutilizando las mismas detecciones y estados, sin ejecutar un modelo adicional.

También se añadió un control básico de errores, de manera que antes de comenzar se comprueba la disponibilidad del modelo, del archivo de plazas y de la fuente de vídeo, y si alguno no puede abrirse o se produce un fallo durante la ejecución, la interfaz informa del error y evita presentar el sistema como operativo.

Aunque el vídeo es la entrada y se utiliza para mostrar el funcionamiento, la salida funcional del programa es el estado estructurado del aparcamiento, con los identificadores de las plazas, su situación libre u ocupada, los contadores, las incidencias activas y el estado de la barrera. En el

prototipo, el núcleo y la interfaz se ejecutan en el mismo ordenador y esa información se intercambia dentro del programa, mientras que en la solución final distribuida ese mismo estado se convertiría en mensajes enviados desde los nodos *edge* a la aplicación central, sin transmitir el vídeo completo, según la estructura de comunicaciones que se desarrolla en el apartado 5.8.

Las funciones de reconexión automática, supervisión simultánea de varios nodos, almacenamiento permanente e intercambio de mensajes por red no forman parte de la aplicación implementada, sino que se mantienen como requisitos de la solución final y se incorporan durante su despliegue.

5.4 Demostración y validación funcional del prototipo

La demostración se realiza con un clip de la UEM (el utilizado como piloto) que representa la entrada de una cámara fija y permite enseñar de forma reproducible el comportamiento de la aplicación. El recorrido cubre el arranque y la conexión con la fuente, el primer análisis, la actualización y estabilización de la ocupación, la parada y reactivación de la visión, el cambio de límite, el funcionamiento automático y manual de la barrera y la aparición y cierre de un aviso de doble plaza. La Tabla 25 reúne los casos de prueba con su resultado esperado.

Prueba	Acción realizada	Resultado esperado
Inicio de la aplicación	Ejecutar el programa	Se carga el modelo, se abre la fuente y se muestra el primer estado del aparcamiento.
Reproducción y muestreo	Mantener activa la visión	El vídeo se reproduce de forma continua y YOLO actualiza los resultados cada dos segundos.
Parada de la visión	Pulsar Detener visión	El vídeo continúa reproduciéndose, pero desaparecen las plazas y los resultados superpuestos.
Reactivación de la visión	Pulsar Activar visión	Se analiza el fotograma más reciente y vuelven a mostrarse los estados.
Ocupación de una plaza	Mantener un vehículo detectado sobre una plaza libre	La plaza pasa a ocupada después de tres detecciones consecutivas.
Liberación de una plaza	Dejar de detectar el vehículo sobre una plaza ocupada	La plaza pasa a libre después de cinco ausencias consecutivas.
Cambio del límite	Modificar el control deslizante y aplicar el nuevo valor	Se actualiza el número de plazas ocupadas necesario para considerar el aparcamiento completo.

Prueba	Acción realizada	Resultado esperado
Barrera automática	Alcanzar el límite configurado durante dos análisis	El sistema muestra el estado completo y representa la barrera cerrada.
Barrera manual	Seleccionar el modo manual	Se habilitan los botones de apertura y cierre.
Incidencia de doble plaza	Mantener un vehículo asociado con dos plazas	El aviso se genera después de seis análisis consecutivos.
Cierre de incidencia	Desaparece la condición o se finaliza manualmente	La incidencia pasa a finalizada y se registra la hora de cierre.

Tabla 25. Casos de prueba de la validación funcional de la aplicación. Elaboración propia

Las pruebas realizadas permiten comprobar la integración entre la captura, la inferencia, la estabilización, las incidencias, la HMI y la lógica de control de acceso, aunque esta validación se limita a la aplicación y todavía no incluye una cámara IP por RTSP (real time streaming protocol), una red de comunicaciones, el nodo en *edge*, un panel exterior ni una barrera física. Estos elementos pertenecen a la solución final propuesta y tendrían que validarse en su implementación final..

5.5 Despliegue y cobertura de las cámaras sobre el aparcamiento

5.5.1 Análisis de reutilización de cámaras

El sistema parte de la infraestructura existente, así que la primera cuestión es si las cámaras de la UEM pueden aprovecharse, ya que reutilizar una cámara reduce el coste de compra, cableado y montaje. El acceso a ese material fue limitado, porque no se facilitaron las grabaciones de todas las cámaras del campus sino solo un conjunto de vídeos, y de entre ellos se seleccionaron las vistas que de verdad servían. Varios clips se descartaron porque la cámara apuntaba al carril de acceso, porque quedaba demasiado lejos para distinguir las plazas o porque su perspectiva era tan baja que un coche tapaba al de detrás, situaciones en las que no se puede medir la ocupación de forma fiable. Quedaron dos vistas que sí permitían distinguir los vehículos en su zona útil, y son las que se han utilizado durante todo el trabajo.

El criterio que sostiene esa selección es que la cámara mantenga un encuadre estable, porque las plazas se definen mediante un archivo asociado a coordenadas de imagen y cualquier giro o cambio de zoom invalida esa correspondencia. Las dos vistas reutilizadas proceden de cámaras con zoom y movimiento, así que se toman como fijas, es decir, el sistema las usa con el encuadre bloqueado en una posición. Esto refleja una limitación actual del método, que necesita un encuadre conocido, pero con la vista fija el modelo funciona sobre ellas igual que sobre cualquier cámara fija, de modo que las dos vistas se incorporan como si lo fueran y el resto del aparcamiento se cubre con las cámaras nuevas que se despliegan para ello. Superar esa limitación, con una identificación automática de las plazas que admita cámaras en movimiento, se recoge en las líneas futuras del trabajo.

5.5.2 Despliegue completo

El objetivo no es cubrir el aparcamiento de forma sobrada, sino cubrir todas las plazas con el menor número razonable de cámaras. Lo que se plantea es una propuesta base optimizada sobre el plano CAD del campus, calculada con el modelo actual y todavía sin probar en el aparcamiento real, de modo que su confirmación corresponde a una prueba piloto. El aparcamiento se divide en sectores porque una sola cámara que abarque demasiada superficie deja las últimas filas muy pequeñas y hace que las primeras oculten a las posteriores. Las cámaras se montan sobre las farolas existentes, que ya disponen de canalización eléctrica, a una altura de 8 m frente a los 6 m de las cámaras de vigilancia actuales, porque una posición más alta reduce las oclusiones entre filas y da un ángulo más cenital sobre los coches.

El primer paso es elegir la cámara, porque sus ángulos ópticos son los que después fijan el cálculo de la cobertura. Se opta por una *bullet* IP fija de 4 megapíxeles con lente de 2,8 mm, descartando la domo, menos cómoda de orientar en farola, y la PTZ, cuyo movimiento rompe la relación cámara-plaza de la que depende el archivo de plazas de la aplicación que se ha desarollado. El modelo de referencia es la Hikvision DS-2CD1043G2-I de 2,8 mm (Hikvision, s.f.). Cumple los requisitos de exterior del apartado 2.4.1, y para esta instalación interesan especialmente las características que tiene:

- Sensor de 4 megapíxeles (2.560 × 1.440), resolución suficiente para que el coche del fondo conserve píxeles sin disparar el coste ni el tráfico de red frente a un sensor de 8 y dejando margen por si fuera necesario usar su resolucion completa, aunque el modelo actual no la use.
- Lente fija de 2,8 mm, que da un ángulo amplio y cubre muchas plazas por cámara y es más barata que la varifocal.
- Protección IP67, necesaria porque la cámara va a la intemperie sobre la farola y tiene que aguantar agua y polvo todo el año.
- WDR de 120 dB, que es un rango dinámico alto y permite detector bien zonas muy iluminadas y oscuras.
- Infrarrojo de hasta 30 m, que mantiene la detección de noche en las filas cercanas, apoyado por el alumbrado del aparcamiento.
- Compresión H.265, que reduce el tráfico hacia el nodo y es compatible con el códec de las cámaras del campus.
- ONVIF (*Open Network Video Interface Forum*), que permite la integración de la cámara con otros sietamas de otras márcas al ser un estándar común IP, y PoE (*Power over Ethernet*), que llevan datos y corriente por un solo cable, lo que simplifica el montaje en la farola.
- Variante sin micrófono, para no captar audio en un espacio público, en línea con el tratamiento de datos del proyecto.

El modelo seleccionado se muestra en la Figura 53.



Figura 51. Camara Hikvision DS-2CD1043G2-I seleccionada para despliegue. De VISIOTECH

De su óptica de 2,8 mm salen los dos ángulos de los que depende el resto del cálculo, un campo de visión horizontal HFOV de 99° y uno vertical VFOV de 55° que se espefician en el *datasheet* del fabricante. La lente de 4 mm y la varifocal de gamas equivalentes quedan como alternativa para los puntos críticos que lo requieran tras el piloto, que se han valorado como opciones pero se ha decidico optar por la fija para reducir el coste y aumentar el campo de vision.

Con esa cámara, el alcance sobre el suelo se obtiene de la altura y la inclinación. La cámara está a una altura h de 8 m e inclinada 32° por debajo de la horizontal, que se elige así para poder cubrir las plzas cercanas y manetenr alcance de las lejanas, y como su campo vertical es de 55°, el borde inferior del encuadre mira 27,5° más abajo del eje y el superior 27,5° más arriba, es decir, a 59,5° y a 4,5° por debajo de la horizontal.

Aplicando la expresión de distancia al suelo definida en el apartado 2.2.4:

$$d = \frac{h}{\tan(\alpha)}$$

Aplicándolo a cada borde, el rayo superior, a 4,5°, alcanza:

$$\frac{8}{\tan(4,5°)} \approx 102 \text{ m}$$

El inferior sitúa la primera fila en 4,7 m y el eje que está a 32°, deja el centro en 12,8 m. La plaza más lejana del despliegue está a unos 71 m, de manera que todas caen dentro del alcance geométrico de la cámara y lo que limita el sistema no es la geometría, sino la resolución con la que se ve el coche del fondo.

El ancho cubierto a cada profundidad se calcula mediante la expresión del campo horizontal presentada en el apartado 2.2.4 también:

$$W = 2 \cdot D \cdot \tan\left(\frac{\theta}{2}\right)$$

Con θ el campo horizontal de 99° y su mitad 49,5°. En el centro, a 12,8 m por ejemplo , el ancho es:

$$2 \cdot 12{,}8 \cdot \tan(49{,}5°) \approx 30 \text{ m}$$

Con este valor se ocupan unas 12 plazas, teniendo en cuanta que son de de 2,5 m por fila, y a 68 m la escena llegía a unos 159 m. Como cada cámara cubre así del orden de varias decenas de plazas, hasta cerca de noventa en las posiciones centrales del *parking* del banco Santander, el número de cámaras necesario es bajo, pero reducuendo el tamaño de los pixels de los vehiculos más lejanos

Las farolas se habían localizado antes recorriendo el aparcamiento a pie y situándolas sobre el plano CAD para medir su distancia real a cada a cada plaza. Con todos estos datos ya solo quedaba por decidir el azimut de cada cámara, es decir, su orientación horizontal sobre el plano, hacia qué lado del aparcamiento apunta.

Para no elegir esa orientación a ojo, se realizó una comprobación sencilla sobre el plano. En cada farola se probaron distintas orientaciones y, para cada una, se calculó qué plazas quedaban dentro de su campo de visión y se eligió la combinación que cubre todas las plazas con el menor número de cámaras, que fueron 8 en el *parking* del Santander y 6 en el *parking* del edificio E.

La comprobación de cobertura se hizo tomando el centro de cada plaza como punto de referencia. Para cada cámara candidata se verificó que ese punto quedara dentro del alcance de trabajo y dentro del campo horizontal de la cámara, es decir, a menos de 49,5° a cada lado de su eje de visión. Cuando una misma plaza podía verse desde varias cámaras, se asignó a la más cercana, ya que en esa vista el vehículo ocupa más píxeles y, por tanto, resulta más favorable para la detección. Al finalizar el barrido, todas las plazas del plano quedaron asignadas a alguna cámara, sin plazas sin cubrir sobre el total de 1.211.

El resultado es la propuesta de las Figuras 54 y 55, donde se puede visualizer que farolas están en uso y con que orientación y cuanto *parking* abarcarian de forma visual.

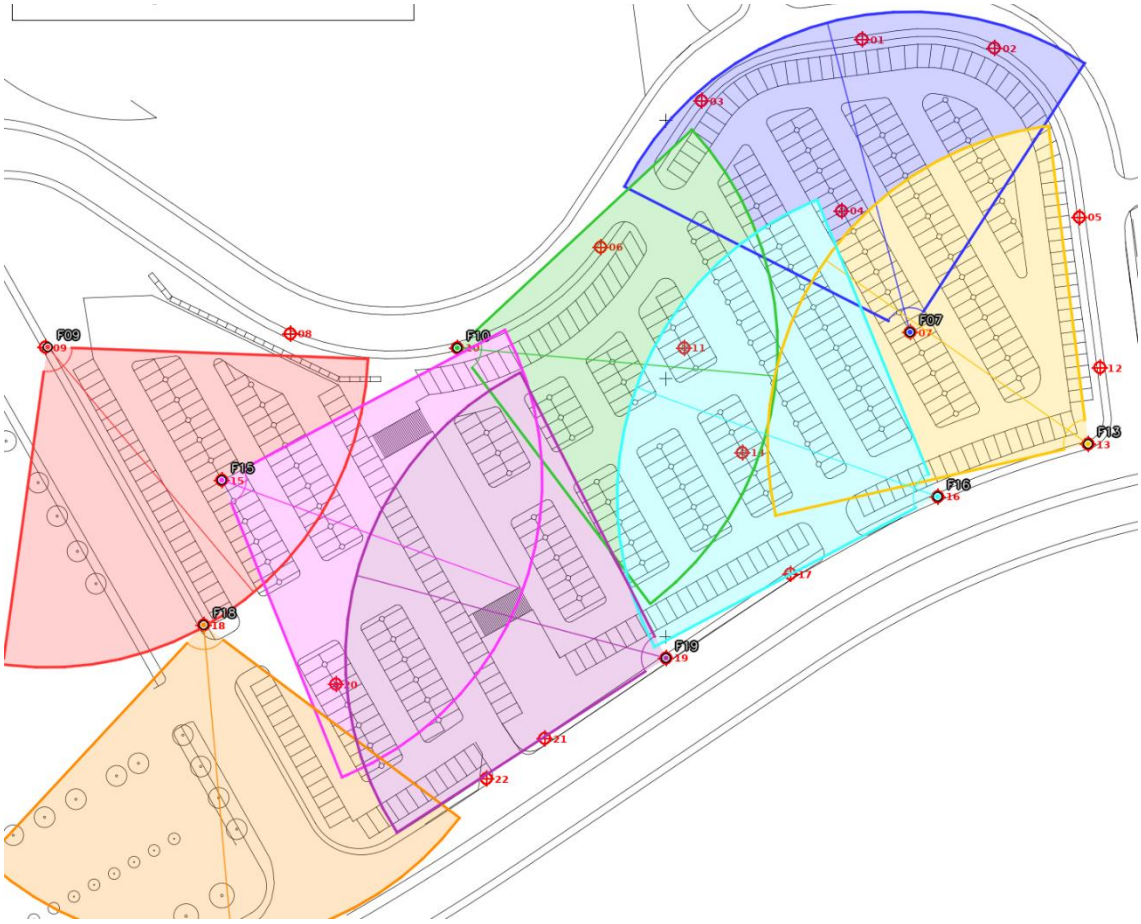


Figura 52. Propuesta base de despliegue sobre el plano del aparcamiento del Santander. Elaboración propia



Figura 53. Propuesta base de despliegue sobre el plano del aparcamiento edificio E. Elaboración propia

La segunda condición es el tamaño en píxeles con el que se ve el coche más lejano de cada cámara, porque es lo que de verdad está viendo la cámara.

Al conoccerse el ancho W ya calculado, el tamaño del coche en píxeles se obtiene por proporcionalidad como se explica en el apartado 2.2.4.:

$$p_x = \frac{a_v}{W} \cdot N_{px}$$

El cálculo se toma como una referencia conservadora, porque se usa la anchura del coche, aproximadamente 1,8 m. En la práctica, algunos vehículos se ven más de lado y pueden ocupar más píxeles por su longitud.

El criterio se basa en el rendimiento del YOLO11s afinado y no en la escala DORI ya mencionada, que se mantiene solo como referencia general de videovigilancia por estar pensada para un operador humano.

En las pruebas realizadas, el coche completo más pequeño correctamente detectado y asociado a una plaza ocupó unos 49 px en la imagen original de 2560 px, equivalentes a unos 37 px con la inferencia a 1920 px:

$$49 \cdot \frac{1920}{2560} \approx 37 \text{ px}$$

Valor que se toma como referencia conservadora del coche más lejano de cada cámara. No es el límite del detector, porque fuera de las plazas marcadas el modelo localizó vehículos de 31 a 36 px, pero esos casos no se usan como criterio al no estar asociados a plazas evaluadas.

Aplicando la relación con la lente de 2,8 mm y la inferencia a 1.920 px, el coche de 1,8 m mantiene los 37 px hasta unos 40 m:

$$W(40) = 2 \cdot 40 \cdot \tan\left(\frac{99^\circ}{2}\right) \approx 94 \text{ m}$$

$$p_x(40) = \frac{1{,}8}{94} \cdot 1920 \approx 37 \text{ px}$$

y baja a unos 22 px a 68 m, de modo que la huella llega de sobra al fondo del sector pero el detalle de las plazas más lejanas queda al límite. La lente angular se eligió precisamente para estudiar la cobertura máxima desde cada farola y reducir el número de cámaras, no para asegurar el máximo detalle en todas las plazas, así que esas filas lejanas son las zonas críticas que condicionan el diseño y que el piloto debe confirmar.

Como el diseño busca la máxima eficiencia y parte de pocas cámaras, esas zonas críticas se cubren con un plan escalonado de menor a mayor coste, pensado para aplicarse si el piloto lo necesita.

Primero se afina la orientación y la inclinación en la instalación. Si una fila lejana sigue corta, se procesa esa zona por recortes, analizando la subregión lejana sin reducirla en lugar de la imagen completa reescalada, lo que recupera resolución efectiva sobre el coche del fondo manteniendo la misma cámara y la misma inferencia a 1920 px. Otro possible ajuste sería una lente de 4 mm o varifocal en esa posición, que cierra el campo y agranda el coche a costa de cubrir menos ancho, y como último recurso se añadirían una o dos cámaras de apoyo, de manera que el sistema admite ampliarse hasta 16 cámaras sin rehacer el diseño. Subir la resolución de inferencia se contempla solo como ajuste puntual en cámaras críticas, mientras la configuración base se mantiene en 1920 px.

5.6 Procesamiento y hardware

El procesamiento se hace en el borde y no en un servidor central, siguiendo el análisis del apartado 2.3, porque reduce mucho el tráfico hacia la cabina

Con este enfoque, la primera decisión es no poner un equipo de procesamiento por cámara, sino compartir un nodo entre las cámaras de un sector. Una cámara que analiza una imagen cada dos segundos genera una carga muy baja, así que dedicarle un equipo entero capaz de ejecutar el modelo lo desaprovecha y multiplica el coste sin necesidad. El nodo recibe los flujos de su sector,

ejecuta el modelo y envía solo el estado de ocupación, de modo que un fallo afecta únicamente a su sector y el sistema puede crecer añadiendo sectores o cámaras simplemente cuando se pueda.

La cámara y el nodo no tienen por qué estar en el mismo sitio. La cámara va en la farola, pero colgar un nodo de cada farola obligaría a llevar corriente y obra civil hasta todas, así que el nodo se sitúa en un punto del sector con alimentación y red. La entrada del sector es un buens sitio porque igualmente allí se instala la barrera con su acometida eléctrica y su obra civil, y el nodo aprovecha esos mismos trabajos en lugar de generar una instalación aparte.

La conexión de las cámaras con el nodo y del nodo con la cabina central se resuelve con dos tramos de comunicación distintos, que se definen en el apartado 5.8.

Para el nodo, el marco teórico comparó la familia NVIDIA Jetson con una Raspberry Pi y un acelerador como Coral o Hailo. La Raspberry con acelerador es más barata, pero encaja mejor con cargas ligeras o pocas cámaras y solo ejecuta modelos cuantizados, lo que obligaría a modificar el detector ya entrenado y a revalidar la detección justo en los coches pequeños y lejanos. El procesamiento integrado en cada cámara queda como evolución futura porque habría que diseñar una placa específica para ello para que sea eficiente en coste. Como este sistema comparte un nodo entre varias cámaras y el modelo de visión está fijo, se selecciona la NVIDIA Jetson Orin *Nano* Super de 8 GB, que ejecuta el YOLO11s entrenado a 1.920 px sin reconvertirlo y tiene memoria para el modelo y varios flujos a la vez. NVIDIA la especifica con 67 TOPS en INT8, 8 GB de memoria y un consumo de 7 a 25 W (NVIDIA, s. f.). En coste queda por encima de la Raspberry con acelerador y muy por debajo de un ordenador con tarjeta gráfica dedicada, un punto intermedio razonable para su función y teniendo en cuenta la cantidad de cámaras que va a poder abarcar un nodo. La Figura 56 muestra la plataforma.



Figura 54. NVIDIA Jetson Orin Nano Super, plataforma de procesamiento seleccionada para los nodos. De Amazon.com

El número de cámaras por nodo depende de cómo se procesen. Analizándolas en paralelo manda la memoria, y con un consumo de referencia de 1,15 GB por cámara más 2 a 3 GB para el sistema y la base de datos local caben unas cuatro en los 8 GB. Analizándolas por turnos, que tiene sentido porque cada cámara solo envía una imagen cada dos segundos, el modelo se carga una sola vez y

el límite pasa a ser el tiempo de inferencia, con margen para atender más, de forma orientativa del orden de seis a ocho cámaras. Estas cifras se apoyan en medidas tomadas en el equipo de desarrollo y no en el Jetson, así que el diseño se fija en cuatro cámaras por nodo, que es el caso seguro por memoria, y deja el resto como margen a confirmar sobre la plataforma final ejecutando el nodo con sus cámaras durante al menos una hora y midiendo memoria, latencia, temperatura y fotogramas perdidos.

Como regla se coloca un nodo por sector, y se añadía un segundo donde el número de cámaras o la separación de las farolas lo pidan, repartiendo el sector en dos nodos más pequeños en lugar de forzar uno solo. El nodo se monta en una caja estanca con ventilación al estar a la intemperie.

Para evitar inferencias innecesarias se usa un modo de *wakeup*. Se planteó al principio como un componente aparte, pero la cámara seleccionada ya incorpora detección de movimiento, así que el nodo permanece en modo ligero manteniendo las comunicaciones y solo ejecuta la inferencia completa ante un evento de movimiento o en la comprobación periódica.

5.7 Comunicaciones, alimentación, señal y elementos del sistema

La integración del sistema se plantea mediante una red propia del aparcamiento, sin depender necesariamente de la red del campus. Como las cámaras se instalarían en farolas, cada punto de captación aprovecharía la alimentación disponible en ese punto para alimentar la cámara y el equipo de comunicación asociado. Esta alimentación no se aplicaría directamente a los equipos, sino que debería adaptarse mediante una fuente o convertidor dentro de una caja técnica, convirtiendo la tensión disponible en la farola a los niveles requeridos por la cámara IP y el equipo inalámbrico. En caso de emplear cámaras compatibles, podría utilizarse PoE local para simplificar el tramo corto de conexión de la cámara, pero no como enlace cableado hasta el nodo.

La transmisión del vídeo desde las cámaras hasta el nodo *edge* del sector se realizaría mediante una red inalámbrica local del sistema. Al tratarse de cámaras IP, el vídeo llegaría al nodo como vídeo comprimido, por ejemplo mediante RTSP sobre H.264 o H.265. Para varios puntos de captación dentro de un mismo sector, la solución más adecuada sería una topología punto a multipunto, con un equipo principal situado junto al nodo y equipos cliente en las farolas. De esta forma, varias cámaras pueden enviar su vídeo al mismo nodo sin necesidad de tender cableado por todo el aparcamiento.

Una vez recibido el vídeo, el nodo *edge* procesa localmente la información y obtiene los estados de ocupación, las incidencias y los avisos. A partir de ese punto ya no se transmite vídeo continuo hacia la gestión central, sino únicamente datos ligeros. Esta comunicación entre el nodo y la HMI puede realizarse mediante MQTT u otro protocolo similar, ya que solo se envían estados, eventos y señales de control. El vídeo quedaría reservado para consulta bajo demanda por parte del operario, no como flujo permanente.

La gestión central recibe la información de los nodos y permite supervisar el estado del aparcamiento desde la HMI. Desde este nivel se actualizan los elementos de señalización y control de acceso, como el panel exterior de plazas libres, el semáforo de acceso, el controlador de barrera

y la barrera de entrada. En caso de que algún nodo no tenga conexión directa estable con la central, se podría recurrir a repetidores, puntos intermedios o a una conexión disponible en el entorno, algo que debería validarse durante una prueba piloto.

Con esta estructura quedan separados los dos tipos de comunicación del sistema, el tramo de mayor carga entre cámaras y nodo, y el tramo ligero de gestión entre el nodo y la central.

La Figura 57 resume la integración entre las cámaras, el nodo, los radioenlaces, la señal exterior y el controlador de la barrera.

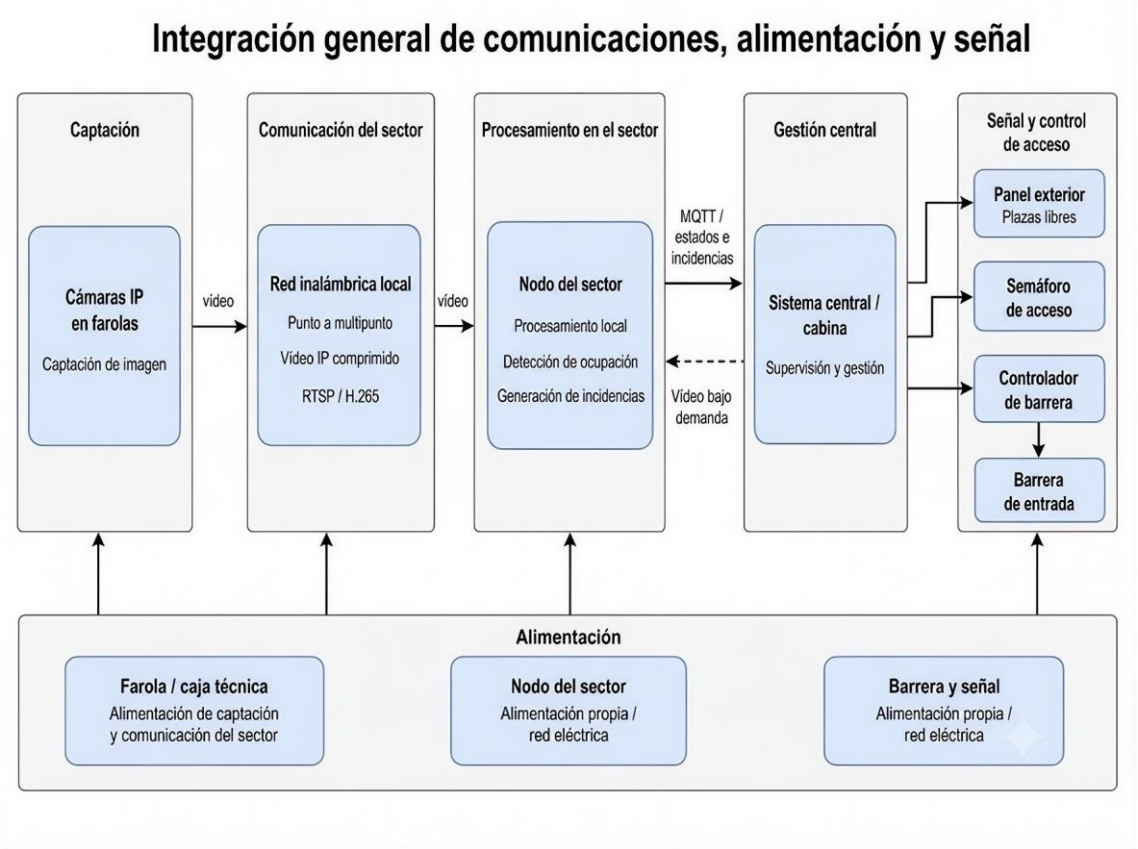


Figura 55. Resumen integración de arquitectura entre camaras, nodos y enlaces de comunicación. Elaboración propia

El acceso lo controla una barrera de brazo con su propio controlador. No se abre con cada coche, sino que actúa como restricción cuando un sector pasa a completo o vuelve a tener plazas libres, así que queda abierta la mayor parte del tiempo. El cambio lo decide el nodo del sector, que es el que conoce la ocupación, y lo comunica al controlador con una orden por contacto seco, sin accionar el motor directamente, de forma que las fotocélulas y las seguridades siguen dependiendo del controlador de la barrera. Si la ocupación no es válida o se pierde la comunicación, se pasa a control manual que ya esta incorporado. Como referencia se toma la FAAC B614 (FAAC, s.f.), una barrera electromecánica de 24 V para exterior con señalización roja y verde en el propio cuerpo, lo que evita poner semáforos aparte demás, su acabado rojo resulta coherente con la estética corporativa de la UEM, aunque la elección se justifica

principalmente por su adecuación técnica al control de acceso del aparcamiento. Además, se pueden instalar un panel informativo en cada entrada del aparcamiento para mostrar el número de plazas libres.

Se propone una barrera por entrada, teniendo en cuenta que tres de los cuatro sectores tienen un único acceso y el cuarto entra por ambos lados con dos barreras, serían cinco, pero sumando las 3 necesarias para los sectores que están cubiertos por las cámaras quedan 8 en total. La Figura 58 muestra la barrera tomada como referencia, siendo el último componente que quedaba para completar el sistema.



Figura 56. FAAC B614, barrera automática de exterior seleccionada. De FAAC

En conjunto, el despliegue queda en 4 sectores en los que se utilizan 4 notos, pero demás para las cámaras reutilizadas se añade un nodo adicional, haciendo 5 en total. En total quedan 16 cámaras, 2 de ellas siendo las reutilizadas en la universidad.

5.8 Tratamiento de datos y ciberseguridad

El marco normativo, con el RGPD, la LOPDGDD, los criterios de la AEPD y los principios de minimización y conservación, se expuso en el marco teórico, así que aquí solo se concreta cómo el diseño los pone en práctica. La decisión de procesar en el borde y transmitir solo el estado es, en sí misma, la aplicación del principio de minimización, porque las imágenes no salen del punto de captura y lo que circula no incluye matrícula, rostro ni biometría. Sobre esa base, el sistema usa solo las cámaras necesarias, limita el campo de visión al interior del aparcamiento, no incorpora reconocimiento facial ni lectura de matrículas, no realiza una grabación analítica continua y aplica máscaras de privacidad en las zonas que no deban captarse. Si las cámaras de seguridad reutilizadas almacenan imágenes, esa grabación sigue la política de la UEM, con la referencia de la supresión transcurrido un mes salvo los supuestos para autoridades o tribunales (AEPD, 2025), una decisión que se revisa con el delegado de protección de datos.

En ciberseguridad, las cámaras y los nodos forman parte de una red y se protegen cambiando las credenciales por defecto, separando las cámaras en una red virtual de área local (VLAN), cifrando las comunicaciones con TLS, gestionando el acceso a la interfaz por roles y registrando los accesos y los cambios de configuración. El uso del sistema queda acotado a la ocupación y las incidencias del aparcamiento, y no debe ampliarse a la identificación de personas o matrículas sin una evaluación independiente.

5.9 Arquitectura final y resumen de decisiones

En la Figura 59 se recoge de forma conjunta la arquitectura completa del sistema propuesto, integrando los bloques que se han ido desarrollando por separado en los apartados anteriores de captación, comunicaciones, procesamiento en el borde, gestión central y actuación sobre el acceso. La figura no plantea una arquitectura nueva, sino que reúne en un único esquema la parte física del despliegue y la lógica interna del nodo de procesamiento, para mostrar cómo quedaría el sistema completo aplicado a un sector tipo del aparcamiento. De esta forma se puede ver de forma resumida la solución final propuesta.

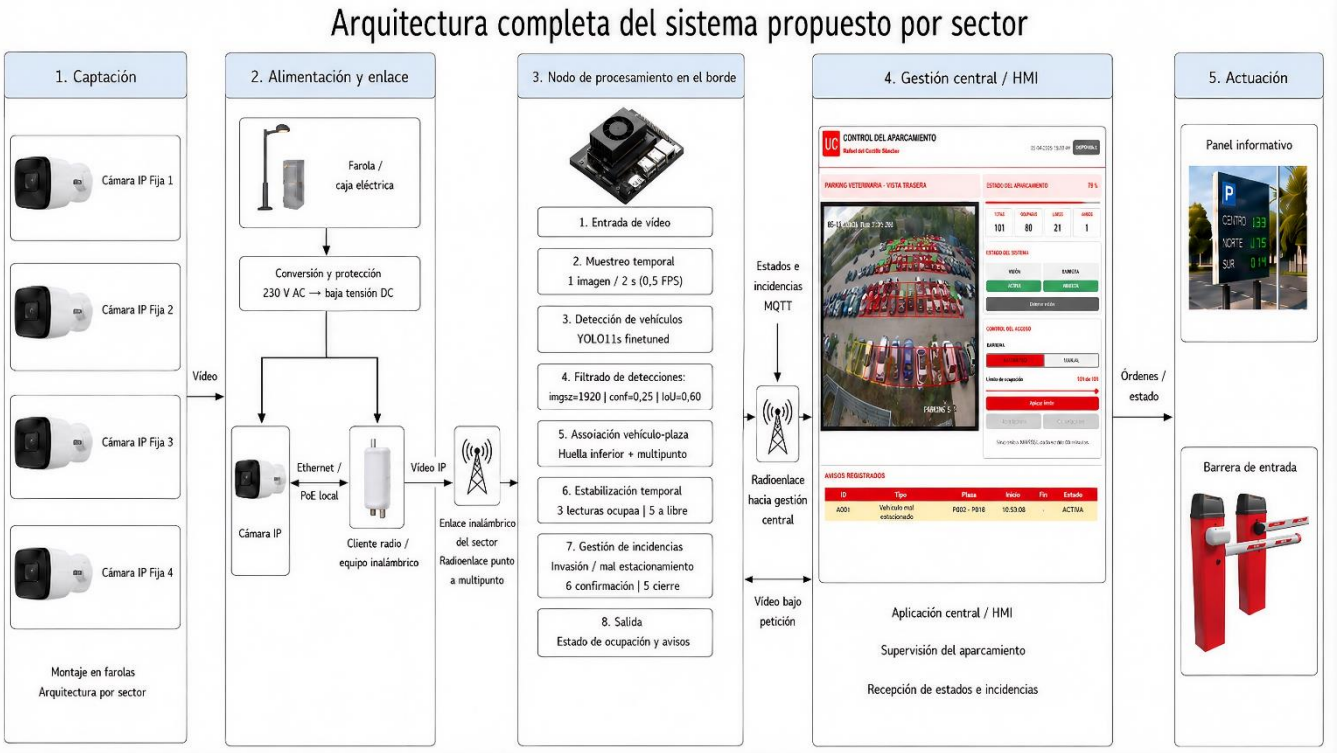


Figura 57. Arquitectura completa del sistema propuesto. Elaboración propia

5.10 Implantación y pruebas de campo antes del despliegue

Definida la arquitectura, antes de darla por válida el sistema pasaría por una prueba piloto en el propio aparcamiento, primero sobre un sector y después escalando al resto, para comprobar que el diseño funciona fuera del entorno de desarrollo.

La prueba empezaría por una inspección del emplazamiento, comprobando sobre el terreno la ubicación de las cámaras, las alturas, la alimentación, la conectividad y los obstáculos como árboles, sombras u oclusiones. A partir de ahí se ajustaría el encuadre de cada cámara y se generaría su archivo de plazas sobre la imagen real, verificando que las plazas más lejanas mantienen el tamaño en píxeles que el detector necesita.

El objetivo no es repetir la demostración del prototipo, sino comprobar lo que puede fallar en una implantación real. Se validarían al menos:

- Las condiciones de visión adversas, con distintas iluminaciones, contraluz y reflejos, lluvia y niebla, y oclusiones por árboles o vehículos altos.
- La estabilidad de la ocupación cuando los vehículos entran, salen o maniobran.
- La pérdida y recuperación de una cámara y la caída o degradación de la comunicación entre cámara, nodo y central.
- La carga del nodo con sus cámaras a la vez y la latencia entre el cambio de ocupación y su reflejo en la HMI, el panel o la barrera.

Como criterio de aceptación se propone que cada cámara supere el 95 % de exactitud en las plazas válidas de su sector, sin falsos libres persistentes y con transiciones estables, entendido como objetivo de ingeniería para la puesta en marcha y no como un resultado ya obtenido. También se comprobaría que el nodo funciona de forma continuada y que, ante una pérdida de visión o comunicación, el sistema se degrada de forma segura, mostrando un estado no fiable en lugar de dar información incorrecta. En esta fase se afinarían además los parámetros fijados en el prototipo para una sola vista, como el intervalo de análisis, los umbrales de asociación y los tiempos de estabilización, que pueden variar según la altura, el ángulo y la distancia de cada sector.

Quedarían fuera de esta fase los elementos que no son imprescindibles para validar el funcionamiento básico, como el almacenamiento histórico permanente, la detección automática de plazas o una aplicación para los usuarios, que se plantean como ampliaciones futuras. Con esta prueba se cierra el diseño técnico, de modo que el despliegue no es una instalación directa, sino una evolución progresiva, primero la validación sobre un sector, después el ajuste de cámaras y parámetros y por último la ampliación al resto del aparcamiento.

Una vez definida la propuesta técnica, se trasladaron también algunas cuestiones finales al responsable de seguridad y al departamento de operaciones de la UEM, con el objetivo de contrastar la posible continuidad del proyecto y su aplicación real en el campus. En esta revisión, la propuesta se consideró viable e interesante como línea de trabajo posterior, y se planteó la posibilidad de seguir desarrollándola después de la entrega del TFG. También se comentaron nuevas funcionalidades que podrían incorporarse en futuras fases, como la gestión específica de plazas para vehículos eléctricos, además de las incidencias ya contempladas en el sistema. No obstante, debido al calendario de entrega y al periodo vacacional, varias cuestiones técnicas y operativas quedaron pendientes de una reunión posterior

Capítulo 6. Estudios económicos y de viabilidad

Una vez definida la arquitectura, en este capítulo se calcula cuánto costaría montar el sistema y se mira si sale rentable en un aparcamiento real. No es un presupuesto cerrado de obra porque para eso haría falta ir a verlo, pedir precios a instaladores y confirmar cómo se alimenta, dónde se sujeta y por dónde va el cableado. Es una estimación razonada, con precios públicos, referencias de otras soluciones y las partidas de instalación e ingeniería que serían necesarias en el caso de la UEM.

La estimación parte de la arquitectura definida en los capítulos anteriores y persigue dos objetivos. El primero es poner un coste razonado a la implantación completa en el caso de la UEM. El segundo es comparar cada bloque del sistema con referencias de mercado de alcance equivalente.

6.1 Alcance económico y criterio de cálculo

El punto de partida del cálculo es el plano CAD del aparcamiento, sobre el que se han contabilizado 1.211 plazas. Esas plazas quedan cubiertas por el despliegue principal de 14 cámaras nuevas y 4 nodos de procesamiento, y son las que se usan para calcular el coste por plaza, porque constituyen el único alcance medido de forma verificable. En adelante, este conjunto se denomina sector de estudio.

El diseño contempla además la reutilización de 2 cámaras existentes de la universidad. Estas amplían la cobertura hacia zonas adicionales, pero las plazas que abarcan no han podido contarse sobre el plano. Por eso su adaptación se suma al coste de la implantación ampliada y no se usa para rebajar artificialmente el coste por plaza.

Con este criterio se distinguen tres lecturas económicas:

- Coste de percepción del sector de estudio, que agrupa las 14 cámaras nuevas, los 4 nodos, las comunicaciones, el software y la ingeniería e instalación asociadas a las 1.211 plazas.
- Coste del sistema completo del sector de estudio, que añade a la capa anterior las barreras automáticas y los paneles informativos.
- Coste de implantación ampliada de la UEM, que suma al sistema completo la adaptación de las 2 cámaras reutilizadas y su quinto nodo.

Para evitar una comparación distorsionada, el presupuesto completo del caso UEM se mantiene separado de la comparativa de mercado. El presupuesto completo incluye captación, procesamiento, comunicaciones, software, ingeniería, instalación, barreras y paneles. La comparativa, en cambio, se organiza por niveles equivalentes. El software se compara con software, el hardware de percepción con sensores o sistemas APMS, y la cámara con inteligencia artificial con cámaras o sensores elevados que integran analítica.

Todos los importes se expresan en euros netos. Cuando una fuente publica precios en dólares estadounidenses, se convierten a euros con el tipo de cambio de referencia del Banco Central Europeo, correspondiéndose a 1,1448 USD por euro, equivalente a 0,8735 EUR por dólar (Banco Central Europeo, 2026).

Para la comparativa del mercado, se ha llevado a cabo una búsqueda de fuentes que contienen información pública disponible, tanto en catálogos, páginas web y contratos públicos como en licitaciones. En gran medida, la mayoría de los precios para soluciones completas de proyecto e implementación, de sus soluciones o productos estaban bajo petición de presupuesto. Aunque la información disponible para realizar la comparativa de mercado es limitada, no se pidió presupuesto para no hacer perder tiempo a las empresas, además de que conlleva un coste al tener que realizar un estudio previo de la zona de estudio.

En la comparativa de mercado, en los apartados a continuación no se mezclan partidas que los proveedores no publican o no incluyen, de modo que cada bloque del sistema se enfrenta únicamente a alternativas de su mismo alcance.

6.2 Referencias de mercado y fuentes utilizadas

Las referencias económicas se han seleccionado priorizando fuentes públicas y verificables, como organismos oficiales, fabricantes, distribuidores conocidos y documentos de contratación. Cuando un proveedor no publica un precio, se utiliza solo como referencia técnica o de modelo de negocio, sin forzar una comparación numérica directa. Las entradas completas de todas estas fuentes figuran en la bibliografía final.

Referencia	Dato utilizado	Fuente / criterio
Tipo de cambio USD/EUR	1 USD = 0,8735 EUR	Banco Central Europeo.
Sensores ultrasónicos plaza a plaza	262,05-436,76 EUR/plaza.	U.S. DOT ITS Knowl*edge* Resources: 300-500 USD/plaza, convertido a EUR.
Sistemas APMS	218,38-698,81 EUR/plaza.	U.S. DOT ITS Knowl*edge* Resources: 250-800 USD/plaza, convertido a EUR.
Sistema de guiado Plaza Mayor Ciudad Real	47.773,15 EUR para 190 plazas.	Documento municipal: sistema de guiado completo para aparcamiento.
Parkinto	60,27 EUR/cámara/mes.	Precio público: 69 USD/cámara/mes, convertido a EUR.
Parquery	Precio por plaza bajo oferta.	Proveedor de software de visión sobre cámaras existentes. No publica tarifa cerrada.

Sharvy	2,00-4,20 EUR/plaza/mes.	Precio público de software de gestión, sin IVA.
Toogethr	Desde 525,00-1.125,00 EUR/mes.	Paquetes públicos de software de gestión. Para grandes aparcamientos puede requerir oferta personalizada.
Cleverciti	2.364,68 EUR/sensor. 2,97 EUR/plaza/mes. 29,70 EUR/plaza de configuración.	Contrato público Sourcewell/NCPA convertido desde USD.
Milesight AI *Parking* Camera	773,38 EUR/cámara.	Precio público de distribuidor para cámara IA de gestión de aparcamiento.
TP-Link Omada EAP215-Bridge KIT	170,00 EUR/kit.	Valor conservador basado en fuente oficial Omada y comparadores europeos.

Tabla 26. Referencias económicas utilizadas para el análisis comparativo. Elaboración propia

6.3 Modelo económico de la solución propuesta

El coste total de implantación se organiza en bloques para poder analizar cada parte por separado. La expresión general utilizada es:

$$C_{total} = C_{captación} + C_{procesamiento} + C_{comunicaciones} + C_{software} + C_{servicios} + C_{actuación}$$

La captación corresponde a las cámaras y a sus elementos locales de alimentación. El procesamiento agrupa los nodos *edge*. Las comunicaciones cubren los enlaces cámara-nodo y nodo-sistema central. El software incluye la visión, la integración digital y la interfaz. Los servicios recogen la ingeniería y la instalación no software, y la actuación engloba las barreras y los paneles informativos.

6.3.1 Coste de captación

La captación se basa en cámaras IP de exterior instaladas sobre farolas o soportes elevados. Para el caso de estudio se toma como referencia una cámara tipo *bullet* Hikvision de 4 MP (Hikvision, s.f.), con protección exterior y resolución suficiente para la arquitectura propuesta. Cada punto incorpora además soporte, pequeño material, conversión de alimentación y un temporizador de carril DIN.

El temporizador se incluye porque la cámara necesita alimentación permanente para funcionar durante el día, mientras que la luminaria debe seguir encendiéndose solo durante el horario nocturno. Por tanto, esta partida representa la adaptación local para separar la alimentación continua del sistema de visión del encendido horario de la luz.

Concepto	Cantidad	Precio unitario	Importe	Fuente / criterio
Cámara IP *bullet* Hikvision 4 MP	14	100,00 EUR	1.400,00 EUR	Precio redondeado a partir de distribuidores europeos con valores próximos a 98-104 EUR.
Soporte, herrajes y pequeño material	14	30,00 EUR	420,00 EUR	Accesorios de montaje para CCTV exterior sobre columna/farola.
Temporizador analógico carril DIN	14	90,00 EUR	1.260,00 EUR	Referencia RS PRO o equivalente, entorno 89,98 EUR netos.
Convertidor local AC-DC 230 V a 12 V	14	10,00 EUR	140,00 EUR	Pequeño material eléctrico de distribución para CCTV.
Total captación del sector de estudio			**3.220,00 EUR**	14 puntos de captación nuevos.

Tabla 27. Desglose del coste de captación para el sector CAD. Elaboración propia

6.3.2 Coste de procesamiento *edge*

El procesamiento se plantea en el borde mediante nodos distribuidos. Cada nodo ejecuta el modelo de visión artificial y agrupa las cámaras de un sector. Esta decisión reduce el envío de vídeo continuo, evita una dependencia fuerte de servicios *cloud* y limita la transmisión a metadatos de ocupación, incidencias y estado del nodo. Además de reducir costes recurrentes de la nube, esta arquitectura disminuye la exposición de imágenes y, por tanto, reduce el riesgo asociado al cumplimiento normativo. El precio del kit de desarrollo se toma de un distribuidor europeo oficial (RS Components, s.f.).

Concepto	Cantidad	Precio unitario	Importe	Fuente / criterio
NVIDIA Jetson Orin *Nano* Super Developer Kit	4	289,31 EUR	1.157,24 EUR	Referencia RS Components, plataforma *edge AI* de hasta 67 TOPS.
Envolvente exterior IP65 con carril DIN y fuente industrial	4	80,00 EUR	320,00 EUR	Rango de catálogo para cajas estancas y fuentes industriales.
Integración interna: microSD industrial,	4	90,90 EUR	363,60 EUR	Kit técnico para integración y protección del nodo en exterior.

ventilación, fusibles y prensaestopas				
Total procesamiento *edge* del sector de estudio			**1.840,84 EUR**	4 nodos principales para las 1.211 plazas contabilizadas.

Tabla 28. Desglose del coste de procesamiento edge. Elaboración propia

6.3.3 Coste de comunicaciones

La partida de comunicaciones incluye de forma explícita la parte emisora y la receptora de la arquitectura. Además del enlace entre nodo y cabina, hay que cubrir la comunicación entre cada cámara y el nodo de su sector. Por ello se presupuestan equipos inalámbricos cliente en las cámaras, equipos base en los nodos y enlaces adicionales hacia el punto de agregación central.

Se toma como referencia el TP-Link Omada EAP215-Bridge KIT o equivalente, un kit exterior de 5 GHz para enlaces PtP/PtMP (Omada Networks, s.f.). El precio utilizado es 170,00 EUR por kit, un valor conservador dentro del rango observado en comparadores europeos (Idealo, s.f.) y coherente con el precio de mercado de este tipo de equipo.

Concepto	Cantidad	Precio unitario	Importe	Fuente / criterio
Kits inalámbricos cámara-nodo	9 kits	170,00 EUR	1.530,00 EUR	18 unidades radio: 14 clientes en cámara y 4 bases en nodo.
Kits inalámbricos nodo-cabina/HMI central	4 kits	170,00 EUR	680,00 EUR	Enlaces de agregación desde los 4 nodos principales.
Switch industrial PoE de nodo	4	100,00 EUR	400,00 EUR	Switch reforzado para gabinete de nodo.
Inyectores PoE/adaptación de alimentación local	14	15,00 EUR	210,00 EUR	Alimentación local de equipos que lo requieran.
Cableado FTP exterior, conectores, herrajes y protecciones	1 partida	800,00 EUR	800,00 EUR	Cableado corto exterior, conectores apantallados y fijaciones.
Total comunicaciones del sector de estudio			**3.620,00 EUR**	Incluye emisor/receptor cámara-nodo y nodo-cabina.

Tabla 29. Desglose corregido del coste de comunicaciones. Elaboración propia.

6.3.4 Coste de un módulo mínimo de percepción

Además del presupuesto global, resulta útil calcular el coste de un módulo mínimo de despliegue. En este trabajo se considera como módulo base el conjunto formado por 4 cámaras, 1 nodo de procesamiento y sus comunicaciones. Esta configuración representa una unidad funcional razonable para aparcamientos grandes, ya que permite agrupar varias cámaras por sector sin sobredimensionar el número de nodos.

Bloque	Cantidad	Precio unitario	Importe
Captación: 4 cámaras con soporte, temporizador y conversión AC-DC	4 puntos	230,00 EUR/punto	920,00 EUR
Procesamiento: 1 nodo *edge* completo	1 nodo	460,21 EUR/nodo	460,21 EUR
Comunicaciones: cámara-nodo, nodo-cabina, *switch*, PoE y cableado proporcional	1 módulo	884,44 EUR/módulo	884,44 EUR
Total módulo mínimo	4 cámaras + 1 nodo		**2.264,65 EUR**
Coste equivalente por cámara del módulo			**566,16 EUR/cámara**

Tabla 30. Coste estimado de un módulo mínimo de percepción. Elaboración propia.

6.3.5 Coste de software, licenciamiento e integración digital

La parte de software se separa explícitamente para poder compararla con proveedores que ofrecen soluciones software sobre cámaras existentes. En esta partida se incluye el desarrollo del pipeline de visión, la lógica de ocupación, la integración MQTT/API/HMI, la calibración de ROIs, la validación funcional y los componentes software necesarios para explotar el sistema.

La valoración utiliza una tarifa base de 25,00 EUR/h, un valor prudente y habitual para perfiles junior o técnicos cualificados en España (TarifaAutonomo, 2026a, 2026b). Queda por debajo de una tarifa comercial senior con margen empresarial completo, pero resulta un coste hora razonable para valorar el trabajo de desarrollo e integración dentro de un TFG.

Para el modelo comercial más coherente con esta arquitectura, se plantea un pago único de desarrollo o licencia local, acompañado de un mantenimiento anual opcional. Esta opción es más adecuada que una suscripción SaaS pura, porque el sistema se ejecuta en *edge* y no requiere pagar infraestructura *cloud* por procesar vídeo. Como referencia conservadora, se considera un mantenimiento anual del 15 % del coste inicial del software, destinado a actualizaciones, correcciones y soporte.

Concepto software	Cálculo	Importe	Criterio
Configuración de firmware, red, nodos y comunicaciones seguras	25 h x 25,00 EUR/h	625,00 EUR	Configuración técnica y securización básica.
Desarrollo del pipeline de visión, integración MQTT/HMI/APIs	40 h x 25,00 EUR/h	1.000,00 EUR	Software de percepción e integración digital.
Calibración fina de ROIs, ajuste de falsos positivos y validación funcional	55 h x 25,00 EUR/h	1.375,00 EUR	Adaptación del software al escenario real.
Pruebas de aceptación, QA del algoritmo y control de latencia	60 h x 25,00 EUR/h	1.500,00 EUR	Validación técnica del comportamiento del sistema.
Licencias, broker MQTT, base de datos y pasarelas API	Partida global	1.998,88 EUR	Componentes software de integración y explotación.
Interfaz de explotación y control lógico de barreras/paneles	Partida global	1.500,00 EUR	HMI y lógica de mando. No incluye equipos físicos.
Total software e integración digital		**7.998,88 EUR**	Base de coste inicial para comparar con soluciones software de mercado.
Mantenimiento anual opcional (15 %)		**1.199,83 EUR/año**	Actualizaciones, correcciones y soporte. No incluye nuevas funcionalidades mayores.

Tabla 31. Desglose del coste de software e integración digital. Elaboración propia

6.3.6 Servicios de ingeniería, instalación y puesta en marcha

Además del software, la implantación requiere tareas de ingeniería de campo, montaje físico, seguridad y coordinación. Estas partidas forman parte del presupuesto completo de la UEM, pero no se utilizan en la comparativa con proveedores que únicamente publican precios de software o de hardware aislado.

Servicio no software	Cálculo	Importe	Criterio
Diseño final de cobertura, simulación geométrica y sectorización	20 h x 25,00 EUR/h	500,00 EUR	Ingeniería previa de despliegue.
Redacción de manuales, transferencia de conocimiento y formación	15 h x 25,00 EUR/h	375,00 EUR	Documentación y formación básica.
Mano de obra de instalación en campo	8 jornadas, 2 técnicos	2.400,00 EUR	Montaje de herrajes, cámaras, cableado y gabinetes.
Conexionado eléctrico, puenteado y pruebas in situ	8 uds. x 220,00 EUR	1.760,00 EUR	Adaptación eléctrica por puntos de agregación.
Alquiler de PEMP para trabajos en altura	8 días	1.200,00 EUR	Medio auxiliar para instalación en farolas.
PRL, balizamiento y seguridad en viales	Partida global	1.000,00 EUR	Seguridad en trabajos de campo.
Dirección de obra y PMO	80 h x 25,00 EUR/h	2.000,00 EUR	Coordinación técnica y control de despliegue.
Reserva técnica y canalizaciones reforzadas	Partida global	740,00 EUR	Provisión de contingencia de campo.
Total servicios		**9.975,00 EUR**	Servicios de implantación y coordinación.

Tabla 32. Desglose de servicios de ingeniería, instalación y puesta en marcha. Elaboración propia

6.3.7 Coste de actuación e información al usuario

La capa de actuación no forma parte estricta de la percepción, pero sí de la explotación completa del aparcamiento. Incluye barreras automáticas para controlar el acceso cuando un sector se encuentre completo y paneles LED exteriores para informar al conductor antes de entrar en una zona saturada.

Concepto	Cantidad	Precio unitario	Importe	Fuente / criterio
Barrera automática FAAC B614 o equivalente	5	2.500,00 EUR	12.500,00 EUR	Precio conservador para kit de barrera y accesorios.
Panel LED exterior informativo	4	1.500,00 EUR	6.000,00 EUR	Rango estimado para panel exterior de guiado/plazas disponibles.
Total actuación e información			**18.500,00 EUR**	No se usa en comparativas con proveedores que no incluyen actuación física.

Tabla 33. Desglose del coste de actuación e información. Elaboración propia

6.4 Presupuesto del sector CAD de 1.211 plazas

Aplicando el modelo anterior al sector contabilizado en el CAD, el coste de la capa de percepción se obtiene sumando captación, procesamiento, comunicaciones, software e integración digital, y servicios no software de implantación. Esta capa representa el coste necesario para medir y explotar digitalmente la ocupación de las 1.211 plazas, sin incluir todavía barreras ni paneles físicos.

Bloque	Importe
Captación	3.220,00 EUR
Procesamiento *edge*	1.840,84 EUR
Comunicaciones corregidas	3.620,00 EUR
Software e integración digital	7.998,88 EUR
Servicios de ingeniería e instalación no software	9.975,00 EUR
Total capa de percepción del sector de estudio	**26.654,72 EUR**

Tabla 34. Presupuesto de la capa de percepción para las 1.211 plazas contabilizadas. Elaboración propia

El coste unitario de la capa de percepción queda por tanto:

$$\frac{26.654{,}72\ EUR}{1.211\ plazas} = 22{,}01\ EUR/Plaza$$

Si se añade la capa de actuación formada por barreras y paneles, se obtiene el coste del sistema completo del sector de estudio:

Bloque	Importe
Capa de percepción del sector de estudio	26.654,72 EUR
Barreras automáticas y paneles informativos	18.500,00 EUR
Total sistema completo del sector de estudio	**45.154,72 EUR**

Tabla 35. Presupuesto del sistema completo para las 1.211 plazas contabilizadas. Elaboración propia

El coste unitario del sistema completo para el sector de estudio queda:

$$\frac{45.154{,}72 EUR}{1.211 plazas} = 37{,}29\ EUR/Plaza$$

Este valor se utiliza para valorar el caso UEM completo, pero no para compararlo con proveedores que solo publican precios de software o de hardware aislado.

6.5 Ampliación mediante cámaras reutilizadas

El diseño final contempla la reutilización de 2 cámaras existentes de la UEM. Estas cámaras no se adquieren de nuevo, por lo que suponen un ahorro respecto a instalar dos puntos de captación adicionales. Sin embargo, sí requieren adaptación de comunicaciones, alimentación local si fuera necesaria, configuración y conexión con un quinto nodo de procesamiento.

Como las plazas cubiertas por estas cámaras no han podido contabilizarse de forma verificable sobre el plano del CAD, este coste se suma al total de implantación ampliada, pero no se utiliza para recalcular el coste por plaza de las 1.211 plazas. También, se añaden las barreras y paneles que corresponderían a estas zonas de aparcamiento.

Concepto adicional	Cálculo	Importe
Nodo *edge* adicional	1 x 460,21 EUR	460,21 EUR
Kits inalámbricos cámara-nodo para cámaras reutilizadas	2 kits x 170,00 EUR	340,00 EUR
Kit inalámbrico nodo-cabina para quinto nodo	1 kit x 170,00 EUR	170,00 EUR
Switch industrial PoE adicional	1 x 100,00 EUR	100,00 EUR
Inyectores/adaptación PoE	2 x 15,00 EUR	30,00 EUR
Cableado, herrajes y configuración adicional	Partida global	150,00 EUR
Total módulo adaptación cámaras reutilizadas		**1.250,21 EUR**
Barreras automáticas (3)	3 X 2.500,00 EUR	7.500 EUR
Paneles informativos (2)	2X 1.500,00 EUR	3.000 EUR
Total zona con cámaras reutilizadas		**11.750,21 EUR**

Tabla 36. Coste de ampliación mediante cámaras reutilizadas. Elaboración propia

Con esta ampliación, el coste total de implantación ampliada de la UEM asciende a 56.904,93 EUR. Esta cifra representa el coste real de dejar integrada la arquitectura ampliada con 16 cámaras y 5 nodos, aunque el coste por plaza se mantiene calculado sobre las 1.211 plazas verificadas del plano CAD.

Aplicando un coste de contingencias del 5% (de 2.845,25 EUR) para cubrir cualquier imprevisto sobre el coste total de implementación en el caso de UEM, el coste asciende a **59.750,18 EUR**.

COSTE TOTAL CASO EUM **59.750,18 EUR**

6.6 Comparativa económica por alcance equivalente

La comparación económica se hace por niveles equivalentes, sin enfrentar el coste completo de implantación a productos que solo publican precios de software o de una cámara aislada. Se organizan tres bloques, uno de software, otro de hardware de percepción y un tercero de cámara con inteligencia artificial.

En las soluciones comerciales donde no se publica el precio se mantiene la referencia técnica, pero no se calcula un coste total. Es el caso de proveedores como Parquery o Urbiotica, que trabajan bajo oferta personalizada porque su precio depende del número de plazas, del estado de la infraestructura y del nivel de servicio contratado (Parquery, s.f.-b). Esta opacidad es la práctica

habitual en las soluciones a medida del sector, y explica que la comparativa se apoye únicamente en fuentes con precios públicos verificables.

6.6.1 Comparativa de software y modelo de licenciamiento

En este bloque se compara la parte software del TFG frente a soluciones que publican precios de software o de servicio digital. No se incluyen cámaras, nodos, comunicaciones, instalación, barreras ni paneles.

Para que la comparación tenga sentido frente a modelos de suscripción, el software del TFG se presenta como licencia local o pago único, con mantenimiento anual opcional. Esta opción es coherente con el diseño *edge.* No hay que pagar una nube para procesar vídeo, el sistema no depende de una suscripción por cámara y la información sensible se mantiene localmente. El coste a cinco años se calcula como el coste inicial más cinco anualidades de mantenimiento.

Solución	Alcance comparable	Coste en EUR	Observación
Software TFG	Pipeline de visión, ocupación, MQTT/API/HMI, calibración, QA e interfaz lógica.	7.998,88 EUR pago único + 1.199,83 EUR/año mantenimiento opcional. 13.998,03 EUR a 5 años.	Modelo local *edge.* Sin cuota *cloud* por procesar vídeo.
Parkinto	SaaS de detección por cámara.	60,27 EUR/cámara/mes. 10.125,79 EUR/año para 14 cámaras. 50.628,93 EUR en 5 años.	No incluye cámaras ni instalación.
Cleverciti Space Management	Software/servicio recurrente por plaza sobre sensores Cleverciti.	2,97 EUR/plaza/mes. 43.150,57 EUR/año para 1.211 plazas. configuración inicial 29,70 EUR/plaza.	Referencia de contrato público Sourcewell/NCPA. No es una oferta española.
Parquery	Software de visión sobre cámaras existentes.	Precio no público. Tarifa por plaza bajo oferta.	Referencia técnica relevante, pero sin coste cerrado verificable.
Sharvy	Software de gestión de plazas.	2,00-4,20 EUR/plaza/mes. 29.064,00-61.034,40 EUR/año para 1.211 plazas.	No es percepción visual directa. Referencia de software de gestión.

Toogethr	Software de gestión de *parking*.	Desde 525,00-1.125,00 EUR/mes. 6.300,00-13.500,00 EUR/año.	Para 1.211 plazas probablemente requeriría tarifa personalizada.

Tabla 37. Comparativa de software con alcance equivalente. Elaboración propia

La comparación muestra que el software del TFG tiene un coste inicial bajo frente a servicios de suscripción por cámara o por plaza (Parkinto, s.f.; Sharvy, s.f.; Sourcewell, s.f.; Toogethr, s.f.). Su ventaja principal es que no depende de una infraestructura *cloud* de pago para ejecutar la percepción, porque el procesamiento se realiza en *edge*. Esto reduce el coste recurrente y también disminuye el riesgo normativo al limitar la salida de imágenes del entorno de captación. A cambio, en una implantación real sí sería recomendable contratar un mantenimiento anual para actualizaciones, corrección de errores y soporte técnico.

6.6.2 Comparativa de hardware de percepción

En este segundo bloque se compara el hardware de percepción del TFG, formado por cámaras, nodos y comunicaciones, frente a sistemas de sensorización o sistemas APMS que se expresan en coste por plaza. No se incluyen software, instalación detallada, ingeniería, barreras ni paneles, para no mezclar alcances diferentes.

Solución	Alcance comparable	Coste en EUR aplicado a 1.211 plazas	Observación
Hardware percepción TFG	14 puntos de captación, 4 nodos *edge* y comunicaciones cámara-nodo/nodo-cabina	8.680,84 EUR (7,16 EUR/plaza)	Economía de escala. Una cámara cubre muchas plazas.
Sensores plaza a plaza	Sensor instalado por plaza	262,05-436,76 EUR/plaza. Total 317.348,01-528.913,35 EUR	Referencia USDOT para sensores ultrasónicos, convertida a EUR.
Sistemas APMS	Sistema avanzado de gestión de aparcamiento por plaza	218,38-698,81 EUR/plaza. Total 264.456,67-846.261,36 EUR	Rango amplio. Puede incluir partidas distintas según caso.
Sistema de guiado Plaza Mayor Ciudad Real	Sistema de guiado completo en aparcamiento con 190 plazas	47.773,15 EUR. 251,44 EUR/plaza	Referencia pública española. Aparcamiento interior, no exterior.

Tabla 38. Comparativa de hardware de percepción con sensores y APMS. Elaboración propia.

El resultado refleja la principal ventaja económica de la visión compartida. El hardware de percepción del TFG queda en torno a 7,16 EUR por plaza, mientras que las referencias de

sensorización individual o de APMS se mueven en cientos de euros por plaza (Ayuntamiento de Ciudad Real, s.f.; U.S. Department of Transportation, 2008, 2020). La comparación debe leerse como orden de magnitud, porque cada referencia puede incluir o excluir partidas distintas.

6.6.3 Comparativa de cámara con inteligencia artificial

En este tercer bloque se compara la solución TFG como unidad funcional de cámara y nodo *edge* frente a cámaras comerciales con analítica integrada. Para evitar duplicidades, no se vuelve a incluir el coste del software desarrollado, ya que este se ha comparado por separado en el apartado 6.6.1. Por tanto, el coste comparable se limita a la captación, la parte proporcional del nodo de procesamiento y las comunicaciones de percepción, sin incluir instalación, actuación física ni software.

Solución	Alcance comparable	Coste en EUR	Observación
TFG cámara + nodo *edge* distribuido	Captación + parte proporcional de nodo/comunicaciones.	620,06 EUR/cámara. 8.680,84 EUR para 14 cámaras. 7,16 EUR/plaza.	No incluye software, instalación ni actuación física, el software se compara aparte en el apartado 6.6.1.
Milesight AI *Parking* Camera	Cámara específica con IA integrada para *parking*.	773,38 EUR/cámara. 10.827,32 EUR para 14 cámaras.	Fabricante declara hasta 100 plazas/cámara. No incluye integración completa, barreras ni paneles.

Tabla 39. Comparativa de cámara con inteligencia artificial. Elaboración propia.

Solo se ha comparado con soluciones de las que existen datos publicados en abierto. Por eso no se han podido usar referencias como Cleverciti Sensor, Urbiotica U-Spot VISIO o el sistema de detección de plazas de la plaza Severo Ochoa en Cartagena, que no publican precios cerrados.

En esta comparación, la solución del TFG se plantea como una arquitectura abierta de percepción distribuida más que como una cámara cerrada. Las cámaras comerciales con IA integrada pueden ser competitivas como producto unitario (IoT-Shop, s.f.; Milesight, s.f.), pero la propuesta separa el hardware de captación y procesamiento del software propio, lo que permite comparar el coste físico de la arquitectura sin duplicar el desarrollo ya valorado en el bloque anterior.

6.7 Impacto ambiental y contribución a los ODS

Más allá de la viabilidad económica, el sistema tiene un impacto ambiental directo, porque su función es eliminar un tráfico que no responde a ningún desplazamiento necesario. La literatura permite dimensionar ese tipo de impacto, donde Jones et al. (2017) estimaron que el cruising genera en Bristol más de 790 toneladas de CO_2 al año y 368.000 £ de combustible desperdiciado, y Shoup (2011) calculó que la búsqueda de aparcamiento en quince manzanas del Upper West

Side de Manhattan producía 590.000 vehículos-kilómetro y 295 toneladas de CO_2 anuales. Para traducir combustible en emisiones se utiliza el factor de referencia de la Agencia de Protección Ambiental estadounidense (EPA), según el cual la combustión de un litro de gasolina libera aproximadamente 2,35 kg de CO_2.

Al no disponer de datos de tráfico del campus, la estimación se plantea como un orden de magnitud a partir de supuestos preliminares y declarados, pendientes de contrastar con mediciones reales. Si en hora punta buscaran plaza en torno a 1.000 vehículos al día, con una media de 3 minutos de búsqueda y 200 días lectivos al año, se acumularían unas 10.000 horas anuales de circulación dedicadas solo a aparcar. Con un consumo en marcha lenta del orden de 1 litro por hora, esto supone aproximadamente 10.000 litros de combustible, unas 23,5 toneladas de CO_2 y un coste cercano a los 15.000 € anuales.

Sobre esa base se puede estimar el ahorro que aportaría el sistema. Las evaluaciones de sistemas ya desplegados sitúan la reducción del tiempo de búsqueda entre el 43 % medido en el programa SFpark de San Francisco (Krishnamurthy y Ngo, 2020) y el 65 % del experimento controlado de Dalla Chiara et al. (2022), donde la información de disponibilidad redujo la espera media de 3,1 a 1,1 minutos; en el contexto español, Rodríguez et al. (2024) estiman además una reducción del 37 % en emisiones de partículas. Aplicando ese rango del 40 al 65 % a la estimación anterior, el sistema podría evitar entre 4.000 y 6.500 horas de circulación, entre 9 y 15 toneladas de CO_2 y entre 6.000 y 9.750 € de combustible cada año. Son cifras condicionadas a los supuestos de partida, pero el propio sistema permite validarlas, porque una vez desplegado registra la ocupación y los tiempos de forma continua, lo que se plantea como parte de la evaluación del piloto.

Este impacto conecta con los compromisos institucionales de la universidad. La Universidad Europea fue la primera universidad privada de España en obtener la certificación ISO 14001 de gestión medioambiental, cuenta con iniciativas como el Sistema Integral de Gestión Medioambiental (SIGMA) y su política de campus saludable, y su labor se evalúa en el ranking THE Impact Rankings, que mide la aportación de las universidades a los Objetivos de Desarrollo Sostenible de la Agenda 2030. Reducir el tráfico de búsqueda contribuye de forma directa al ODS 11 (ciudades y comunidades sostenibles), por la mejora de la movilidad y la reducción de la congestión en el campus, y al ODS 13 (acción por el clima), por la reducción de emisiones estimada. La solución se alinea además con el marco de la Ley 7/2021 de cambio climático, orientado a reducir el tráfico contaminante en entornos urbanos, de modo que un despliegue como el propuesto no solo mejora la gestión del espacio, sino que aporta un resultado medible a los objetivos de sostenibilidad de la universidad.

6.8 Consideraciones sobre mantenimiento y operación

El presupuesto calculado corresponde al CAPEX inicial. En una implantación real habría que prever limpieza de lentes, revisión de cajas estancas, comprobación de enlaces inalámbricos, actualización de software, sustitución de componentes, recalibración si se desplaza alguna cámara y soporte técnico.

La arquitectura *edge* reduce los costes recurrentes asociados a la nube, ya que el vídeo no tiene que enviarse de forma continua a un servidor externo para ser procesado. En lugar de eso, se transmiten metadatos de ocupación e incidencias. Esto reduce ancho de banda, dependencia de conectividad y exposición de datos personales. No elimina las obligaciones normativas, pero sí reduce el riesgo frente a una arquitectura basada en vídeo continuo en la nube.

Además, el mantenimiento físico es menor que en una solución plaza a plaza. En lugar de más de mil sensores distribuidos por el pavimento, el sistema concentra la captación en 14 cámaras nuevas, 2 cámaras reutilizadas y 5 nodos en la implantación ampliada. Esto simplifica las intervenciones y permite sectorizar los fallos.

6.9 Conclusión de viabilidad

El análisis económico muestra que la solución propuesta es viable y competitiva cuando se compara con alternativas de alcance equivalente. Para el sector de estudio de 1.211 plazas, la capa de percepción completa tiene un coste estimado de 26.654,72 EUR, equivalente a 22,01 EUR por plaza. Si se añaden barreras y paneles, el sistema completo asciende a 45.154,72 EUR, equivalente a 37,29 EUR por plaza teniendo en cuenta las necesidades de todo el *parking* (sin incluir las que necesitarían las cámaras extra a reutilizar).

La implantación ampliada, incluyendo la adaptación de las 2 cámaras reutilizadas y el quinto nodo (junto a las barreras y los paneles correspondientes), alcanza 56.904,93 EUR. Y, aplicando el coste de contingencias, se alcanzan los 59.750,18 EUR. Este valor representa el coste real de la arquitectura completa prevista para la UEM, pero no se utiliza para reducir el coste por plaza porque las plazas adicionales cubiertas por esas cámaras no se han podido contabilizar en el plano del CAD.

La comparación por niveles permite extraer varias conclusiones. El software propio resulta competitivo frente a servicios de suscripción, especialmente si se considera un horizonte de varios años y la ausencia de costes de nube. El hardware de percepción basado en cámaras y nodos presenta una ventaja muy clara frente a sensores individuales, porque una sola cámara cubre decenas de plazas. Las cámaras comerciales con IA integrada son una alternativa relevante, pero no ofrecen necesariamente el mismo grado de control, adaptación e integración que una arquitectura propia.

En conjunto, el sistema propuesto combina bajo coste por plaza, escalabilidad, reutilización parcial de infraestructura existente y procesamiento local en el borde. Aunque una implantación real requeriría una prueba piloto prolongada para validar comunicaciones, estabilidad y mantenimiento, el análisis económico permite concluir que la propuesta es técnicamente razonable y económicamente competitiva para un aparcamiento exterior de gran superficie como el de la UEM.

Capítulo 7. Conclusiones

Tras haber desarrollado todo el modelo de visión, comparado varias configuraciones y modelos, integrando todo en una aplicación software, seleccionando los componentes para funcionar con ella y haciendo una propuesta de despliegue con su arquitectura completa, para finalmente estudiar la viabilidad económica, en este capítulo se desarrollan las conclusiones de todo el trabajo.

Se abarcan las conclusiones, las posibles líneas futuras, implementaciones o mejoras que puede tener a partir de aquí el trabajo y por último, menciona brevemente que conocimientos del grado se han aplicado.

7.1 Conclusiones generales

La detección de ocupación en aparcamientos exteriores no se resuelve con una sola decisión, se ha podido ver que depende perspectiva, la distancia a las plazas del fondo, las oclusiones y la posición de la cámara pesan tanto como el modelo, así que el sistema salió de evaluar por separado el detector, la asociación vehículo-plaza y la estabilización, y de corregir cada parte a partir de sus fallos.

Un resultado muy interesante el del detector, que se entrenó con un conjunto de imágenes limitado y sin ninguna del aparcamiento de la UEM, y aun así respondió bien sobre las cámaras del campus, alcanzando un F1 en torno al 0,99, con la sensibilidad subiendo del 0,73 al 0,99 tras el ajuste al evaluarlo sobre imágenes reales del campus que el modelo no había visto y con las plazas desde ángulos distintos .

Aunque son pruebas acotadas, validar sobre un escenario con el que no se entrenó dice bastante, porque podría indicar que la estrategia podría trasladarse a otros aparcamientos, y deja ver que con un *dataset* mayor todavía hay margen de mejora

Ese detector se eligió por criterio de ingeniería, no por métrica. Un modelo pequeño dio resultados equiparables a otros mayores, así que se optó por él para reducir cómputo y consumo, pensando en un nodo autónomo capaz de agrupar varias cámaras. En un sistema encendido que puede estar encendido largos periodos de tiempo, esa eficiencia importa tanto como el acierto.

Detectar vehículos no es conocer la ocupación. La asociación entre la caja del coche y la plaza es un problema propio, y resolverlo con la huella inferior y el solape fue lo que dio una lectura estable, de lo más reutilizable en otras escenas, pudiéndose ver que funcionó en ambas pruebas piloto.

El procesamiento en el borde encaja con el campus, porque evita enviar vídeo continuo y transmite solo estados e incidencias, lo que reduce ancho de banda, dependencia de red y exposición de imágenes. El estudio económico resulta viable y aunque apenas hay precios públicos de soluciones a medida, frente a las alternativas con tarifa abierta el coste queda muy

por debajo, en 22,01 € por plaza la capa de percepción y 37,29 € el sistema completo, frente a los cientos de euros por plaza de la sonorización individual.

La propuesta se contrastó de forma preliminar con responsables de la UEM, que la vieron viable e interesante para continuar, aunque la definición de una prueba piloto real quedó pendiente de una reunión posterior.

Este trabajo se ha desarrollado a la vez que cursaba las asignaturas del último año y hacía las prácticas, y a nivel personal ha sido un reto grande. He tenido que aprender a manejar herramientas que no conocía y a resolver cosas que no salían a la primera. Pero he aprendido mucho, sobre inteligencia artificial y entrenamiento de modelos, y sobre un tema que parecía menor y resultó tener bastante detrás. El mundo del aparcamiento me parece ahora un problema del futuro que merecería más atención de la que recibe. Y si con tan pocos recursos se ha podido montar algo tan funcional y a un coste que no es alto para lo que aporta, cuesta entender que estos sistemas no estén ya en muchos más sitios. Ha sido duro, pero he disfrutado el proceso y me llevo un aprendizaje real de este campo.

7.2 Líneas futuras

El trabajo se puede continuar en dos horizontes distintos. Por un lado, los pasos que dan continuidad directa a lo ya hecho y lo acercarían a un uso real. Por otro, las ideas de más alcance que habría desarrollado con más tiempo y recursos, y que marcan hacia dónde puede crecer el proyecto.

7.2.1 Continuación directa del proyecto

Con el sistema actual, el paso inmediato es la prueba piloto. Se trata de llevarlo del entorno de vídeo en el que se ha validado a una instalación real sobre el aparcamiento, con la cámara, el nodo, la red y la interfaz funcionando en campo. A partir de ahí habría que dejarlo trabajando durante más tiempo y en las condiciones que aquí no se pudieron cubrir del todo, como la lluvia, la noche, las sombras fuertes o la suciedad en la lente. Es lo que convierte una propuesta probada sobre grabaciones en un sistema probado en funcionamiento.

En paralelo, el modelo puede seguir mejorando. Se trataría de entrenar con más imágenes, probar los modelos que vayan saliendo y hacerlo más robusto ampliando el conjunto de datos con condiciones variadas de las que aquí no se disponía. Probarlo en otras zonas del campus y en aparcamientos distintos confirmaría además hasta dónde generaliza cuando cambian la geometría, la altura de la cámara o el tipo de plaza, algo que este trabajo apunta pero no cierra, porque solo se probó sobre la UEM.

Todo esto se plantea avanzando junto a los responsables del aparcamiento, viendo sobre el terreno nuevas necesidades y desarrollándolas con ellos, como se ha hecho hasta ahora.

7.2.2 Evolución de mayor alcance

Más allá de esos pasos, hay ideas que no dieron tiempo a desarrollar pero que resultan especialmente interesantes.

La primera es que el sistema detecte las plazas por sí solo. Ahora se definen a mano, y si el sistema las reconociera automáticamente, instalarlo en un sitio nuevo dejaría de exigir ese trabajo manual, ahorrando ingeniería y tareas repetitivas. Eso abre además la puerta a usar cámaras con zoom o giratorias para cubrir más superficie con menos equipos, algo que hoy no es viable porque al mover el encuadre las plazas dejarían de coincidir. Sobre esa base se podrían ampliar también las incidencias, como la gestión de las plazas de recarga de vehículos eléctricos que surgió hablando con la UEM.

Otra línea es el guiado al conductor. Cuando el sistema sabe con precisión qué plaza está libre, el paso lógico es hacerle llegar esa información, por ejemplo con una aplicación móvil que muestre en tiempo real las plazas libres del aparcamiento, para que sepa dónde aparcar antes de dar vueltas. Es lo que más cambia la experiencia de quien busca sitio, que es al final el problema que motiva todo el trabajo.

La idea más ambiciosa, y a la que en realidad se quería llegar, es diseñar una electrónica propia para el nodo. En lugar de depender de placas genéricas, se trataría de crear una placa a medida que integre el procesador y todo lo que la cámara necesita, pensada para competir con las cámaras de IA integrada del mercado pero con una diferencia, concebir el nodo como una unidad autónoma que se alimente con panel solar y batería y se monte sobre un poste sin depender de la red eléctrica. Así podría desplegarse en zonas sin corriente y convertirse en un producto escalable, instalable sin obra. No dio tiempo a abordarlo, pero queda como la evolución natural del sistema.

Por último, la integración con los elementos que la aplicación hoy solo simula, como la barrera, los paneles de plazas libres y la plataforma del campus, junto a un histórico de datos con estadísticas por horas y zonas y un autodiagnóstico que vigile el propio sistema y avise si una cámara se ha movido, una lente está sucia o un nodo ha dejado de responder. Es lo que llevaría la propuesta de prototipo a un sistema explotable y mantenible en el tiempo.

7.3 Competencias y conocimientos del grado aplicados

A lo largo del trabajo me di cuenta de que estaba juntando piezas de asignaturas muy distintas del grado, y merece la pena reconocer de dónde sale cada una. Hay nociones de casi todas, pero unas pocas han sostenido el peso.

La que más ha pesado es la visión, que viene directamente de Sistemas de Percepción en Robótica. Ahí aprendí cómo una máquina interpreta una imagen, y de ahí sale buena parte del criterio que he aplicado. Muy cerca está Automatización Industrial, porque es la que me hizo mirar el aparcamiento no como un problema de detectar coches sino como un proceso que se puede automatizar de principio a fin, desde que el sistema percibe una plaza hasta que actúa sobre una barrera o un panel. Toda la parte de programar, montar los scripts y manejar los datos se apoya en Fundamentos de Informática para la Ingeniería y en Informática Industrial. Para pensar el

sistema sobre un equipo pequeño y de bajo consumo me sirvieron Electrónica Digital y Microprocesadores, y para el consumo y la parte eléctrica del despliegue, Electrotecnia.

Para medir y comparar resultados está Estadística para la Ingeniería, detrás de todas las métricas y de los esquemas y figuras. Ya en lo menos técnico, el análisis de costes y la viabilidad vienen de Organización de Empresas y Sistemas de Producción Industrial, y el enfoque del presupuesto y de la protección de datos, de Proyectos y Legislación, que además me dio la forma de llevar un proyecto de este tamaño.

Pero por encima de cualquier asignatura, lo que más me llevo no salió de una clase concreta. Son cuatro años haciendo proyectos, presentándolos y defendiéndolos, y esa experiencia es la que de verdad me ha enseñado a enfrentarme a algo así. Coger un problema grande y abierto, partirlo en pasos, avanzar probando, ver qué falla y decidir el siguiente con criterio. También a buscar información por mi cuenta y a leer documentación técnica con sentido crítico para quedarme solo con lo que servía, sobre todo cuando las cosas no salían a la primera y tocaba replantear. Al final es eso, más que ninguna asignatura suelta, lo que me ha hecho sacar el trabajo adelante.

BIBLIOGRAFÍA

Agencia Española de Protección de Datos [AEPD]. (s.f.). *Principios*. https://www.aepd.es/derechos-y-deberes/cumple-tus-deberes/principios

Agencia Española de Protección de Datos [AEPD]. (2025). *Guía sobre el uso de videocámaras para seguridad y otros fines*. https://www.aepd.es

Almeida, P. R. L., Oliveira, L. S., Britto, A. S., Silva, E. J. y Koerich, A. L. (2015). PKLot: A robust dataset for parking lot classification. *Expert Systems with Applications*, *42*(11), 4937–4949. https://doi.org/10.1016/j.eswa.2015.02.009

Amato, G., Carrara, F., Falchi, F., Gennaro, C., Meghini, C. y Vairo, C. (2017). Deep learning for decentralized parking lot occupancy detection. *Expert Systems with Applications*, *72*, 327–334. https://doi.org/10.1016/j.eswa.2016.10.055

Asociación Española de Fabricantes de Automóviles y Camiones [ANFAC]. (2025). *Informe del parque de vehículos en España 2024* [nota de prensa]. https://anfac.com/wp-content/uploads/2025/02/NP-ANFAC-Informe-parque-de-vehiculos-Espana-2024.pdf

Axis Communications. (s.f.). *Network cameras*. https://www.axis.com/products/network-cameras

Ayuntamiento de Ciudad Real. (s.f.). *Formalización del contrato del sistema de guiado a plazas libres del aparcamiento de la Plaza Mayor*. https://www.ciudadreal.es/documentos/transparencia/concesiones/48_formalizacion_contrato_parking_Plaza_Mayor.pdf

Ayuntamiento de Sevilla. (2019, 6 de febrero). *Nuevo proyecto piloto de parking inteligente en el Parque Científico y Tecnológico para facilitar el aparcamiento y la movilidad sostenible*. https://www.sevilla.org/actualidad/noticias/2019/nuevo-proyecto-piloto-de-parking-inteligente-en-el-parque-cientifico-y-tecnologico-para-facilitar-el-aparcamiento-y-la-movilidad-sostenible

Banco Central Europeo. (2026, 3 de julio). *Euro foreign exchange reference rates: US dollar*. https://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/eurofxref-graph-usd.en.html

Bradski, G. (2000). The OpenCV library. *Dr. Dobb's Journal of Software Tools*, *25*(11), 120–125.

Centro Demostrador TIC de Extremadura. (s.f.). *Sistema de parking inteligente y gestión del tráfico*. https://cdticextremadura.es/oficina-smartcity/proyectos/sistema-de-parking-inteligente-y-gestion-del-trafico/3

Chen, L.-C., Sheu, R.-K., Peng, W.-Y., Wu, J.-H. y Tseng, C.-H. (2020). Video-based parking occupancy detection for smart control system. *Applied Sciences*, *10*(3), 1079. https://doi.org/10.3390/app10031079

Cisco. (s.f.). *What is Power over Ethernet (PoE)?* https://www.cisco.com/site/us/en/learn/topics/networking/what-is-power-over-ethernet.html

Cleverciti. (s.f.). *Cleverciti Sensor*. https://www.cleverciti.com/en/innovations/cleverciti-sensor

Comisión Europea. (2019). *El Pacto Verde Europeo* (COM(2019) 640 final). https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX:52019DC0640

Dalla Chiara, G., Krutein, K. F., Ranjbari, A. y Goodchild, A. (2022). Providing curb availability information to delivery drivers reduces cruising for parking. *Scientific Reports*, *12*, 19355. https://doi.org/10.1038/s41598-022-23987-z

Encío, L., Díaz, C., del-Blanco, C. R., Jaureguizar, F. y García, N. (2023). Visual parking occupancy detection using extended contextual image information via a multi-branch output ConvNeXt network. *Sensors*, *23*(6), 3329. https://doi.org/10.3390/s23063329

Environmental Protection Agency [EPA]. (2023). *Tailpipe greenhouse gas emissions from a typical passenger vehicle* (EPA-420-F-23-014). https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle

ESMARTCITY. (2026, 12 de junio). *Urbiotica amplía su plataforma de smart parking con nuevas funcionalidades basadas en LPR*. https://www.esmartcity.es/2026/06/12/urbiotica-amplia-plataforma-smart-parking-nuevas-funcionalidades-basadas-lpr

European Data Protection Board [EDPB]. (2019). *Guidelines 3/2019 on processing of personal data through video devices*. https://www.edpb.europa.eu

Eurostat. (2024). *Stock of vehicles at regional level*. Statistics Explained. https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Stock_of_vehicles_at_regional_level

FAAC. (s.f.). *B614: Barrera automática*. https://www.faac.es

Gillies, S., van der Wel, C. y Van den Bossche, J. (s.f.). *Shapely: Manipulation and analysis of geometric objects* [software]. https://github.com/shapely/shapely

Hampshire, R. C. y Shoup, D. (2018). What share of traffic is cruising for parking? *Journal of Transport Economics and Policy*, *52*(3), 184–201.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hikvision. (s.f.). *DS-2CD1043G2 series network camera* [ficha técnica]. https://www.hikvision.com/en/products/IP-Products/Network-Cameras/value-series/ds-2cd1043g2-liu-f-/

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

IBM. (2024). *What is deep learning?* https://www.ibm.com/think/topics/deep-learning

Idealo. (s.f.). *TP-Link EAP215-Bridge KIT: Comparación de precios*. https://www.idealo.de/preisvergleich/OffersOfProduct/204066800_-eap215-bridge-kit-tp-link.html

INRIX. (2017). *The impact of parking pain in the US, UK and Germany* (G. Cookson y B. Pishue, autores). INRIX Research. https://inrix.com/press-releases/cod-us/

International Electrotechnical Commission [IEC]. (2002). *IEC 62262:2002. Degrees of protection provided by enclosures for electrical equipment against external mechanical impacts (IK code)*.

International Electrotechnical Commission [IEC]. (2013). *IEC 60529:1989+AMD1:1999+AMD2:2013. Degrees of protection provided by enclosures (IP Code)*.

International Electrotechnical Commission [IEC]. (2015). *EN 62676-4:2015. Video surveillance systems for use in security applications. Part 4: Application guidelines*.

IoT-Shop. (s.f.). *Milesight PMC8266-FPE AI Outdoor Parking Management Pro Bullet Plus Camera*. https://iot-shop.de/en/shop/mil-pmc8266-fpe-milesight-pmc8266-fpc-ai-outdoor-parking-management-pro-*bullet*-plus-kamera-7490

Jabbar, W. A., Wei, C. W., Azmi, N. A. A. M. y Haironnazli, N. A. (2021). An IoT Raspberry Pi-based parking management system for smart campus. *Internet of Things*, *14*, 100387. https://doi.org/10.1016/j.iot.2021.100387

Jocher, G. y Qiu, J. (2024). *Ultralytics YOLO11* [software]. https://github.com/ultralytics/ultralytics

Jones, M., Khan, A., Kulkarni, P., Carnelli, P. y Sooriyabandara, M. (2017). ParkUs 2.0: Automated cruise detection for parking availability inference. En *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2017)* (pp. 242–251). ACM. https://doi.org/10.1145/3144457.3144495

Kim, K., Lee, J., Jeong, I., Jung, J. y Cho, J. (2025). Real-time parking space management system based on a low-power embedded platform. *Sensors*, *25*(22), 7009. https://doi.org/10.3390/s25227009

Krishnamurthy, C. K. B. y Ngo, N. S. (2020). The effects of smart-parking on transit and traffic: Evidence from SFpark. *Journal of Environmental Economics and Management*, *99*, 102273. https://doi.org/10.1016/j.jeem.2019.102273

Ley 7/2021, de 20 de mayo, de cambio climático y transición energética. *Boletín Oficial del Estado*, 121, de 21 de mayo de 2021. https://www.boe.es/eli/es/l/2021/05/20/7

Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales. *Boletín Oficial del Estado*, 294, de 6 de diciembre de 2018. https://www.boe.es/eli/es/lo/2018/12/05/3

Liu, B., Lai, H., Kan, S. y Chan, C. (2023). Camera-based smart parking system using perspective transformation. *Smart Cities*, *6*(2), 1167–1184. https://doi.org/10.3390/smartcities6020056

Marek, M. (2021). *Image-based parking space occupancy classification: Dataset and baseline*. arXiv. https://arxiv.org/abs/2107.12207

McKinney, W. (2010). Data structures for statistical computing in Python. En *Proceedings of the 9th Python in Science Conference* (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

Milesight. (s.f.). *AI Outdoor Parking Management Pro Bullet Plus Camera*. https://www.milesight.com/security/product/ai-parking-management-pro-*bullet*-plus-camera

Minott, D., Siddiqui, S. y Haddad, R. J. (2025). Benchmarking edge AI platforms: Performance analysis of NVIDIA Jetson and Raspberry Pi 5 with Coral TPU. En *SoutheastCon 2025* (pp. 1384–1389). IEEE. https://doi.org/10.1109/SOUTHEASTCON56624.2025.10971592

Mordor Intelligence. (2026). *Europe smart parking market: Analysis, share and size (2026–2031)*. https://www.mordorintelligence.com/industry-reports/europe-smart-parking-market

MQTT.org. (s.f.). *MQTT: The standard for IoT messaging*. https://mqtt.org

Nguyen, T. T. y Sartipi, M. (2024). Smart camera parking system with auto parking spot detection. En *Proceedings of the Asian Conference on Computer Vision (ACCV) Workshops* (pp. 232–241). https://doi.org/10.1007/978-981-96-2641-0_16

NVIDIA. (s.f.). *Jetson Orin Nano Super Developer Kit*. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-orin/

Omada Networks. (s.f.). *EAP215-Bridge KIT*. https://www.omadanetworks.com/nl/business-networking/omada-wifi-wireless-bridge/eap215-bridge-kit/

Paidi, V., Fleyeh, H., Håkansson, J. y Nyberg, R. G. (2018). Smart parking sensors, technologies and applications for open parking lots: A review. *IET Intelligent Transport Systems*, *12*(8), 735–741. https://doi.org/10.1049/iet-its.2017.0406

Paidi, V., Håkansson, J., Fleyeh, H. y Nyberg, R. G. (2022). CO2 emissions induced by vehicles cruising for empty parking spaces in an open parking lot. *Sustainability*, *14*(7), 3742. https://doi.org/10.3390/su14073742

ParkHelp. (2025, 3 de noviembre). *ParkHelp acquires Cleverciti, creating the world's most advanced platform for parking guidance and curb management* [nota de prensa]. https://www.parkhelp.com/parkhelp-acquires-cleverciti-creating-the-worlds-most-advanced-platform-for-parking-guidance-and-curb-management/

Parkinto. (s.f.). *Pricing*. https://parkinto.com/pricing/

Parquery. (s.f.-a). *Upgrade any camera with artificial intelligence to resolve mobility questions*. https://parquery.com

Parquery. (s.f.-b). *Pricing*. https://parquery.com/pricing/

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*, 8026–8037.

PATH (California Partners for Advanced Transportation Technology). (2008). *Smart parking field test for the California Department of Transportation and the Bay Area Rapid Transit* [datos de coste recogidos en la base ITS Deployment Evaluation del U.S. Department of Transportation]. https://www.itskrs.its.dot.gov

Pham, T. N., Tsai, M.-F., Nguyen, D. B., Dow, C.-R. y Deng, D.-J. (2015). A *cloud*-based smart-parking system based on Internet-of-Things technologies. *IEEE Access*, *3*, 1581–1591. https://doi.org/10.1109/ACCESS.2015.2477299

Redmon, J., Divvala, S., Girshick, R. y Farhadi, A. (2016). You only look once: Unified, real-time object detection. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779–788). https://doi.org/10.1109/CVPR.2016.91

Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos. *Diario Oficial de la Unión Europea*, L 119, de 4 de mayo de 2016. https://eur-lex.europa.eu/eli/reg/2016/679/oj

Ren, S., He, K., Girshick, R. y Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, *28*, 91–99.

Roboflow. (s.f.). *Roboflow* [plataforma de gestión y anotación de datasets]. https://roboflow.com

Rodríguez, A., Alonso, B., Moura, J. L. y dell'Olio, L. (2024). Analysis of user behavior in urban parking under different level of information scenarios provided by smart devices or connected cars. *Travel Behaviour and Society*, *37*, 100847. https://doi.org/10.1016/j.tbs.2024.100847

Rodríguez, A., Cordera, R., Alonso, B., dell'Olio, L. y Benavente, J. (2022). Microsimulation parking choice and search model to assess dynamic pricing scenarios. *Transportation Research Part A: Policy and Practice*, *156*, 253–269. https://doi.org/10.1016/j.tra.2021.12.013

RS Components. (s.f.). *NVIDIA Jetson Orin Nano Super Developer Kit* (ref. 264-7384). https://nl.rs-online.com/web/p/processor-development-tools/2647384

Saki, S. y Hagen, T. (2024). Cruising for parking again: Measuring the ground truth and using survival analysis to reveal the determinants of the duration. *Transportation Research Part A: Policy and Practice*, *183*, 104045. https://doi.org/10.1016/j.tra.2024.104045

Schimansky, T. (s.f.). *CustomTkinter* [software]. https://github.com/TomSchimansky/CustomTkinter

Sharvy. (s.f.). *Pricing*. https://www.sharvy.com/en/pricing/

Shi, W., Cao, J., Zhang, Q., Li, Y. y Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

Shoup, D. C. (2006). Cruising for parking. *Transport Policy*, *13*(6), 479–486. https://doi.org/10.1016/j.tranpol.2006.05.005

Shoup, D. C. (2011). *The high cost of free parking* (ed. actualizada). Planners Press.

Sourcewell. (s.f.). *Cooperative purchasing contract 120423-CLVR: Cleverciti*. https://www.sourcewell-mn.gov/cooperative-purchasing/120423-CLVR

Sui, X., Ye, X., Wang, T., Yan, X., Chen, J. y Ran, B. (2022). Microscopic simulating the impact of cruising for parking on traffic efficiency and emission with parking-and-visit test data. *International Journal of Environmental Research and Public Health*, *19*(15), 9127. https://doi.org/10.3390/ijerph19159127

TarifaAutonomo. (2026a). *Tarifa por hora de un ingeniero freelance en España*. https://tarifaautonomo.com/blog/tarifa-hora-ingeniero-espana

TarifaAutonomo. (2026b). *Tarifa por hora de un técnico informático freelance en España*. https://tarifaautonomo.com/blog/tarifa-hora-tecnico-informatico-espana

Toogethr. (s.f.). *Parking packages*. https://www.toogethr.com/en/parking/packages

Universidad Europea. (s.f.). *Escuela de Sostenibilidad*. https://universidadeuropea.com/conocenos/escuela-sostenibilidad/

U.S. Department of Transportation. (2008). *Advanced parking management systems* (ITS Deployment Evaluation, 2008-SC00131). https://www.itskrs.its.dot.gov/2008-sc00131

U.S. Department of Transportation. (2020). *Ultrasonic parking space availability sensors* (ITS Deployment Evaluation, 2020-SC00464). https://www.itskrs.its.dot.gov/2020-sc00464

U.S. Department of Transportation. (2025). *Parking reforms*. Climate Change Center. https://www.transportation.gov/sites/dot.gov/files/2025-01/Parking%20Reforms.pdf

Weinberger, R. R., Millard-Ball, A. y Hampshire, R. C. (2020). Parking search caused congestion: Where's all the fuss? *Transportation Research Part C: Emerging Technologies*, *120*, 102781. https://doi.org/10.1016/j.trc.2020.102781

ANEXOS

BIBLIOGRAFÍA 159
ANEXOS 164
ANEXO 1. Presupuesto desglosado para la instalación completa en la UEM 165
ANEXO 2. USO DE LA IA 167
ANEXO 3. ENTRENAMIENTO DE MODELOS 168
Anexo 3.1 Definición de parámetros de entrenamiento 168
Anexo 3.2 Primer modelo 171
Anexo 3.3 Segundo modelo 174
Anexo 3.4 Tercer modelo 177
Anexo3.5 Cuarto modelo 180
Anexo 3.6 Quinto modelo 183
Anexo 3.7 Sexto modelo 186
Anexo 3.8 Séptimo modelo 189
Anexo 3.9 Octavo modelo 192
Anexo 3.10 Noveno modelo 195
ANEXO 4. CÓDIGO DE LA APLICACIÓN 198
Anexo 4.1 Código de la interfaz de supervisión 198
Anexo 4.2 Código del sistema de visión y lógica interna 209

ANEXO 1. Presupuesto desglosado para la instalación completa en la UEM

En anexo muestra el presupuesto desglosado con todos los costes que supondría la implantación del sistema en la UEM.

Se recogen todos los costes ya mencionados en una sola tabla que los agrupa.

Bloque	Partida	Cant.	Precio ud.	Importe
BLOQUE A. CAPTACIÓN				
A	Cámara IP bullet Hikvision 4 MP	14	100,00 €	1.400,00 €
A	Soporte, herrajes y pequeño material	14	30,00 €	420,00 €
A	Temporizador analógico carril DIN	14	90,00 €	1.260,00 €
A	Convertidor local AC-DC 230 V a 12 V	14	10,00 €	140,00 €
A	**Subtotal captación**			**3.220,00 €**
BLOQUE B. PROCESAMIENTO EDGE				
B	NVIDIA Jetson Orin Nano Super Developer Kit	4	289,31 €	1.157,24 €
B	Envolvente exterior IP65 con carril DIN y fuente industrial	4	80,00 €	320,00 €
B	Integración interna: microSD industrial, ventilación, fusibles y prensaestopas	4	90,90 €	363,60 €
B	**Subtotal procesamiento edge**			**1.840,84 €**
BLOQUE C. COMUNICACIONES INALÁMBRICAS Y RED LOCAL				
C	Kits inalámbricos cámara-nodo	9 kits	170,00 €	1.530,00 €
C	Kits inalámbricos nodo-cabina/HMI central	4 kits	170,00 €	680,00 €
C	Switch industrial PoE de nodo	4	100,00 €	400,00 €
C	Inyectores PoE/adaptación de alimentación local	14	15,00 €	210,00 €
C	Cableado FTP exterior, conectores, herrajes y protecciones	1 partida	800,00 €	800,00 €
C	**Subtotal comunicaciones**			**3.620,00 €**
BLOQUE D. SOFTWARE E INTEGRACIÓN DIGITAL				
D	Configuración de firmware, red, nodos y comunicaciones seguras	25 h	25,00 €/h	625,00 €
D	Desarrollo del pipeline de visión, integración MQTT/HMI/APIs	40 h	25,00 €/h	1.000,00 €
D	Calibración fina de ROIs, ajuste de falsos positivos y validación funcional	55 h	25,00 €/h	1.375,00 €
D	Pruebas de aceptación, QA del algoritmo y control de latencia	60 h	25,00 €/h	1.500,00 €
D	Licencias, broker MQTT, base de datos y pasarelas API	1 partida	1.998,88 €	1.998,88 €
D	Interfaz de explotación y control lógico de barreras/paneles	1 partida	1.500,00 €	1.500,00 €
D	**Subtotal software e integración digital**			**7.998,88 €**
BLOQUE E. SERVICIOS DE INGENIERÍA, INSTALACIÓN Y PUESTA EN MARCHA				
E	Diseño final de cobertura, simulación geométrica y sectorización	20 h	25,00 €/h	500,00 €
E	Redacción de manuales, transferencia de conocimiento y formación	15 h	25,00 €/h	375,00 €
E	Mano de obra de instalación en campo	8 jornadas, 2 técnicos	300,00 €/j	2.400,00 €
E	Conexionado eléctrico, puenteado y pruebas in situ	8 uds.	220,00 €	1.760,00 €

Bloque	Partida	Cant.	Precio ud.	Importe
E	Alquiler de PEMP para trabajos en altura	8 días	150,00 €/día	1.200,00 €
E	PRL, balizamiento y seguridad en viales	1 partida	1.000,00 €	1.000,00 €
E	Dirección de obra y PMO	80 h	25,00 €/h	2.000,00 €
E	Reserva técnica y canalizaciones reforzadas	1 partida	740,00 €	740,00 €
E	**Subtotal servicios de ingeniería e instalación**			**9.975,00 €**
BLOQUE F. ACTUACIÓN E INFORMACIÓN AL USUARIO				
F	Barrera automática FAAC B614 o equivalente	8	2.500,00 €	20.000,00 €
F	Panel LED exterior informativo	6	1.500,00 €	9.000,00 €
F	**Subtotal actuación e información**			**29.000,00 €**
BLOQUE G. AMPLIACIÓN MEDIANTE CÁMARAS REUTILIZADAS				
G	Nodo edge adicional	1	460,21 €	460,21 €
G	Kits inalámbricos cámara-nodo para cámaras reutilizadas	2 kits	170,00 €	340,00 €
G	Kit inalámbrico nodo-cabina para quinto nodo	1 kit	170,00 €	170,00 €
G	Switch industrial PoE adicional	1	100,00 €	100,00 €
G	Inyectores/adaptación PoE	2	15,00 €	30,00 €
G	Cableado, herrajes y configuración adicional	1 partida	150,00 €	150,00 €
G	**Subtotal adaptación cámaras reutilizadas**			**1.250,21 €**
RESUMEN FINAL				
	Total implantación ampliada UEM sin contingencia			**56.904,93 €**
	Contingencia técnica 5 %			**2.845,25 €**
	TOTAL ESTIMADO CON CONTINGENCIA			**59.750,18 €**

ANEXO 2. USO DE LA IA

Para la elaboración de este Trabajo de Fin de Grado se ha utilizado la herramienta de inteligencia artificial ChatGPT (OpenAI) como apoyo puntual, con la autorización previa del tutor y dentro de los usos que contempla el reglamento de la Universidad Europea para la mejora de la ortografía, la gramática y la edición en general.

En concreto, la herramienta se ha empleado para revisar y mejorar la redacción de algunos apartados, corregir erratas y faltas, y ayudar a organizar y ordenar la estructura de ciertas secciones del documento. En todos los casos se ha partido de texto y de ideas propias, y la herramienta ha actuado sobre un contenido ya elaborado, sin generar el trabajo en sí.

No se ha utilizado la inteligencia artificial para el diseño del sistema, el desarrollo de la parte experimental, el entrenamiento y la comparación de los modelos, la toma de decisiones técnicas ni la obtención de los resultados, que son de elaboración propia. Las fuentes y referencias empleadas se han verificado de forma independiente.

El estudiante asume la autoría del trabajo y la responsabilidad sobre su contenido, habiendo revisado y contrastado con criterio propio todas las aportaciones de la herramienta antes de incorporarlas.

ANEXO 3. ENTRENAMIENTO DE MODELOS

En este anexo se recoge el proceso de entrenamiento seguido para ajustar los modelos de detección utilizados en el sistema. Se describen los principales hiperparámetros empleados, las configuraciones probadas y la evolución de los distintos entrenamientos realizados. El objetivo del anexo no es repetir el análisis del capítulo 4, sino dejar documentada la parte experimental que permitió comparar las distintas versiones del detector y justificar la selección final del modelo YOLO11s afinado.

Los entrenamientos se realizaron mediante *fine tuning* a partir de modelos preentrenados de la familia YOLO, utilizando imágenes de aparcamientos exteriores distintas a las empleadas en la validación final sobre la UEM. De esta forma se evitó entrenar y evaluar sobre el mismo escenario, reduciendo el riesgo de sobreajuste y permitiendo comprobar mejor la capacidad de generalización del sistema.

Básicamente, aquí se puede observar las diferentes pruebas que se realizaron a la hora de entrenar el modelo y que antes no habían sido detalladas.

Anexo 3.1 Definición de parámetros de entrenamiento

La librería de Ultralytics permite ajustar diversos hiperparámetros relacionados con el número de pasadas sobre el conjunto de entrenamiento, las tasas de aprendizaje y los pesos de las distintas componentes de la función de pérdida, entre otros. Según la variante de YOLO seleccionada, el número de hiperparámetros accesibles puede variar. En este trabajo se realizan entrenamientos por fine-tuning con las versiones YOLO11 y YOLO26; esta última ofrece opciones adicionales de configuración. A continuación se listan y describen los hiperparámetros empleados para el modelo basado en YOLO11:

- Epochs (épocas): Número máximo de pasadas completas por el conjunto de entrenamiento. Valores altos permiten explotar mejor los datos, pero aumentan el riesgo de sobreajuste. En este trabajo se han usado valores entre 280 y 600, combinados siempre con early-stopping mediante el hiperparámetro patience
- imgsz (Tamaño de las imágenes): Tamaño cuadrado al que se redimensionan las imágenes (NxN). Debe ser múltiplo de 32 para ajustarse al *stride* de la red (p. ej. 512, 960, 1024). Un imgsz mayor mejora la detección de objetos pequeños o lejanos, a costa de más consumo de memoria y tiempo de entrenamiento.
- batch: Número de imágenes procesadas en cada iteración de entrenamiento antes de actualizar los pesos. Un *batch* mayor produce un gradiente más estable, pero requiere más

memoria de GPU; uno más pequeño introduce más ruido en el gradiente, lo que a veces mejora la generalización en datasets pequeños.

- optimizar: Algoritmo utilizado para ajustar los pesos del modelo durante el entrenamiento. Se ha utilizado principalmente AdamW, recomendado por Ultralytics para *fine-tuning* en datasets pequeños, ya que converge rápido y mantiene una buena estabilidad sin requerir un ajuste fino tan delicado del lr como SGD.
- lr0: Tasa de aprendizaje inicial, controla qué tan grandes son los pasos al ajustar los pesos. Valores más altos permiten explorar soluciones más agresivas, pero pueden desestabilizar el entrenamiento.
- lrf: Factor de la tasa de aprendizaje final, que se usa para reducir gradualmente la tasa de aprendizaje durante el entrenamiento. Valores bajos fuerzan que el *learning rate* caiga más, favoreciendo un ajuste fino al final.
- warmup_epochs (épocas de calentamiento): Número de épocas en las que la tasa de aprendizaje aumenta gradualmente al inicio para evitar inestabilidades al comienzo.
- Pesos relativos para las diferentes componentes de la función de pérdida:
  - box: pérdida de la caja delimitadora. Incrementarlo hace que el modelo priorice el ajuste geométrico de las cajas frente a la clasificación.
  - cls: pérdida de clasificación. Subirlo penaliza más los falsos positivos y negativos de clase.
  - dfl: pérdida de DFL . Aumentarlo afina aún más los bordes de las cajas.
- Parámetros para la aumentación de datos, estrategia conocida como *augmentation*. Generando escenas artificiales que mejoran la robustez en *datasets* pequeños, aunque valores muy altos pueden producir composiciones poco realistas:
  - mosaic y mixup: combinan varias imágenes.
  - hsv_h, hsv_s, hsv_v: controlan variaciones de tono, saturación y brillo.
  - degrees, translate, scale: controlan rotaciones, traslaciones y escalado.
  - flipud, fliplr: definen volteos verticales y horizontales.
- patience: Número de épocas consecutivas sin mejora significativa en la métrica de validación a partir del cual se detiene el entrenamiento (*early stopping*). Valores demasiado bajos pueden cortar el aprendizaje prematuramente; valores demasiado altos pueden permitir sobreajuste en un conjunto de datos pequeño.

Para el modelo YOLO versión 26, aparte de los hiperparámetros ya listados, se amplía la posibilidad de personalización de entrenamiento con los listados a continuación:

- weight_decay: aplica “Regularización L2” sobre los pesos para reducir sobreajuste.
- cos_lr: activa un programador de tasa de aprendizaje con decaimiento cosenoidal, que tiende a mejorar la convergencia en *fine-tuning* prolongados.
- label_smoothing: suaviza las etiquetas de clase, evitando que el modelo se vuelva excesivamente confiado y mejorando la calibración de las probabilidades.
- close_mosaic: desactiva el *mosaic* en las últimas N épocas para que el modelo se ajuste más a imágenes realistas al final del entrenamiento.

Se han entenado varios modelos, variando los parámetros listados hasta obtener un resultado fiable y estable para la aplicación del alcance de este proyecto. Se van a establecer en más detalle el motivo de elección del valor de cada uno de los hiperparámetros para el primer modelo, y según se avance en la explicación de los modelos siguientes hasta llegar al último, se van explicar los

motivos del cambio de valor de los hiperparámetros para ese nuevo modelo, pero sólo con los modificados.

Aparte, como para comprobar el resultado del entrenamiento antes de realizar la prueba del despliegue del modelo entrenado, durante el fine-tuning de cada modelo se registran las siguientes métricas de entrenamiento y validación:

- train/box_loss, train/cls_loss, train/dfl_loss: evolución de las tres componentes de la función de pérdida sobre el conjunto de entrenamiento (caja, clasificación y distribución de bordes).
- val/box_loss, val/cls_loss, val/dfl_loss: las mismas pérdidas, pero sobre el conjunto de validación.
- metrics/precision(B): precisión de detección a nivel de *bounding box*, es decir, proporción de cajas predichas como "coche" que realmente corresponden a un coche.
- metrics/recall(B): *recall* a nivel de *bounding box*, es decir, proporción de coches del conjunto de validación que el modelo consigue detectar.
- metrics/mAP50(B): media del *Average Precision* usando un umbral de solapamiento IoU=0.5 entre predicción y *ground truth*.
- metrics/mAP50-95(B): mAP promedio para umbrales de IoU entre 0.5 y 0.95 en pasos de 0.05, una métrica más exigente que refleja tanto detección como ajuste fino de las cajas.

Las curvas de estas métricas permiten analizar la curva de aprendizaje de cada modelo, que dice si las pérdidas de entrenamiento y validación descienden de forma paralela, sin separarse, y las métricas de precisión, *recall* y mAP se estabilizan en un plateau, se considera que el modelo ha convergido adecuadamente. En cambio, si las pérdidas de entrenamiento siguen bajando mientras las de validación empeoran, se interpreta como señal de sobreentrenamiento. El uso de *early stopping* mediante el hiperparámetro patience ayuda a detener el entrenamiento cuando la mejora en validación deja de ser significativa.

Además de las curvas, para cada modelo se calcula una matriz de confusión sobre el conjunto de validación, separando verdaderos positivos (coches correctamente detectados), falsos positivos (fondo marcado como coche), verdaderos negativos (fondo correctamente ignorado) y falsos negativos (coches no detectados). A partir de esta matriz se pueden estimar la precisión, el *recall*, el F1-score y la tasa de falsos positivos, lo que permite cuantificar con más detalle el equilibrio entre aciertos y errores en la detección de vehículos.

Se empezó optimizando los hiperparámetros para el modelo YOLO11s como semilla, y cuando se obtenga el resultado esperado, se entrenará con el YOLO11n y el YOLO26s para comparar si se obtienen resultados parecidos con menos consumo o mejores.

Anexo 3.2 Primer modelo

Hiperparámetros del primer modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=280 imgsz=512 batch=16 optimizer="AdamW"	lr0=0.001 lrf=0.1 warmup_epochs=2.0 patience=20	box=10.0 cls=0.5 dfl=1.5	mosaic=0.9 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 degrees=5 translate=0.1 scale=0.5 flipud=0.0 fliplr=0.5

Se parte de la semilla YOLO11s y en este primer modelo se utilizó imgsz=512 y batch=16 para ajustarse a las limitaciones de memoria de la GPU T4 de Google Colab gratuito, garantizando al mismo tiempo un número suficiente de imágenes por iteración para obtener un gradiente relativamente estable.

Se eligió AdamW como optimizador por su buen comportamiento en *fine-tuning* con datasets pequeños, y una tasa de aprendizaje inicial lr0=0.001 con lrf=0.1, siguiendo rangos habituales recomendados por Ultralytics para comenzar los experimentos. El calentamiento de 2 épocas (warmup_epochs=2.0) es suficiente para estabilizar el entrenamiento sin alargar innecesariamente la fase de *warmup*.

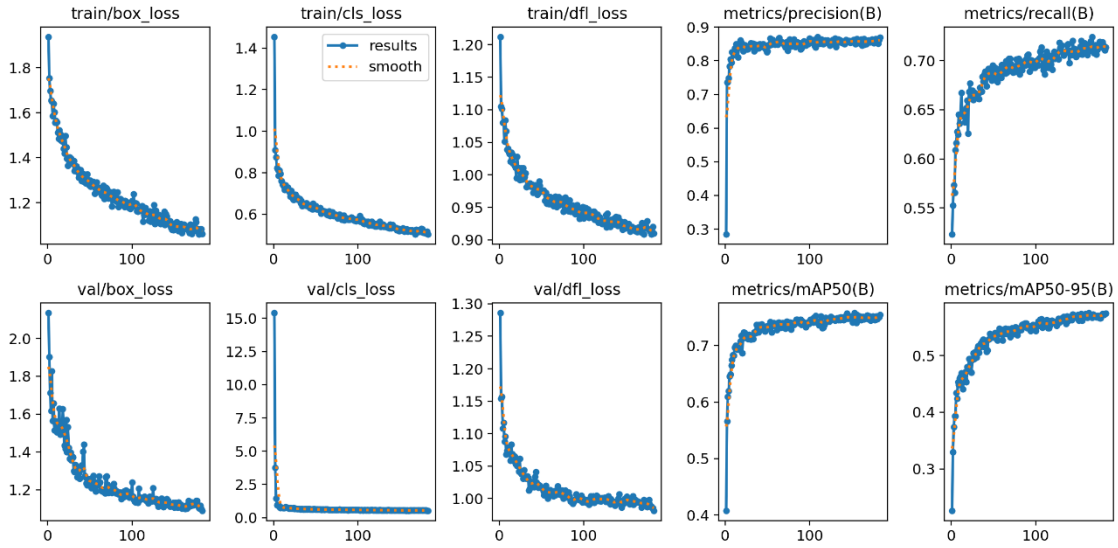
En cuanto a la función de pérdida, se partió de unos pesos moderados (box=10.0, cls=0.5, dfl=1.5) para no forzar todavía un ajuste muy agresivo de las cajas ni de la clasificación. Dado el tamaño reducido del dataset, se utilizó una aumentación de datos intensiva (mosaic=0.9) para aumentar la diversidad de ejemplos y mejorar la capacidad de generalización del modelo. Finalmente, patience=20 permite que el *early stopping* corte el entrenamiento si la métrica de validación deja de mejorar durante un número razonable de épocas, reduciendo el riesgo de sobreajuste en esta primera prueba.

Este entrenamiento llevó 183/280 épocas durante un tiempo de 28 minutos y 16 segundos. Consumiendo en cada época 5.4 GB de RAM de la GPU T4.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    183/280       5.4G      1.062     0.5035     0.9098        724        512: 100% ━━━━━━━━━━━━ 31/31 
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━
                   all        108       5090      0.869      0.714      0.755      0.575
EarlyStopping: Training stopped early as no improvement observed in last 20 epochs. Best results observe
To update EarlyStopping(patience=20) pass a new patience value, i.e. `patience=300` or use `patience=0`

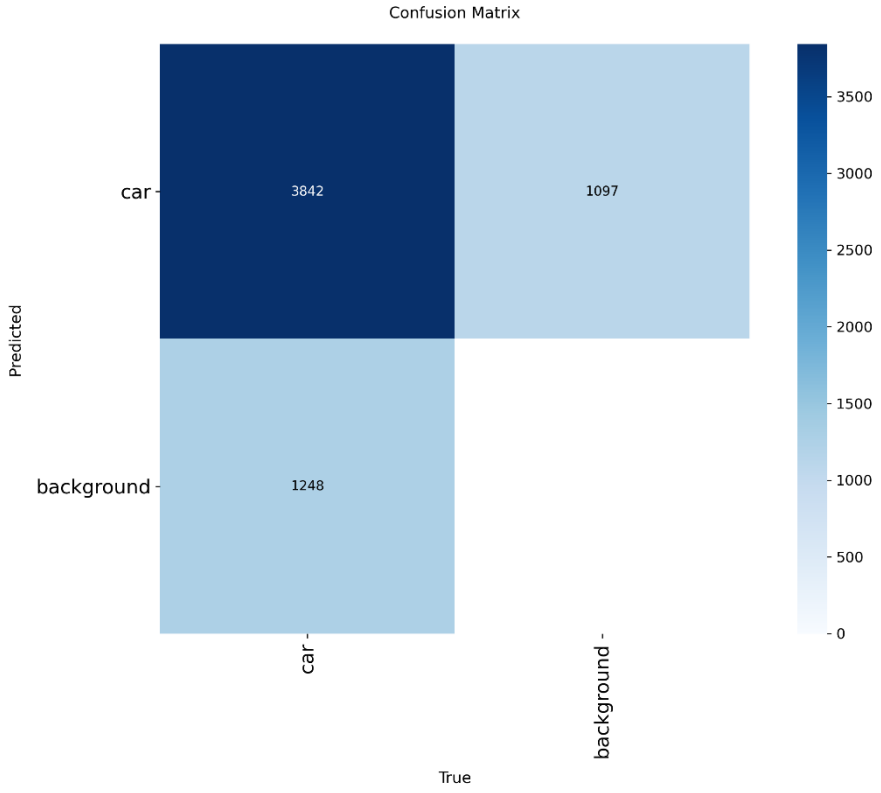
183 epochs completed in 0.471 hours.
```

El entrenamiento del primer modelo se detuvo en la época 183 de 280 por activarse el *early stopping* con patience=20, lo que indica que, a partir de ese punto, la métrica de validación ya no mostraba mejoras significativas. Las curvas de train/box_loss, train/cls_loss y train/dfl_loss descienden rápidamente durante las primeras épocas y luego se aplanan, mientras que las curvas de validación siguen una tendencia paralela, sin separarse de forma apreciable. Las métricas mAP50 y mAP50-95 crecen de forma rápida al principio y alcanzan un plateau aproximado entre las épocas 120 y 150. En conjunto, las curvas sugieren una convergencia razonable sin signos claros de sobreentrenamiento, aunque el modelo no llega a explotar completamente todos los datos disponibles.



La matriz de confusión del modelo 1 muestra un número elevado de verdaderos positivos pero todavía una cantidad importante de falsos negativos, lo que se traduce en un *recall* moderado. La precisión es aceptable, es decir, la mayoría de las cajas que el modelo etiqueta como “coche” son

correctas, pero deja sin detectar una parte relevante de los vehículos en imágenes complejas



Al desplegar el modelo en el caso piloto del aparcamiento de la universidad, el cual no se usó para el entrenamiento, se observó que mejoró respecto al modelo YOLO11 pero le costaba la detección de vehículos lejanos, de los coches poco visibles en zonas agrupadas y con oclusiones por troncos o ramas de árboles.

Anexo 3.3 Segundo modelo

Hiperparámetros del segundo modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=280 imgsz=960 batch=6 optimizer="AdamW"	lr0=0.001 lrf=0.1 warmup_epochs=2.0 patience=20	box=10.0 cls=0.5 dfl=1.5	mosaic=0.9 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 degrees=5 translate=0.1 scale=0.5 flipud=0.0 fliplr=0.5

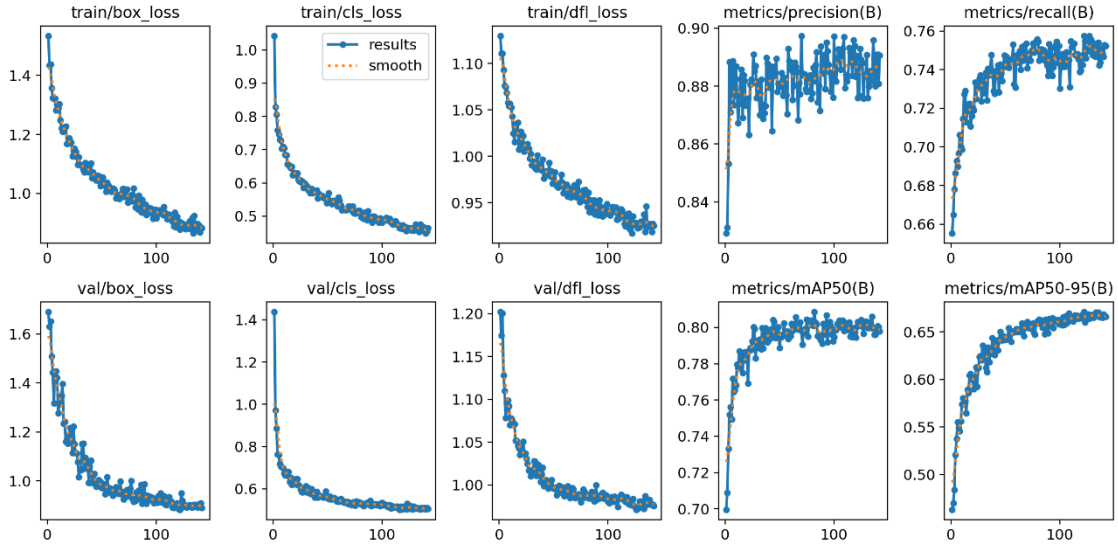
Este segundo modelo también se entrenó con la versión gratuita y con la semilla YOLO11s. Para estar trabajando con un tamaño de 512x512 de imágenes, respecto al tamaño objetivo de despliegue de 1920, se observó un muy buen resultado. Pero para poder detectar los vehículos más lejanos y agrupados, se optó por pasar el tamaño del entrenamiento de 512 a 960. Esto incrementaba bastante el cómputo requerido, por lo que se redujo el número de imágenes procesadas en cada iteración de 16 a 6. El resto de los parámetros los mantiene igual. Este cambio busca mejorar la detección de vehículos pequeños y lejanos, más cercanos al tamaño real de los objetos en el escenario de despliegue (1920×1080), a costa de un entrenamiento más lento.

Este entrenamiento llevó 142/280 épocas durante un tiempo de 1 hora 28 minutos y 30 segundos. Consumiendo en cada época 7.56 GB de RAM de la GPU T4.

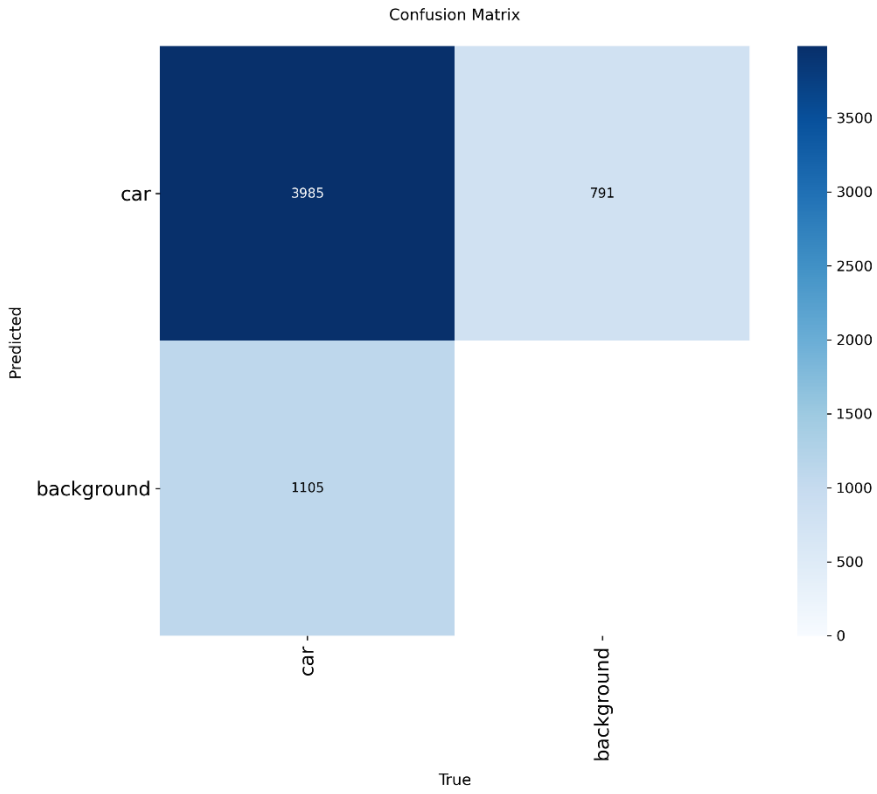
```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    142/280      7.56G     0.8823     0.4642     0.9253        139        960: 100% ━━━━━━━━━━━━ 82/82 2
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━
                   all        108       5090      0.891      0.752      0.798      0.666
EarlyStopping: Training stopped early as no improvement observed in last 20 epochs. Best results observe
To update EarlyStopping(patience=20) pass a new patience value, i.e. `patience=300` or use `patience=0`

142 epochs completed in 1.475 hours.
```

En el segundo modelo se incrementa la resolución a 960×960 manteniendo el resto de la configuración, salvo el batch, que se reduce de 16 a 6 por limitaciones de memoria. El entrenamiento se detiene en la época 142/280. Las curvas de pérdida muestran un patrón similar al modelo 1, con descensos rápidos al inicio y estabilización posterior, pero los valores finales de val/box_loss, val/cls_loss y val/dfl_loss son algo menores, y las curvas de metrics/mAP50 y metrics/mAP50-95 alcanzan un nivel superior. No se aprecia separación entre las curvas de entrenamiento y validación, por lo que no hay evidencias claras de sobreentrenamiento.



La matriz de confusión muestra un incremento de verdaderos positivos y una ligera disminución de falsos negativos respecto al modelo 1, lo que se traduce en un recall algo mejor. La precisión se mantiene en un rango similar, de modo que la tasa de falsos positivos no aumenta de forma apreciable.



Al desplegar el modelo en el ZONA PILOTO DE LA UEM aún se observan problemas con coches muy oclusos, vehículos en sombra y agrupaciones densas de coches, lo que motiva seguir ajustando los pesos de la pérdida y el resto de hiperparámetros.

Anexo 3.4 Tercer modelo

Hiperparámetros del tercer modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=280 imgsz=960 batch=6 optimizer="AdamW"	lr0=0.001 lrf=0.1 warmup_epochs=2.0 patience=20	box=15.0 cls=0.5 dfl=1.5	mosaic=0.9 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 degrees=5 translate=0.1 scale=0.5 flipud=0.0 fliplr=0.5

Como el cómputo era limitado y se agotó, se eligió pasar a usar el entorno Google Colab Pro. También se ha partido de la semilla YOLO11s en este entrenamiento.

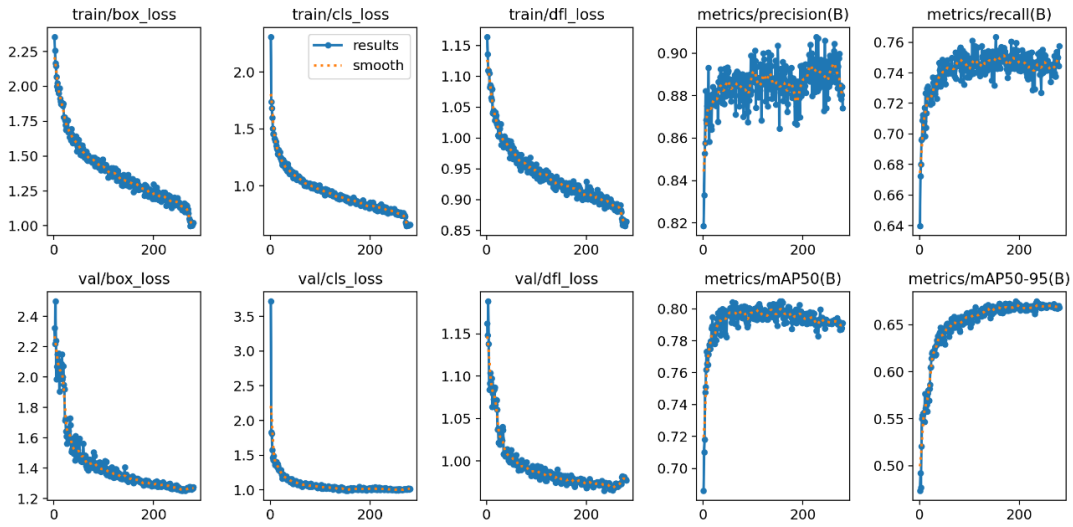
Se ha observado que la estrategia del early-stopping estaba haciendo efecto y terminando las épocas, se optó por hacer una comprobación de si de verdad el modelo estaba dejando de aprender y se estaba cortando a tiempo o si se estaba cortando el entrenamiento antes de tiempo. El tercer modelo mantiene la configuración del modelo 2 salvo por un incremento del peso box de 10.0 a 15.0. La intención era penalizar más los errores en la geometría de las cajas para reducir las cajas excesivamente grandes que se habían observado, especialmente en zonas donde el modelo agrupaba varios coches y parte del fondo en una sola detección

Este entrenamiento llevó 280/280 épocas durante un tiempo de 18 minutos y 30 segundos. Consumiendo en cada época 9.94 GB de RAM de la GPU A100.

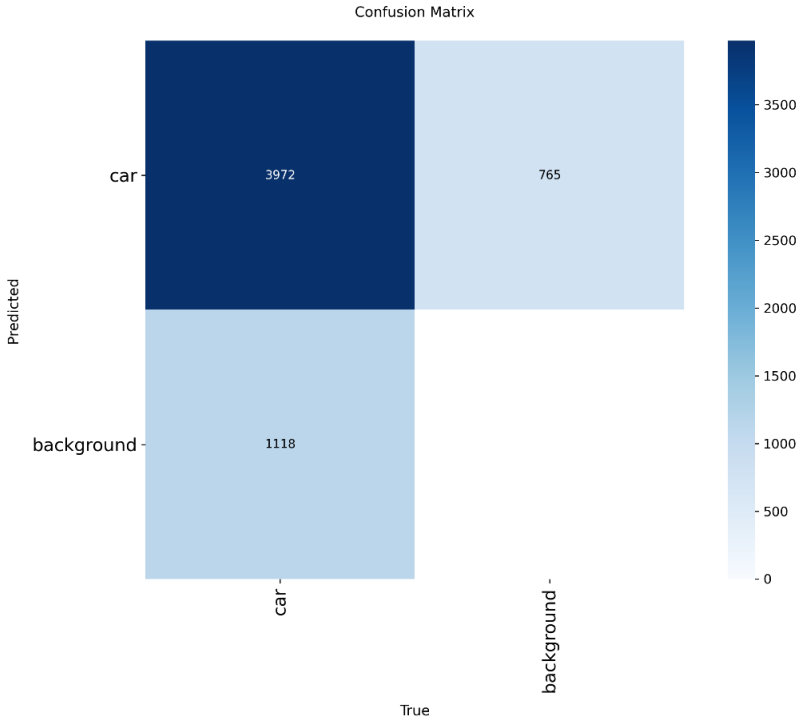
```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    280/280      9.94G      1.022     0.6586     0.8643        108        960: 100% ━━━━━━━━━━━━ 82/82
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━
                   all        108       5090      0.874      0.758      0.791      0.668

280 epochs completed in 0.315 hours.
```

El tercer modelo mantiene la configuración del modelo 2, pero incrementa el peso de la pérdida de caja de box=10.0 a box=15.0 con el objetivo de penalizar más las cajas mal ajustadas y reducir las cajas excesivamente grandes. En este caso el entrenamiento recorre las 280 épocas completas sin que se active el *early stopping*, lo que indica que la métrica de validación continúa mejorando, aunque muy lentamente, hasta el final. Las curvas de pérdida muestran una reducción continua, pero cada vez más plana; las métricas de mAP mejoran de forma marginal en las últimas épocas. No se observan divergencias claras entre entrenamiento y validación, de modo que no hay sobreajuste evidente, pero sí un rendimiento incremental muy limitado.



La matriz de confusión del modelo 3 es similar a la del modelo 2, con ligeras variaciones en verdaderos positivos y falsos negativos que no representan una mejora clara. En términos cualitativos, el modelo no reduce de manera significativa la tasa de falsos positivos ni mejora el recall de coches difíciles.



Al desplegar el modelo en el ZONA PILOTO DE LA UEM, se observa un cierto empeoramiento: el modelo tiende a ser más conservador, detecta menos vehículos lejanos y sigue teniendo problemas importantes con coches oclusos y con sombras. Esto sugiere que el incremento de box

ha hecho al modelo demasiado exigente con la geometría de la caja en ejemplos limpios, sin resolver el problema de las cajas grandes de fondo, por lo que se decide explorar ajustes más profundos en el siguiente modelo.

Anexo3.5 Cuarto modelo

Hiperparámetros del cuarto modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=500 imgsz=960 batch=16 optimizer="AdamW"	lr0=0.005 lrf=0.05 warmup_epochs=2.0 patience=75	box=20.0 cls=2.0 dfl=2.0	mosaic=0.7 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.6 degrees=5 translate=0.1 scale=0.5 flipud=0.0 fliplr=0.5

Como los modelos anteriores daban problemas con las zonas más conflictivas y todo indicaba que se centraban en aprender las zonas más directas y sencillas. El cuarto modelo supone un rediseño más profundo de la configuración, partiendo den la semilla YOLO11s. Se mantiene imgsz=960 pero se aumenta el *batch* de 6 a 16 aprovechando la GPU A100 de Colab Pro, lo que proporciona gradientes más estables. La tasa de aprendizaje inicial lr0 se incrementa de 0.001 a 0.005 y lrf se reduce de 0.1 a 0.05, buscando que el modelo explore soluciones más variadas al principio y realice un ajuste más fino al final.

Los pesos de pérdida se incrementan a box=20.0, cls=2.0 y dfl=2.0, dando más importancia tanto a la geometría de las cajas como a la clasificación. El objetivo es que el modelo sea más estricto, diferencie vehículos del fondo y reduzca las cajas mal ajustadas. Para evitar que el entrenamiento se corte demasiado pronto, patience se aumenta a 75 y el límite de épocas a 500. Por último, se reduce mosaic de 0.9 a 0.7 y se aumenta hsv_v a 0.6, buscando un equilibrio entre diversidad de iluminaciones y realismo, ya que un *mosaic* demasiado agresivo puede generar escenas poco representativas que dificultan la convergencia.

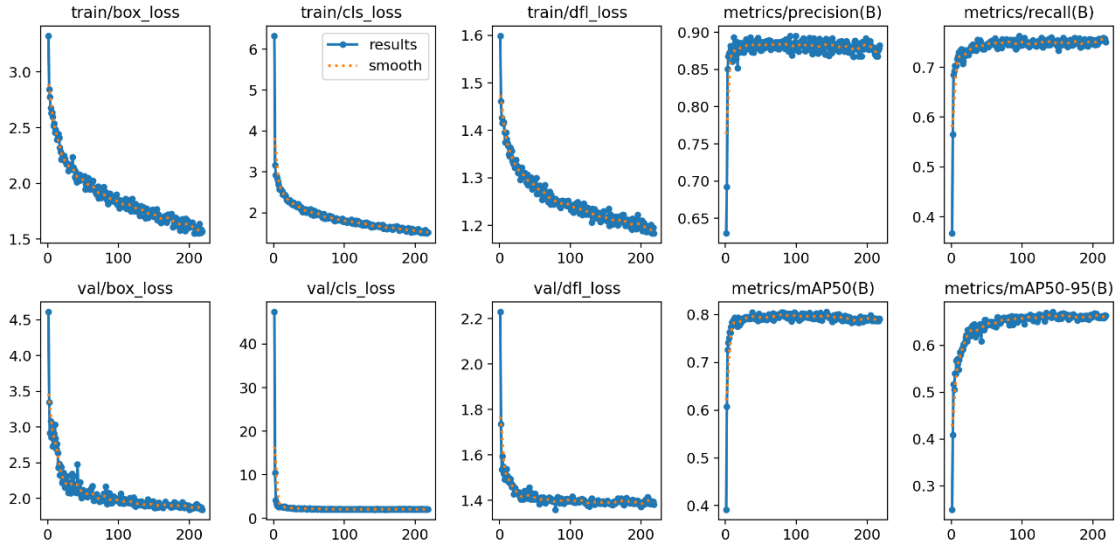
Este entrenamiento llevó 218/500 épocas durante un tiempo de 24 minutos y 25 segundos. Consumiendo en cada época 28.6 GB de RAM de la GPU A100.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    218/500      28.6G      1.571      1.517      1.183        877        960: 100% ━━━━━━━━━━━━ 31/31 5
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━
                   all        108       5090      0.882       0.75      0.791      0.664
EarlyStopping: Training stopped early as no improvement observed in last 75 epochs. Best results observe
To update EarlyStopping(patience=75) pass a new patience value, i.e. `patience=300` or use `patience=0`

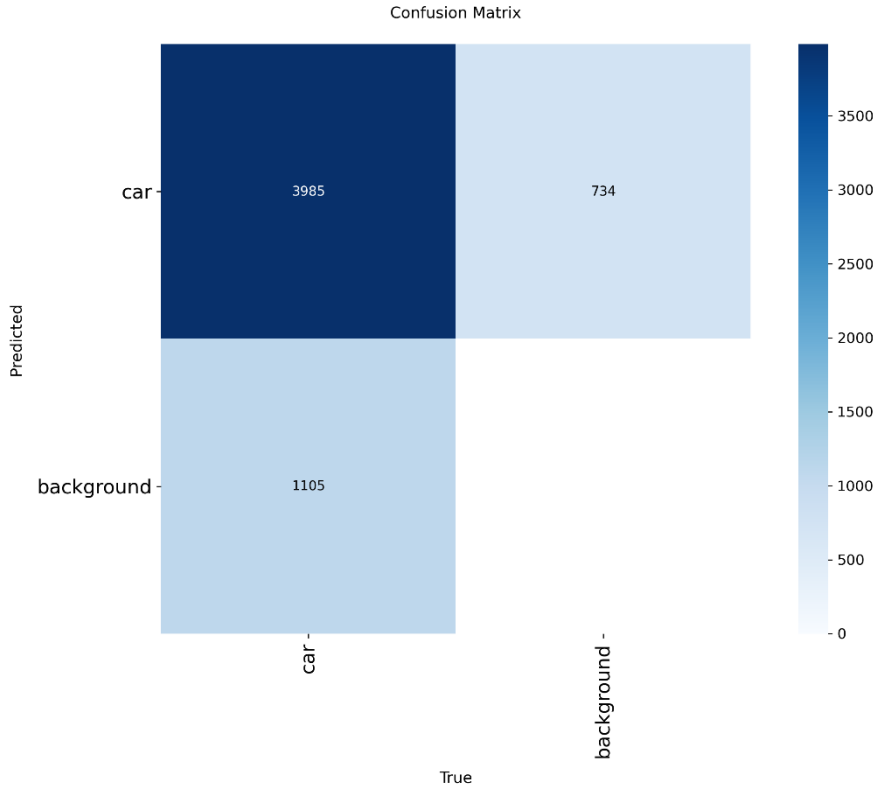
218 epochs completed in 0.407 hours.
```

Las curvas de pérdida de entrenamiento y validación descienden de forma suave y casi paralela, sin separación visible, y las métricas de mAP50 y mAP50-95 alcanzan un plateau estable

aproximadamente entre las épocas 150 y 180. No se observan indicios de sobreentrenamiento importante; más bien, se aprecia una convergencia robusta a un rendimiento alto y estable.



La matriz de confusión del modelo 4 presenta un incremento significativo de verdaderos positivos y una reducción de falsos negativos respecto a los modelos 1–3, con una ligera reducción también de falsos positivos. En términos aproximados, la precisión se sitúa en torno al 0.84 y el *recall* alrededor de 0.78, lo que da lugar a un F1 de 0.81 (frente a valores claramente inferiores en los modelos iniciales), lo que indica un buen equilibrio entre aciertos y errores. La proporción de detecciones incorrectas (tasa de falsos positivos) es relativamente baja, es decir, la mayoría de las cajas marcadas como coche corresponden efectivamente a un vehículo.



Al desplegar el modelo en el ZONA PILOTO DE LA UEM se observa que el modelo detecta con alta confianza tanto vehículos cercanos como muchos de los coches lejanos y oclusos, incluidos dos vehículos oscuros y en sombra al fondo del aparcamiento. El principal defecto que persiste es la presencia ocasional de cajas muy grandes que agrupan varios coches y parte del fondo, aunque este problema es manejable mediante filtrado en la fase de despliegue.

Con este modelo se pudo observar que con ajustes en el despliegue de la “iou”, se ha obtenido un modelo que tiene buena generalización y confianza en la clasificación de coches en aparcamientos.

Anexo 3.6 Quinto modelo

Hiperparámetros del quinto modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=500 imgsz=960 batch=32 optimizer="AdamW"	lr0=0.005 lrf=0.05 warmup_epochs=2.0 patience=150	box=25.0 cls=2.0 dfl=2.0	mosaic=0.7 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.6 degrees=5 translate=0.1 scale=0.5 flipud=0.0 fliplr=0.5

Con la misma semilla YOLO11s, y partiendo del modelo 4, el quinto modelo mantiene la mayoría de hiperparámetros pero aumenta el batch de 16 a 32, aprovechando la GPU A100 para ver si un gradiente más estable ayuda a reducir falsos positivos y a mejorar la delimitación de las cajas. Además, se incrementa el peso box de 20.0 a 25.0 y patience de 75 a 150. La motivación era doble:

- forzar al modelo a penalizar todavía más los errores geométricos de las cajas, intentando reducir las cajas grandes que englobaban varios coches y parte del fondo;
- dar más margen de épocas "útiles" al entrenamiento antes de que se active el early stopping, por si el modelo necesitaba más tiempo para aprender las zonas conflictivas.

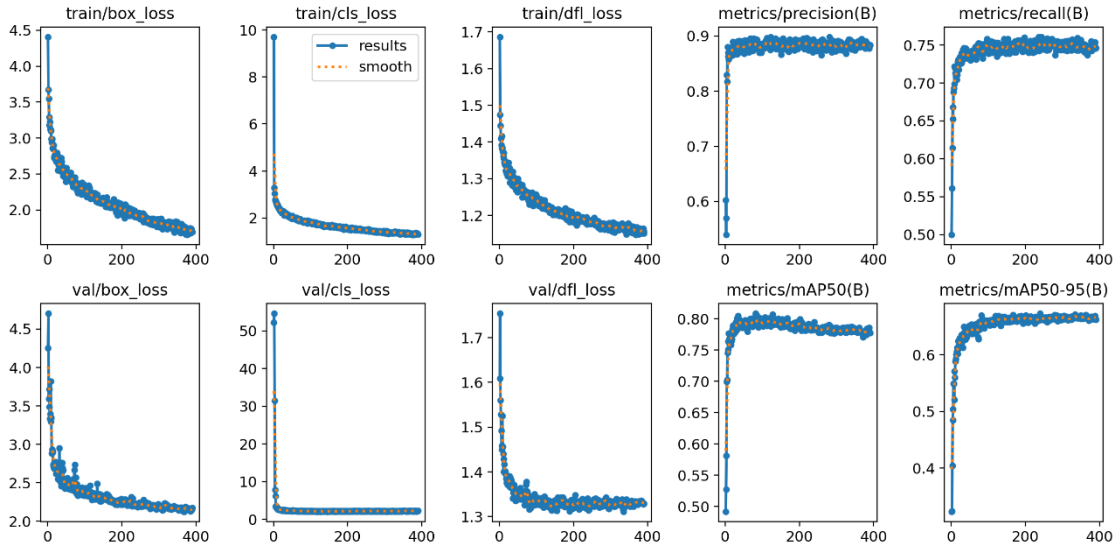
Este entrenamiento llevó 390/500 épocas durante un tiempo de 40 minutos y 5 segundos. Consumiendo en cada época 39.1 GB de RAM de la GPU A100.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    390/500      39.1G      1.697      1.307      1.151        703        960: 100% ──────────── 16/16
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ────────
                   all        108       5090      0.884      0.746      0.778      0.662
EarlyStopping: Training stopped early as no improvement observed in last 150 epochs. Best results obse
To update EarlyStopping(patience=150) pass a new patience value, i.e. `patience=300` or use `patience=

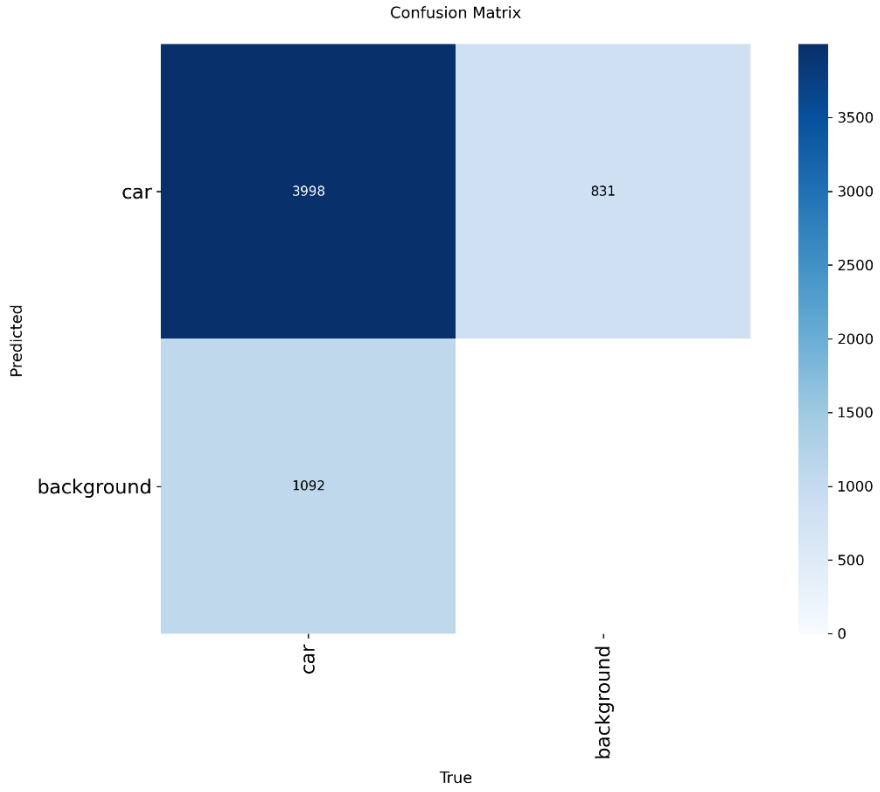
390 epochs completed in 0.668 hours.
```

El entrenamiento se detiene en la época 390/500, lo que indica que el modelo sigue mejorando la métrica de validación durante más tiempo que el modelo 4. Las curvas de train/box_loss, train/cls_loss y train/dfl_loss muestran un descenso prolongado, mientras que las curvas de validación descienden también pero con una pendiente cada vez menor a partir de aproximadamente la época 250. Las métricas mAP50 y mAP50-95 crecen rápidamente al principio, pero luego entran en un plateau con mejoras muy pequeñas, sin signos claros de sobreajuste fuerte, aunque sí de rendimiento marginal decreciente.

En resumen, el modelo 5 parece exprimir algo más el ajuste en validación, pero con una ganancia muy modesta respecto al esfuerzo de entrenamiento adicional.



La matriz de confusión del modelo 5 muestra un número de verdaderos positivos muy similar al del modelo 4 y una ligera variación en falsos positivos y falsos negativos. Aproximadamente, la precisión se mantiene en torno a 0.83–0.84 y el *recall* alrededor de 0.78, con un F1-score del orden de 0.80. La tasa de falsos positivos (1 − precisión) se mantiene en torno al 16–17 %, lo que indica que el incremento de box no ha eliminado de forma significativa las cajas incorrectas.



Al desplegar el modelo en el ZONA PILOTO DE LA UEM e observa que el modelo 5 **sigue generando cajas grandes** que agrupan coches, cielo y árboles y, además, pierde sensibilidad en coches agrupados y con oclusiones: algunos vehículos laterales y parcialmente ocultos dejan de detectarse o aparecen con confianza muy baja. Esto sugiere que el aumento simultáneo de box, batch y patience ha llevado al modelo a una solución demasiado "conservadora": prefiere pocas cajas más grandes y seguras, sacrificando el *recall* en casos difíciles. Por ello, a pesar de tener métricas agregadas parecidas al modelo 4, su comportamiento cualitativo en las escenas más críticas es peor y se descarta como modelo candidato.

Anexo 3.7 Sexto modelo

Hiperparámetros del sexto modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=500 imgsz=960 batch=12 optimizer="AdamW"	lr0=0.005 lrf=0.05 warmup_epochs=2.0 patience=75	box=12.0 cls=1.5 dfl=2.0	mosaic=0.6 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.4 degrees=5 translate=0.1 scale=0.6 flipud=0.0 fliplr=0.5

El sexto modelo surge como reacción a los problemas observados en el modelo 5. En lugar de seguir aumentando el peso de la pérdida de caja, se reduce de forma notable box=25.0 → 12.0 y también cls=2.0 → 1.5, buscando un equilibrio más razonable entre precisión geométrica y semántica. La idea es que el modelo deje de "castigar" en exceso pequeños errores de caja que pueden ocurrir en coches oclusos, y centre más su capacidad en diferenciar coche/fondo.

Paralelamente, se ajustan las aumentaciones de datos para hacerlas algo menos agresivas: mosaic pasa de 0.7 a 0.6, hsv_v de 0.6 a 0.4 y scale de 0.5 a 0.6. Con ello se pretende reducir la probabilidad de generar escenas artificiales que favorezcan cajas de fondo extrañas y, al mismo tiempo, seguir aportando diversidad suficiente para generalizar en distintas iluminaciones. El *batch* se fija en 12 como compromiso entre estabilidad y ruido útil en el gradiente, manteniendo patience=75 y epochs=500.

Este entrenamiento llevó 192/500 épocas durante un tiempo de 23 minutos y 35 segundos. Consumiendo en cada época 17 GB de RAM de la GPU A100.

```
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ──────────
                 all        108       5090      0.884      0.756      0.801      0.671

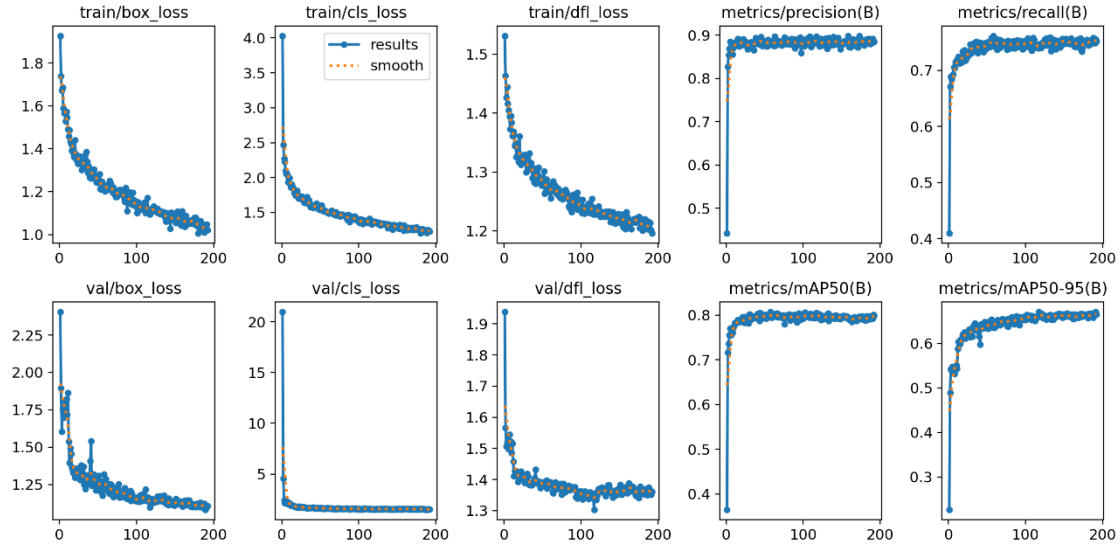
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    192/500        17G      1.022       1.23      1.196        807        960: 100% ──────────── 41/41 
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ──────────
                 all        108       5090      0.886      0.752      0.799      0.665
EarlyStopping: Training stopped early as no improvement observed in last 75 epochs. Best results observe
To update EarlyStopping(patience=75) pass a new patience value, i.e. `patience=300` or use `patience=0`

192 epochs completed in 0.393 hours.
```

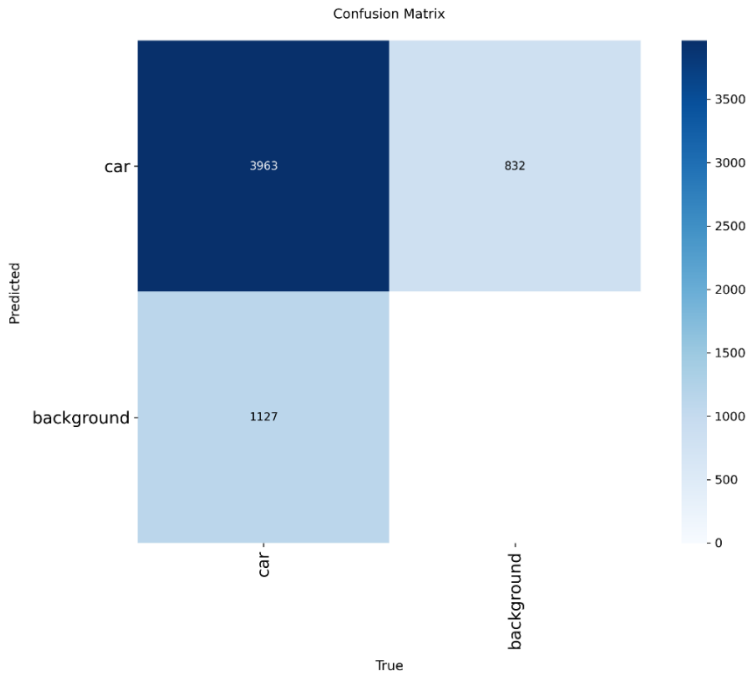
El entrenamiento del modelo 6 se detiene en la época 192/500 por *early stopping*. Las curvas de pérdida muestran un descenso rápido en las primeras épocas y una estabilización razonable a

partir de ~150 épocas, con trayectorias de entrenamiento y validación muy próximas, lo que indica buena generalización y ausencia de sobreajuste apreciable. Las métricas mAP50 y mAP50-95 alcanzan valores similares a los del modelo 4 en torno a la época 150, y las mejoras posteriores son marginales.

En términos de dinámica de aprendizaje, el modelo 6 parece converger antes que el modelo 5 y no se beneficia de un número muy alto de épocas, lo que confirma que la nueva configuración de pérdidas y aumentaciones es más eficiente.



En la matriz de confusión de puede observar que



La matriz de confusión del modelo 6 presenta valores de verdaderos positivos y falsos negativos muy próximos a los del modelo 4, con ligeras oscilaciones en falsos positivos. A grandes rasgos, la precisión se sitúa de nuevo en torno al 0.82–0.83 y el recall en torno al 0.77–0.78, con un F1-score cercano a 0.80. La tasa de falsos positivos se mantiene en un rango similar al del modelo 4 y 5, lo que indica que la reducción de box por sí sola no basta para eliminar las cajas grandes de fondo.

En las pruebas sobre el ZONA PILOTO DE LA UEM, el modelo 6 muestra un comportamiento **intermedio**: reduce ligeramente algunos casos de cajas excesivas, pero no de forma sistemática, y no supone una mejora clara en la detección de coches oclusos o agrupados respecto al modelo 4. Dado que tampoco ofrece una ventaja evidente en coches lejanos, se concluye que el modelo 6 no aporta un beneficio suficiente para ser considerado sustituto del modelo 4, aunque sí ayuda a entender el efecto de reequilibrar los pesos de pérdida y las aumentaciones.

Anexo 3.8 Séptimo modelo

Hiperparámetros del séptimo modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=400 imgsz=960 batch=16 optimizer="AdamW"	lr0=0.005 lrf=0.05 warmup_epochs=2.0 patience=75	box=20.0 cls=2.0 dfl=2.0	mosaic=0.7 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.6 degrees=5 translate=0.1 scale=0.5 flipud=0.0 fliplr=0.5

El séptimo modelo mantiene exactamente la receta de hiperparámetros del modelo 4, pero cambiando la semilla al modelo base YOLO26s. El objetivo es analizar si la nueva arquitectura aporta mejoras en detección manteniendo el mismo esquema de entrenamiento, de forma que la comparación sea lo más justa posible.

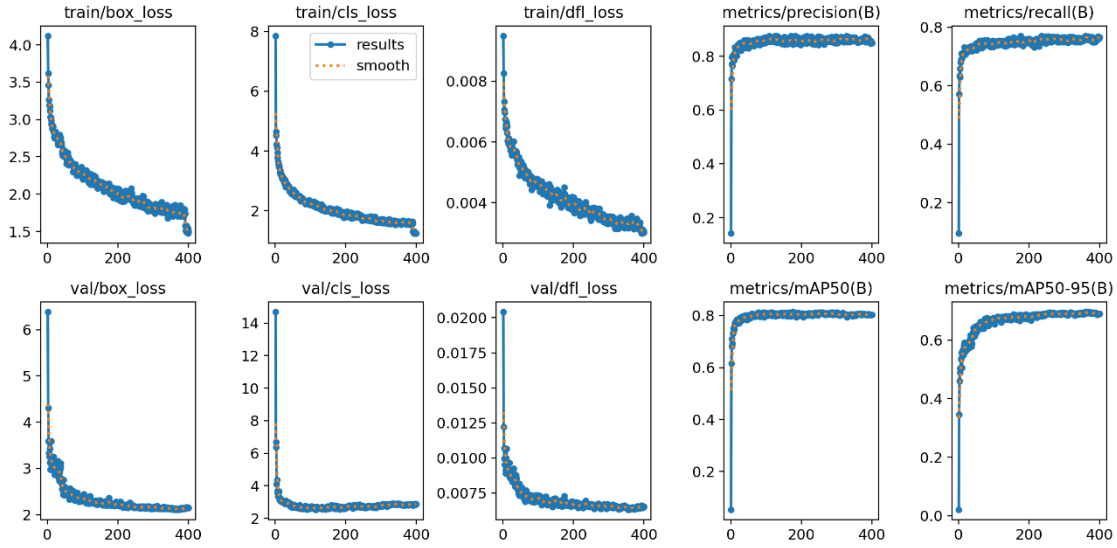
Este entrenamiento llevó 400/400 épocas durante un tiempo de 56 minutos y 53 segundos. Consumiendo en cada época 30.2 GB de RAM de la GPU A100.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    400/400      30.2G      1.494      1.243   0.003036        460        960: 100% ━━━━━━━━━━━━ 31/31 
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━
                   all        108       5090      0.851      0.766      0.804       0.69

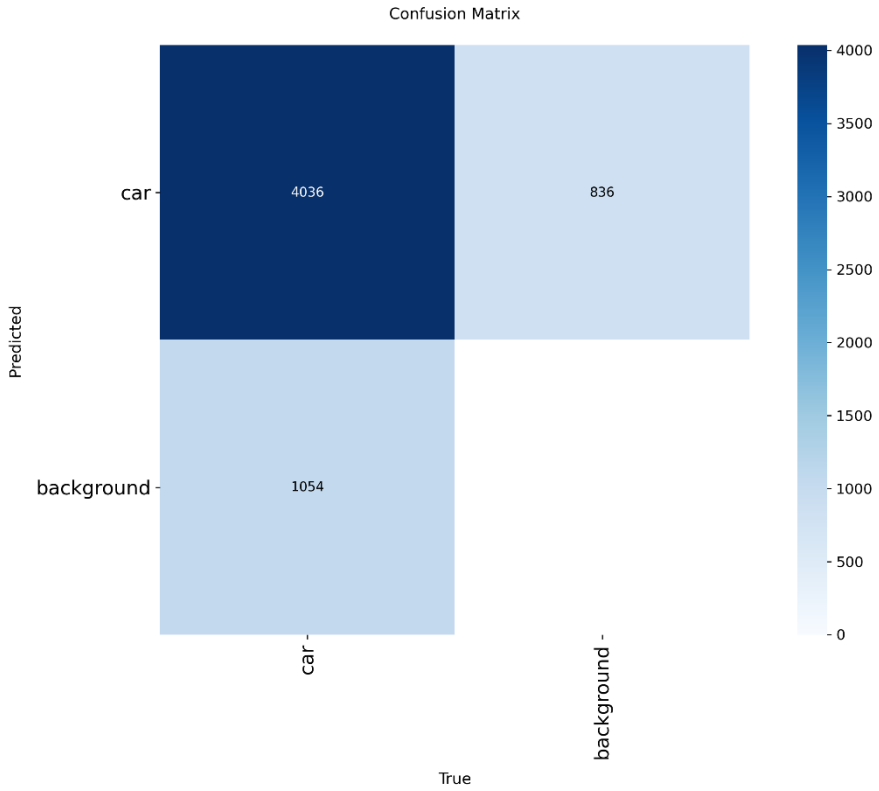
400 epochs completed in 0.948 hours.
```

El séptimo modelo replica la configuración del modelo 4 pero utilizando como semilla la arquitectura YOLO26s en lugar de YOLO11s, con el fin de comparar de forma justa el efecto de cambiar de backbone. El entrenamiento recorre las 400 épocas completas (400/400), lo que significa que, con patience=75, la métrica de validación sigue mejorando hasta el límite de épocas, aunque de forma cada vez más marginal.

Las curvas de pérdida de entrenamiento y validación muestran una evolución muy suave, con valores finales ligeramente inferiores a los del modelo 4, y las métricas de mAP50 y mAP50-95 se estabilizan alrededor de la época 250, con pequeñas oscilaciones posteriores. Tampoco se aprecia divergencia entre train y val, por lo que no hay evidencias claras de sobreajuste; simplemente el modelo sigue realizando mejoras pequeñas durante más tiempo.



La matriz de confusión del modelo 7 refleja un ligero incremento de verdaderos positivos y una pequeña reducción de falsos positivos respecto al modelo 4, mientras que los falsos negativos se mantienen en un rango similar. Esto se traduce en una precisión algo mayor y un recall comparable al del modelo 4, con un F1-score muy cercano. La tasa de falsos positivos es ligeramente menor, por lo que el modelo es algo más conservador.



Al desplegar el modelo en el ZONA PILOTO DE LA UEM, este modelo ofrece un comportamiento muy competitivo: mantiene una alta confianza en la mayoría de coches, mejora la detección de algunos vehículos lejanos y responde bien a oclusiones por árboles y ramas. Sin embargo, en determinadas vistas se observa que deja de detectar algún coche ocluido por otro vehículo y que, en escenas muy cargadas, pierde algunas detecciones de coches pequeños y lejanos con el paso del tiempo. Por tanto, el modelo 7 se interpreta como un trade-off respecto al modelo 4: ofrece ventajas puntuales en vehículos lejanos y ciertas oclusiones, pero su comportamiento es menos consistente en todas las condiciones. Por ello se mantiene como alternativa interesante, pero se selecciona el modelo 4 como “ganador clásico” del proyecto.

Anexo 3.9 Octavo modelo

Hiperparámetros del octavo modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=600 imgsz=960 batch=16 optimizer="AdamW"	lr0=0.005 lrf=0.05 warmup_epochs=2.0 patience=60 weight_decay=0.0005 cos_lr=True	box=20.0 cls=2.0 dfl=2.0 label_smoothing=0.03	mosaic=0.7 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.6 degrees=5 translate=0.1 scale=0.5 flipud=0.0 fliplr=0.5 close_mosaic=20

En el octavo modelo se parte de la configuración del modelo 7 y se explotan las opciones adicionales de YOLO26: se añaden weight_decay=0.0005 para introducir regularización L2, cos_lr=True para emplear un plan de tasa de aprendizaje cosenoidal y label_smoothing=0.03 para suavizar ligeramente las etiquetas de clase. El objetivo es mejorar la generalización y la calibración de las probabilidades, reduciendo la sobre-confianza del modelo 7. Además, se aumenta el límite de épocas a 600 y se reduce patience a 60, de modo que el modelo pueda seguir aprendiendo si hay mejoras reales pero el entrenamiento se corte cuando estas mejoras se vuelvan marginales. Se fija close_mosaic=20 para desactivar el *mosaic* en las últimas 20 épocas en caso de llegar al final del entrenamiento.

Este entrenamiento llevó 272/600 épocas durante un tiempo de 39 minutos y 47 segundos. Consumiendo en cada época 30.2 GB de RAM de la GPU A100.

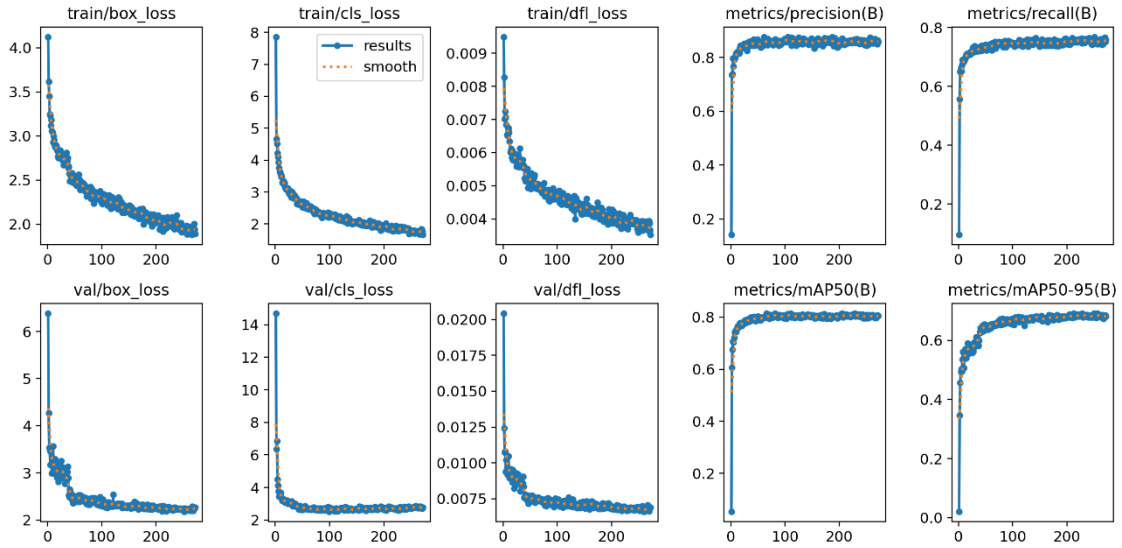
```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    272/600      30.2G       1.89      1.663   0.003521        445        960: 100% ━━━━━━━━━━━━ 31/31
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━
                   all        108       5090      0.858      0.758      0.805      0.682
EarlyStopping: Training stopped early as no improvement observed in last 60 epochs. Best results observ
To update EarlyStopping(patience=60) pass a new patience value, i.e. `patience=300` or use `patience=0`

272 epochs completed in 0.663 hours.
```

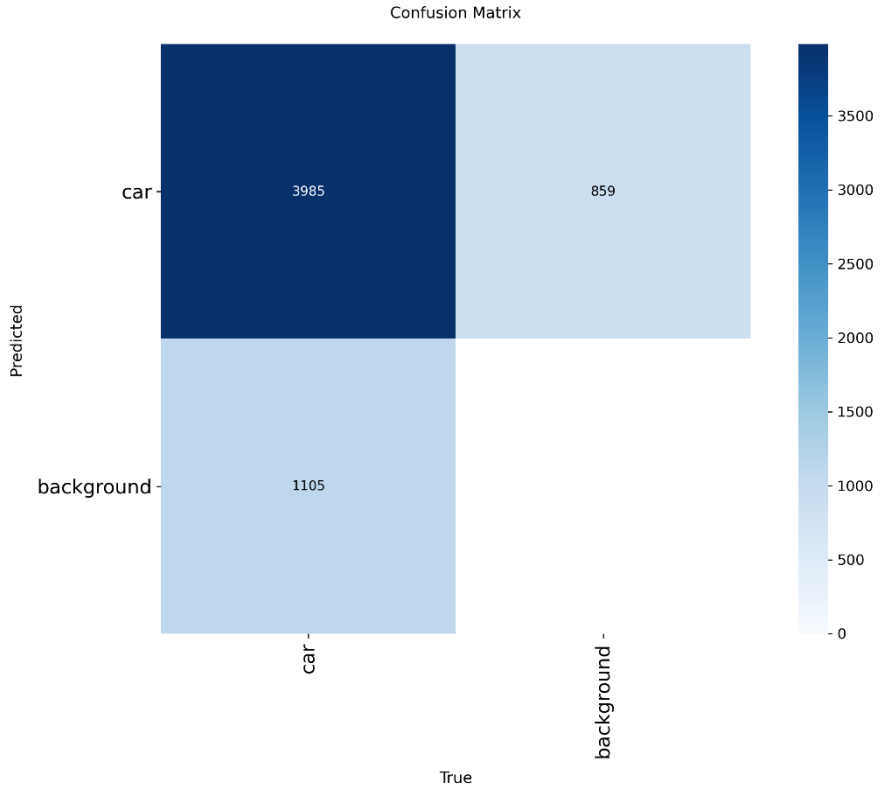
El octavo modelo incorpora las capacidades avanzadas de configuración de YOLO26: a la receta del modelo 7 se añaden weight_decay=0.0005 como regularización L2, cos_lr=True para emplear un scheduler cosenoidal y label_smoothing=0.03 (valor efectivo utilizado en código), manteniendo el resto de hiperparámetros constantes. El objetivo es mejorar la generalización y suavizar la excesiva confianza observada en algunos casos del modelo 7. Se aumenta el máximo

de épocas a 600 y se reduce patience a 60 para permitir más iteraciones útiles pero cortar cuando la mejora en validación deje de ser relevante.

El entrenamiento se detiene en la época 272/600 por early stopping. Las curvas de pérdida de entrenamiento y validación descienden de forma casi paralela y las métricas de mAP50 y mAP50-95 continúan mejorando hasta cerca de la época 200, con un plateau suave posterior. No se aprecia separación significativa entre train y val, por lo que el modelo parece bien regularizado; las mejoras más allá de la época 250 son pequeñas, coherentes con la activación del early stopping.



La matriz de confusión del modelo 8 muestra un perfil muy parecido al del modelo 7, con una ligera variación en falsos positivos y falsos negativos. A nivel agregado, precisión y recall siguen siendo altos, con un F1 similar al de los mejores modelos



Al desplegar el modelo en el ZONA PILOTO DE LA UEM se observa un comportamiento asimétrico. el modelo mejora claramente la detección de coches muy lejanos, incluso oscuros y con sombras pronunciadas, manteniendo buenas puntuaciones de confianza, pero pierde sensibilidad en coches oclusos y agrupados lateralmente. En algunas secuencias, coches que el modelo 4 y 7 detectaban con confianza razonable dejan de aparecer o lo hacen de forma intermitente.

La combinación de weight_decay, cos_lr y label_smoothing parece haber beneficiado la generalización en situaciones de baja resolución aparente (coches muy pequeños), pero ha penalizado la capacidad del modelo para activar detecciones en casos ambiguos de oclusión. Por esta razón, a pesar de sus buenas métricas agregadas, el modelo 8 se descarta como modelo final.

Anexo 3.10 Noveno modelo

Hiperparámetros del noveno modelo			
Base	**Aprendizaje**	**Pesos función de pérdida**	**Aumentación de datos**
epochs=500 imgsz=960 batch=16 optimizer="AdamW"	lr0=0.005 lrf=0.05 warmup_epochs=2.0 patience=60	box=20.0 cls=2.0 dfl=2.0	mosaic=0.7 mixup=0.05 hsv_h=0.015 hsv_s=0.7 hsv_v=0.6 degrees=5 translate=0.1 scale=0.5 flipud=0.0 fliplr=0.5

Debido a la limitación de unidades de cómputo disponibles y al hecho de que el modelo 4 ya ofrecía un rendimiento muy satisfactorio, se decidió no seguir afinando la configuración de YOLO26s y dedicar los recursos restantes a evaluar un modelo más ligero, YOLO11n. El noveno modelo reutiliza esencialmente la receta de entrenamiento del modelo 4 (tasa de aprendizaje, pesos de pérdida y aumentaciones), con epochs=500 y patience=60 para permitir un entrenamiento relativamente profundo pero con corte cuando la mejora en validación se estabilice.

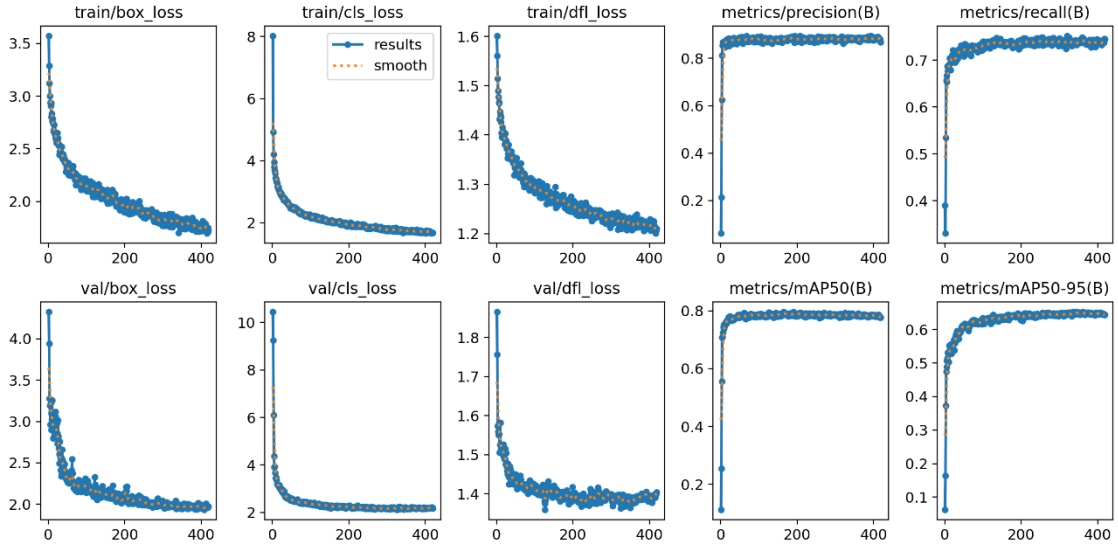
Este entrenamiento llevó 419/500 épocas durante un tiempo de 41 minutos y 56 segundos. Consumiendo en cada época 27.6 GB de RAM de la GPU A100.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    419/500      27.6G      1.728       1.68       1.21        653        960: 100% ━━━━━━━━━━━━ 31/31 
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━
                   all        108       5090       0.87      0.745      0.778      0.644
EarlyStopping: Training stopped early as no improvement observed in last 75 epochs. Best results observ
To update EarlyStopping(patience=75) pass a new patience value, i.e. `patience=300` or use `patience=0`

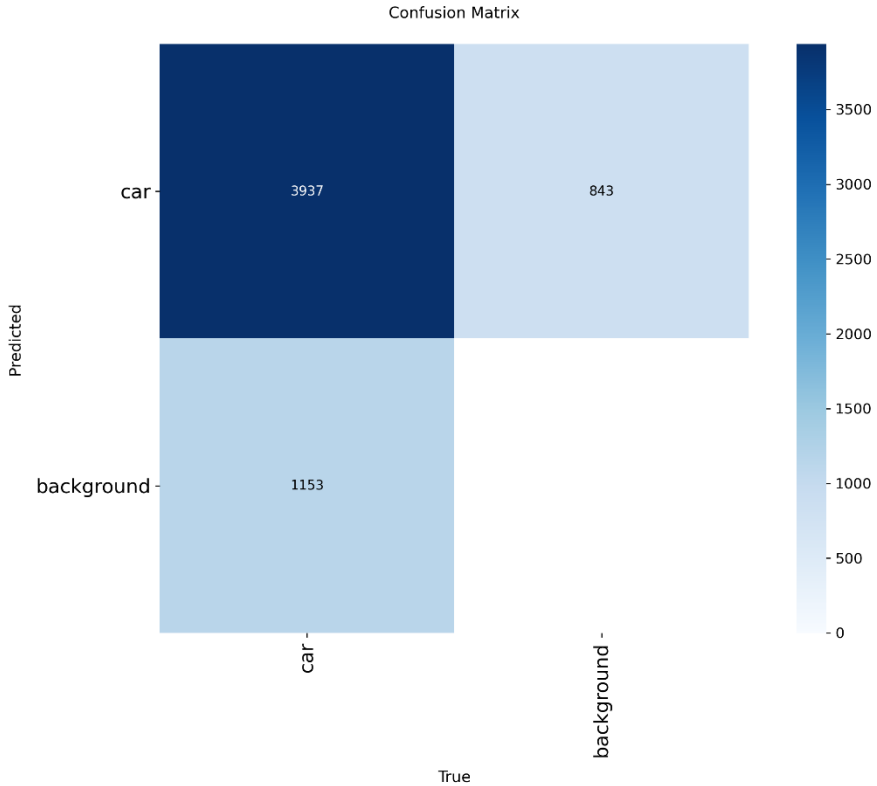
419 epochs completed in 0.699 hours.
```

El noveno modelo utiliza la arquitectura YOLO11n, más ligera, con el fin de evaluar si es posible mantener un rendimiento razonable con menor coste computacional. Se reutiliza la configuración de entrenamiento del modelo 4 con epochs=500 y patience=60, de forma que el early stopping corte el entrenamiento cuando la métrica de validación se estabilice.

El entrenamiento se detiene en la época 419/500, lo que indica que el modelo sigue mejorando lentamente durante buena parte del proceso. Las curvas de pérdida descienden de manera suave, pero los valores finales de val/box_loss, val/cls_loss y val/dfl_loss son peores que los obtenidos por los modelos 4, 7 y 8, y las métricas mAP50 y mAP50-95 se sitúan visiblemente por debajo de ellos. No se observan signos fuertes de sobreentrenamiento, pero la capacidad del modelo parece insuficiente para alcanzar el mismo nivel de ajuste que las arquitecturas más grandes.



La matriz de confusión del modelo 9 refleja un aumento de falsos negativos en comparación con el modelo 4 y 7, mientras que los falsos positivos se mantienen en un rango similar. En consecuencia, la precisión sigue siendo razonable, pero el recall disminuye, lo que provoca un F1 inferior. En términos prácticos, esto se traduce en un modelo que es relativamente fiable cuando ve un coche claro, pero deja sin detectar más vehículos, especialmente en situaciones difíciles.



Al desplegar el modelo en la zona piloto de la UEM se comprueba que el modelo 9 tiene problemas evidentes para detectar coches oclusos cercanos y coches agrupados en la zona lateral de la imagen, además de una caída en la detección de vehículos lejanos. Aunque su menor tamaño podría ser interesante para escenarios con restricciones de hardware, en el contexto de este proyecto el compromiso en precisión y recall no resulta aceptable frente al rendimiento alcanzado por el modelo 4. Por ello, el modelo 9 se considera un experimento exploratorio y no un candidato a modelo final.

ANEXO 4. CÓDIGO DE LA APLICACIÓN

Este anexo recoge el código principal del prototipo software, tal como se usó en la validación final. Se separa en dos archivos: la parte visual y la lógica de visión.

app_parking.py es la interfaz, hecha con CustomTkinter. Muestra el vídeo del aparcamiento, el estado de las plazas, los avisos de incidencias y el control de la barrera.

Parking_core.py es el funcionamiento interno, que carga las plazas desde el JSON, ejecuta el modelo YOLO, filtra las detecciones, asocia cada vehículo a su plaza, estabiliza el resultado en el tiempo y genera las incidencias.

No es un producto cerrado, sino la versión funcional que integra los módulos del trabajo y permite comprobar el sistema completo sobre los vídeos reales del aparcamiento de la UEM.

Anexo 4.1 Código de la interfaz de supervisión

```
#
================================================================================
=
#  app_parking.py
#  Ventana de supervision del sistema de aparcamiento (demostrador)
#
#  Trabajo de Fin de Grado
#  Diseno de un sistema de percepcion inteligente para la gestion eficiente
#  de aparcamientos al aire libre
#
#  Autor: Rafael del Castillo Shevelev
#  Universidad Europea de Madrid
#
#  Esta es la parte visual. Toda la vision esta en parking_core.py; aqui solo
#  se monta la ventana, se muestran el video y los contadores, y se dan los
#  botones al operario (parar la vision, cambiar el aforo, manejar la barrera).
#  La ventana le pregunta el estado al nucleo varias veces por segundo y lo
#  unico que hace es pintarlo.
#
================================================================================
=

# --- Librerias: customtkinter para la ventana, OpenCV y PIL para pasar el
#     video al formato de la interfaz, y el nucleo con toda la vision ---
from datetime import datetime
from pathlib import Path
```

```
from tkinter import messagebox, ttk
import customtkinter as ctk
import cv2
from PIL import Image
from parking_core import SistemaParking

#
================================================================================
=
#  Configuracion: donde estan el modelo, el video y las plazas, y con que
#  parametros se analiza. Cambiar de escenario es cambiar estas rutas.
#
================================================================================
=
PROJECT_ROOT = Path('C:\\Users\\Usuario\\Documents\\UEM\\TFG_parking')
MODEL_PATH = PROJECT_ROOT / 'models' / 'yolo' / 'yolo11s_960_v4.pt'
VIDEO_SOURCE = str(PROJECT_ROOT / 'data' / 'videos' / 'clips_uem_h264' /
'parking_facil_10_55_clip_final_h264.mp4') #'parking_facil_10_55_clip_final_h264.mp4'
JSON_PATH = PROJECT_ROOT / 'data' / 'annotations' / 'uem' / 'parking_facil_10m.json'
#parking_facil_10m.json
AUTHOR_NAME = 'Rafael del Castillo Shevelev'
PARKING_NAME = 'PARKING VETERINARIA · VISTA TRASERA'
DEVICE = 'cpu'
IMGSZ = 1920
CONF = 0.25
IOU = 0.6
SAMPLE_INTERVAL_S = 2.0

# --- Colores de la interfaz (los rojos y grises de la Universidad Europea) ---
RED = '#E30613'
DARK_RED = '#B0000A'
SOFT_RED = '#FCE8EA'
WHITE = '#FFFFFF'
LIGHT_GRAY = '#F2F2F2'
MEDIUM_GRAY = '#D2D2D2'
DARK_GRAY = '#555555'
BLACK = '#171717'
GREEN = '#2E9B5F'
DARK_GREEN = '#237849'
YELLOW = '#FFF1B3'

#
================================================================================
=
#  Ventana principal
#
#  Crea el sistema de vision, monta toda la interfaz y arranca dos tareas que
#  se repiten solas: una pone en marcha el sistema y otra refresca la ventana
#  (video, contadores y tabla de avisos) leyendo el estado del nucleo.
```

```
# Los metodos build_* solo construyen trozos de la ventana; los make_* son
# ayudas para crear tarjetas y contadores repetidos.
#
============================================================================
=
class ParkingApp(ctk.CTk):
    """Interfaz principal del demostrador."""

    def __init__(self):
        super().__init__()
        ctk.set_appearance_mode('light')
        self.title('Control del aparcamiento UEM')
        self.geometry('1400x850')
        self.minsize(1120, 720)
        self.configure(fg_color=LIGHT_GRAY)
        self.system = SistemaParking(MODEL_PATH, VIDEO_SOURCE, JSON_PATH, device=DEVICE,
imgsz=IMGSZ, conf=CONF, iou=IOU, sample_interval_s=SAMPLE_INTERVAL_S)
        self.video_image = None
        self.start_error = ''
        self.slider_ready = False
        self.last_alerts = None
        self.configure_table()
        self.build_interface()
        self.protocol('WM_DELETE_WINDOW', self.close_app)
        self.after(300, self.start_system)
        self.after(300, self.update_interface)

    def build_interface(self):
        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure(1, weight=1)
        self.build_header()
        main = ctk.CTkFrame(self, fg_color='transparent')
        main.grid(row=1, column=0, padx=14, pady=(10, 8), sticky='nsew')
        main.grid_columnconfigure(0, weight=7, uniform='main')
        main.grid_columnconfigure(1, weight=4, uniform='main')
        main.grid_rowconfigure(0, weight=1)
        self.build_video(main)
        self.build_controls(main)
        self.build_alerts()

    def build_header(self):
        container = ctk.CTkFrame(self, fg_color=WHITE, corner_radius=0)
        container.grid(row=0, column=0, sticky='ew')
        container.grid_columnconfigure(0, weight=1)
        header = ctk.CTkFrame(container, fg_color=WHITE, corner_radius=0)
        header.grid(row=0, column=0, sticky='ew')
        header.grid_columnconfigure(1, weight=1)
        logo_path = Path(__file__).parent / 'ue_logo.png'
        logo_bgr = cv2.imread(str(logo_path))
```

```
        if logo_bgr is None:
            raise FileNotFoundError(f'No se pudo leer el logo:\n{logo_path}')
        logo = Image.fromarray(cv2.cvtColor(logo_bgr, cv2.COLOR_BGR2RGB))
        self.logo_image = ctk.CTkImage(light_image=logo, dark_image=logo, size=(56, 56))
        ctk.CTkLabel(header, image=self.logo_image, text='').grid(row=0, column=0, padx=(20,
12), pady=9)
        titles = ctk.CTkFrame(header, fg_color='transparent')
        titles.grid(row=0, column=1, sticky='w')
        ctk.CTkLabel(titles, text='CONTROL DEL APARCAMIENTO', text_color=BLACK,
font=ctk.CTkFont(size=23, weight='bold')).pack(anchor='w')
        ctk.CTkLabel(titles, text=AUTHOR_NAME, text_color=RED, font=ctk.CTkFont(size=13,
weight='bold')).pack(anchor='w')
        self.clock_label = ctk.CTkLabel(header, text='', text_color=DARK_GRAY,
font=ctk.CTkFont(size=13))
        self.clock_label.grid(row=0, column=2, padx=15)
        self.parking_state_label = ctk.CTkLabel(header, text='INICIALIZANDO', width=145,
height=38, fg_color=DARK_GRAY, text_color=WHITE, corner_radius=7,
font=ctk.CTkFont(size=14, weight='bold'))
        self.parking_state_label.grid(row=0, column=3, padx=(0, 20))
        ctk.CTkFrame(container, fg_color=RED, height=5, corner_radius=0).grid(row=1, column=0,
sticky='ew')

    def build_video(self, parent):
        card = self.make_card(parent)
        card.grid(row=0, column=0, sticky='nsew')
        card.grid_columnconfigure(0, weight=1)
        card.grid_rowconfigure(1, weight=1)
        top = ctk.CTkFrame(card, fg_color=SOFT_RED, corner_radius=6)
        top.grid(row=0, column=0, padx=10, pady=(10, 7), sticky='ew')
        top.grid_columnconfigure(0, weight=1)
        ctk.CTkLabel(top, text=PARKING_NAME, text_color=RED, font=ctk.CTkFont(size=15,
weight='bold')).grid(row=0, column=0, padx=10, pady=7, sticky='w')
        self.connection_label = ctk.CTkLabel(top, text='● INICIALIZANDO', text_color=DARK_GRAY,
font=ctk.CTkFont(size=11, weight='bold'))
        self.connection_label.grid(row=0, column=1, padx=10, sticky='e')
        self.video_frame = ctk.CTkFrame(card, fg_color=BLACK, corner_radius=7)
        self.video_frame.grid(row=1, column=0, padx=10, pady=(0, 10), sticky='nsew')
        self.video_frame.grid_columnconfigure(0, weight=1)
        self.video_frame.grid_rowconfigure(0, weight=1)
        self.video_label = ctk.CTkLabel(self.video_frame, text='Inicializando modelo y conectando
con la cámara...', text_color=WHITE, fg_color=BLACK, font=ctk.CTkFont(size=14))
        self.video_label.grid(row=0, column=0, sticky='nsew')

    def build_controls(self, parent):
        panel = self.make_card(parent)
        panel.grid(row=0, column=1, padx=(10, 0), sticky='nsew')
        panel.grid_columnconfigure(0, weight=1)
        self.build_summary(panel)
        self.build_system_status(panel)
```

```
        self.build_access(panel)

    def build_summary(self, parent):
        heading = ctk.CTkFrame(parent, fg_color=SOFT_RED, corner_radius=6)
        heading.grid(row=0, column=0, padx=9, pady=(9, 5), sticky='ew')
        heading.grid_columnconfigure(0, weight=1)
        ctk.CTkLabel(heading, text='ESTADO DEL APARCAMIENTO', text_color=RED,
font=ctk.CTkFont(size=13, weight='bold')).grid(row=0, column=0, padx=(10, 5), pady=8,
sticky='w')
        self.percentage_label = ctk.CTkLabel(heading, text='-- %', width=54, text_color=RED,
font=ctk.CTkFont(size=14, weight='bold'))
        self.percentage_label.grid(row=0, column=1, padx=(5, 10), pady=8, sticky='e')
        self.occupancy_bar = ctk.CTkProgressBar(parent, height=6, fg_color=MEDIUM_GRAY,
progress_color=RED)
        self.occupancy_bar.grid(row=1, column=0, padx=11, pady=(0, 5), sticky='ew')
        self.occupancy_bar.set(0)
        counters = ctk.CTkFrame(parent, fg_color='transparent')
        counters.grid(row=2, column=0, padx=6, pady=(0, 2), sticky='ew')
        for column in range(4):
            counters.grid_columnconfigure(column, weight=1)
        self.total_label = self.make_counter(counters, 'TOTAL', 0)
        self.occupied_label = self.make_counter(counters, 'OCUPADAS', 1)
        self.free_label = self.make_counter(counters, 'LIBRES', 2)
        self.alert_count_label = self.make_counter(counters, 'AVISOS', 3)

    def build_system_status(self, parent):
        section = self.make_section(parent, 3, 'Estado del sistema')
        row = ctk.CTkFrame(section, fg_color='transparent')
        row.grid(row=1, column=0, columnspan=2, padx=7, pady=(0, 5), sticky='ew')
        row.grid_columnconfigure(0, weight=1)
        row.grid_columnconfigure(1, weight=1)
        self.vision_status = self.make_status(row, 'VISIÓN', 0)
        self.barrier_status = self.make_status(row, 'BARRERA', 1)
        self.vision_button = ctk.CTkButton(section, text='Detener visión',
command=self.toggle_vision, height=30, fg_color=DARK_GRAY, hover_color=BLACK)
        self.vision_button.grid(row=2, column=0, columnspan=2, padx=9, pady=(2, 8), sticky='ew')

    def build_access(self, parent):
        section = self.make_section(parent, 4, 'Control del acceso')
        ctk.CTkLabel(section, text='BARRERA', text_color=BLACK, font=ctk.CTkFont(size=11,
weight='bold')).grid(row=1, column=0, columnspan=2, padx=9, pady=(0, 3), sticky='w')
        self.barrier_mode = ctk.CTkSegmentedButton(section, values=['AUTOMÁTICO',
'MANUAL'], command=self.change_barrier_mode, height=30, selected_color=RED,
selected_hover_color=DARK_RED, unselected_color=LIGHT_GRAY,
unselected_hover_color=MEDIUM_GRAY, text_color=BLACK)
        self.barrier_mode.grid(row=2, column=0, columnspan=2, padx=9, pady=(0, 7), sticky='ew')
        self.barrier_mode.set('AUTOMÁTICO')
        limit_row = ctk.CTkFrame(section, fg_color='transparent')
        limit_row.grid(row=3, column=0, columnspan=2, padx=9, sticky='ew')
```

```
        limit_row.grid_columnconfigure(0, weight=1)
        ctk.CTkLabel(limit_row, text='Límite de ocupación', text_color=BLACK,
font=ctk.CTkFont(size=11, weight='bold')).grid(row=0, column=0, sticky='w')
        self.threshold_label = ctk.CTkLabel(limit_row, text='Esperando plazas...', text_color=RED,
font=ctk.CTkFont(size=11, weight='bold'))
        self.threshold_label.grid(row=0, column=1, sticky='e')
        self.threshold_slider = ctk.CTkSlider(section, from_=0, to=1, number_of_steps=1,
command=self.preview_threshold, height=14, button_color=RED,
button_hover_color=DARK_RED, progress_color=RED, fg_color=MEDIUM_GRAY)
        self.threshold_slider.grid(row=4, column=0, columnspan=2, padx=11, pady=(5, 4),
sticky='ew')
        self.threshold_slider.set(0)
        ctk.CTkButton(section, text='Aplicar límite', command=self.apply_threshold, height=29,
fg_color=RED, hover_color=DARK_RED).grid(row=5, column=0, columnspan=2, padx=9,
pady=(0, 7), sticky='ew')
        manual = ctk.CTkFrame(section, fg_color='transparent')
        manual.grid(row=6, column=0, columnspan=2, padx=9, pady=(0, 4), sticky='ew')
        manual.grid_columnconfigure(0, weight=1)
        manual.grid_columnconfigure(1, weight=1)
        self.open_button = ctk.CTkButton(manual, text='Abrir barrera',
command=self.system.open_barrier, height=31, fg_color=GREEN, hover_color=DARK_GREEN,
text_color=WHITE)
        self.open_button.grid(row=0, column=0, padx=(0, 4), sticky='ew')
        self.close_button = ctk.CTkButton(manual, text='Cerrar barrera',
command=self.system.close_barrier, height=31, fg_color=RED, hover_color=DARK_RED,
text_color=WHITE)
        self.close_button.grid(row=0, column=1, padx=(4, 0), sticky='ew')
        self.manual_help = ctk.CTkLabel(section, text='Selecciona MANUAL para usar los
botones.', text_color=DARK_GRAY, font=ctk.CTkFont(size=9))
        self.manual_help.grid(row=7, column=0, columnspan=2, padx=9, pady=(0, 7))
        self.set_manual_enabled(False)

    def build_alerts(self):
        card = self.make_card(self)
        card.grid(row=2, column=0, padx=14, pady=(0, 12), sticky='ew')
        card.grid_columnconfigure(0, weight=1)
        top = ctk.CTkFrame(card, fg_color='transparent')
        top.grid(row=0, column=0, columnspan=2, padx=10, pady=(7, 4), sticky='ew')
        top.grid_columnconfigure(0, weight=1)
        ctk.CTkLabel(top, text='AVISOS REGISTRADOS', text_color=RED, font=ctk.CTkFont(size=14,
weight='bold')).grid(row=0, column=0, sticky='w')
        ctk.CTkButton(top, text='Finalizar aviso seleccionado', command=self.finish_alert,
height=29, fg_color=RED, hover_color=DARK_RED, width=215).grid(row=0, column=1,
sticky='e')
        columns = ('id', 'type', 'spaces', 'start', 'end', 'status')
        self.alert_table = ttk.Treeview(card, columns=columns, show='headings', height=4)
        settings = {'id': ('ID', 60), 'type': ('Tipo', 260), 'spaces': ('Plazas', 130), 'start': ('Inicio', 90),
'end': ('Fin', 90), 'status': ('Estado', 100)}
        for column, (title, width) in settings.items():
```

```
            self.alert_table.heading(column, text=title)
            self.alert_table.column(column, width=width, anchor='center')
        self.alert_table.tag_configure('active', background=YELLOW)
        scrollbar = ttk.Scrollbar(card, orient='vertical', command=self.alert_table.yview)
        self.alert_table.configure(yscrollcommand=scrollbar.set)
        self.alert_table.grid(row=1, column=0, padx=(10, 0), pady=(0, 9), sticky='ew')
        scrollbar.grid(row=1, column=1, padx=(3, 10), pady=(0, 9), sticky='ns')

    # arranca el nucleo de vision y guarda el error si algo falla
    def start_system(self):
        try:
            self.system.start()
        except Exception as error:
            self.start_error = str(error)
            messagebox.showerror('Error de inicio', self.start_error)

    def toggle_vision(self):
        self.system.set_vision_enabled(not self.system.vision_activa)

    def preview_threshold(self, value):
        total = len(self.system.plazas)
        if total:
            self.threshold_label.configure(text=f'{int(round(value))} de {total}')

    def apply_threshold(self):
        try:
            self.system.set_full_threshold(int(round(self.threshold_slider.get())))
        except ValueError as error:
            messagebox.showwarning('Límite no válido', str(error))

    def change_barrier_mode(self, value):
        manual = value == 'MANUAL'
        self.system.set_barrier_mode('MANUAL' if manual else 'AUTO')
        self.set_manual_enabled(manual)

    def finish_alert(self):
        selection = self.alert_table.selection()
        if not selection:
            messagebox.showinfo('Avisos', 'Selecciona primero un aviso.')
            return
        alert_id = self.alert_table.item(selection[0], 'values')[0]
        self.system.finish_alert(alert_id)

    # se llama sola cada poco: pide el estado al nucleo y actualiza la pantalla
    def update_interface(self):
        # se pide al nucleo el estado actual (plazas, video, avisos, barrera...)
        snapshot = self.system.get_snapshot()
        # si hubo error al arrancar, se refleja en el estado
        if self.start_error:
```

```
            snapshot['camera_status'] = 'ERROR'
            snapshot['last_error'] = self.start_error
        # se actualiza el reloj y cada parte de la ventana con ese estado
        self.clock_label.configure(text=datetime.now().strftime('%d/%m/%Y  %H:%M:%S'))
        self.update_values(snapshot)
        self.update_status(snapshot)
        self.update_video(snapshot)
        self.update_alert_table(snapshot)
        # se vuelve a llamar sola dentro de 150 ms (refresco continuo)
        self.after(150, self.update_interface)

    def update_values(self, snapshot):
        # contadores de plazas y avisos que se muestran arriba
        total = snapshot['total_spaces']
        vision_enabled = snapshot['vision_enabled']
        ready = snapshot['analysis_ready']
        self.total_label.configure(text=str(total))
        self.alert_count_label.configure(text=str(snapshot['active_alerts']))
        # si la vision esta parada, se muestran guiones en vez de numeros
        if not vision_enabled:
            self.occupied_label.configure(text='—')
            self.free_label.configure(text='—')
            self.percentage_label.configure(text='-- %')
            self.occupancy_bar.set(0)
            self.parking_state_label.configure(text='VISIÓN DETENIDA', fg_color=RED)
            return
        if not ready:
            self.occupied_label.configure(text='—')
            self.free_label.configure(text='—')
            self.percentage_label.configure(text='-- %')
            self.occupancy_bar.set(0)
            self.parking_state_label.configure(text='ANALIZANDO', fg_color=DARK_GRAY)
            return
        # con analisis listo, se calcula el porcentaje de ocupacion y se pinta
        occupied = snapshot['occupied_spaces']
        percentage = occupied / total if total else 0
        self.occupied_label.configure(text=str(occupied))
        self.free_label.configure(text=str(snapshot['free_spaces']))
        self.percentage_label.configure(text=f'{percentage * 100:.0f} %')
        self.occupancy_bar.set(percentage)
        self.parking_state_label.configure(text='COMPLETO' if snapshot['parking_full'] else
'DISPONIBLE', fg_color=RED if snapshot['parking_full'] else DARK_GRAY)

    def update_status(self, snapshot):
        camera = snapshot['camera_status']
        camera_color = GREEN if camera == 'CONECTADA' else RED if camera == 'ERROR' else
DARK_GRAY
        self.connection_label.configure(text=f'● {camera}', text_color=camera_color)
        vision_enabled = snapshot['vision_enabled']
```

```
        self.vision_status.configure(text='ACTIVA' if vision_enabled else 'DETENIDA',
fg_color=GREEN if vision_enabled else RED)
        self.vision_button.configure(text='Detener visión' if vision_enabled else 'Activar visión')
        barrier_open = snapshot['barrier_state'] == 'ABIERTA'
        self.barrier_status.configure(text=snapshot['barrier_state'], fg_color=GREEN if
barrier_open else RED)
        manual = snapshot['barrier_mode'] == 'MANUAL'
        desired_mode = 'MANUAL' if manual else 'AUTOMÁTICO'
        if self.barrier_mode.get() != desired_mode:
            self.barrier_mode.set(desired_mode)
        self.set_manual_enabled(manual)
        total = snapshot['total_spaces']
        if not self.slider_ready and total:
            threshold = snapshot['full_threshold']
            if total > 1:
                self.threshold_slider.configure(from_=1, to=total, number_of_steps=total - 1,
state='normal')
                self.threshold_slider.set(threshold)
            else:
                self.threshold_slider.configure(from_=0, to=1, number_of_steps=1, state='disabled')
                self.threshold_slider.set(1)
            self.threshold_label.configure(text=f'{threshold} de {total}')
            self.slider_ready = True

    def update_video(self, snapshot):
        frame = snapshot['frame']
        if frame is None:
            camera = snapshot['camera_status']
            if camera == 'ERROR':
                text = snapshot['last_error'] or 'Error al abrir la fuente.'
            elif camera == 'CONECTADA':
                text = 'Cámara conectada. Esperando el primer frame...'
            else:
                text = 'Inicializando modelo y conectando con la cámara...'
            self.video_label.configure(image=None, text=text)
            return
        image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        width = max(self.video_frame.winfo_width() - 18, 560)
        height = max(self.video_frame.winfo_height() - 18, 340)
        resampling = getattr(Image, 'Resampling', Image)
        image.thumbnail((width, height), resampling.LANCZOS)
        self.video_image = ctk.CTkImage(light_image=image, dark_image=image, size=image.size)
        self.video_label.configure(image=self.video_image, text='')

    def update_alert_table(self, snapshot):
        alerts = [(alert['id'], alert['type'], alert['spaces'], alert['start_time'], alert['end_time'],
alert['status']) for alert in snapshot['incidents']]
        if tuple(alerts) == self.last_alerts:
            return
```

```
        self.alert_table.delete(*self.alert_table.get_children())
        for alert in alerts:
            tags = ('active',) if alert[-1] == 'ACTIVA' else ()
            self.alert_table.insert('', 'end', values=alert, tags=tags)
        self.last_alerts = tuple(alerts)

    def set_manual_enabled(self, enabled):
        if enabled:
            self.open_button.configure(state='normal', fg_color=GREEN,
hover_color=DARK_GREEN, text_color=WHITE)
            self.close_button.configure(state='normal', fg_color=RED, hover_color=DARK_RED,
text_color=WHITE)
            self.manual_help.configure(text='Control manual activo.', text_color=RED)
        else:
            for button in (self.open_button, self.close_button):
                button.configure(state='disabled', fg_color=MEDIUM_GRAY,
hover_color=MEDIUM_GRAY, text_color=DARK_GRAY)
            self.manual_help.configure(text='Selecciona MANUAL para usar los botones.',
text_color=DARK_GRAY)

    def make_card(self, parent):
        return ctk.CTkFrame(parent, fg_color=WHITE, corner_radius=8, border_width=1,
border_color=MEDIUM_GRAY)

    def make_counter(self, parent, title, column):
        card = self.make_card(parent)
        card.grid(row=0, column=column, padx=3, pady=3, sticky='ew')
        ctk.CTkLabel(card, text=title, text_color=RED, font=ctk.CTkFont(size=9,
weight='bold')).pack(pady=(5, 0))
        value = ctk.CTkLabel(card, text='0', text_color=BLACK, font=ctk.CTkFont(size=19,
weight='bold'))
        value.pack(pady=(0, 5))
        return value

    def make_section(self, parent, row, title):
        section = self.make_card(parent)
        section.grid(row=row, column=0, padx=9, pady=4, sticky='ew')
        section.grid_columnconfigure(1, weight=1)
        ctk.CTkLabel(section, text=title.upper(), text_color=RED, font=ctk.CTkFont(size=11,
weight='bold')).grid(row=0, column=0, columnspan=2, padx=9, pady=(7, 4), sticky='w')
        return section

    def make_status(self, parent, title, column):
        card = ctk.CTkFrame(parent, fg_color=LIGHT_GRAY, corner_radius=7)
        card.grid(row=0, column=column, padx=3, sticky='ew')
        ctk.CTkLabel(card, text=title, text_color=DARK_GRAY, font=ctk.CTkFont(size=9,
weight='bold')).pack(pady=(5, 2))
        value = ctk.CTkLabel(card, text='-', height=24, fg_color=DARK_GRAY, text_color=WHITE,
corner_radius=6, font=ctk.CTkFont(size=10, weight='bold'))
```

```
        value.pack(padx=6, pady=(0, 6), fill='x')
        return value

    def configure_table(self):
        style = ttk.Style()
        style.theme_use('default')
        style.configure('Treeview', background=WHITE, foreground=BLACK,
fieldbackground=WHITE, rowheight=27, borderwidth=0, font=('Segoe UI', 9))
        style.configure('Treeview.Heading', background=RED, foreground=WHITE, relief='flat',
font=('Segoe UI', 9, 'bold'))
        style.map('Treeview.Heading', background=[('active', DARK_RED)])
        style.map('Treeview', background=[('selected', RED)], foreground=[('selected', WHITE)])

    def close_app(self):
        self.system.stop()
        self.destroy()

# --- Punto de entrada: se crea la ventana y se lanza ---
if __name__ == '__main__':
    ParkingApp().mainloop()
```

Anexo 4.2 Código del sistema de visión y lógica interna

```
#
# ============================================================================
# parking_core.py
# Nucleo de vision y logica de ocupacion del sistema de aparcamiento
#
# Trabajo de Fin de Grado
# Diseno de un sistema de percepcion inteligente para la gestion eficiente
# de aparcamientos al aire libre
#
# Autor: Rafael del Castillo Shevelev
# Universidad Europea de Madrid
#
# Aqui esta toda la parte que no es la ventana: cargar las plazas, detectar
# los coches con YOLO, decidir en que plaza esta cada uno, estabilizar el
# resultado y avisar de un coche que ocupa dos plazas. La ventana
# (app_parking.py) solo lee los resultados que salen de aqui.
#
# ============================================================================

# --- Librerias que uso ---
from datetime import datetime
from pathlib import Path
from threading import Lock, Thread
import json
import time
import cv2
import numpy as np
from ultralytics import YOLO

#
# ============================================================================
# 1. Valores de configuracion
#
# Junto aqui todos los numeros que ajustan el sistema, para no tenerlos
# sueltos por el codigo. Las posiciones van en proporcion (de 0 a 1) sobre la
# caja del coche, no en pixeles, para que valgan igual sea grande o pequena.
#
# ============================================================================

# Zona de la caja que uso como "huella" del coche (la parte de abajo, que es
```

```
# la que toca el suelo). Va de 12% a 88% de ancho y de 52% a 100% de alto.
HUELLA_X = (0.12, 0.88)
HUELLA_Y = (0.52, 1.0)

# Cuanto se tiene que solapar la huella con una plaza para darla por ocupada.
SOLAPE_MINIMO = 0.3          # solape principal
SOLAPE_SECUNDARIO_MINIMO = 0.2  # para detectar la segunda plaza invadida
PUNTOS_SECUNDARIA_MINIMO = 9
PUNTOS_RESPALDO_MINIMO = 3

# Rejilla de puntos que reparto sobre el coche cuando el solape no basta y
# tengo que decidir la plaza contando puntos que caen dentro.
REJILLA_X = (0.18, 0.34, 0.5, 0.66, 0.82)
REJILLA_Y_NORMAL = (0.32, 0.5, 0.68, 0.86)
REJILLA_Y_PEQUENA = (0.5, 0.64, 0.78, 0.92)
PUNTOS_RESPALDO = ((0.42, 0.74), (0.58, 0.74), (0.35, 0.88), (0.45, 0.88), (0.55, 0.88), (0.65,
0.88))

# Filtro por tamano de caja. Descarto las cajas imposibles de grandes para ser
# un coche en la escena y pido mas confianza a las grandes. Esto quita las
# cajas falsas que aparecian sobre el fondo despues del fine tuning.
AREA_CAJA_PEQUENA = 0.0015
AREA_CAJA_GRANDE = 0.01
CONFIANZA_MINIMA_CAJA_GRANDE = 0.75
AREA_CAJA_MAXIMA = 0.05

# Estabilizacion en el tiempo. No me creo un cambio hasta verlo varias veces
# seguidas. Cuesta mas liberar (5) que ocupar (3) para no marcar libre una
# plaza por un fallo suelto, que es el error mas molesto para guiar al conductor.
CONFIRMACIONES_PARA_OCUPAR = 3
CONFIRMACIONES_PARA_LIBERAR = 5

# Lo mismo para la incidencia de doble plaza: varias veces seguidas para
# confirmarla y para cerrarla, y asi no dar falsas alarmas.
CONFIRMACIONES_INCIDENCIA = 6
CONFIRMACIONES_CIERRE_INCIDENCIA = 5

# Cuantas veces seguidas tiene que verse el parking lleno para darlo por lleno.
CONFIRMACIONES_LLENO = 2

# Colores para pintar las plazas (en formato BGR, que es el que usa OpenCV).
COLOR_LIBRE = (0, 170, 0)
COLOR_OCUPADA = (0, 0, 220)
COLOR_ALERTA = (0, 210, 255)
COLOR_TEXTO = (255, 255, 255)

#
==========================================================================
=
```

```
# 2. Cargar las plazas
#
# Las plazas no estan en el codigo, estan en un archivo JSON con el poligono
# de cada una. Para cambiar de aparcamiento solo cambio ese archivo.
#
================================================================================
=

def cargar_plazas(ruta_json):
    """Lee las plazas del archivo JSON y las deja listas para usar."""
    # se lee el archivo y, si viene envuelto en un diccionario, se busca la lista
    datos = json.loads(Path(ruta_json).read_text(encoding='utf-8'))
    if isinstance(datos, dict):
        for clave in ('regions', 'parking_spaces', 'spaces', 'rois', 'slots'):
            if isinstance(datos.get(clave), list):
                datos = datos[clave]
                break
    if not isinstance(datos, list):
        raise ValueError('El JSON debe contener una lista de plazas.')

    # se recorre cada plaza y se guardan sus puntos, su area y su centro
    plazas = []
    for indice, elemento in enumerate(datos, start=1):
        puntos_brutos = elemento.get('points') or elemento.get('polygon') or elemento.get('pts')
        if puntos_brutos is None:
            continue
        puntos = np.asarray(puntos_brutos, dtype=np.int32).reshape((-1, 1, 2))
        if len(puntos) < 3:  # con menos de 3 puntos no hay poligono
            continue
        contorno = cv2.convexHull(puntos.astype(np.float32))
        area = abs(float(cv2.contourArea(contorno)))
        if area > 1:
            plazas.append({
                'id': str(elemento.get('id', elemento.get('plaza_id', f'P{indice:03d}'))),
                'points': puntos,
                'hull': contorno,
                'area': area,
                'center': puntos.reshape(-1, 2).mean(axis=0),
            })

    if not plazas:
        raise RuntimeError('No se ha cargado ninguna plaza.')
    return plazas

#
================================================================================
=
# 3. Detectar coches y filtrar cajas imposibles
#
```

```
# YOLO devuelve muchas cajas. Aqui las paso a una lista sencilla y de paso
# quito las que por tamano no pueden ser un coche real en la escena.
#
================================================================================

def sacar_detecciones(resultado, imagen):
    """Convierte las cajas de YOLO en una lista simple y filtra por tamano."""
    if resultado.boxes is None or len(resultado.boxes) == 0:
        return []
    # se sacan las cajas y sus confianzas del resultado de YOLO
    area_imagen = imagen.shape[0] * imagen.shape[1]
    cajas = resultado.boxes.xyxy.cpu().numpy()
    confianzas = resultado.boxes.conf.cpu().numpy()

    detecciones = []
    for id_deteccion, (caja, confianza) in enumerate(zip(cajas, confianzas)):
        # esquinas de la caja y su tamano en proporcion a la imagen
        x1, y1, x2, y2 = map(float, caja[:4])
        area_relativa = max(x2 - x1, 0) * max(y2 - y1, 0) / max(area_imagen, 1)
        # se descarta si es demasiado grande para ser un coche en la escena
        if area_relativa > AREA_CAJA_MAXIMA:
            continue
        # a las cajas grandes se les pide mas confianza (asi caen las falsas del fondo)
                        if area_relativa > AREA_CAJA_GRANDE and confianza <
CONFIANZA_MINIMA_CAJA_GRANDE:
            continue
        # la caja pasa el filtro: se guarda como una deteccion valida
         detecciones.append({'id': id_deteccion, 'x1': x1, 'y1': y1, 'x2': x2, 'y2': y2, 'relative_area':
area_relativa})
    return detecciones

#
================================================================================
=
# 4. Relacionar cada coche con su plaza
#
# Esta es la parte importante: no basta con ver el coche, hay que saber en
# que plaza esta. Uso la huella inferior (la parte de abajo del coche) en vez
# del centro, porque en perspectiva el centro cae sobre la plaza de detras.
#
================================================================================
=

def hacer_huella(deteccion):
    """Devuelve la huella inferior del coche, que es lo que comparo con la plaza."""
    ancho = max(deteccion['x2'] - deteccion['x1'], 1)
    alto = max(deteccion['y2'] - deteccion['y1'], 1)
    x1 = deteccion['x1'] + HUELLA_X[0] * ancho
```

```
    x2 = deteccion['x1'] + HUELLA_X[1] * ancho
    y1 = deteccion['y1'] + HUELLA_Y[0] * alto
    y2 = deteccion['y1'] + HUELLA_Y[1] * alto
    return np.asarray([[x1, y1], [x2, y1], [x2, y2], [x1, y2]], dtype=np.float32).reshape((-1, 1, 2))

def ratio_solape(huella, plaza):
    """Cuanto se solapa la huella del coche con una plaza (de 0 a 1)."""
    try:
        area, _ = cv2.intersectConvexConvex(huella, plaza['hull'].astype(np.float32))
        return max(float(area or 0), 0) / max(plaza['area'], 1)
    except cv2.error:
        return 0.0

def puntos_relativos(deteccion, posiciones):
    """Coloca una lista de puntos sobre la caja del coche segun proporciones."""
    ancho = max(deteccion['x2'] - deteccion['x1'], 1)
    alto = max(deteccion['y2'] - deteccion['y1'], 1)
    return [(deteccion['x1'] + rx * ancho, deteccion['y1'] + ry * alto) for rx, ry in posiciones]

def contar_puntos(puntos, plaza):
    """Cuenta cuantos de esos puntos caen dentro de la plaza."""
    return sum((cv2.pointPolygonTest(plaza['hull'], (float(x), float(y)), False) >= 0 for x, y in
puntos))

def asociar_coches(detecciones, plazas):
    """Devuelve que plazas estan ocupadas ahora mismo y las dobles ocupaciones."""
    ocupadas = [False] * len(plazas)
    info = []

    # PASO 1: a cada coche se le busca su plaza principal
    for deteccion in detecciones:
        # se calcula cuanto solapa la huella del coche con cada plaza
        huella = hacer_huella(deteccion)
        solapes = [ratio_solape(huella, plaza) for plaza in plazas]
        # de las plazas que superan el solape minimo, se queda con la de mas solape
        candidatas = [i for i, valor in enumerate(solapes) if valor >= SOLAPE_MINIMO]
        principal = max(candidatas, key=lambda i: solapes[i]) if candidatas else None
        metodo = 'overlap' if principal is not None else 'none'

        # plan B: si ninguna plaza solapa bastante, se decide por puntos de respaldo
        if principal is None:
            puntos = puntos_relativos(deteccion, PUNTOS_RESPALDO)
            cuentas = [contar_puntos(puntos, plaza) for plaza in plazas]
            # se coge la plaza que mas puntos del coche contiene, si llega al minimo
            mejor = max(range(len(plazas)), key=lambda i: cuentas[i])
            if cuentas[mejor] >= PUNTOS_RESPALDO_MINIMO:
                principal = mejor
                metodo = 'fallback'
        # se guarda toda la info de este coche para usarla en los pasos siguientes
```

```
        info.append({'detection': deteccion, 'overlaps': solapes, 'primary': principal, 'method': metodo})

    # PASO 2: se marcan ocupadas las plazas que ha reclamado algun coche.
    # 'reclamos' guarda que coche ocupa cada plaza, para el paso siguiente.
    reclamos = {}
    for dato in info:
        if dato['primary'] is None:
            continue
        ocupadas[dato['primary']] = True
        reclamos.setdefault(dato['primary'], []).append(dato['detection']['id'])

    # PASO 3: se detecta la incidencia de un coche que ocupa dos plazas
    parejas_incidencia = set()
    for dato in info:
        # solo se mira en coches que tienen plaza principal por solape
        if dato['method'] != 'overlap' or dato['primary'] is None:
            continue
        deteccion = dato['detection']
        # se reparte una rejilla de puntos sobre el coche; los lejanos (pequenos)
        # usan una rejilla concentrada abajo, donde de verdad esta el vehiculo
        rejilla_y = REJILLA_Y_PEQUENA if deteccion['relative_area'] < AREA_CAJA_PEQUENA else REJILLA_Y_NORMAL
        rejilla = puntos_relativos(deteccion, [(x, y) for y in rejilla_y for x in REJILLA_X])
        # se buscan segundas plazas invadidas por ese mismo coche
        candidatas_segunda = []
        for segunda, valor in enumerate(dato['overlaps']):
            # se salta la plaza principal y las que apenas solapan
            if segunda == dato['primary'] or valor < SOLAPE_SECUNDARIO_MINIMO:
                continue
            # cuantos puntos del coche caen en esa segunda plaza
            puntos_dentro = contar_puntos(rejilla, plazas[segunda])
            # y si esa plaza no la ha reclamado ya otro coche distinto
            reclamada_por_otro = any((otro_id != deteccion['id'] for otro_id in reclamos.get(segunda, [])))
            if puntos_dentro >= PUNTOS_SECUNDARIA_MINIMO and (not reclamada_por_otro):
                candidatas_segunda.append((puntos_dentro, valor, segunda))
        # si hay segunda plaza invadida, se marca ocupada y se registra la pareja
        if candidatas_segunda:
            _, _, segunda = max(candidatas_segunda)
            ocupadas[segunda] = True
            ids = sorted((plazas[dato['primary']]['id'], plazas[segunda]['id']))
            parejas_incidencia.add(f'{ids[0]}|{ids[1]}')

    return (ocupadas, parejas_incidencia)

#
==========================================================================
=
```

```
# 5. Pintar las plazas sobre la imagen
#
# Colorea cada plaza segun su estado (verde libre, rojo ocupada, amarillo
# alerta). Solo pinta las plazas, no las cajas de los coches, para que se vea
# limpio.
#
================================================================================
=

def dibujar_parking(imagen, plazas, estados, plazas_en_alerta):
    """Devuelve una copia de la imagen con las plazas coloreadas segun su estado."""
    salida = imagen.copy()
    capa = salida.copy()
    # primero el relleno semitransparente
    for indice, plaza in enumerate(plazas):
        if plaza['id'] in plazas_en_alerta:
            color = COLOR_ALERTA
        else:
            color = COLOR_OCUPADA if estados[indice] else COLOR_LIBRE
        cv2.fillPoly(capa, [plaza['points']], color)
    cv2.addWeighted(capa, 0.15, salida, 0.85, 0, salida)
    # luego el borde y el nombre de cada plaza
    for indice, plaza in enumerate(plazas):
        alerta = plaza['id'] in plazas_en_alerta
        color = COLOR_ALERTA if alerta else COLOR_OCUPADA if estados[indice] else COLOR_LIBRE
        cv2.polylines(salida, [plaza['points']], True, color, 4 if alerta else 2, cv2.LINE_AA)
        x, y = np.round(plaza['center']).astype(int)
            cv2.putText(salida, plaza['id'], (x - 14, y + 4), cv2.FONT_HERSHEY_SIMPLEX, 0.34,
COLOR_TEXTO, 1, cv2.LINE_AA)
    return salida

#
================================================================================
=
# 6. El sistema completo
#
# Junta todo lo anterior y lo pone a funcionar sobre un video usando dos
# tareas en paralelo (hilos):
#
#   - Una reproduce el video sin parar, para que se vea fluido.
#   - La otra analiza una imagen cada dos segundos, que es lento y no hace
#     falta hacerlo mas seguido en un aparcamiento.
#
# Para que no choquen, la primera aparta una foto cada dos segundos y le pone
# un numero (un "token"). La segunda solo trabaja cuando ese numero cambia, es
# decir, cuando hay una foto nueva. El Lock evita que las dos toquen los
# mismos datos a la vez.
```

```
#
================================================================================
=

class SistemaParking:
    """Maneja una camara: reproduce el video y analiza la ocupacion en paralelo."""

    def __init__(self, ruta_modelo, fuente_video, ruta_json, device='cpu', imgsz=1920, conf=0.25,
iou=0.6, sample_interval_s=2.0, full_threshold=None, loop_video=True):
        # rutas y parametros de inferencia
        self.ruta_modelo = Path(ruta_modelo)
        self.fuente_video = str(fuente_video)
        self.ruta_json = Path(ruta_json)
        self.device = device
        self.imgsz = imgsz
        self.conf = conf
        self.iou = iou
        self.intervalo_muestreo = sample_interval_s
        self.aforo = full_threshold
        self.repetir_video = loop_video

        # candado para que los dos hilos no pisen los mismos datos
        self.lock = Lock()

        # estado general
        self.en_marcha = False
        self.vision_activa = True
        self.plazas = []
        self.estados = []
        self.inicializado = False
        self.reiniciar_en_siguiente = False

        # contadores de la estabilizacion temporal
        self.cuenta_ocupada = []
        self.cuenta_libre = []

        # imagenes que se pasan entre los dos hilos
        self.imagen_bruta = None      # ultimo frame leido (lo que se ve)
        self.imagen_mostrada = None    # frame ya con las plazas pintadas
        self.imagen_pendiente = None   # foto apartada para analizar
        self.token_pendiente = 0      # sube cada vez que hay foto nueva
        self.token_procesado = 0      # ultimo token que ya se analizo

        # incidencias de doble ocupacion
        self.estado_incidencias = {}
        self.incidencias = []
        self.siguiente_numero_alerta = 1

        # estado de camara, barrera y lleno
```

```
        self.estado_camara = 'DETENIDA'
        self.ultimo_error = ''
        self.modo_barrera = 'AUTO'
        self.estado_barrera = 'ABIERTA'
        self.parking_lleno = False
        self.contador_lleno = 0

    # --- Arranque y parada ---

    def start(self):
        """Comprueba que existen modelo y JSON, carga las plazas y lanza los dos hilos."""
        if self.en_marcha:
            return
        if not self.ruta_modelo.exists():
            raise FileNotFoundError(f'No se encontro el modelo:\n{self.ruta_modelo}')
        if not self.ruta_json.exists():
            raise FileNotFoundError(f'No se encontro el JSON:\n{self.ruta_json}')
        self.plazas = cargar_plazas(self.ruta_json)
        self.aforo = self.aforo or len(self.plazas)
        self.en_marcha = True
        self.estado_camara = 'CONECTANDO'
        Thread(target=self._bucle_captura, daemon=True).start()
        Thread(target=self._bucle_inferencia, daemon=True).start()

    def stop(self):
        self.en_marcha = False

    # --- Controles que usa la ventana ---

    def set_vision_enabled(self, activa):
        """Activa o para la vision sin cortar el video."""
        with self.lock:
            self.vision_activa = bool(activa)
            if not self.vision_activa:
                self.imagen_pendiente = None
                return
            # al reactivar, se apunta la imagen actual para analizarla ya
            if self.imagen_bruta is not None:
                self.imagen_pendiente = self.imagen_bruta.copy()
                self.token_pendiente += 1

    def set_full_threshold(self, aforo):
        """Cambia cuantas plazas ocupadas se consideran 'lleno'."""
        aforo = int(aforo)
        with self.lock:
            total = len(self.plazas)
            if total == 0:
                raise ValueError('Las plazas todavia no se han cargado.')
            if not 1 <= aforo <= total:
```

```
            raise ValueError(f'El valor debe estar entre 1 y {total}.')
        self.aforo = aforo
        self.contador_lleno = 0
        self.parking_lleno = False
        self._actualizar_barrera()

    def set_barrier_mode(self, modo):
        with self.lock:
            if modo in ('AUTO', 'MANUAL'):
                self.modo_barrera = modo
                self._actualizar_barrera()

    def open_barrier(self):
        with self.lock:
            if self.modo_barrera == 'MANUAL':
                self.estado_barrera = 'ABIERTA'

    def close_barrier(self):
        with self.lock:
            if self.modo_barrera == 'MANUAL':
                self.estado_barrera = 'CERRADA'

    def finish_alert(self, id_alerta):
        """Cierra a mano una alerta desde la ventana."""
        with self.lock:
            alerta = self._buscar_alerta(id_alerta)
            if not alerta or alerta['status'] != 'ACTIVA':
                return
            alerta['status'] = 'FINALIZADA'
            alerta['end_time'] = datetime.now().strftime('%H:%M:%S')
            estado = self.estado_incidencias.get(alerta['pair'])
            if estado:
                estado['active'] = False
                estado['blocked'] = True
                estado['alert_id'] = None

    def get_snapshot(self):
        """Devuelve una foto del estado actual para que la ventana lo pinte."""
        with self.lock:
            total = len(self.plazas)
            ocupadas = int(sum(self.estados)) if self.inicializado else 0
            return {
                'frame': self.imagen_mostrada.copy() if self.imagen_mostrada is not None else None,
                'analysis_ready': self.inicializado,
                'total_spaces': total,
                'occupied_spaces': ocupadas,
                'free_spaces': total - ocupadas if self.inicializado else 0,
                'parking_full': self.parking_lleno,
                'full_threshold': self.aforo or total,
```

```
            'vision_enabled': self.vision_activa,
            'camera_status': self.estado_camara,
            'last_error': self.ultimo_error,
            'barrier_mode': self.modo_barrera,
            'barrier_state': self.estado_barrera,
            'active_alerts': sum((a['status'] == 'ACTIVA' for a in self.incidencias)),
            'incidents': [dict(a) for a in self.incidencias],
        }

    # --- Hilo 1: reproduce el video y aparta una foto cada dos segundos ---

    def _bucle_captura(self):
        captura = None
        try:
            captura = cv2.VideoCapture(self.fuente_video)
            if not captura.isOpened():
                raise RuntimeError('No se pudo abrir la camara o el video.')
            es_archivo = Path(self.fuente_video).exists()
            fps = float(captura.get(cv2.CAP_PROP_FPS)) or 25.0
            # si es un archivo se respeta su velocidad; si es camara se va rapido
            espera_frame = 1.0 / fps if es_archivo else 0.01
            # cada cuantos frames toca apartar una foto (para archivos)
            paso_muestreo = max(1, int(round(fps * self.intervalo_muestreo)))
            indice_frame = 0
            ultima_muestra_camara = 0.0

            with self.lock:
                self.estado_camara = 'CONECTADA'
                self.ultimo_error = ''

            while self.en_marcha:
                inicio_ciclo = time.monotonic()
                ok, frame = captura.read()
                if not ok or frame is None:
                    # si es archivo y se acaba, se vuelve a empezar
                    if es_archivo and self.repetir_video:
                        captura.set(cv2.CAP_PROP_POS_FRAMES, 0)
                        indice_frame = 0
                        self._preparar_nueva_vuelta()
                        continue
                    raise RuntimeError('Se ha perdido la entrada de video.')

                ahora = time.monotonic()
                with self.lock:
                    # se guarda siempre el ultimo frame (es lo que se ve en pantalla)
                    self.imagen_bruta = frame.copy()
                    vision_activa = self.vision_activa
                    # se decide si toca apartar una foto para analizar:
                    # en un archivo, cada cierto numero de frames; en camara, por tiempo
```

```
            if es_archivo:
                tomar_muestra = indice_frame % paso_muestreo == 0
            else:
                tomar_muestra = ahora - ultima_muestra_camara >= self.intervalo_muestreo
            # si toca, se aparta la foto y se sube el token para avisar al otro hilo
            if vision_activa and tomar_muestra:
                self.imagen_pendiente = frame.copy()
                self.token_pendiente += 1
                if not es_archivo:
                    ultima_muestra_camara = ahora
            # se prepara lo que se muestra: con las plazas pintadas si ya hay analisis
            if vision_activa and self.inicializado:
                                mostrada = dibujar_parking(frame, self.plazas, self.estados,
self._plazas_en_alerta())
            else:
                mostrada = frame.copy()
            self.imagen_mostrada = mostrada

        indice_frame += 1
        # se espera lo justo para ir al ritmo del video
        transcurrido = time.monotonic() - inicio_ciclo
        time.sleep(max(espera_frame - transcurrido, 0))
    except Exception as error:
      self._marcar_error(error)
    finally:
      if captura is not None:
        captura.release()
      self.en_marcha = False

  # --- Hilo 2: analiza la ultima foto apartada y actualiza el estado ---

  def _bucle_inferencia(self):
    try:
      modelo = YOLO(str(self.ruta_modelo))
      while self.en_marcha:
        # se comprueba si hay una foto nueva que analizar (token distinto)
        imagen = None
        with self.lock:
          if self.vision_activa and self.imagen_pendiente is not None and (self.token_pendiente
!= self.token_procesado):
            imagen = self.imagen_pendiente.copy()
            self.token_procesado = self.token_pendiente
        # si no hay foto nueva, se espera un poco y se vuelve a mirar
        if imagen is None:
          time.sleep(0.02)
          continue
        # se pasa el modelo YOLO sobre la foto y se filtran las detecciones
            resultado = modelo.predict(source=imagen, imgsz=self.imgsz, conf=self.conf,
iou=self.iou, device=self.device, verbose=False)[0]
```

```
            detecciones = sacar_detecciones(resultado, imagen)
            # se decide que plazas estan ocupadas y si hay doble ocupacion
            ocupadas_ahora, parejas_incidencia = asociar_coches(detecciones, self.plazas)
            # se vuelca el resultado al estado del sistema (con estabilizacion)
            with self.lock:
                if not self.vision_activa:
                    continue
                self._actualizar_ocupacion(ocupadas_ahora)
                self._actualizar_incidencias(parejas_incidencia)
                self._actualizar_lleno()
        except Exception as error:
            self._marcar_error(error)

    # --- Estabilizacion: no creerse un cambio hasta verlo varias veces ---

    def _actualizar_ocupacion(self, ocupadas_ahora):
        """Aplica las reglas de 3 para ocupar y 5 para liberar."""
        # la primera vez se toma el resultado tal cual
        if not self.inicializado or self.reiniciar_en_siguiente:
            self.estados = [bool(v) for v in ocupadas_ahora]
            self.cuenta_ocupada = [0] * len(ocupadas_ahora)
            self.cuenta_libre = [0] * len(ocupadas_ahora)
            self.inicializado = True
            self.reiniciar_en_siguiente = False
            return
        # despues, cada plaza necesita varias lecturas seguidas para cambiar de estado
        for i, ahora_ocupada in enumerate(ocupadas_ahora):
            estable = self.estados[i]
            # si la lectura coincide con el estado actual, no hay cambio: se reinician contadores
            if ahora_ocupada == estable:
                self.cuenta_ocupada[i] = 0
                self.cuenta_libre[i] = 0
            # la plaza aparece ocupada pero estaba libre: se cuenta hacia ocupar
            elif ahora_ocupada:
                self.cuenta_ocupada[i] += 1
                self.cuenta_libre[i] = 0
                # solo tras 3 veces seguidas se da por ocupada de verdad
                if self.cuenta_ocupada[i] >= CONFIRMACIONES_PARA_OCUPAR:
                    self.estados[i] = True
                    self.cuenta_ocupada[i] = 0
            # la plaza aparece libre pero estaba ocupada: se cuenta hacia liberar
            else:
                self.cuenta_libre[i] += 1
                self.cuenta_ocupada[i] = 0
                # cuesta mas liberar (5) para no marcar libre una plaza por un fallo suelto
                if self.cuenta_libre[i] >= CONFIRMACIONES_PARA_LIBERAR:
                    self.estados[i] = False
                    self.cuenta_libre[i] = 0
```

```
    def _actualizar_incidencias(self, parejas_detectadas):
        """Confirma una incidencia tras 6 lecturas y la cierra tras 5 sin verla."""
        for pareja in parejas_detectadas:
            self.estado_incidencias.setdefault(pareja, {'positives': 0, 'negatives': 0, 'active': False,
'blocked': False, 'alert_id': None})
        for pareja, estado in self.estado_incidencias.items():
            detectada = pareja in parejas_detectadas
            if estado['blocked']:
                # ya se cerro a mano: se espera a que desaparezca para reactivarla
                estado['negatives'] = 0 if detectada else estado['negatives'] + 1
                if estado['negatives'] >= CONFIRMACIONES_CIERRE_INCIDENCIA:
                    estado['blocked'] = False
                    estado['negatives'] = 0
            elif estado['active']:
                estado['negatives'] = 0 if detectada else estado['negatives'] + 1
                if estado['negatives'] >= CONFIRMACIONES_CIERRE_INCIDENCIA:
                    self._cerrar_alerta_automatica(estado['alert_id'])
                    estado.update({'positives': 0, 'negatives': 0, 'active': False, 'alert_id': None})
            else:
                estado['positives'] = estado['positives'] + 1 if detectada else 0
                if estado['positives'] >= CONFIRMACIONES_INCIDENCIA:
                    id1, id2 = pareja.split('|')
                    estado['alert_id'] = self._crear_alerta(pareja, id1, id2)
                    estado['active'] = True
                    estado['negatives'] = 0

    def _actualizar_lleno(self):
        """Marca el parking como lleno si se supera el aforo varias veces seguidas."""
        if not self.inicializado:
            return
        self.contador_lleno = self.contador_lleno + 1 if sum(self.estados) >= self.aforo else 0
        self.parking_lleno = self.contador_lleno >= CONFIRMACIONES_LLENO
        self._actualizar_barrera()

    # --- Manejo de alertas y barrera ---

    def _crear_alerta(self, pareja, id1, id2):
        id_alerta = f'A{self.siguiente_numero_alerta:03d}'
        self.siguiente_numero_alerta += 1
        self.incidencias.append({'id': id_alerta, 'type': 'Vehiculo mal estacionado', 'spaces': f'{id1} -
{id2}', 'space_ids': [id1, id2], 'pair': pareja, 'start_time': datetime.now().strftime('%H:%M:%S'),
'end_time': '', 'status': 'ACTIVA'})
        return id_alerta

    def _cerrar_alerta_automatica(self, id_alerta):
        alerta = self._buscar_alerta(id_alerta)
        if alerta and alerta['status'] == 'ACTIVA':
            alerta['status'] = 'FINALIZADA'
            alerta['end_time'] = datetime.now().strftime('%H:%M:%S')
```

```
def _buscar_alerta(self, id_alerta):
    return next((a for a in self.incidencias if a['id'] == id_alerta), None)

def _plazas_en_alerta(self):
    return {sid for a in self.incidencias if a['status'] == 'ACTIVA' for sid in a['space_ids']}

def _actualizar_barrera(self):
    # en automatico, la barrera se cierra si el parking esta lleno
    if self.modo_barrera == 'AUTO':
        self.estado_barrera = 'CERRADA' if self.parking_lleno else 'ABIERTA'

def _preparar_nueva_vuelta(self):
    """Al repetir el video desde el principio, se borra el estado anterior."""
    with self.lock:
        for a in self.incidencias:
            if a['status'] == 'ACTIVA':
                a['status'] = 'FINALIZADA'
                a['end_time'] = datetime.now().strftime('%H:%M:%S')
        self.reiniciar_en_siguiente = True
        self.cuenta_ocupada = []
        self.cuenta_libre = []
        self.estado_incidencias = {}
        self.contador_lleno = 0

def _marcar_error(self, error):
    with self.lock:
        self.estado_camara = 'ERROR'
        self.ultimo_error = str(error)
    self.en_marcha = False
```