



INSTITUTO POLITÉCNICO NACIONAL

Unidad Profesional Interdisciplinaria
en Ingeniería Campus Tlaxcala

Reporte Técnico

ESTIMACIÓN DE DISTANCIAS A TRANSEÚNTES EN ENTORNOS DINÁMICOS MEDIANTE VISIÓN COMPUTACIONAL MONOCULAR

TT2025-2_IA06

Presentan:

**Castelan Rosete Marco A.; Ing. Inteligencia Artificial
Tapia Acosta Uriel; Ing. Inteligencia Artificial**

Asesor:

Reyes Cocoletzi Lauro

Tlaxcala, Tlax. a 14 de noviembre de 2025

"La Técnica al Servicio de la Patria"

Resumen

En el presente proyecto se aborda el desarrollo de un sistema de estimación de distancias a transeúntes en entornos dinámicos utilizando visión computacional monocular desde la perspectiva de un peatón. Para obtener referencias métricas confiables, se estimaron distancias reales aproximadas mediante un modelo geométrico que integra las propiedades intrínsecas de la cámara y las dimensiones físicas de los sujetos observados. Considerando la variabilidad natural en las estaturas de distintas personas, el sistema incorporó un dominio de alturas representativo con el fin de mejorar la robustez y la capacidad de generalización del modelo. Con estas mediciones se construyó un conjunto de entrenamiento que permitió ajustar una red neuronal recurrente (RNN) orientada a predecir distancias en secuencias de video, logrando un sistema capaz de adaptarse a escenarios dinámicos y a la diversidad de características de los peatones.

Palabras clave: Estimación de distancias, entornos dinámicos, visión computacional monocular, transeúntes, modelo geométrico, RNN.

Índice

1. Introducción	4
1.1. Antecedentes	5
1.2. Planteamiento del problema	6
1.2.1. Definición del problema	6
1.2.2. Objetivos	7
1.2.3. Justificación	8
1.3. Hipótesis	9
1.4. Aportación científica y/o tecnológica	9
2. Marco Teórico	11
2.1. Visión Computacional	11
2.1.1. Visión Monocular	11
2.1.2. Modelo de Detección: YOLOv8	12
2.1.3. Modelo de seguimiento: DeepSORT	13
2.2. Redes Neuronales Recurrentes	15
2.2.1. LSTM (Long Short-Term Memory)	17
2.2.2. GRU (Gated Recurrent Unit)	17
3. Metodología	18
3.1. Recolección de datos	18
3.2. Generación del conjunto de datos	20
3.3. Análisis y Preprocesamiento de datos	24
3.3.1. Depuración de datos	24
3.3.2. Análisis de datos	25
3.3.3. Segmentación temporal	27
3.3.4. Partición del conjunto de datos	28
3.3.5. Normalización de características	29
3.4. Entrenamiento del modelo recurrente	30
3.4.1. Procesamiento por lotes	30
3.4.2. Configuración de entrenamiento y definición de hiperparámetros	30
3.4.3. Arquitectura GRU	32
3.4.4. Arquitectura LSTM	33

4. Resultados	35
4.1. Dinámica de entrenamiento	35
4.2. Evaluación en conjunto de prueba	36
4.2.1. Métricas globales	36
4.2.2. Visualización de resultados	36
4.3. Inferencia del modelo sobre videos	38
5. Conclusiones	39
Referencias	43

1. Introducción

La estimación de distancias es fundamental para el funcionamiento seguro y eficiente de sistemas móviles y autónomos, como vehículos autónomos, robots móviles y asistentes personales, y se realiza tradicionalmente mediante sistemas de imágenes estéreo, multicámara o mediciones LiDAR. Sin embargo, estos enfoques presentan limitaciones en términos de costo, casos de uso, sincronización, complejidad de equipamiento y consumo energético [1]. En contraste, la visión monocular, potenciada por avances en aprendizaje profundo, ha emergido como una alternativa viable y más accesible, permitiendo la estimación de profundidad a partir de una única imagen RGB . [27].

En la actualidad, los avances en visión computacional monocular han permitido desarrollar sistemas capaces de interpretar información visual de manera automatizada. Se han realizado estudios sobre la estimación de distancias entre vehículos [28], así como otros que amplían el análisis a múltiples objetos y entornos, aunque todavía se basan en perspectivas vehiculares [29]. También existen enfoques basados en análisis geométrico que pueden escalarse a distintos objetos y escenarios [18]. Sin embargo, a pesar de los avances y los buenos resultados obtenidos, los videos y conjuntos de datos utilizados siguen predominando en contextos de movimiento y captura vehicular, destacando el conjunto de datos de detección de objetos KITTI [5], que fue recopilado desde una camioneta.

En este contexto, el proyecto desarrolla y valida un sistema de estimación de distancias a partir de video monocular filmado desde la perspectiva de un peatón, integrando un enfoque geométrico que aprovecha la altura real de los sujetos y los parámetros intrínsecos de la cámara, a partir de los cuales se construyó un conjunto de datos propio; el análisis estadístico de esas mediciones mostró que características visuales como la altura en píxeles y la coordenada vertical del centro del cuadro envolvente de cada persona correlacionan de manera consistente con la distancia real, lo cual orientó la selección de entradas del modelo. Sobre esa base se diseñaron e implementaron dos variantes de arquitectura recurrente (una basada en GRU y otra en LSTM) que fueron entrenadas para la regresión de distancia en secuencias temporales, y se integró un detector YOLOv8 —sin reentrenamiento— para localizar peatones en cada cuadro y alimentar el sistema. Los resultados experimentales, muestran la capacidad del enfoque para generalizar a personas de alturas variables en escenarios dinámicos y establecen las métricas de error y limitaciones que guían futuras mejoras.

1.1. Antecedentes

Artículo	Método	Métricas de desempeño	Contribución principal	Limitaciones
Liang et al. (2022) <i>Self-Supervised Object Distance Estimation Using a Monocular Camera</i>	YOLO + ShuffleNet con aprendizaje auto-supervisado	Evaluado en KITTI; mejora en estimación de distancia específica de objetos	Integración eficiente de detección y estimación de distancia con calibración de cámara	No se especifican métricas cuantitativas detalladas; validación limitada a ciertos escenarios
Zhu et al. (2019) <i>Learning Object-Specific Distance from a Monocular Image</i>	YOLOv5 + Autoencoder con pérdida de proyección	RMSE: 6.870 AbsRel: 0.251	Primer modelo de aprendizaje profundo de extremo a extremo para distancias específicas de objetos	Alto consumo computacional; rendimiento decreciente en escenas con curvas o pendientes pronunciadas
Masoumian et al. (2021) <i>Absolute Distance Prediction Based on Deep Learning Object Detection and Monocular Depth Estimation Models</i>	YOLOv5 + Autoencoder con pérdidas geométricas	Precisión: 96 % RMSE: 0.203	Mejora precisión en entornos complejos mediante enfoque en objetos detectados	No aborda interacciones dinámicas cámara-objetos; validación en escenas estáticas
Magaña et al. (2017) <i>Estimación de la Distancia a un Objeto con Visión Computacional</i>	Sistema auto-supervisado (detección + calibración monocular + multi-escala)	Error: 0.99 % σ : 0.1 % Éxito: 96.1 %	Bajo consumo computacional y adaptabilidad en escenarios dinámicos	Limitado en escenas con movimiento impredecible o alta oclusión

Tabla 1: Comparativa de trabajos relacionados en estimación de distancia con cámara monocular.

En los últimos años, la visión computacional monocular ha experimentado avances notables en la comprensión tridimensional de escenas. La estimación de profundidad a partir de una sola imagen ha sido el enfoque predominante para inferir la estructura espacial del entorno. Sin embargo, es crucial distinguir entre la estimación de profundidad y la estimación de distancias: mientras que la primera busca reconstruir un mapa de profundidad de la

escena, la segunda se centra en calcular la separación métrica entre la cámara y objetos específicos, lo cual es esencial para tareas como la navegación autónoma y la prevención de colisiones.

Frente a este panorama, diversos estudios recientes han abordado el problema de la estimación de distancias desde una perspectiva orientada a objetos, explorando enfoques que combinan detección y percepción espacial a partir de imágenes monoculares. Por ejemplo, en [13] se propone un algoritmo basado en geometría y visión artificial que estima la distancia entre un robot y un objeto específico, alcanzando un error promedio inferior a 10 cm en entornos controlados. De forma complementaria, [30] introduce una arquitectura de aprendizaje profundo que fusiona detección mediante YOLOv5 y mapas de profundidad relativa obtenidos con un autoencoder, logrando una tasa de error del 5.3 % en condiciones reales al aire libre. Por otro lado, [15] plantea una red de aprendizaje profundo de extremo a extremo con funciones de pérdida geométrica, que mejora la precisión en escenas complejas hasta en un 18 % respecto a métodos tradicionales. Estos trabajos evidencian avances notables en precisión y eficiencia en la estimación de distancias orientada a objetos en visión monocular.

Sin embargo, un aspecto que permanece poco explorado en la mayoría de estos enfoques es la dinámica de la escena, particularmente la interacción entre el movimiento de la cámara y el de los objetos dentro de entornos cambiantes. Aunque [26] introduce un sistema auto-supervisado eficiente con bajo consumo computacional y buena adaptabilidad, su desempeño se ve limitado cuando se enfrenta a condiciones no estáticas.

1.2. Planteamiento del problema

1.2.1. Definición del problema

Las tecnologías como LiDAR y las cámaras estéreo son reconocidas por su alta precisión en la estimación de distancias. Sin embargo, presentan desafíos significativos en términos de costo, tamaño y consumo energético, lo que limita su implementación en dispositivos portátiles o de bajo costo. Por ejemplo, el uso de LiDAR en aplicaciones como la conducción autónoma es común debido a su precisión, pero su alto costo y complejidad técnica dificultan su adopción en soluciones más accesibles [25].

Gran parte de los estudios actuales en estimación de profundidad y detección de objetos se desarrollan sobre escenarios vehiculares, utilizando conjuntos de

datos como KITTI, Cityscapes o nuScenes, todos recolectados desde vehículos en movimiento y orientados a aplicaciones de conducción autónoma [16]. Esta concentración en contextos vehiculares limita la generalización de los modelos a otros escenarios como los entornos peatonales o interiores, donde las dinámicas de movimiento y los objetos en escena difieren significativamente. Incluso en conjuntos como DIODE, que incluye escenas interiores, se privilegian escenarios estáticos o controlados [12].

En contraste, la visión computacional monocular, al usar únicamente una cámara para estimar las distancias y desplazamientos, se perfila como una alternativa más accesible frente a tecnologías más complejas. Sin embargo, aún presenta retos importantes. Las soluciones que proponen métodos de estimación no supervisados presentan deficiencias como ambigüedad de escala y rendimiento limitado debido a restricciones geométricas y a la falta de una verdad fundamental de referencia [22].

En el caso del aprendizaje supervisado, se han utilizado métodos basados en secuencias temporales de imágenes para estimar la posición y orientación de la cámara mediante redes convolucionales recurrentes, como en DistanceNet [10]. Sin embargo, la estimación de profundidad monocular continúa enfrentando limitaciones de generalización, especialmente cuando existen variaciones en los parámetros intrínsecos de la cámara, la resolución o el preprocesamiento de las imágenes. Estudios recientes demuestran que los modelos entrenados bajo configuraciones de cámara específicas tienden a degradar su rendimiento al aplicarse en dominios con características ópticas o de captura distintas [9]. Incluso los métodos actuales de estado del arte que integran ajustes para estimación métrica absoluta, evidencian discrepancias métricas notables cuando se evalúan con cámaras o resoluciones distintas a las utilizadas durante su entrenamiento. Estos resultados destacan la relevancia de adaptar los modelos de estimación de distancia a las propiedades particulares de la cámara empleada —incluyendo su resolución, parámetros intrínsecos y de preprocesamiento— para lograr una estimación métrica consistente.

1.2.2. Objetivos

Objetivo general

Evaluar la eficacia de un sistema de estimación de distancias en entornos dinámicos, basado en visión computacional monocular y técnicas de aprendizaje profundo, desarrollado a partir de la integración de distintos enfoques de modelado, detección y análisis secuencial de información visual, con el fin

de determinar su precisión y aplicabilidad en contextos peatonales.

Objetivos específicos

- Generar un conjunto de datos propio compuesto por secuencias de video capturadas desde la perspectiva de un peatón, que sirviera como base para el entrenamiento, validación y análisis del sistema propuesto.
- Explorar dos variantes de modelado temporal mediante redes recurrentes para la estimación de distancia en un marco de visión monocular, considerando las variaciones propias del movimiento en la escena.
- Analizar el rendimiento del sistema desarrollado mediante métricas cuantitativas (MAE , $MAPE$, $RMSE$ y R^2) y la comparación de resultados frente a valores de referencia obtenidos experimentalmente.

1.2.3. Justificación

La detección y estimación de distancias de objetos en entornos dinámicos es fundamental para mejorar la seguridad y la eficiencia de sistemas móviles. Aunque existen soluciones basadas en sensores especializados como LiDAR o sistemas estéreo, estos suelen implicar un costo elevado, mayor consumo energético y limitaciones de integración en dispositivos ligeros. Por otro lado, las metodologías puramente monoculares ofrecen una alternativa más accesible, aunque presentan dificultades para obtener estimaciones métricas precisas, especialmente cuando la cámara se encuentra en movimiento y las condiciones de captura difieren de las utilizadas durante el entrenamiento de los modelos.

En este sentido, en el trabajo presente se propone abordar dicha brecha mediante

1. la captura y anotación de un dataset propio, recolectado con la cámara y resolución reales del sistema objetivo
2. el entrenamiento supervisado de dos enfoques basados en redes recurrentes (GRU y LSTM) para modelar la dinámica temporal y aproximar la distancia en escala métrica.

De esta forma, se busca determinar el grado en que modelos supervisados ajustados al dominio específico pueden superar las inconsistencias métricas de aproximaciones de propósito general.

1.3. Hipótesis

La estimación inicial de distancias, derivada de los parámetros intrínsecos de la cámara y de características observables de los sujetos en la escena, establece una aproximación métrica útil que sirve como base para el entrenamiento supervisado; al alimentar con secuencias de video (es decir, aprovechando las dependencias temporales entre fotogramas) modelos recurrentes, el sistema puede aprender una representación de la escala métrica más estable y generalizable frente a la variabilidad de altura entre individuos, mejorando así la precisión de las estimaciones de distancia en escenarios dinámicos registrados con una cámara monocular.

1.4. Aportación científica y/o tecnológica

El presente trabajo plantea como aportación principal el desarrollo y validación de un modelo de percepción visual basado en visión computacional monocular, orientado a la estimación métrica absoluta de distancias a transeúntes en entornos dinámicos. A diferencia de enfoques genéricos entrenados en dominios externos o con sensores especializados, el sistema propuesto se construyó a partir de la captura y anotación de un dataset propio, grabado con la cámara y resolución reales del sistema objetivo, lo que permitió ajustar el modelo a las condiciones particulares de observación y movimiento.

Como eje metodológico, se combina una estimación inicial métrica basada en parámetros de la cámara con el entrenamiento supervisado de dos variantes recurrentes (GRU y LSTM) sobre secuencias de video; esto permitió explorar cómo la información temporal contribuye a mejorar la coherencia y la generalización de las estimaciones de distancia en escenas dinámicas.

El trabajo constituye así una aportación tecnológica y experimental al demostrar la viabilidad de obtener estimaciones métricas de profundidad con una sola cámara, sin recurrir a sensores costosos como LiDAR o configuraciones estéreo.

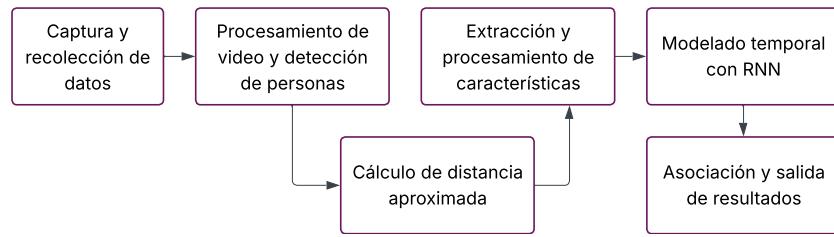


Figura 1: Flujo de trabajo

2. Marco Teórico

2.1. Visión Computacional

La visión computacional es una rama de la inteligencia artificial que busca emular la capacidad humana de interpretar información visual del entorno mediante algoritmos capaces de procesar imágenes y extraer conocimiento significativo. Este campo ha evolucionado de manera considerable en las últimas décadas, impulsado por avances en el aprendizaje profundo y el procesamiento de grandes volúmenes de datos visuales.

Desde un enfoque teórico, la visión computacional se fundamenta en principios de óptica, geometría proyectiva, aprendizaje automático y procesamiento de señales, disciplinas que en conjunto permiten modelar la relación entre el espacio tridimensional (3D) y su representación bidimensional (2D) en una imagen. Uno de los desafíos más relevantes es precisamente la recuperación de información tridimensional a partir de imágenes planas, lo cual constituye la base de tareas como la estimación de profundidad, reconstrucción 3D, localización y navegación autónoma.

Entre los principales paradigmas se encuentran la visión computacional estereoscópica, que emplea múltiples cámaras para inferir profundidad mediante triangulación directa entre puntos correspondientes en distintas vistas, y la visión computacional monocular, que enfrenta el reto de estimar la estructura 3D del entorno a partir de una única imagen o secuencia de imágenes de una sola cámara.

Este último enfoque, aunque más desafiante debido a la ambigüedad inherente en la proyección 2D, resulta especialmente atractivo por su bajo costo computacional y de hardware, y es el foco principal del presente trabajo.

2.1.1. Visión Monocular

La visión monocular se fundamenta en principios geométricos que permiten inferir información tridimensional a partir de proyecciones bidimensionales. En este contexto, la *geometría proyectiva* proporciona el marco matemático que describe la formación de imágenes mediante una cámara, estableciendo cómo los puntos del espacio tridimensional se transforman en coordenadas del plano imagen. A diferencia de la geometría euclíadiana, la

geometría proyectiva no preserva distancias ni ángulos, sino relaciones de incidencia y proporciones invariantes, lo que la hace adecuada para modelar los efectos de la proyección perspectiva [2].

El modelo más utilizado para representar este proceso es el modelo de cámara estenopeica (o *pinhole camera model*), el cual asume una lente ideal que proyecta los rayos de luz a través de un punto denominado *centro óptico* sobre un plano imagen situado a una distancia focal f . Bajo este modelo, la proyección de un punto del espacio $P = (X, Y, Z)$ en coordenadas del mundo hacia un punto $p = (x, y)$ en el plano imagen se describe mediante la semejanza de triángulos [19]:

$$\frac{x}{X} = \frac{f}{Z}, \quad \frac{y}{Y} = \frac{f}{Z}. \quad (1)$$

De esta relación se obtiene la transformación perspectiva directa:

$$x = f \frac{X}{Z}, \quad y = f \frac{Y}{Z}, \quad (2)$$

la cual refleja que la proyección de un punto en el plano imagen depende de la razón entre su posición en el espacio y su profundidad Z . En términos geométricos, esto significa que, a medida que un objeto se aleja de la cámara (aumenta Z), su tamaño proyectado en la imagen disminuye de forma inversamente proporcional.

A partir de esta formulación se deriva una relación fundamental entre la distancia al objeto y su tamaño aparente. Considerando un objeto de altura real H cuya proyección en la imagen tiene una altura h , se cumple la siguiente proporción [18]:

$$\frac{h}{f} = \frac{H}{Z} \quad \Rightarrow \quad Z = f \frac{H}{h}. \quad (3)$$

Esta expresión constituye la base de la estimación de distancia mediante visión monocular, ya que permite inferir la profundidad Z a partir del tamaño proyectado h , siempre que se conozca la distancia focal f de la cámara y la altura real H del objeto. En sistemas digitales, esta relación puede ajustarse considerando las dimensiones del sensor y el tamaño de píxel.

2.1.2. Modelo de Detección: YOLOv8

El modelo de detección utilizado en este proyecto es YOLOv8, una de las versiones más recientes de la familia *You Only Look Once*, desarrollada

por Ultralytics. Este enfoque pertenece a los detectores de una sola etapa, donde la predicción de cajas delimitadoras y categorías se realiza en un único proceso inferencial sobre la imagen completa, permitiendo lograr velocidades en tiempo real sin sacrificar precisión [3].

A nivel conceptual, el detector toma una imagen de entrada $I \in \mathbb{R}^{H \times W \times 3}$ y produce un conjunto de predicciones $\{\hat{\mathbf{b}}_i, \hat{c}_i, \hat{p}_i\}_{i=1}^N$, donde $\hat{\mathbf{b}}_i = (\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i)$ corresponde a la caja delimitadora, \hat{c}_i a la categoría predicha, y \hat{p}_i a la probabilidad asociada.

YOLOv8 conserva la estructura clásica de los detectores modernos: un *backbone* para extraer características jerárquicas, un *neck* para fusionar información multiescala mediante arquitecturas tipo FPN/PAN, y una *cabeza* de predicción encargada de estimar cajas, clases y objetividad [20].

Cada celda en los mapas de características predice un vector de salida del tipo:

$$p_c = (o_c, \Delta x_c, \Delta y_c, \Delta w_c, \Delta h_c, s_c), \quad (4)$$

donde o_c es la probabilidad de presencia de objeto, $(\Delta x_c, \Delta y_c, \Delta w_c, \Delta h_c)$ son desplazamientos relativos para reconstruir la caja final, y s_c contiene las puntuaciones de clase.

La decodificación de una caja predicha se realiza mediante transformaciones paramétricas que ajustan los desplazamientos a la geometría del mapa de características:

$$\begin{aligned} \hat{x}_c &= \sigma(\Delta x_c) + c_x, & \hat{y}_c &= \sigma(\Delta y_c) + c_y, \\ \hat{w}_c &= e^{\Delta w_c} \cdot a_w, & \hat{h}_c &= e^{\Delta h_c} \cdot a_h, \end{aligned}$$

donde (c_x, c_y) representa el centro de la celda y (a_w, a_h) son factores de escala adaptados al enfoque *anchor-free* adoptado por YOLOv8 [20].

Tras la predicción, se aplica una etapa de post–procesado basada en *Non-Maximum Suppression* (NMS) o *Soft-NMS*, encargada de eliminar cajas redundantes y garantizar que el conjunto final de detecciones contenga instancias únicas y confiables.

2.1.3. Modelo de seguimiento: DeepSORT

El modelo de seguimiento DeepSORT integra un modelo dinámico clásico con una métrica de asociación aprendida para mejorar la robustez frente a occlusiones y cambios de apariencia [23]. DeepSORT opera sobre el conjunto de detecciones por cuadro provistas por el detector y resuelve, en línea, la

correspondencia entre detecciones y trayectorias activas (tracks) mediante una combinación de información de movimiento y de apariencia.

A nivel formal, cada track mantiene una estimación del estado dinámico del objeto modelado por un vector de estado

$$\mathbf{x} = [x \ y \ a \ h \ v_x \ v_y \ v_a \ v_h]^\top, \quad (5)$$

donde x, y son las coordenadas del centro de la caja, a la razón de aspecto (ancho/alto), h la altura de la caja, y v_x, v_y, v_a, v_h las correspondientes velocidades. El propósito de este espacio de estado es permitir una predicción cinemática simple (modelo de velocidad constante) y cuantificar la incertidumbre asociada a dicha predicción [23].

La actualización del estado se realiza mediante un filtro de Kalman lineal discreto. Denotando por $\mathbf{x}_{k|k-1}$ y $P_{k|k-1}$ la predicción a priori del estado y su covarianza en el instante k , y por \mathbf{z}_k la observación (medición) obtenida a partir de la detección, las ecuaciones básicas son:

$$\text{Predict: } \mathbf{x}_{k|k-1} = F\mathbf{x}_{k-1|k-1}, \quad P_{k|k-1} = FP_{k-1|k-1}F^\top + Q, \quad (6)$$

$$\text{Update: } K_k = P_{k|k-1}H^\top(HP_{k|k-1}H^\top + R)^{-1}, \quad (7)$$

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + K_k(\mathbf{z}_k - H\mathbf{x}_{k|k-1}), \quad (8)$$

$$P_{k|k} = (I - K_k H)P_{k|k-1}, \quad (9)$$

donde F es la matriz de transición de estado, H la matriz de observación, Q y R las covarianzas de proceso y de medición respectivamente, y K_k la ganancia de Kalman. Esta formulación permite predecir la posición y tamaño esperados de cada objeto y asignar una medida de confianza (covarianza) a la predicción [8].

Para realizar la asociación entre predicciones de tracks y detecciones observadas, DeepSORT construye una matriz de costos que combina dos componentes complementarios:

1. **Componente de movimiento (Mahalanobis):** mide la discrepancia entre la detección \mathbf{d}_j y la predicción del filtro de Kalman para el track i , teniendo en cuenta la incertidumbre del estado. Su forma es

$$d_M^2(i, j) = (\mathbf{d}_j - H\mathbf{x}_{i,k|k-1})^\top S_i^{-1}(\mathbf{d}_j - H\mathbf{x}_{i,k|k-1}),$$

donde $S_i = HP_{i,k|k-1}H^\top + R$ es la covarianza de la predicción proyectada al espacio de medición. En la práctica se aplica un *gating* estadístico:

asociaciones cuyo d_M^2 excede un umbral (por ejemplo, cuantiles de la distribución χ^2) se consideran imposibles y se descartan antes de la asignación [14].

2. **Componente de apariencia (similaridad):** para cada detección se extrae un *embedding* visual $f_j \in \mathbb{R}^D$ mediante una red CNN preentrenada para re-identificación (ReID). La similitud entre el embedding de la detección y el embedding representativo del track i se evalúa típicamente mediante la distancia coseno:

$$d_{\cos}(i, j) = 1 - \frac{f_i^\top f_j}{\|f_i\| \|f_j\|}.$$

DeepSORT mantiene un histórico corto de embeddings por track y usa la mínima o la media de distancias sobre ese banco para robustecer la comparación frente a variaciones de vista y occlusión .

Estos dos términos se combinan en un costo total ponderado

$$c(i, j) = \lambda d_M^2(i, j) + (1 - \lambda) d_{\cos}(i, j), \quad (10)$$

con $\lambda \in [0, 1]$ controlando el balance entre la confianza en la predicción de movimiento y la evidencia visual de apariencia. La combinación lineal permite adaptar el comportamiento del tracker a distintos escenarios.

Dada la matriz de costos $C = [c(i, j)]$, la correspondencia entre M tracks y N detecciones se obtiene resolviendo un problema de asignación con costo mínimo. En la práctica DeepSORT emplea el algoritmo húngaro para obtener la asignación $X = [x_{ij}]$ que minimiza $\sum_{i,j} c(i, j)x_{ij}$ sujeto a restricciones de unicidad [11]. Para mejorar la robustez en presencia de tracks con distintas edades, DeepSORT aplica una *matching cascade* que procesa primero tracks recientemente actualizados y deja para etapas posteriores los tracks más antiguos, reduciendo así los cambios de identidad (ID switches) en ventanas temporales con occlusiones parciales [23].

2.2. Redes Neuronales Recurrentes

Las redes neuronales recurrentes (RNN) pertenecen a una variante de arquitecturas diseñadas para procesar datos secuenciales donde la información útil en un instante depende de eventos previos. A diferencia de las redes feed-forward, las RNN incorporan un *estado oculto* que actúa como memoria a

corto plazo y se actualiza paso a paso a medida que la secuencia es procesada. Esta propiedad las vuelve adecuadas para tareas donde la dinámica temporal o la dependencia contextual son fundamentales. El modelo de RNN se puede describir mediante las ecuaciones siguientes:

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \quad (11)$$

$$y_t = \psi(W_{hy}h_t + b_y), \quad (12)$$

donde $x_t \in \mathbb{R}^{d_x}$ es la entrada, $h_t \in \mathbb{R}^{d_h}$ el estado oculto, y_t la salida (si aplica), W_{xh}, W_{hh}, W_{hy} las matrices de pesos, b_h, b_y los sesgos, ϕ una no linealidad (p.ej. tanh) y ψ la función de salida (p.ej. softmax o identidad). La ecuación (11) explica cómo la red integra información nueva con su memoria previa.

En cada instante temporal t la RNN combina la entrada actual x_t con su memoria previa h_{t-1} para producir un nuevo estado h_t . El estado en h_t resume la información relevante vista hasta el tiempo t . El reuso de los mismos parámetros a lo largo del tiempo (weight sharing) permite que la red generalice a secuencias de distinta longitud y aprenda patrones temporales sin aprender parámetros independientes por posición.

Si definimos una pérdida por paso $\ell(y_t, \hat{y}_t)$, la función de coste sobre una secuencia de longitud T es:

$$\mathcal{L} = \sum_{t=1}^T \ell(y_t, \hat{y}_t). \quad (13)$$

El entrenamiento se realiza mediante *Backpropagation Through Time* (BPTT), que aplica la regla de la cadena a la red desplegada en el tiempo, acumulando gradientes en cada paso temporal.

Al calcular gradientes a través de muchas composiciones en el tiempo, las derivadas involucran productos iterados de jacobianos. Esto puede producir:

- **Explosión del gradiente:** normas de gradiente que crecen exponencialmente, provocando inestabilidad en el entrenamiento.
- **Desvanecimiento del gradiente:** normas que decrecen exponencialmente, imposibilitando el aprendizaje de dependencias a largo plazo.

Formalmente, la contribución del paso k al gradiente en el tiempo t contiene términos del tipo

$$\prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^t J_i,$$

donde J_i es el jacobiano local $\partial h_i / \partial h_{i-1}$. Si los autovalores de J_i son menores que 1 en módulo, el producto tiende a cero (desvanecimiento); si son mayores, tiende a infinito (explosión) [17]. Para aprender dependencias a largo plazo sin sufrir tanto el problema del gradiente, se introdujeron celdas con *puertas*.

2.2.1. LSTM (Long Short-Term Memory)

Una celda LSTM incorpora una *celda de memoria* c_t y tres puertas (forget, input, output) que controlan cuánto de la memoria previa se olvida, cuánto de la nueva entrada se almacena y qué parte de la memoria se expone como salida. Una formulación compacta es:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (14)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (15)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (16)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (17)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (18)$$

$$h_t = o_t \odot \tanh(c_t), \quad (19)$$

donde σ es la sigmoide y \odot el producto elemento a elemento. Estas compuertas permiten mantener información relevante en c_t durante muchos pasos, facilitando dependencias de largo alcance [7].

2.2.2. GRU (Gated Recurrent Unit)

La GRU simplifica la LSTM combinando algunas puertas y estados en una estructura más compacta. Sus ecuaciones típicas son:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad (20)$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad (21)$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \quad (22)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \quad (23)$$

donde z_t es la *update gate* y r_t la *reset gate*. Las GRU a menudo logran desempeño comparable al de LSTM con menor número de parámetros [4].

3. Metodología

Para abordar la estimación de distancia a peatones desde una sola cámara RGB se diseñó una arquitectura que combina técnicas geométricas con modelado temporal, tal como se ilustra en la Fig. [1]. Esta arquitectura integra en un solo flujo los procesos de detección, seguimiento y análisis secuencial de características visuales, siendo capaz de operar en escenarios dinámicos desde la perspectiva de un peatón.

El núcleo del sistema consiste en identificar personas en cada ventana y construir, para cada individuo, una representación temporal basada en atributos geométricos derivados de su proyección en la imagen. Estos atributos capturan variaciones de escala y desplazamiento que reflejan de manera indirecta los cambios de distancia respecto a la cámara. Sobre estas secuencias se aplica un modelo recurrente entrenado para inferir la distancia instantánea a partir de patrones temporales.

La metodología empleada para desarrollar y validar esta arquitectura incluyó la construcción de un conjunto de datos propio mediante un modelo geométrico de referencia, la identificación y seguimiento del peatón objetivo a lo largo de cada video, la depuración y organización de las secuencias resultantes y, finalmente, el entrenamiento del modelo recurrente utilizando ventanas temporales de frames consecutivos. En las siguientes secciones se describen con detalle cada uno de estos componentes, así como su integración dentro del flujo completo de procesamiento.

3.1. Recolección de datos

La fase inicial consistió en la captura de secuencias de video que representen condiciones urbanas reales, grabadas con la cámara principal de un teléfono celular **Google Pixel 6 Pro** a una resolución de 1080p/30 FPS, idealmente bajo condiciones de iluminación naturales.

La cámara se sostuvo manualmente a una altura aproximada de 1.50 m en orientación horizontal, procurando mantener una posición y ángulo estable durante cada grabación. Las distancias de referencia fueron marcadas en el suelo utilizando un flexómetro, midiendo desde la posición de la cámara hasta cada punto marcado. Cada voluntario se desplazó sucesivamente hacia estas posiciones, para cada distancia marcada, el voluntario realizó desplazamientos laterales dentro de un radio acotado, manteniéndose siempre a la misma distancia aproximada de la cámara. (ver Fig. 2).

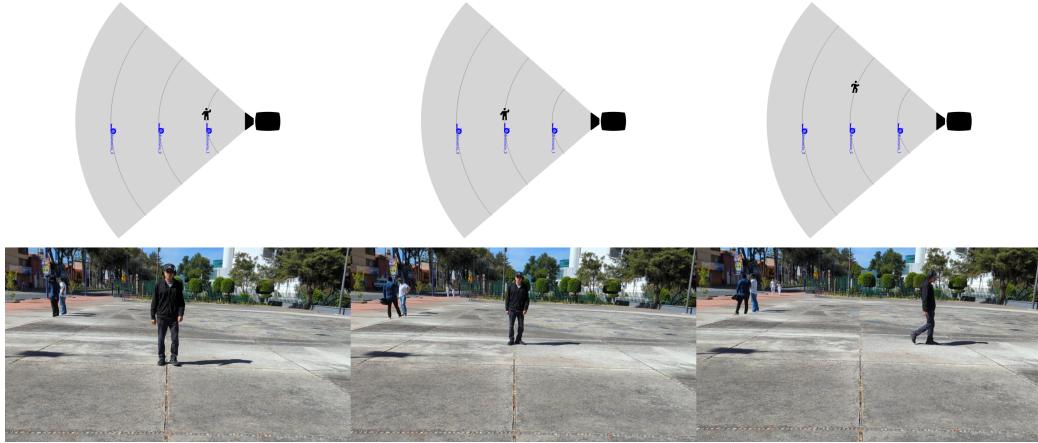
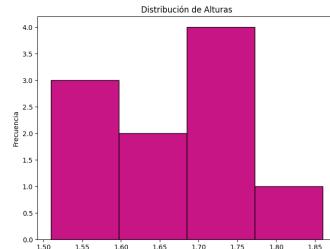


Figura 2: Esquema de grabación

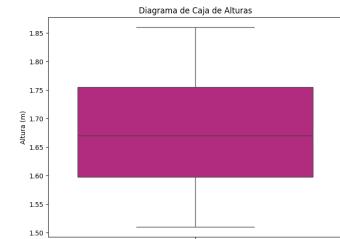
La grabación se realizó con diez voluntarios, cada uno con sus respectivas alturas (Figura 3), medidas previamente a su participación; se buscó mantener alturas variadas, con el fin de evitar el sesgo de la altura más común que es de 1.70 m [21].

Target ID	Altura (m)
1	1.58
2	1.86
3	1.71
4	1.51
5	1.77
6	1.77
7	1.65
8	1.62
9	1.59
10	1.69

(a) Tabla



(b) Histograma



(c) Gráfico de caja

Figura 3: Alturas registradas de los voluntarios para la grabación de video, se presentan un histograma y un gráfico de caja para analizar las características de los datos

3.2. Generación del conjunto de datos

En esta etapa se procesaron las grabaciones para obtener, por cada frame relevante, el cuadro envolvente (*bbox*) que delimita al sujeto de interés y las anotaciones métricas necesarias para la etapa de modelado. En el diagrama de flujo (ver Fig. 4) se representa de manera general el procedimiento de esta etapa.

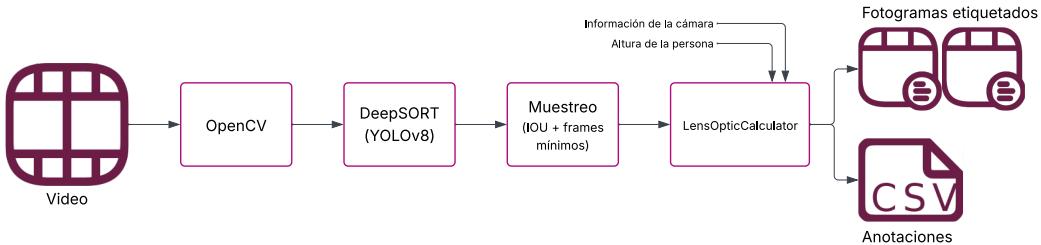


Figura 4: Pasos para etiquetar los videos y obtener anotaciones en CSV

En primer lugar el video se subdivide en los frames correspondientes, a través de OpenCV, debido a que es importante mantener la cantidad de píxeles originales para hacer el cálculo de la distancia no se redimensiona la imagen, por lo tanto, cada frame se procesa con un tamaño de 1080x1920 píxeles, para conservar la relación entre tamaño en píxeles y distancia física del sujeto.

A continuación se pasa cada frame al procedimiento de detección y seguimiento (YOLOV8 + DeepSORT), con el objetivo de detectar a personas en el video, identificar quién de ellas es el objetivo principal, aprendiendo características del individuo en particular y realizar un seguimiento de la entidad a lo largo de toda la trayectoria que esta tomará en el transcurso de grabación. Lo anterior permite generar un *bbox* de la misma persona en cada frame, y con ello las coordenadas que lo conforman.

Para evitar redundancia en el conjunto, se aplica un muestreo que elimina frames muy similares: un frame se conserva si ha transcurrido al menos 5 frames anteriormente o si por medio del cálculo de la intersección sobre la unión (IOU por sus siglas en inglés) se determina que los frames no son parecidos; cualquiera de estas dos situaciones permite que el frame siga procesándose, en caso contrario se omite y se pasa al frame siguiente.

Con la información proporcionada por el cuadro envolvente se calcula la altura de la persona en píxeles que junto con los parámetros intrínsecos de la

cámara y la altura real de la persona (en milímetros) se realiza la estimación de la distancia asociada a cada frame mediante la relación física (24) descrita en el trabajo mencionado en el estado del arte por [24].

$$\text{distancia} = \frac{\text{altura}_{\text{real}}[\text{mm}] \cdot \text{distancia}_{\text{focal}}[\text{mm}] \cdot \text{tamaño}_{\text{sensor}}[\text{px}]}{\text{altura}_{\text{frame}}[\text{px}] \cdot \text{tamaño}_{\text{sensor}}[\text{mm}]} \quad (24)$$

Los parámetros intrínsecos de la cámara (distancia focal y dimensiones del sensor) se obtuvieron inicialmente de las especificaciones del fabricante [6] y posteriormente se ajustaron empíricamente mediante un procedimiento de validación en campo. Para la validación se colocaron sujetos en marcas de referencia en el suelo (2.5, 5.0, 7.5, ..., 25.0 m), se registraron frames representativos y se compararon las distancias calculadas mediante la ecuación (Ec. 24) con las medidas del flexómetro. Los valores finales de los parámetros intrínsecos usados en los experimentos se recogen en la Tabla 2.

Parámetro	Valor	Unidad
Altura del sensor	9.6	mm
Resolución vertical del sensor	1080	píxeles
Distancia focal	15	mm

Tabla 2: Parámetros intrínsecos utilizados para la estimación de distancia.

El desempeño de la relación geométrica se describe en la Tabla 3 y en la Figura 5, donde se muestra la distribución del error absoluto frente a la distancia de referencia.

GT (m)	N muestras	Pred. mediana (m)	Error abs. (m)	Error %
2.5	4	2.8805	0.3805	15.22 %
5.0	4	5.0020	0.0020	0.04 %
7.5	3	7.4320	0.0680	0.91 %
10.0	3	9.9610	0.0390	0.39 %
12.5	3	12.2600	0.2400	1.92 %
15.0	3	15.2590	0.2590	1.73 %
17.5	2	17.4390	0.0610	0.35 %
20.0	2	19.7880	0.2120	1.06 %
22.5	2	22.4135	0.0865	0.38 %
25.0	1	24.5190	0.4810	1.92 %

Tabla 3: Resumen de la precisión por punto de referencia (mediana de las predicciones por punto).

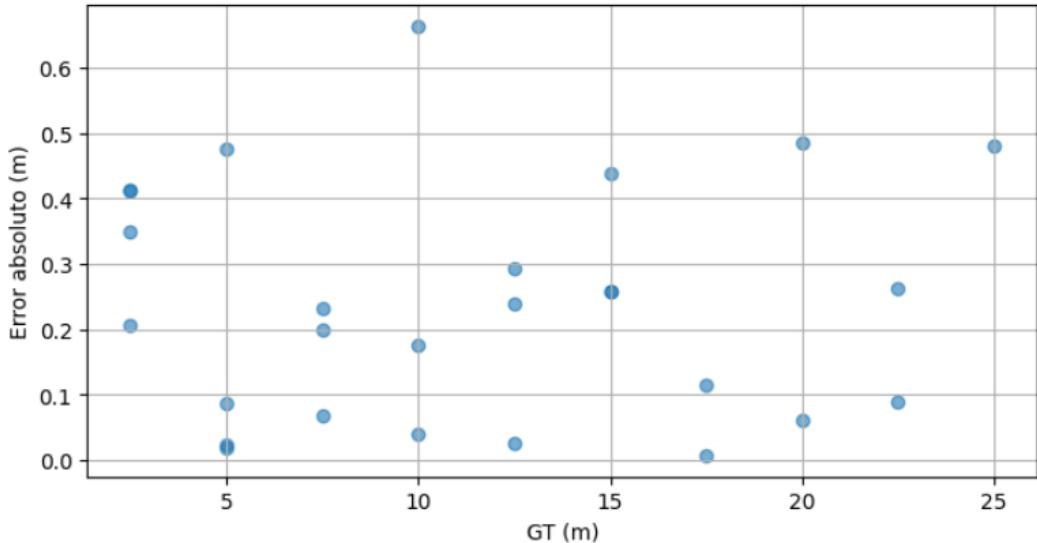


Figura 5: Dispersión del error absoluto $|d_{\text{pred}} - d_{\text{gt}}|$ respecto a la distancia real d_{gt} .

La única distancia que presenta un error significativamente mayor en el resultado es 2.5 m ($\approx 15,22 \%$). Este efecto se debe a que a distancias muy cortas el sujeto puede no aparecer completo dentro del campo de visión de

la cámara (por ejemplo, la parte superior o inferior del cuerpo queda fuera del encuadre). En ese caso el detector ajusta el *bbox* al área visible y la altura en píxeles registrada para la persona queda reducida en comparación con la altura completa esperada. Aunque el error global obtenido ($\approx 3\%$) incorpora todas las distancias evaluadas, este valor está condicionado por el comportamiento del modelo a 2.5 m, donde la persona no siempre aparece completamente en el encuadre y la altura del *bbox* queda limitada por la propia proyección en la imagen. Este fenómeno genera una sobreestimación de la distancia y eleva el error promedio total. Para el resto del rango evaluado (que corresponde al intervalo donde la cámara capta la figura completa con mayor estabilidad) el error porcentual medio se mantiene entre 1% y 1.2%, lo que refleja el desempeño real del modelo en operación.

Finalmente, una vez validados los parámetros intrínsecos y obtenidas las mediciones correspondientes, se almacenaron todos los *frames* etiquetados. Cada registro conserva su identificador de video y de persona, junto con las anotaciones generadas: coordenadas del cuadro envolvente, posición del punto central, altura y ancho del *bbox*, así como la distancia asociada para cada instancia. Toda esta información se consolidó en un único archivo CSV que sirve como la base estructurada del conjunto de datos utilizado en el proyecto. La Figura 6 muestra la distribución de distancias presentes en el conjunto de datos.

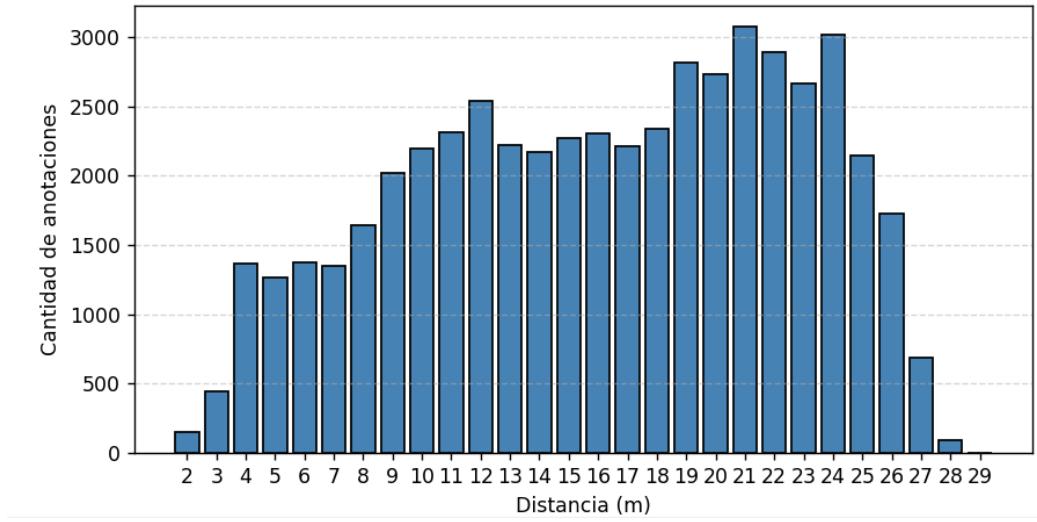


Figura 6: Distribución final de anotaciones por metro

3.3. Análisis y Preprocesamiento de datos

A partir del archivo con las anotaciones consolidadas se llevó a cabo un estudio detallado del conjunto de datos, orientado primero a comprender el comportamiento de las variables disponibles y posteriormente a adecuarlas para su uso en el modelo. Este proceso involucró analizar la relación entre las características geométricas y la distancia real para identificar aquellas con mayor capacidad predictiva, y, con base en ello, aplicar una serie de transformaciones destinadas a garantizar la coherencia y calidad temporal de las secuencias, homogenizar la escala de las variables y organizar la información en segmentos adecuados para su procesamiento por redes recurrentes. El flujo completo de esta etapa se resume en la Figura 7. Dicho esquema ilustra las etapas que conforman el procedimiento, desde la depuración inicial de las anotaciones hasta la organización final de las ventanas temporales utilizadas durante el entrenamiento.

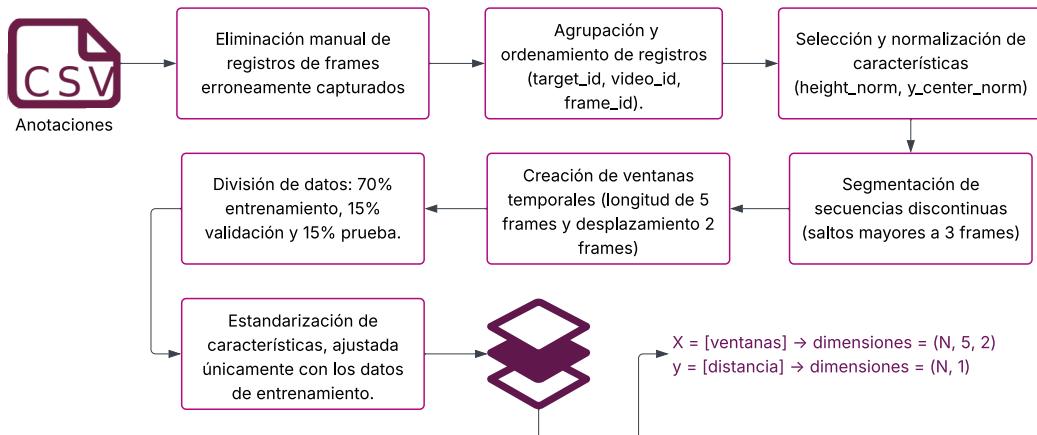


Figura 7: Etapas de tratamiento de las anotaciones

3.3.1. Depuración de datos

Durante la inspección de calidad de las anotaciones se observó que existían errores de identificación y seguimiento provistos por el algoritmo Deep-SORT (ver Fig.4). Esto debido a que, ante occlusiones o cambios bruscos en la apariencia o posición, el algoritmo prioriza la asociación que minimiza su costo interno (apariencia + predicción del filtro Kalman), lo que puede conducir a confundir detecciones cercanas. Estas reasignaciones dieron lugar a distancias anotadas que no correspondían al sujeto objetivo.

Para detectar estos fallos se generaron versiones del video con las anotaciones (bbox, identificador de track y distancia anotada) superpuestas por cada target; esta revisión visual inicial permitió localizar y contextualizar los instantes en que las anotaciones resultaban incongruentes con la distancia conocida del sujeto.

A partir de esta referencia visual, la depuración consistió en verificar, la correspondencia entre la bbox y el sujeto de interés, revisando el historial temporal del track para detectar cambios de identidad y se contrastaron los valores de distancia anotada con la observación directa en el vídeo. Cuando la evidencia mostraba que la detección correspondía a otra persona u objeto la anotación se consideró errónea y se eliminó del conjunto de trabajo

3.3.2. Análisis de datos

A partir del archivo con las anotaciones ya depuradas se realizó un análisis exploratorio para justificar la selección de variables a emplear en el entrenamiento del modelo. La selección se apoyó en una matriz de correlación entre las variables anotadas y la distancia. La matriz de correlación (Figura 8) se calculó utilizando la correlación de Pearson sobre las columnas numéricas relevantes y se visualizó mediante un *heatmap*.

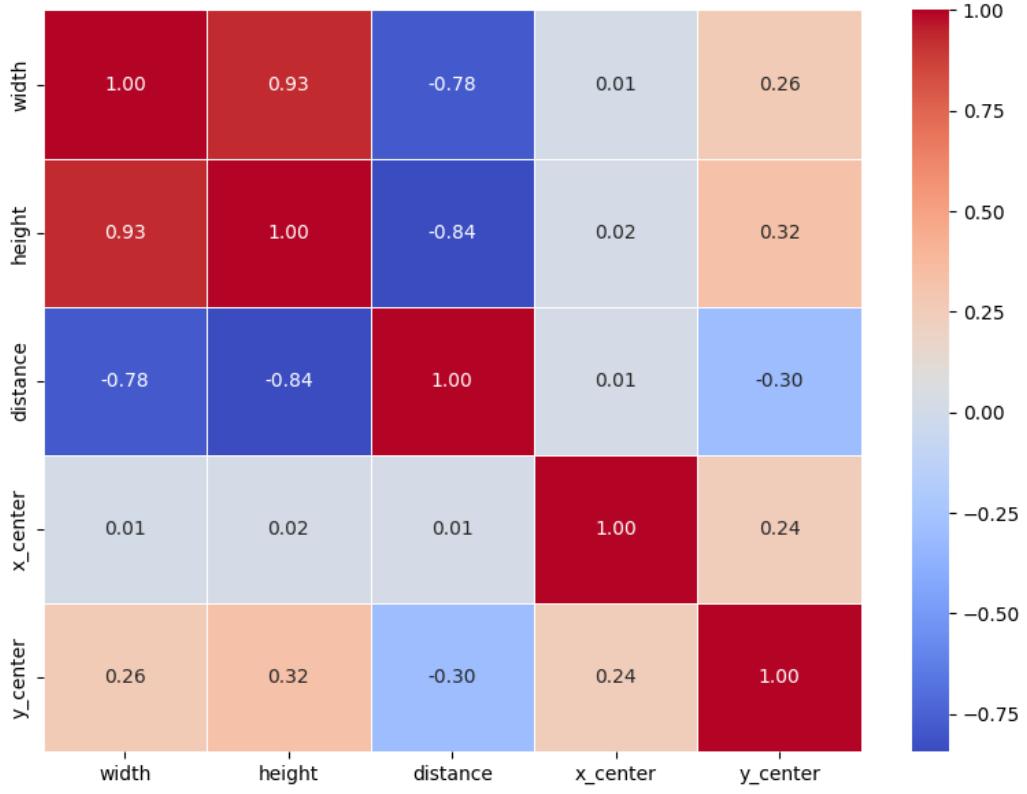


Figura 8: Matriz de correlación entre características geométricas y distancia

A partir de esta matriz se identificaron dos características con mayor correlación negativa respecto a la distancia: la altura del cuadro envolvente (*bbox height*, -0.84) y la coordenada vertical de su centro (*y_center*, -0.30). Ambas variables se relacionan de manera inversa con la distancia real, ya que, conforme una persona se aleja de la cámara, su tamaño proyectado en la imagen disminuye y, en consecuencia, la altura del *bbox* se reduce.

De forma complementaria, la perspectiva de la cámara provoca que el centro del objeto se desplace gradualmente hacia la parte superior del fotograma; es decir, valores menores de *y_center* suelen asociarse con objetos más lejanos. Para reducir multicolinealidad y favorecer que el modelo aprenda características claramente informativas, se optó por seleccionar una sola variable del grupo redundante (en este caso la altura) dado que presentó la correlación más alta con la distancia. La coordenada vertical del centro (*y_center*) se conservó porque no evidenció redundancia con la altura y aporta informa-

ción geométrica complementaria ligada a la perspectiva de la cámara. En consecuencia, las variables utilizadas como entradas principales del modelo fueron la altura del *bbox* y la coordenada vertical del centro, decisión que además simplifica la interpretación del modelo y reduce la dimensionalidad del espacio de entrada.

3.3.3. Segmentación temporal

Una vez definido el conjunto depurado de anotaciones, las muestras se organizaron siguiendo una estructura jerárquica basada en el individuo registrado (*target_id*), el video correspondiente (*video_id*) y el índice temporal de cada imagen (*frame_id*). Esta organización permitió reconstruir las secuencias originales de cada persona y preservar la relación temporal entre los cuadros capturados, condición necesaria para la posterior creación de ventanas temporales.

Posteriormente, se verificó la continuidad temporal de cada secuencia. La depuración previa de cuadros con anotaciones inválidas podía introducir saltos abruptos en el índice de fotogramas, comprometiendo la coherencia temporal necesaria para el aprendizaje secuencial. Para corregirlo, se realizaron segmentaciones; cada video se recorrió en busca de pérdidas superiores a tres *frames* consecutivos, de manera que si esto ocurre se divide la secuencia en dos diferentes, con el punto de corte justo en la posición de la pérdida de frames. Al finalizar este proceso se obtienen múltiples cadenas de frames con una correcta consistencia temporal.

A partir de las subsecuencias temporalmente consistentes obtenidas tras la segmentación, se procede a la construcción de ventanas deslizantes que constituyen las muestras de entrada para el modelo. Cada ventana está definida por una longitud fija (*seq_len*) y un desplazamiento entre ventanas consecutivas (*stride*).

La longitud de ventana propuesto es de diez valores, de cada secuencia original se tomarán estos cinco valores y se seleccionarán con un desplazamiento de dos frames hacia adelante. Lo anterior permite aumentar el número de muestras y mantiene la dinámica temporal de la secuencia.

Formalmente, para una segmento de longitud *L*, el número de ventanas generadas se calcula como:

$$N_{\text{ventanas}} = \left\lfloor \frac{L - \text{longitud_ventana}}{\text{desplazamiento}} \right\rfloor + 1 \quad (25)$$

En la figura 9 se ilustra la cantidad de ventanas generadas por target dada la ecuación 25.

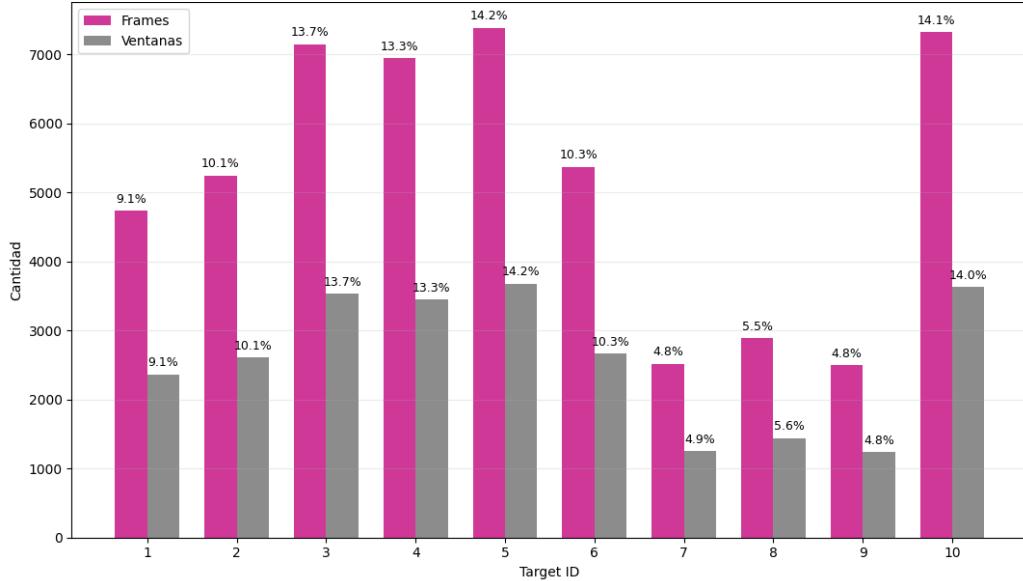


Figura 9: Número de ventanas generadas por individuo-video

3.3.4. Partición del conjunto de datos

Con las ventanas generadas por individuo, se procede a la partición del conjunto de datos en subconjuntos de entrenamiento (*train*), validación (*val*) y prueba (*test*).

Antes de aplicar la partición, se definió manual y deliberadamente qué *targets* pertenecerían a cada subconjunto; esta asignación buscó favorecer la capacidad de generalización del modelo, procurando que *train*, *val* y *test* incluyeran individuos con distinta variabilidad (por ejemplo, rangos de estatura y condiciones de captura diversas), siempre garantizando que ningún *target* se repita entre subconjuntos.

A partir de las listas manuales, la separación efectiva se llevó a cabo mediante máscaras booleanas: para cada ventana se comprueba si su *target_id* aparece en la lista correspondiente, generando filtros que seleccionan todas las ventanas asociadas a los individuos asignados. Este mecanismo asegura que cada persona quede íntegramente contenida en un único subconjunto, evita fugas de información entre entrenamiento y evaluación y contribuye a

que el rendimiento reportado refleje de manera más fiel la capacidad de extrapolación del modelo.

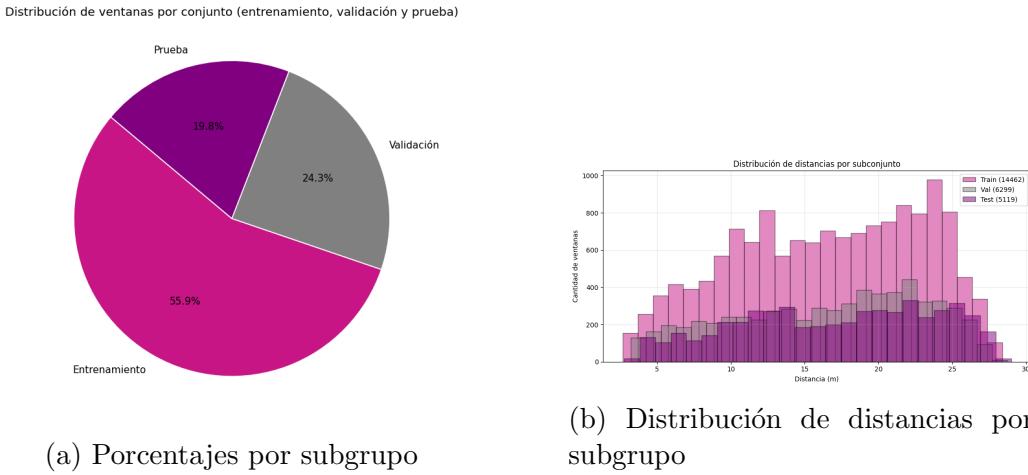


Figura 10: Distribución de los subgrupos de entrenamiento, prueba y validación

La proporción resultante entre los subgrupos y la cobertura de los rangos de distancia en cada subconjunto se ilustran en la Figura 10, lo que permite verificar empíricamente tanto el balance de tamaños como la representatividad de las distancias.

3.3.5. Normalización de características

La normalización se realizó después de la partición en *train*, *val* y *test* descrita en la sección previa, empleando el escalado Min–Max calculado exclusivamente sobre los datos de entrenamiento. Esta elección responde a la necesidad de conservar la distribución relativa de las dimensiones geométricas extraídas del *bbox* y a la ventaja práctica de producir valores dentro de un rango acotado [0,1], lo que suele acelerar la convergencia y estabilizar el entrenamiento. Así mismo, al estimar los parámetros únicamente sobre *train* se evita una fuga de información hacia los subconjuntos de validación y prueba, previendo sesgos optimistas en la evaluación.

Formalmente, denotando por x una componente cualquiera de las características, el escalado aplicado es:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min} + \varepsilon}, \quad (26)$$

Donde x_{\min} y x_{\max} son el mínimo y el máximo de la característica calculados sobre todos los fotogramas contenidos en las ventanas del conjunto de entrenamiento, y ε es una constante pequeña ($\varepsilon = 10^{-8}$) para evitar divisiones por cero cuando $x_{\min} = x_{\max}$.

3.4. Entrenamiento del modelo recurrente

En esta etapa se implementa y compara una arquitectura basada en LSTM y otra basada en GRU, ambas orientadas a la tarea de regresión de la distancia en metros a partir de las secuencias temporales generadas y de las características geométricas seleccionadas tras el análisis de correlación. A lo largo de la sección se describirá la estrategia de agrupación en lotes (*batching*), la arquitectura de cada red, los hiperparámetros de entrenamiento y las métricas empleadas para la evaluación.

3.4.1. Procesamiento por lotes

Dada la naturaleza secuencial de las muestras, la estrategia de agrupación en lotes (*batching*) se diseñó para preservar la coherencia temporal y evitar la mezcla de ventanas de distintos individuos dentro del mismo lote (*batch*). Para ello se emplea una clase de generación de lotes (*batches*) que agrupa las ventanas por *target_id* (cada batch contiene únicamente ventanas pertenecientes al mismo individuo). Este procedimiento facilita el aprendizaje de dinámicas temporales coherentes y permite controlar explícitamente el barajado a nivel de batches —barajado activo en entrenamiento y desactivado en validación—, contribuyendo a una evaluación más estable y representativa.

3.4.2. Configuración de entrenamiento y definición de hiperparámetros

La configuración de entrenamiento se definió de forma común para las dos variantes de arquitecturas (LSTM y GRU). En ella se establecen los criterios y parámetros que se mantendrán idénticos para permitir comparaciones controladas y reproducibles.

El procedimiento de entrenamiento inicia con la carga de las secuencias de

entrenamiento y validación, junto con sus etiquetas de distancia y los identificadores de individuo necesarios para la generación de los lotes. Estas secuencias se organizan mediante el esquema de batching descrito previamente: los lotes se mantienen homogéneos por *target_id* y sólo se barajan en el conjunto de entrenamiento.

Una vez generados los lotes, se instancia el modelo recurrente correspondiente (LSTM o GRU) mediante una función constructora que recibe como parámetros la longitud de secuencia y el número de características. De esta forma, la estructura del entrenamiento permanece fija y únicamente la arquitectura interna del bloque recurrente varía entre experimentos.

Para controlar el proceso de entrenamiento se emplean callbacks estándar pero afinados para el problema de regresión en distancia:

- **EarlyStopping** monitorizando `val_mae` y recuperando los mejores pesos mediante `restore_best_weights`, con el fin de detener el entrenamiento cuando no se observe mejora y evitar sobreajuste.
- **ReduceLROnPlateau** monitorizando `val_mae`, para disminuir la tasa de aprendizaje cuando el progreso en validación se estabiliza.
- **ModelCheckpoint** para guardar el mejor modelo (formato `.keras`) durante el entrenamiento.
- **CSVLogger** para conservar un registro de las métricas por época y facilitar análisis posteriores.

A modo de referencia, la configuración experimental de hiperparámetros usada en el presente trabajo se resume en la Tabla 4.

Parámetro	Valor (por defecto)
Optimizer	Adam
Learning rate inicial	1×10^{-4}
Loss	MSE
Métrica principal	MAE
Epochs máximas	200
Batch size (seq. por batch)	32
EarlyStopping (monitor)	val_mae, patience=30, min_delta=0.01
ReduceLROnPlateau	factor=0.5, patience=15, min_lr=1e-7
ModelCheckpoint	guarda mejor val_mae (.keras)
CSVLogger	registra por-época training_log.csv

Tabla 4: Hiperparámetros de entrenamiento.

3.4.3. Arquitectura GRU

La variante GRU utilizada en este trabajo intenta equilibrar capacidad de modelado temporal y simplicidad operativa. Dada la entrada temporal estructurada en ventanas y compuesta por dos variables por paso (las características geométricas seleccionadas durante el análisis); el procesamiento de dichas ventanas y el batching por *target_id* siguen el procedimiento descrito previamente en la sección de preprocesamiento y se ilustran de manera general en la Figura 11.

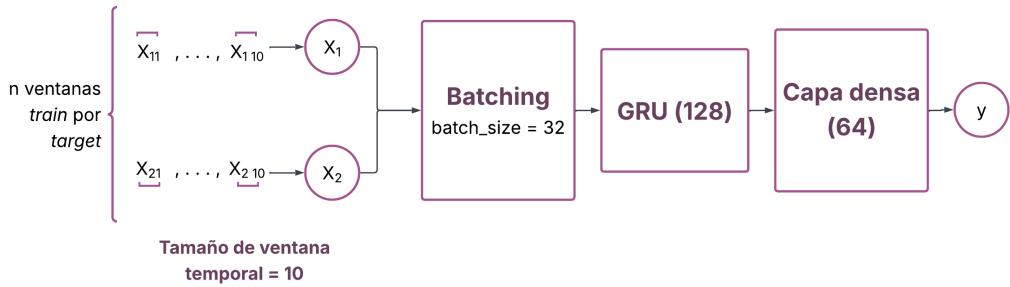


Figura 11: Arquitectura de la RNN basada en GRU

A partir de esa entrada, la red dispone de un único bloque recurrente GRU (en la implementación experimental con 128 unidades) que resume la dinámica temporal de cada ventana en un vector de estado final para obtener

una representación compacta por ventana.

La salida del bloque recurrente se proyecta a través de una capa densa intermedia de 64 unidades con activación ReLU, que aporta no linealidad y facilita la transformación de la representación temporal a un espacio más adecuado para la regresión. Tras esa capa densa se aplica regularización por **dropout** (en la configuración usada se fijó en 0.2) para mitigar el sobreajuste. Finalmente, la capa de salida consiste en una única neurona con activación lineal que produce la predicción continua de la distancia en metros, decisión coherente con el objetivo de regresión.

3.4.4. Arquitectura LSTM

La variante basada en LSTM parte de la misma estructura de entrada —ventanas temporales con dos características por paso— cuyo procesamiento y agrupamiento por *target_id* siguen exactamente el flujo de batching descrito; el esquema de la arquitectura se muestra en la Figura 12.

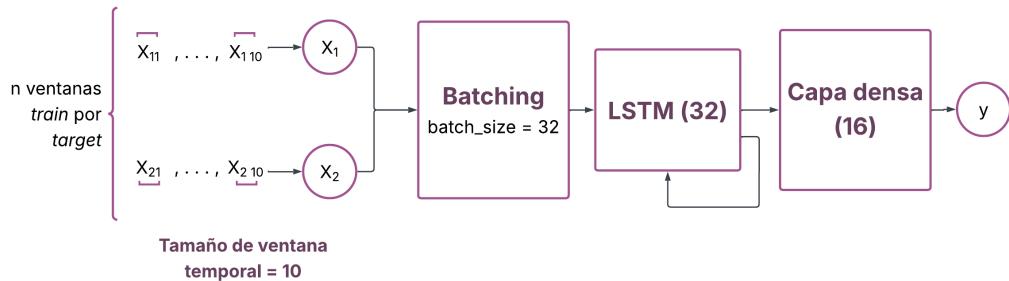


Figura 12: Arquitectura de la RNN basada en LSTM

A continuación, la secuencia entra a una única capa LSTM de 32 unidades (con `return_sequences=False`), cuyo estado final resume la dinámica temporal de la ventana. La elección de un número reducido de unidades —en contraste con las 128 usadas en la variante GRU— se debe a que las LSTM tienen una estructura interna más compleja y costosa (compuertas y estado de celda), por lo que incrementar su dimensionalidad eleva significativamente el coste computacional. Al igual que en la arquitectura previa, se aplica regularización mediante **dropout** (configurado en 0.1) tanto en las conexiones de entrada como en las recurrentes con el fin de mitigar sobreajuste sin incrementar la complejidad del modelo.

La representación generada por el LSTM se proyecta finalmente a una capa densa de 16 unidades con activación `relu`, que actúa como transformación no lineal previa a la estimación. La salida corresponde a una única neurona lineal, para producir el valor continuo de distancia que el modelo debe predecir.

4. Resultados

En esta sección se presentan los resultados obtenidos al evaluar las dos variantes de la red recurrente (GRU y LSTM) entrenadas bajo la configuración descrita previamente. Primero se muestran las curvas de entrenamiento para comprobar la convergencia (ver Fig 13); a continuación, se reportan las métricas agregadas en *test* y se realiza un análisis de error detallado: distribución global (ver Tabla 5) y por rangos de distancia (ver Fig 14). Finalmente se muestran ejemplos cualitativos y una discusión sobre el comportamiento observado.

4.1. Dinámica de entrenamiento

La figura 13 muestra la evolución de la pérdida (MSE) durante el entrenamiento de ambas variantes (GRU a la izquierda y LSTM a la derecha). Se observa una caída pronunciada de la pérdida en las primeras épocas seguida de una meseta intermedia y una nueva reducción posterior.

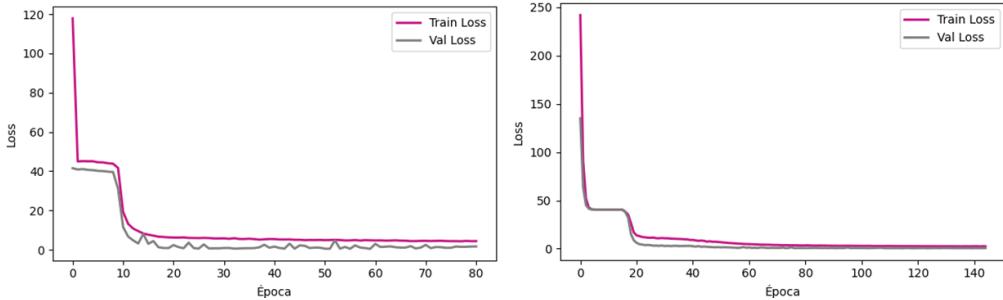


Figura 13: Evolución de la pérdida (MSE) durante el entrenamiento para GRU (izquierda) y LSTM (derecha).

Proponemos como hipótesis que este comportamiento en dos fases está relacionado con la naturaleza estocástica del entrenamiento y con la forma en que se construyen los mini-batches. En una primera etapa, el modelo aprende patrones generales y la pérdida desciende rápidamente; después aparece una meseta cuando las actualizaciones dejan de producir mejoras claras. Con el paso de las épocas, la variación entre batches —así como la posible agrupación de ejemplos por *target_id*— introduce suficiente diversidad como para que el optimizador salga de ese estancamiento y logre una segunda etapa de refinamiento.

Otro aspecto relevante observado durante el entrenamiento fue la diferencia en convergencia y tiempo entre las dos variantes. El modelo GRU alcanzó su mejor validación mucho antes (época 51) que la LSTM (época 115), lo que indica una convergencia más rápida en las condiciones experimentales usadas. Por otra parte, la LSTM alcanzó un mejor ‘val_mae’ final (0.5016 m frente a 0.5288 m para la GRU), a costa de más épocas y mayor tiempo por época. Estas diferencias pueden presentarse debido a las características internas de cada bloque recurrente (capacidad de modelado y coste computacional) como a la configuración experimental (número de unidades por bloque, batching por *target_id*, etc.).

4.2. Evaluación en conjunto de prueba

4.2.1. Métricas globales

Las métricas globales se calcularon sobre el conjunto de prueba para comparar de forma directa el desempeño de las dos variantes entrenadas (LSTM y GRU). Se reportan MAE (con su desviación estándar del error absoluto), RMSE, R^2 , MAPE y el número de secuencias evaluadas. La tabla 5 resume los resultados obtenidos por ambos modelos.

Métrica	LSTM	GRU
MAE (m)	$0,7106 \pm 0,6116$	$0,7358 \pm 0,5726$
RMSE (m)	0,9376	0,9324
R^2	0,9792	0,9794
MAPE (%)	4,11	4,30

Tabla 5: Comparativa de métricas globales en el conjunto de prueba para las variantes LSTM y GRU.

La LSTM muestra un MAE y un MAPE ligeramente mejores que la GRU, mientras que la GRU presenta un RMSE marginalmente menor y un R^2 prácticamente idéntico. Las diferencias son pequeñas en magnitud.

4.2.2. Visualización de resultados

A continuación, en la Figura 14 se presentan las visualizaciones principales que permiten analizar el comportamiento de ambas variantes en el conjunto de prueba. Estas figuras complementan las métricas numéricas previamente

reportadas y ofrecen una vista directa sobre la relación entre las predicciones y los valores reales, así como sobre la distribución y magnitud de los errores obtenidos para cada arquitectura.

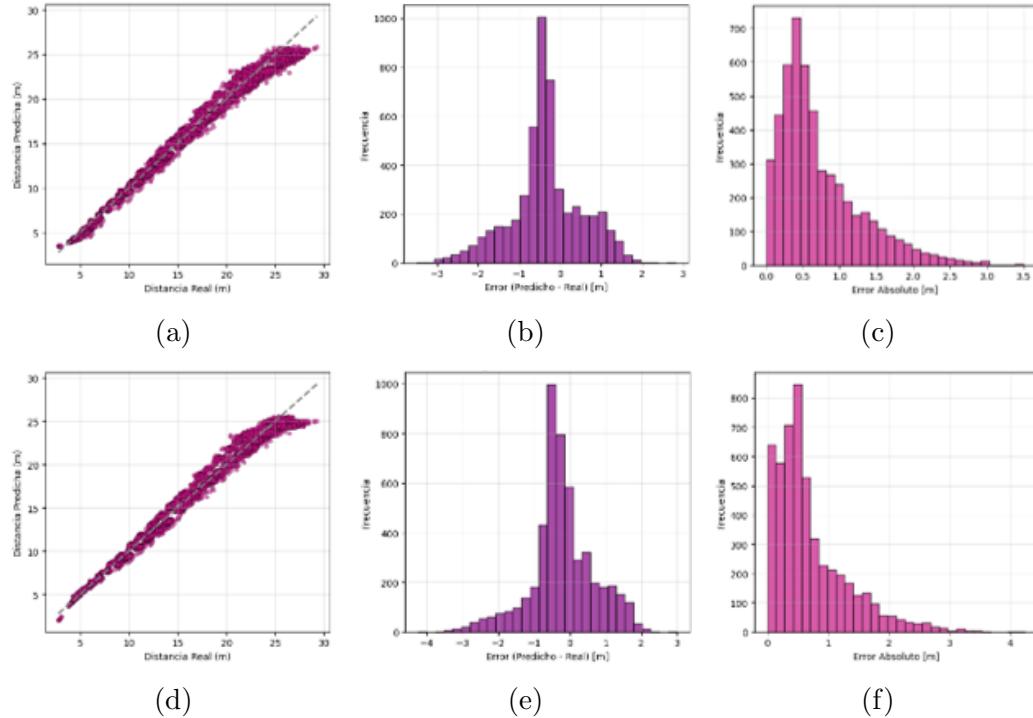


Figura 14: Comparativa visual del desempeño en test: fila superior = GRU; fila inferior = LSTM. (a)/(d) Predicción vs. Real con línea identidad; (b)/(e) histograma de errores (Pred – Real); (c)/(f) histograma de errores absolutos.

A partir de estas representaciones se puede contrastar directamente el comportamiento de ambas arquitecturas.

En términos del análisis de la alineación entre las predicciones y los valores reales, las subfiguras 14a y 14d permiten observar que tanto el modelo GRU como el de LSTM siguen una tendencia cercana a la línea identidad, lo que indica que, en general, el modelo logra replicar adecuadamente la progresión de distancias del conjunto de prueba. La GRU muestra una dispersión un poco mayor, especialmente en distancias altas, mientras que la LSTM mantiene un agrupamiento ligeramente más consistente alrededor de la referencia. En esta última también se aprecia una leve subestimación en el intervalo aproximado de 2–3 m, aunque se trata de un efecto reducido.

Las distribuciones de error presentadas en las subfiguras correspondientes (ver Fig. 14b, 14e) muestran una forma aproximadamente gaussiana centrada en valores cercanos a cero, lo que sugiere ausencia de sesgos marcados y estabilidad general en las predicciones. En la variante GRU, la media del error es $\mu = -0,348$ m y su dispersión alcanza $\sigma = 0,865$ m, reflejando una leve subestimación acompañada de una variabilidad moderada. Por otra parte, la LSTM presenta una media aún más próxima a cero, $\mu = -0,218$ m, junto con una desviación estándar ligeramente mayor, $\sigma = 0,912$ m.

En las distribuciones de errores absolutos mostradas en las subfiguras 14c y 14f permiten evaluar la magnitud típica de las desviaciones en las predicciones. En ambos modelos se observa una concentración alta de valores por debajo de 1 m, lo que indica que la mayoría de las estimaciones se mantienen dentro de un rango de error reducido. En la GRU, la mayor concentración de valores se agrupa ligeramente más cerca del origen, reflejando su MAE de menor magnitud. En la LSTM, aunque también predomina la zona de errores pequeños, la cola de la distribución se extiende un poco más, consistente con la presencia ocasional de errores absolutos mayores.

4.3. Inferencia del modelo sobre videos



Figura 15: Frames de resultado tras aplicar el modelo de red neuronal sobre videos

5. Conclusiones

El proyecto cumplió con el objetivo principal de desarrollo y validación de un sistema de estimación de distancias basado en visión monocular desde la perspectiva de un peatón en movimiento. A diferencia de los trabajos previos centrados en escenarios vehiculares, este estudio abordó un entorno peatonal real, construyendo un conjunto de datos propio con distancias medidas empíricamente y un análisis de características visuales relevantes, lo cual permitió entrenar un modelo adaptado al caso de uso.

Los resultados demostraron que el sistema es capaz de estimar distancias de manera estable dentro del rango predominante del conjunto de datos, alcanzando un MAE aproximado de 1 m y un coeficiente de determinación R^2 cercano a 0.95 para los videos utilizados en el conjunto de prueba cuyos sujetos tenían una variabilidad de altura relativamente distinta entre sí. Por otro lado, las curvas de entrenamiento evidencian una convergencia consistente entre el error de entrenamiento y de validación, lo que indica que el modelo logró captar las relaciones geométricas esenciales entre la altura del sujeto en la imagen, la posición vertical del cuadro envolvente y la distancia real.

De manera general, el enfoque mostró una capacidad de generalización aceptable, donde las predicciones mantuvieron alta fidelidad respecto a las distancias medidas. Esto confirma que la combinación entre parámetros intrínsecos de cámara, variables geométricas, modelado recurrente constituye una estrategia efectiva para escenarios peatonales monoculares, aun sin recurrir a información de profundidad, sensores adicionales o reentrenamiento del detector.

Referencias

- [1] P. Agand, M. Chang, and M. Chen. Dmode: Differential monocular object distance estimation module without class specific information. In *2024 13th International Workshop on Robot Motion and Control (RoMoCo)*, pages 261–266, 2024.
- [2] S. Birchfield. An introduction to projective geometry (for computer vision), March 1998. Unpublished manuscript.
- [3] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv:2004.10934*, 2020.
- [4] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. ACL, 2014.
- [5] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.
- [6] Google. Pixel phone hardware tech specs - pixel phone help, 2025. URL <https://support.google.com/pixelphone/answer/7158570?hl=en#zippy=%2Cpixel-pro>.
- [7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [8] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(1):35–45, 1960.
- [9] Karlo Koledic, Ivan Markovic, and Ivan Petrovic. Towards camera-parameters invariant monocular depth estimation in autonomous driving. In *European Conference on Mobile Robots (ECMR)*, 2023.

- [10] R. Kreuzig, M. Ochs, and R. Mester. Distancenet: Estimating traveled distance from monocular images using a recurrent convolutional neural network. arXiv preprint arXiv:1905.09957, 2019.
- [11] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955.
- [12] J. Li, J. Hu, Y. Huang, Z. Chen, B. Gao, J. Jiang, and Y. Zhang. A synthetic digital city dataset for robustness and generalisation of depth estimation models. *Scientific Data*, 11:301, 2024.
- [13] Hong Liang, Zizhen Ma, and Qian Zhang. Self-supervised object distance estimation using a monocular camera. *Sensors*, 22(8):2936, 2022. doi: 10.3390/s22082936.
- [14] Prasanta Chandra Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1): 49–55, 1936.
- [15] Armin Masoumian, David G. F. Marei, Saddam Abdulwahab, Julián Cristiano, Domenec Puig, and Hatem A. Rashwan. Absolute distance prediction based on deep learning object detection and monocular depth estimation models. 2023.
- [16] Y. Niu, Z. Xu, E. Xu, G. Li, Y. Huo, and W. Sun. Monocular pedestrian 3d localization for social distance monitoring. *Sensors*, 21(17):5908, 2021.
- [17] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1310–1318. PMLR, 2013.
- [18] Xuepeng Shi, Qi Ye, Xiaozhi Chen, Chuangrong Chen, Zhixiang Chen, and Tae-Kyun Kim. Geometry-based distance decomposition for monocular 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15172–15181, 2021. doi: 10.1109/ICCV48922.2021.01474.
- [19] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, Cham, Switzerland, 2nd edition, 2022. ISBN 978-3-031-01847-5.

- [20] Ultralytics. Yolov8 documentation. <https://docs.ultralytics.com>, 2023.
- [21] Ricardo Velez. ¿cuál es la estatura promedio de los mexicanos? datos revelados por el inegi, Apr 2025.
- [22] Y. Wan, Q. Zhao, C. Guo, C. Xu, and L. Fang. Multi-sensor fusion self-supervised deep odometry and depth estimation. *Remote Sensing*, 14(5):1228, 2022.
- [23] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017.
- [24] Yiran Wu, Sihao Ying, and Lianmin Zheng. Size-to-depth: A new perspective for single image depth estimation, 2018. URL <https://arxiv.org/abs/1801.04461>.
- [25] H. Yang, T. Tang, and C. Gao. Train distance estimation in turnout area based on monocular vision. *Sensors*, 23(21):8778, 2023.
- [26] J. B. Magaña Z., J. R. Atoche E., J. C. Molina C., M. Blanco V., and E. Pérez C. Estimación de la distancia a un objeto con visión computacional. 2017. Recibido: 30 de junio de 2017; Aprobado: 19 de septiembre de 2017.
- [27] Chaoqiang Zhao, Qiyu Sun, Chongzhen Zhang, Yang Tang, and Feng Qian. Monocular depth estimation based on deep learning: An overview. *arXiv preprint arXiv:2003.06620v2*, 2020. URL <https://arxiv.org/abs/2003.06620>. Accessed: June 6, 2025.
- [28] T. Zhe, L. Huang, Q. Wu, J. Zhang, C. Pei, and L. Li. Inter-vehicle distance estimation method based on monocular vision using 3d detection. *IEEE Transactions on Vehicular Technology*, 69(5):4907–4919, May 2020.
- [29] J. Zhu and Y. Fang. Learning object-specific distance from a monocular image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3839–3848, 2019.

- [30] Jing Zhu, Yi Fang, Husam Abu-Hamed, Kuo-Chin Lien, Dongdong Fu, and Junli Gu. Learning object-specific distance from a monocular image. *arXiv preprint arXiv:1909.04182*, 2019.