UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Localización de robots bípedos usando transformadores visuales y filtros de partículas

TESIS

Que para obtener el título de

Ingeniera Mecatrónica

P R E S E N T A

Ruth Getzemaní Moreno Cedano

DIRECTOR DE TESIS

Dr. Marco Antonio Negrete Villanueva

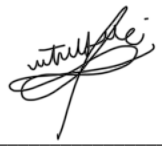
Ciudad Universitaria, Cd. Mx., 2026

PROTESTA UNIVERSITARIA DE INTEGRIDAD Y HONESTIDAD ACADÉMICA Y PROFESIONAL
(Titulación con trabajo escrito)

De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado LOCALIZACION DE ROBOTS BIPEDOS USANDO TRANSFORMADORES VISUALES Y FILTROS DE PARTICULAS. que presenté para obtener el titulo de INGENIERA MECATRÓNICO es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.



RUTH GETZEMANI MORENO CEDANO
Número de cuenta: 419048451

Dedicado a mi madre, Hilda,
por su apoyo incondicional y por ser
el pilar más importante en mi vida.

Agradecimientos

Este trabajo se realizó con el apoyo del proyecto PAPIIT IN118226 “Aprendizaje de máquina para el desarrollo de robots bípedos autónomos”.

Agradezco sinceramente:

A mi tutor de tesis, el Dr. Marco Negrete, por ser una figura fundamental en mi formación académica. Le agradezco profundamente su invaluable guía y liderazgo, tanto en esta investigación como en el trabajo conjunto con el laboratorio.

Al Dr. Luis Contreras, por su guía, hospitalidad y orientación durante el desarrollo y enriquecimiento de este trabajo.

A todos los docentes que forjaron mi desarrollo académico, pero principalmente al profesor Crail.

A Xihuitl, por su amistad y constante compañía y por hacer memorables los últimos años de mi carrera.

A los amigos que hice en la carrera y me han acompañado: Alejandra, Jacquelin, Candy y Rafael, así como a mis amigas, que a la distancia siempre me han apoyado: Ivette, Samantha, Ana, Rayza, Valeria, Karla, Fernanda y Frida.

A Miguel y Edith, por brindarme un hogar y una familia a lo largo de esta etapa.

Asimismo, a mi familia: Adriana, Yessica, Susana, Hugo, Jaqueline e Idalia, por su constante aliento aún a la distancia.

Finalmente, a Miguel Ángel, por todo lo que me ha enseñado y hemos compartido, por ser mi compañero a lo largo de este trayecto.

Índice general

Capítulo 1

Introducción

El área de los robots bípedos, debido a su capacidad de movimiento bidimensional semejante en gran medida a la locomoción humana, es una de las más prometedoras de la robótica. Esta similitud con el patrón de marcha de los humanos los convierte en plataformas ideales para operar en entornos diseñados para personas, desde hogares hasta espacios públicos. Dentro de este contexto, un ambiente de pruebas que resulta excelente para desarrollar y evaluar su anatomía es un campo de fútbol, en donde el robot debe percibir, planificar y tomar decisiones en tiempo real.

La localización robusta es una capacidad fundamental para el comportamiento de robots humanoides autónomos en el dominio de la liga Humanoid Soccer de la competencia Robocup [23] en ambientes dinámicos, como una cancha de fútbol. La habilidad de una localización confiable permite a su vez una navegación adecuada, toma de decisiones e interacción con el entorno. [47] En el fútbol de humanoides, la localización es particularmente retadora debido a la percepción ambigua, oclusiones frecuentes y la ausencia de marcas distintivas que rompan esta ambigüedad.

Las propuestas de localización mediante visión generalmente se basan en las detecciones del campo, incluyendo porterías, líneas e intersecciones [34]. A diferencia de los objetos rígidos, estas marcas de campo no tienen bordes y su apariencia varía significativamente dependiendo del punto de vista e iluminación. Como resultado, las métricas de evaluación de detección de objetos estándar basadas en superposición de cuadros delimitadores pueden no ser la mejor opción para reflejar cuán útiles son para la localización.

En este trabajo, se investigó el impacto de la calidad en detección de marcas en localización de robot humanoides. Se evaluaron tres detectores de objetos del estado del arte: YOLOv8, Deformable DETR [57], y RT-DETR, centrándose en su capacidad para proporcionar observaciones de puntos de referencia espacialmente precisas, más que en la precisión de las dimensiones de los objetos. El error de localización basado en el centro es una métrica más adecuada para evaluar el reconocimiento de marcas sin restricciones espaciales que las medidas basadas en la intersección sobre la unión (IoU). El módulo de detección

analizado en este artículo se integra en un marco de localización Monte Carlo (MCL) basado en visión [52], en el que las observaciones de puntos de referencia se utilizan para actualizar la estimación de la postura del robot.

1.1. Planteamiento del problema

Consideramos el problema de estimar la pose plana de un robot humanoide en un campo de fútbol, representada por el vector de estado $\mathbf{x} = (x, y, \theta)$. El robot está equipado con una cámara monocular y una unidad de medición inercial sin magnetómetro. La localización se realiza mediante un marco probabilístico que fusiona odometría y observaciones visuales de puntos de referencia [52].

En cada paso de tiempo, el sistema de percepción proporciona un conjunto de puntos de referencia detectados, cada uno asociado con una etiqueta de clase y una posición estimada en el espacio de imagen. Estas observaciones se convierten en mediciones angulares relativas al marco del robot y se utilizan para actualizar la creencia sobre la pose del robot. Dado que las estimaciones de distancia son poco confiables para puntos de referencia lejanos o parcialmente observados [34], el marco de localización se basa principalmente en información de orientación angular.

La precisión y consistencia de las detecciones de puntos de referencia afectan directamente la calidad de la actualización de la localización. Un sesgo espacial en las posiciones estimadas conduce a una ponderación incorrecta de las partículas y a una convergencia tardía del filtro. Por lo tanto, dado que el sistema opera con elementos del entorno no delimitados donde las dimensiones exactas de las cajas de detección (bounding boxes) resultan secundarias, el módulo de visión debe priorizar la exactitud de la coordenada posicional enviada al modelo de observación.

Los puntos de referencia del campo, como las intersecciones de líneas y las marcas del terreno, no corresponden a objetos finitos con geometría fija. Su apariencia visual está definida por el contraste y la topología, más que por su extensión física. En consecuencia, las anotaciones de cajas delimitadoras son inherentemente ambiguas, y pueden existir múltiples anotaciones válidas para un mismo punto de referencia.

Para abordar este desafío, evaluamos los modelos de detección de objetos en función de su capacidad para producir centros de puntos de referencia consistentes y espacialmente precisos. Este criterio de evaluación se alinea con los requisitos de la tarea de localización posterior.

Para evaluar la idoneidad de los detectores de objetos modernos en la localización de robots humanoides, evaluamos tres arquitecturas representativas: YOLOv8, Deformable DETR [57] y RT-DETR. Estos modelos fueron seleccionados para cubrir tanto paradigmas de detección basados en anclas como basados en transformers, así como diferentes compromisos entre precisión y eficiencia computacional.

YOLOv8 representa un detector de una sola etapa ampliamente adoptado, optimizado para rendimiento en tiempo real, mientras que RT-DETR introduce

una alternativa ligera basada en transformadores diseñada para mayor eficiencia. Deformable DETR actúa como un detector basado en transformers de alta precisión, con mecanismos de atención deformable que mejoran la localización espacial.

1.2. Hipótesis

En esta propuesta se trabajará con las siguientes hipótesis:

- Es posible localizar un robot bípedo observando marcas sin restricciones espaciales y usando filtros de partículas.
- Los transformadores visuales pueden tener mejores resultados en la detección y localización de marcas sin restricciones espaciales que los métodos basados en redes neuronales convolucionales.
- El error de localización basado en el centro es una medida adecuada para evaluar el desempeño de reconocimiento de objetos sin restricciones espaciales que las métricas tradicionales como la intersección sobre la unión.

1.3. Objetivos

- Construir un dataset de marcas de un campo de fútbol para el entrenamiento de modelos de reconocimiento.
- Comparar el desempeño de tres métodos: Yolov8, RT-DETR y Deformable-DETR.
- Desarrollar un sistema de localización basado en filtros de partículas usando como observaciones las marcas de un campo de fútbol.
- Implementar y evaluar el sistema de localización en un robot bípedo real.

1.4. Descripción del documento

En el capítulo 2 se abordan los conceptos necesarios para el desarrollo del sistema de localización usando marcas del campo de juego.

En el capítulo 3 se describe el sistema de localización visual. Se comienza con la construcción del dataset, el entrenamiento de los modelos y se finaliza con el algoritmo de localización.

El capítulo 4 presenta los resultados tanto del desempeño de la detección y localización de marcas como el desempeño del sistema de localización en general.

Finalmente en el capítulo 5 se dan las conclusiones y se esboza el trabajo futuro.

Capítulo 2

Marco teórico

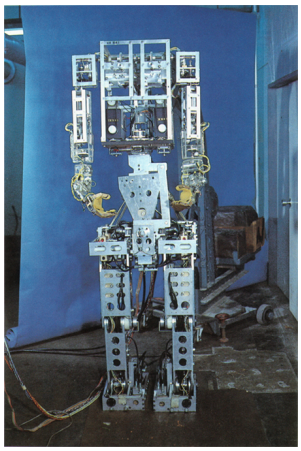
En este capítulo se abordan los conceptos básicos sobre robots bípedos autónomos, el problema de la localización, el problema del reconocimiento de marcas sin restricciones espaciales, la localización por el método de Monte Carlo y finalmente se aborda el trabajo relacionado con la propuesta de este trabajo.

2.1. Robots bípedos autónomos

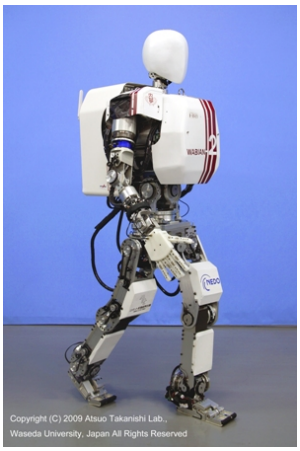
Los robots bípedos son aquellos que utilizan dos extremidades inferiores para locomoción, replicando el patrón de marcha humana. Una importante característica de los robots bípedos es su forma antropomórfica. Pueden ser construidos para tener las mismas dimensiones que los humanos; esto los convierte en excelentes vehículos para la investigación en interacciones humano-robot. [48] Desde una perspectiva histórica, el desarrollo de los robots bípedos ha estado marcado por importantes avances en actuadores, control y percepción, los cuales han permitido su evolución desde prototipos experimentales hasta plataformas de investigación más complejas.

En el sector comercial, Honda y Sony destacaron significativamente con el desarrollo de nuevos servomotores "inteligentes" que proporcionaron una actuación fuerte y flexible mediante control de lazo cerrado y detección de par. [47]. En 1973, en la Universidad Waseda, se presentó el primer robot antropomórfico, el WABOT-1, que contaba con sistemas de visión, conversación y control de extremidades. Más adelante se desarrolló el robot WABIAN-2R [48] con fines de investigación; fue diseñado para emular movimientos humanos, incluso bailar como uno.

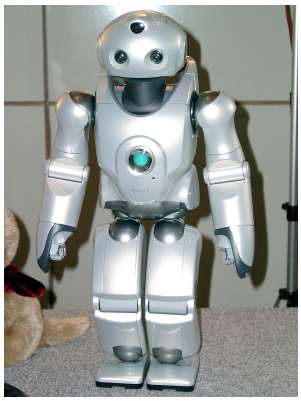
Años más tarde, en el año 1997, se originó el proyecto QRIO como SDR (Sony Dream Robot), buscando realizar un humanoide bípedo con fines de entretenimiento. Aquí, el desarrollo del ISA, un actuador triple compuesto por una unidad motriz, una de control embebida y otra unidad de engranaje, [19] fue la aportación técnica más significativa, logrando una salida dinámica de alta potencia a través de un actuador pequeño y ligero. [33]



(a) WABOT-1 (1973): primer humanoide a escala completa. GDL: 11 (piernas), 4 c/mano. Act.: hidráulicos (WL-5 + WAM-4). Altura: ∼1.60 m.



(b) WABIAN-2R (2006): primer humanoide con marcha de rodilla extendida. GDL: 41. Act.: motores DC Maxon + harmonic drive. Altura: 1.48 m. Peso: 64 kg.



(c) QRIO (2003): primer humanoide bípedo en correr. GDL: 38. Act.: ISA (motor + control + engranaje). Altura: 0.58 m. Peso: 7 kg.

Figura 2.1: Robots humanoides pioneros y sus características principales.

Como parte de la investigación de Honda en robótica humanoide, en 1986 se desarrolló la serie E (E0–E6), donde el E0 lograba únicamente marcha estática en línea recta, y el E2 (1991) alcanzó la marcha dinámica sobre superficie plana; la serie P (P1–P3, 1993–1997) incorporó torso y brazos, y redujo progresivamente peso y dimensiones para que el robot pudiera operar en entornos humanos. El resultado de ese proceso fue ASIMO (Advanced Step in Innovative Mobility), presentado en 2000, cuyas capacidades se agrupan en: movilidad con marcha estable en superficies irregulares, control ZMP (Zero Moment Point) y posicionamiento de pies, además de carrera a 9 km/h alcanzada en el modelo 2011; ejecución de tareas como manipulación con manos de cinco dedos; comunicación y reconocimiento de voz e imagen y expresión corporal para interacción con personas. En su etapa más avanzada, ASIMO incorporó una arquitectura de generación autónoma de comportamiento denominada intelligence loop.[46]

En conjunto, estos desarrollos marcaron las bases tecnológicas de los robots bípedos modernos, al integrar avances en actuadores, control de estabilidad, percepción y autonomía. A partir de estas contribuciones pioneras, la investigación y el desarrollo industrial evolucionaron hacia plataformas comerciales cada vez más robustas, versátiles y adaptadas a aplicaciones prácticas.

Figura 2.2: Robot NAO v6.

2.2. Evolución de las plataformas comerciales

El desarrollo de la robótica humanoide ha sido impulsado por la creación de plataformas estandarizadas lo que ha permitido un avance significativo en el desarrollo de algoritmos de locomoción, manipulación e inteligencia artificial. Estas plataformas han ido evolucionando a lo largo de los años atendiendo las necesidades no sólo de la industria, si no también del ámbito académico.

Transitando de prototipos con limitaciones de movimiento hasta las plataformas más modernas que hoy en día lideran esta área, con locomoción más natural, movimientos menos rígidos y sistemas de visión artificial más robustos.

2.2.1. Modelo educativo *Nao* de Aldebaran

El robot NAO fue desarrollado por Aldebaran Robotics (Francia) en el año 2004 y lanzado comercialmente en 2008 como una plataforma abierta con fines de investigación y educación. Sus características incluyen una altura de 0.57 m, peso aproximado de 4.5 kg, un diseño cinemático de pelvis y un sistema de actuación propietario basado en motores DC con escobillas, 25 grados de libertad distribuidos en cabeza (2 DOF), brazos (5 DOF), pelvis (1 DOF), piernas (5 DOF) y manos (1 DOF), con encoders magnéticos rotatorios. [21]. Su plataforma de software es modificable por el usuario y tanto la cabeza como los antebrazos son módulos intercambiables para facilitar su evolución [21].

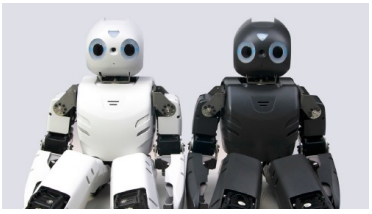
NAO constituyó el primer robot bípedo asequible y de alto rendimiento disponible para laboratorios de investigación y el mercado general, tiene una herramienta central de programación de movimiento, que es el software Choregraphe, una interfaz gráfica que permite definir y refinar comportamientos motrices.[21] Este diseño integral fue determinante para que fuera seleccionado como plataforma estándar de la RoboCup Standard Platform League en 2008, en sustitución del robot cuadrúpedo AIBO de Sony, siendo así utilizado en diversas universidades y laboratorios de todo el mundo. [21]

El robot experimentó sucesivas mejoras de hardware: la versión 4 *Next Gen* (2011) incorporó cámaras de alta densidad a 960p y casi duplicó la capacidad de batería de 27.6 Wh a 48.6 Wh; la versión 5 *Evolution* (2014) actualizó el

procesador a un Intel Atom E3845, mejoró las cámaras a sensores OmniVision OV5640 de 5 megapíxeles y reforzó las articulaciones con piezas metálicas de mayor durabilidad; y la ultima versión, el NAOv6 (2018) mantuvo el mismo procesador, pero introdujo nuevos motores con mayor vida útil y menor sobrecalentamiento, conectividad Bluetooth y Wi-Fi mejorada, además de actualizar el sistema operativo a NAOqi OS 2.8 con capacidades de inteligencia artificial más avanzadas. [4].

2.2.2. Robot DARwIn-OP OP3 de ROBOTIS

El DARwIn-OP (*Dynamic Anthropomorphic Robot with Intelligence - Open Platform*) fue desarrollado por ROBOTIS en colaboración con Virginia Tech, Purdue University y la Universidad de Pensilvania. Su tercera generación, el OP3, sustituyó los actuadores MX-28 por servomotores XM-430 con protocolo DYNAMIXEL 2.0, mejorando torque y control por corriente, y reemplazó la SBC basada en Intel Atom por un Intel NUC con procesador i3, lo que incrementó sustancialmente la capacidad de cómputo [41]. Cuenta con 20 grados de libertad y fue desarrollado íntegramente bajo ROS (*Robot Operating System*), facilitando la integración con el ecosistema de paquetes de código abierto [54]. Su principal aportación fue consolidarse como la plataforma de referencia para investigación en robótica humanoide a bajo costo, capaz de ganar la liga *Kid Size* de RoboCup en 2011, 2012 y 2013.



(a) DARwIn-OP2

(b) DARwIn-OP3

Figura 2.3

2.2.3. Robot G1 de Unitree

El robot Unitree G1 fue presentado en 2024 por Unitree Robotics como plataforma humanoide de propósito general orientada a investigación, educación

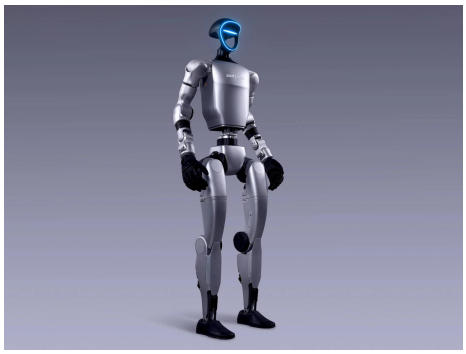


Figura 2.4: Unitree G1

e inteligencia artificial encarnada (*embodied AI*) [42]. En su configuración base dispone de 23 grados de libertad (6 por pierna, 5 por brazo y 1 de cintura). Mide 132 cm de altura, pesa alrededor de 35 kg y alcanza una velocidad máxima de 2 m/s; sus articulaciones están accionadas por motores síncronos de imán permanente (PMSM) de rotor interno de baja inercia, con torque máximo de rodilla de 90 Nm en la versión base y 120 Nm en la versión EDU. Integra sensores como LiDAR 3D Livox MID-360, cámara de profundidad Intel RealSense D435i, IMU y encoders duales por articulación para retroalimentación de posición y torque. En términos de cómputo, la versión EDU incorpora un módulo NVIDIA Jetson Orin NX con 100 TOPS de rendimiento para inferencia de inteligencia artificial en tiempo real, aprendizaje por refuerzo y operación autónoma. La plataforma es compatible con ROS2, C++ y Python [53].

Como hallazgo destacado en el ámbito competitivo y de desempeño, el G1 ha demostrado capacidades de locomoción dinámica avanzada, incluyendo salto largo de 1.4 m, giro lateral y transición de posición horizontal. [53]

2.2.4. Plataformas de desarrollo Booster K1 y T1

Booster Robotics, fundada en 2023 por ingenieros del Laboratorio de Control de Robots de la Universidad de Tsinghua, desarrolló dos plataformas humanoides complementarias orientadas a investigación, educación e inteligencia artificial encarnada: el T1 y el K1.

El **Booster T1** es la plataforma de mayor escala, con 118 cm de altura y 30 kg de peso [13]. En su configuración base dispone de 23 DOF distribuidos en piernas (6 DOF $\times$ 2), brazos (4 DOF $\times$ 2), cintura (1 DOF) y cabeza (2 DOF), ampliables a 31 DOF con grippers o hasta 41 DOF con manos diestras. Sus actuadores son juntas de control de fuerza con encoders duales que soportan control mixto de torque, velocidad y posición, con un torque pico máximo de 130 Nm [43]. El sistema de cómputo combina un procesador Intel i7 de 14 núcleos con una GPU NVIDIA Jetson AGX Orin de 200 TOPS para tareas de percepción, aprendizaje por refuerzo y control en tiempo real. Su sistema de

sensores integra cámara RGB-D, IMU de 9 ejes, arreglo de micrófonos y altavoz, con conectividad Wi-Fib 6 y Bluetooth 5.2. La plataforma es completamente de código abierto, compatible con ROS2 y con entornos de simulación Isaac Sim, MuJoCo y Webots, lo que facilita flujos de trabajo *sim-to-real*. El **Booster K1**

Figura 2.5: Booster T1

es la plataforma compacta del ecosistema, con 95 cm de altura y 19.5 kg de peso. Cuenta con 22 DOF, piernas (6 DOF $\times$ 2), brazos (4 DOF $\times$ 2) y cabeza (2 DOF), accionados por juntas de control de fuerza completo con encoders duales y torque máximo de rodilla de 60 Nm. Según la configuración, su cómputo varía entre 48, 117 y 200 TOPS mediante módulos NVIDIA Jetson Orin NX, permitiendo escalar la carga de trabajo desde prototipos hasta inferencia de redes neuronales embebidas [24].

Figura 2.6: Booster K1

Al igual que el T1, es compatible con ROS2, Python y C++, e incluye soporte para reconocimiento de voz (ASR), síntesis de texto a voz (TTS) y algoritmos de visión como YOLO [9]. Su batería admite entre 40 y 80 minutos de operación activa dependiendo de la versión [24].

Ambas plataformas comparten una arquitectura de hardware abierta con documentación y SDK de libre acceso [56], marcos de aprendizaje por refuerzo basados en Isaac Lab publicados públicamente [12]. Su validación competitiva es su hallazgo más destacado: el T1 obtuvo el primer lugar en la categoría AdultSize de RoboCup 2025 y en las competencias de carrera y navegación del IEEE-RAS Humanoids 2024, mientras que el K1 conquistó el primer y segundo lugar en la categoría KidSize de RoboCup 2025 con ambos finalistas utilizando hardware idéntico, convirtiendo la competencia en una prueba pura de algoritmos de software [24, 56].

2.3. Redes neuronales profundas

Las *redes neuronales profundas* **DNN** fueron desarrolladas en el año 2006 como una extensión del *Perceptrón Multi Capa*, basadas en la idea de que redes neuronales más profundas pueden aprender más información [6]. El aprendizaje profundo (*Deep Learning*) se fundamenta en el uso de arquitecturas de redes neuronales artificiales compuestas por múltiples capas, diseñadas para extraer progresivamente características de alto nivel a partir de datos crudos [15]. Este enfoque es fundamental en la robótica autónoma moderna, ya que dota a los agentes de capacidades de percepción complejas necesarias para interpretar entornos dinámicos [6].

A diferencia de los métodos tradicionales de aprendizaje superficial, cuyos algoritmos presentan una capacidad limitada para representar funciones complejas con un número finito de muestras y unidades computacionales, el aprendizaje profundo logra aproximar funciones complejas mediante el aprendizaje de estructuras de red no lineales profundas. Su principal característica radica en construir modelos con múltiples capas ocultas que, a través de grandes volúmenes de datos de entrenamiento, aprenden características progresivamente más abstractas: los *modelos profundos* son el medio, y el *aprendizaje de características* es el objetivo [15]. Esto se traduce en dos ventajas fundamentales: *la profundidad del modelo*, que generalmente comprende decenas de capas o más, y *la transformación capa por capa* de la representación de los datos desde el espacio original hacia un nuevo espacio de características, facilitando así la clasificación o predicción sin depender de reglas definidas manualmente [15].

Uno de los principales tipos de redes neuronales profundas son las *redes neuronales convolucionales* **CNNs**. Pueden ser entrenadas para clasificación, regresión y reconocimiento de imágenes [6].

2.3.1. Modelo de un perceptrón

El perceptrón es un sistema auto-organizado, es decir, que es capaz de aprender y modificar su comportamiento de forma autónoma a partir de datos. Propuesto por Rosenblatt [44]. Este modelo matemático se inspiró en el funcionamiento de una neurona biológica, consiste en una retina de sensores que están conectados [11]. Su arquitectura se basa en un discriminador lineal que recibe un vector de señales de entrada x, las multiplica por un vector de pesos sinápticos w, y suma un término de sesgo (*bias*) b. El resultado de esta combinación lineal pasa a través de una función de activación no lineal (tradicionalmente, una función escalón) para producir una única señal de salida y [20].

Matemáticamente, la salida de un perceptrón se define como:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

donde f es la función de activación.

Aunque el perceptrón sentó las bases algorítmicas del aprendizaje automático al introducir el concepto de pesos ajustables basados en el error, su principal

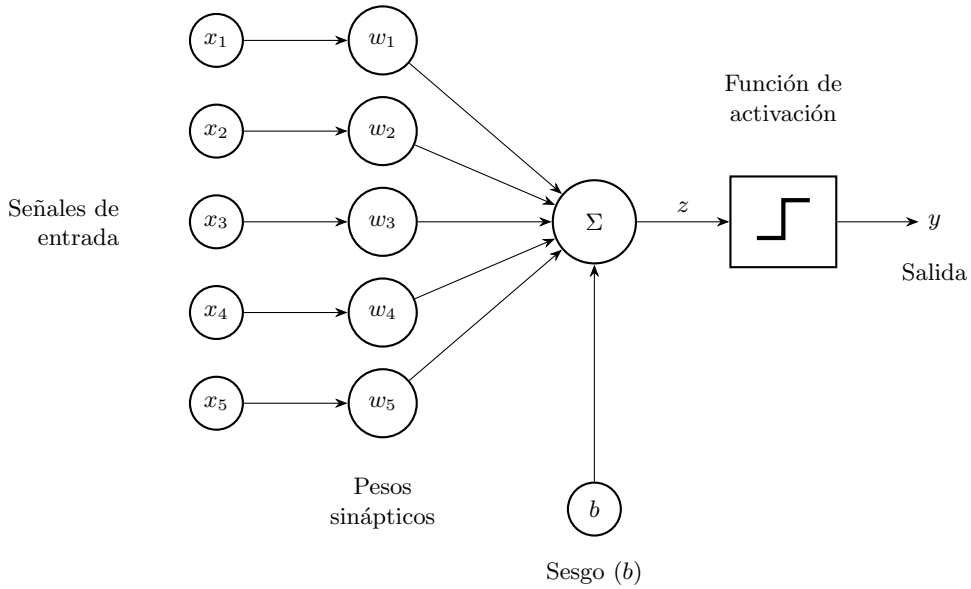


Figura 2.7: Diagrama del Perceptrón simple con cinco entradas, pesos sinápticos, unión sumadora y función de activación escalón.

limitación algorítmica radica en que solo es capaz de resolver problemas que son linealmente separables (como las funciones lógicas AND y OR, pero no la función XOR)[6]. Esta restricción estructural motivó el desarrollo de arquitecturas más profundas.

Para superar las restricciones del perceptrón simple, se desarrolló el Perceptrón Multicapa (*MLP*, por sus siglas en inglés). Un MLP es una red neuronal prealimentada (*FNN*) que incorpora una o más "capas ocultas" de neuronas interconectadas entre la capa de entrada y la capa de salida [6]. En esta arquitectura, la salida de cada neurona es una capa que se convierte en la entrada de las neuronas de la capa subsiguiente, estableciendo un flujo de información secuencial.

El poder representacional del MLP proviene de la inclusión de funciones de activación continuas y no lineales (como la función Sigmoide, Tangente Hiperbólica o ReLU) en sus capas ocultas. Casi cualquier función no lineal puede utilizarse para este propósito, a excepción de las funciones polinomiales. Gracias a estas no linealidades, el modelo adquiere la capacidad de aproximar cualquier función continua, convirtiéndose en un aproximador universal. Esta propiedad está respaldada formalmente por dos resultados teóricos fundamentales [15]:

- **Teorema de Kolmogorov:** Dada cualquier función continua $f : [0, 1]^n \to \mathbb{R}^m$, f puede ser implementada por una red neuronal prealimentada (*feed-forward*) de tres capas, donde la capa de entrada tiene n neuronas, la capa oculta tiene $2n + 1$ neuronas, y la capa de salida tiene m neuronas.

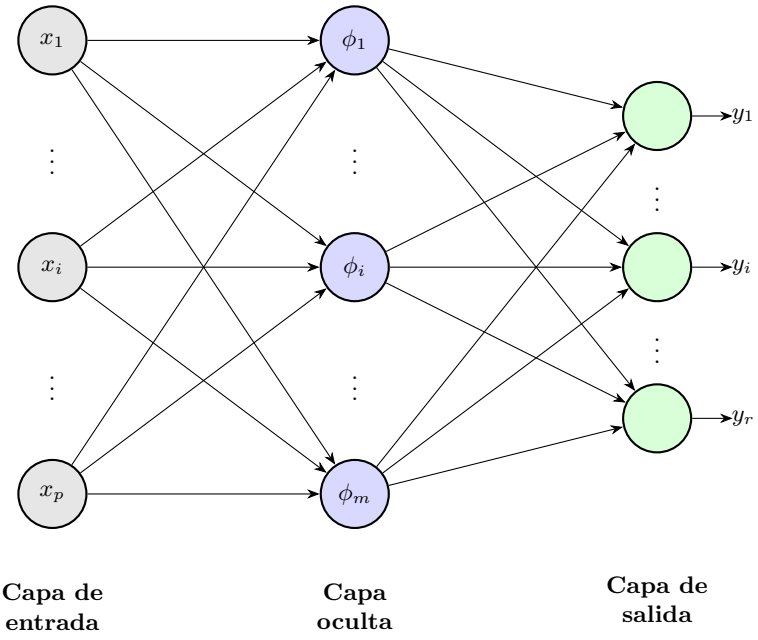


Figura 2.8: Arquitectura del Perceptrón Multicapa (MLP) con p entradas, m neuronas en la capa oculta y r salidas.

- **Teorema de aproximación L^2:** Dado cualquier $\varepsilon > 0$, para cualquier función continua de tipo L^2 $f : [0,1]^n \to \mathbb{R}^m$, existe una red MLP de tres capas capaz de aproximar f con un error cuadrático menor que ε.

Sin embargo, aunque una red MLP de tres capas es teóricamente suficiente para aproximar cualquier función continua, en la realidad el número de neuronas requeridas en la capa oculta puede crecer de forma exponencial con la complejidad del problema. Popescu et al. [36] señalan que el MLP opera como un clasificador que forma superficies de decisión no lineales en el espacio de entrada, y que el número de neuronas en una capa debe ser más de tres veces mayor que el de la capa siguiente, para que cada capa pueda proveer al menos tres aristas por cada región convexa identificada por la capa posterior. Esto implica que, apilar múltiples capas ocultas resulta más viable computacionalmente que expandir indefinidamente una sola capa, dando lugar a las arquitecturas profundas. [36].

En aplicaciones de robótica y visión por computadora, el apilamiento masivo de estas capas da lugar a las 'redes neuronales profundas"(*Deep Neural Networks*), las cuales son capaces de aprender representaciones jerárquicas complejas; por ejemplo, las primeras capas pueden detectar bordes en un campo de fútbol, mientras que las capas más profundas pueden identificar patrones semánticos complejos, como la forma de un robot o un balón [20].

2.3.2. Redes neuronales convolucionales

Una red neuronal convolucional (*Convolutional Neural Network*, CNN) es un tipo de red neuronal de alimentación directa, (*Feedforward Neural Network, FNN*), que es capaz de extraer características a partir de datos con estructuras convolucionales. A diferencia de los métodos tradicionales de visión por computadora, en los que las características debían diseñarse manualmente, las *CNN*, como una arquitectura de aprendizaje profundo, fueron propuestas para minimizar los requerimientos de preprocesamiento de datos; aprenden de manera automática representaciones relevantes, lo que ha contribuido de forma decisiva al avance del reconocimiento y la detección visual en múltiples aplicaciones [26, 27, 15].

La idea central de una *CNN* es el uso de la operación de **convolución**, mediante la cual, un conjunto de filtros o *kernels* recorre la imagen de entrada para detectar patrones locales, tales como bordes, esquinas, texturas o cambios de intensidad [35]. Este proceso se organiza en una estructura jerárquica: una pequeña región de la imagen sirve como entrada en el nivel más bajo, y la información se transmite progresivamente hacia capas superiores, donde cada capa aplica un filtro digital para extraer las características más significativas de los datos observados [15]. Gracias a este mecanismo, las *CNN* son capaces de capturar características invariantes a traslaciones, cambios de escala y rotaciones, ya que el campo receptivo local permite a las neuronas acceder a las características más elementales de la imagen, como bordes orientados o esquinas [15].

A diferencia de una red totalmente conectada, donde cada neurona recibe como entrada todos los píxeles, las *CNN* utilizan tres propiedades fundamentales: la **conectividad local**, el **compartimiento de pesos** y el **submuestreo espacial** (*downsampling*).

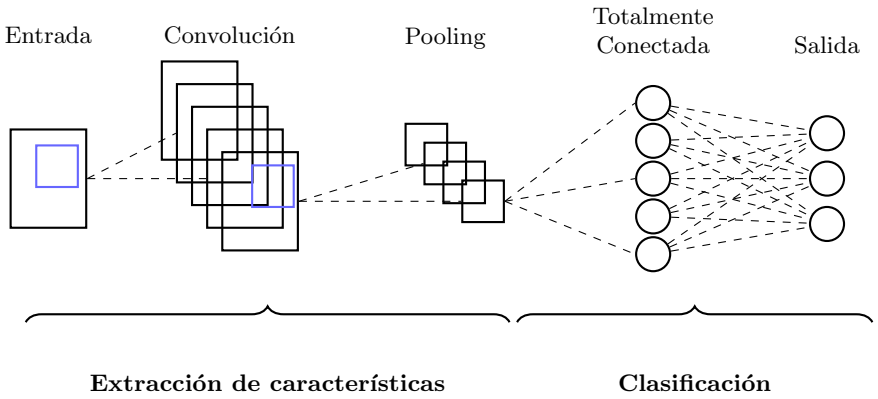


Figura 2.9: Esquema básico de una red neuronal convolucional.

La conectividad local permite reducir parámetros y aumentar la velocidad de convergencia, debido a que ahora una neurona procesa únicamente una sección reducida de la imagen, mientras que el compartimiento de pesos permite

que un mismo filtro se utilice en todas las posiciones espaciales. Por su parte, el submuestreo, implementado frecuentemente mediante capas de agrupamiento (*pooling*), reduce progresivamente la resolución de los mapas de características; esto no solo disminuye aún más el costo computacional y el riesgo de sobreajuste, sino que otorga al modelo invarianza frente a pequeñas traslaciones y distorsiones en la entrada [26, 27].

La imagen 2.9 muestra la estructura básica de una CNN como una red neuronal multicapa formada por planos bidimensionales de neuronas independientes. Primero, la imagen de entrada pasa por una etapa de convolución, donde filtros entrenables y un sesgo extraen características locales y generan mapas de características; después, una etapa de pooling resume esa información y reduce su tamaño. Finalmente, los mapas obtenidos se transforman en un vector y se envían a una capa totalmente conectada para producir la salida de clasificación.

2.3.3. Métodos de entrenamiento

El aprendizaje de una red neuronal profunda consiste en un proceso de optimización iterativa, donde el objetivo es ajustar los pesos sinápticos y los sesgos de la red para minimizar una función de pérdida o coste (*Loss Function*). Esta función calcula empíricamente la discrepancia entre las predicciones del modelo y las etiquetas reales de los datos de entrenamiento [20].

El algoritmo fundamental que hace posible el entrenamiento de redes profundas es el **Descenso del Gradiente**. Este método actualiza los parámetros de la red en la dirección opuesta al gradiente de la función de pérdida con respecto a dichos parámetros. La regla de actualización general para un peso w en la iteración t es:

$$w_{t+1} = w_t - \eta \frac{\partial L}{\partial w_t}$$

donde η es la tasa de aprendizaje y L es la función de pérdida.

Para calcular eficientemente estos gradientes a través de múltiples capas ocultas, se emplea el algoritmo de **Retropropagación** (*Backpropagation*). Este algoritmo aplica sistemáticamente la regla de la cadena del cálculo diferencial, propagando el error desde la capa de salida hacia atrás, hasta llegar a la capa de entrada. [6].

En la práctica moderna de entrenamiento de modelos visuales como YOLO o DETR, rara vez se evalúa el gradiente sobre todo el conjunto de datos a la vez debido a restricciones de memoria de GPU. En su lugar, se utilizan variantes como el **Descenso de Gradiente Estocástico (SGD)** o el SGD por minilotes (*Mini-batch SGD*), los cuales estiman el gradiente utilizando subconjuntos aleatorios de los datos. Además, para acelerar la convergencia computacional y evitar que el entrenamiento converja a mínimos locales cuando la función de pérdida presenta una geometría compleja, el estado del arte emplea optimizadores adaptativos, tales como Adam (Estimación de Momento Adaptativo) o AdamW, que ajustan dinámicamente la tasa de aprendizaje para cada parámetro basándose en los momentos históricos de los gradientes [20]. Otra característica del proceso de entrenamiento es que los distintos ejemplos son presentados más de una

vez a la red. Cuando todo el conjunto de datos de entrenamiento es presentado completo, sin importar el orden en que se presenten los ejemplos, a esto se le conoce como una época (*epoch*).

Es importante entender que los pesos, si bien son ajustados durante el entrenamiento, no son inicializados a cero ante cada nuevo ejemplo que se presenta. Por lo tanto, los pesos representan lo aprendido por el perceptrón. Normalmente se presentan los datos de entrada utilizando más de una época para poder minimizar los errores mediante el ajuste de los pesos.

2.3.4. Modelo YOLOv8

YOLOv8, desarrollado por Ultralytics, forma parte de la familia de detectores de objetos de una sola etapa (*You Only Look Once*). YOLOv8 utiliza una arquitectura de red neuronal convolucional profunda (CNN) basada en una columna vertebral de red (*backbone*) CSPDarknet modificado y un cabezal de detección desacoplado [22]. Esta arquitectura está optimizada para inferencia de alto rendimiento, lo que la convierte en una opción muy utilizada para la percepción en tiempo real, incluyendo el dominio de la RoboCup Humanoid League [39].

Una distinción clave de YOLOv8 es su diseño libre de anclas (*anchor-free*). A diferencia de iteraciones anteriores que dependían de cajas ancla *anchor boxes* predefinidas, YOLOv8 predice directamente el centro de un objeto y las distancias a los bordes de la caja delimitadora (*bounding box*). Este enfoque simplifica la función de pérdida y mejora la capacidad del modelo para generalizar a través de diferentes relaciones de aspecto [49, 50]. Para la localización de humanoides, la eficiencia de YOLOv8 permite que el pipeline de visión opere a altas tasas de fotogramas, lo cual es crucial para mantener un estado de creencia actualizado [51].

Sin embargo, al ser un modelo basado en CNN, YOLOv8 depende de campos receptivos locales, lo que puede limitar su capacidad para capturar dependencias espaciales de largo alcance en comparación con los mecanismos de atención presentes en los Transformers. En escenarios de puntos de referencia poco estructurados, esto puede traducirse en una mayor varianza en la estimación del centro de la detección, especialmente cuando los hitos están parcialmente ocluidos o sufren distorsión de perspectiva marcada [22].

2.4. Transformadores visuales

Los modelos de visión basados en transformadores han emergido como una alternativa de alto rendimiento a las redes neuronales convolucionales (CNN) para tareas de percepción visual en robótica. Su principal ventaja radica en el mecanismo de atención *Self-Atention*(autoatención), que permite modelar dependencias globales entre todas las regiones de una imagen simultáneamente [14]. En el contexto de la localización robótica, esta capacidad de razonamiento espacial global es importante para identificar puntos de referencia estructurados

(*landmarks*) como intersecciones de líneas o porterías en entornos dinámicos con oclusiones parciales.

2.4.1. Modelos basados en transformadores

La arquitectura Transformer fue concebida originalmente para el procesamiento de lenguaje natural por Vasawani et al. [55], basándose íntegramente en mecanismos de atención multicabeza (*Multi-Head Attention*) que relacionan cada elemento de una secuencia con todos los demás a través de ternas de vectores: Consulta (*Query*), Clave (*Key*) y Valor (*Value*). En lugar de aplicar una única función de atención con consultas, claves y valores de dimensión d_{model}, el Transformer proyecta linealmente estas representaciones h veces mediante matrices distintas, obteniendo subespacios de menor dimensión en los que la atención se calcula en paralelo. Cada cabeza produce una salida de dimensión d_v, y posteriormente todas las salidas se concatenan y se proyectan de nuevo para obtener la representación final. Esta estrategia permite que el modelo atienda simultáneamente a información ubicada en distintos subespacios representacionales y en diferentes posiciones de la secuencia, lo que sería menos efectivo con una sola cabeza de atención [55].

Una función de atención puede describirse como un mapeo de una consulta y un conjunto de pares clave-valor a una salida que se calcula como una suma ponderada de los valores, donde el peso asignado a cada valor se obtiene mediante una función de compatibilidad entre la consulta y la clave correspondiente.La función de atención se expresa como:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

donde d_k es la dimensionalidad de las claves, que escala el producto punto para estabilizar los gradientes, Q, K y V son matrices que guardan un conjunto de consultas, claves y valores, respectivamente [55]. La atención multicabeza se define como:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

donde

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

y las matrices de proyección son $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ y $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ [55].

Como se muestra en la Figura 2.10, la arquitectura del transformer se compone por un codificador (*encoder*) y un decodificador (*decoder*), cada uno replicado N veces. El encoder recibre la secuencia de entrada como un *Input Embedding* al que se le suma un elemento de codificación posicional (*Positional Encoding*) para conservar el orden de los elementos, y la procesa mediante bloques de atención multicabeza (*Multi-Head Attention*) y redes prealimentadas (*Feed Forward*), con conexiones residuales y normalización (*Add & Norm*) después de cada subcapa.

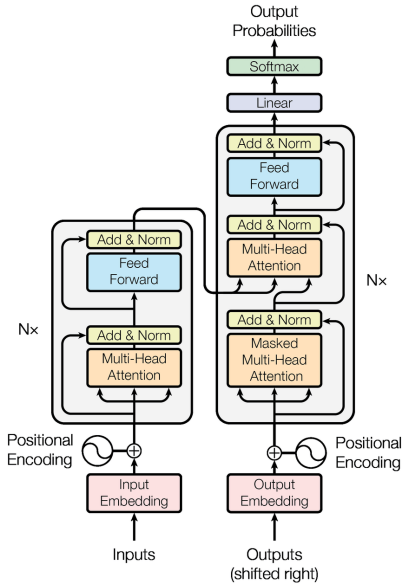


Figura 2.10: Arquitectura del Transformer tomada de [55]

El decoder sigue una estructura similar, pero incorpora además una capa de atención multicabeza enmascarada (*Masked Multi-Head Attention*) que impide que cada posición atienda a posiciones futuras, y una capa de *Multi-Head Attention* cruzada que conecta la salida del encoder con el decoder. Finalmente, una capa lineal seguida de una función *Softmax* produce las probabilidades de salida sobre el vocabulario.

Los transformers fueron adaptados al dominio visual por Dosovitskiy et al. en el *Vision Transformer* (ViT), publicado en 2020 [17]. La idea central del ViT es dividir una imagen de entrada en una cuadrícula de tamaño fijo (e.g., 16×16 píxeles), llamada parche, proyectar linealmente cada parche en un vector de baja dimensionalidad (*embedding*), y tratar la secuencia resultante como si fuera una secuencia de palabras para un Transformer estándar. A esta secuencia de *embeddings* se le suman codificaciones posicionales aprendibles para preservar al información espacial, ya que la operación de atención es intrínsicamente invariante al orden. Aunque el ViT original demostró rendimientos superiores a las CNN en tareas de clasificación a gran escala, su costo computacional crece cuadráticamente con el número de parches, lo que motiva el desarrollo de arquitecturas más eficientes [32].

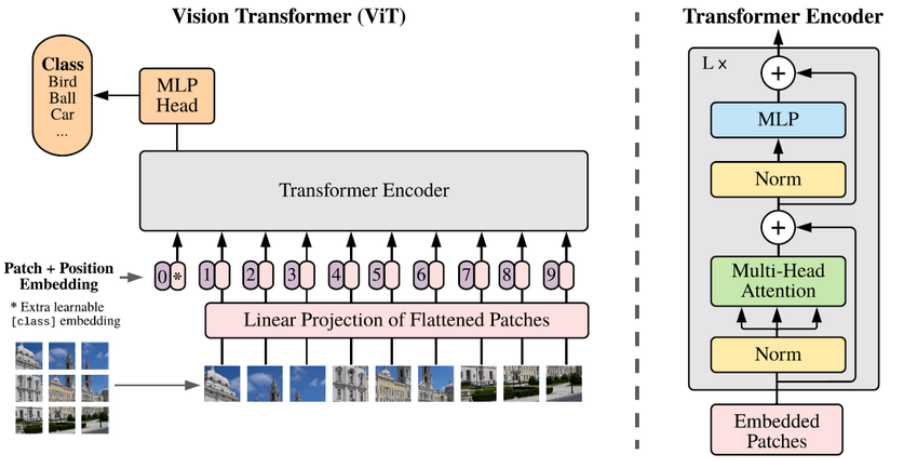


Figura 2.11: Arquitectura del Vision Transformer (ViT) propuesta por Dosovitskiy et al.[17].

En la figura 2.11, la imagen de entrada se divide en parches de tamaño fijo, los cuales se aplanan y se proyectan linealmente (*Linear Projection of Flattened Patches*) para obtener una secuencia de *embeddings*. A esta secuencia se le antepone un token especial `[class]` (posición 0*) y se le suma un *Positional Embedding* para preservar la información espacial de cada parche. Los tokens resultantes se procesan mediante un *Transformer Encoder* compuesto por L bloques apilados, cada uno con normalización (*Norm*), atención multi cabeza (*Multi-Head Attention*) y una red densa (*MLP*), con conexiones residuales en cada subcapa. Finalmente, la representación del token `[class]` es procesada por un cabezal de clasificación (*MLP Head*) que produce las probabilidades de salida para cada clase. A diferencia del Transformer original, el ViT emplea únicamente el encoder, prescindiendo del decoder al tratarse de una tarea de clasificación y no de generación de secuencias.

2.4.2. Modelo RT-DETR

El *Real-Time DEtection TRansformer* (RT-DETR), propuesto por Zhao et al. en 2024, representa el primer detector de objetos basado en transformadores que logra rendimiento en tiempo real, superando a las familias YOLO tanto en velocidad como en precisión [30]. En el trabajo original, se reporta que el modelo RT-DETR-R50 alcanza aproximadamente 53.1 % AP en COCO con alrededor de 108 FPS en GPU T4, mientras que RT-DETR-R101 obtiene 54.3 % AP con 74 FPS, superando en ambos casos a versiones avanzadas de YOLO en términos de velocidad y exactitud [30].

RT-DETR conserva la idea de un detector de extremo a extremo sin post-procesamiento NMS propia de DETR, pero introduce un codificador híbrido y eficiente, junto con un *backbone* convolucional, para reducir el costo compu-

tacional y permitir detección en tiempo real [30, 38]. El modelo también soporta ajuste flexible de la velocidad de inferencia mediante el número de capas del decodificador, sin necesidad de reentrenar, lo que lo hace adaptable a distintos escenarios de borde y robótica [30]. Esta combinación de arquitectura transformers y optimización para tiempo real lo convierte en una alternativa relevante para pipelines de visión robótica, aunque con requerimientos de cómputo y memoria típicamente mayores que los detectores basados en CNN, como YOLO [30, 38].

2.4.3. Modelo Deformable-DETR

El marco original DETR (*DEtection TRansformer*), introducido por Carion et al. en 2020 [14], reformuló la detección de objetos como un problema de predicción de conjuntos (*set prediction*) mediante una red transformador codificador-decodificador entrenada extremo a extremo, eliminando componentes artesanales como los cuadros ancla y la supresión no máxima (NMS) [8]. Sin embargo, su adopción práctica fue limitada por su lenta convergencia (requería $\sim$500 épocas de entrenamiento) y su dificultad para detectar objetos pequeños, ambos problemas derivados del uso de atención densa en los mapas de características completos.

Deformable DETR extiende el marco original de DETR mediante la introducción de módulos de **atención deformable multi-escala**, que atienden a un conjunto disperso y adaptativo de puntos de muestreo alrededor de ubicaciones de referencia en lugar de atender a la totalidad del mapa de características [57]. Para cada consulta q de atención, el módulo aprende a predecir un pequeño conjunto de desplazamientos *offsets* (Δp_{mqk}) con respecto a una ubicación de referencia $\hat{p}_q$, de modo que la función de atención deformable queda definida como:

$$\text{DeformAttn}(z_q, \hat{p}_q, x) = \sum_{m=1}^{M} W_m \left[\sum_{k=1}^{K} A_{mqk} \cdot W'_m x \left(\hat{p}_q + \Delta p_{mqk} \right) \right]$$

donde z_q es el vector de características de la consulta q, x es el mapa de características de entrada sobre el que se realiza el muestreo, M es el número de cabezas de atención, K es el número de puntos de muestreo por cabeza, A_{mqk} son los pesos de atención aprendidos, W_m, W'_m son matrices de proyección lineal [57].

Este diseño reduce la complejidad computacional de $\mathcal{O}(HWC^2)$ de la atención densa a $\mathcal{O}(2HWC \cdot K)$ [14], donde H y W son las dimensiones espaciales del mapa de características, C es el número de canales y K es el número de puntos de muestreo por cabeza, mejorando drásticamente la velocidad de convergencia y la precisión de localización para objetos a múltiples escalas.

A diferencia de los detectores basados en anclas, Deformable DETR no depende de *priors* predefinidos de cajas delimitadoras, lo que lo hace menos sensible a las variaciones de escala en los datos de entrenamiento. Esta propiedad

es particularmente beneficiosa para detectar puntos de referencia no restringidos, cuyos tamaños de caja delimitadora varían ampliamente entre anotaciones. Combinado con la capacidad de procesamiento multi-escala mediante mapas de características FPN (*Feature Pyramid Network*), Deformable DETR logra un excelente compromiso entre precisión de localización y eficiencia computacional [14].

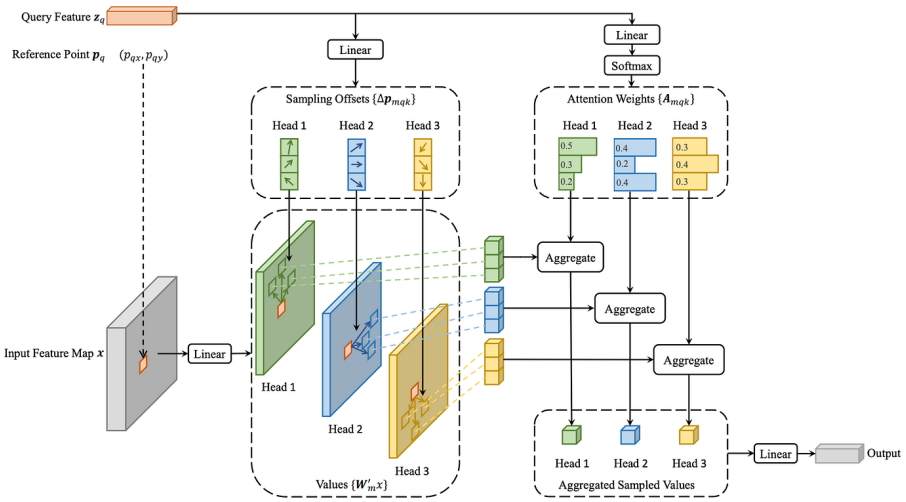


Figura 2.12: Ilustración del módulo de atención deformable propuesto en Deformable DETR [57].

En la figura 2.12 se muestra que a partir de una característica de consulta $\mathbf{z}_q$ y un punto de referencia $\mathbf{p}_q = (p_{qx}, p_{qy})$, el módulo predice para cada cabeza de atención un conjunto de desplazamientos de muestreo $\{\Delta\mathbf{p}_{mqk}\}$ y pesos de atención $\{A_{mqk}\}$. Los desplazamientos se aplican sobre el mapa de características de entrada x para obtener un conjunto reducido de puntos de muestreo, cuyos valores $\{W'_m x\}$ se agregan ponderadamente por cabeza y se proyectan linealmente para producir la salida.

En este trabajo se emplea Deformable DETR como detector de puntos de referencia visuales (*landmarks*) en el entorno de nuestro robot. Las detecciones producidas por el modelo se utilizan como observaciones z_t dentro de un esquema de localización probabilística basado en filtros de partículas, descrito en la siguiente sección. La elección de un modelo basado en transformadores deformables se justifica por su capacidad para manejar objetos de escala variable sin restricciones geométricas rígidas, lo que resulta adecuado para un entorno en el que los puntos de referencia pueden aparecer a distintas distancias y orientaciones.

2.5. Localización por filtros bayesianos

La localización de un robot móvil es el problema de estimar las coordenadas de un robot relativas a un marco de referencia externo. Es decir, estimar su pose $x_t = (x, y, \theta)^T$ (posición y orientación) en un mapa conocido del entorno a partir de su historial de comandos de control $u_{1:t}$ y observaciones sensoriales $z_{1:t}$. Este problema de localización específico es conocido como localización global, en donde un robot es puesto en algún lugar de un mapa conocido y tiene que localizarse por sí sólo desde el principio.

El enfoque probabilístico modela la *creencia* momentánea del robot por medio de una función de densidad de probabilidad sobre el espacio de todas las localidades. Thrun et al.[51] definen la creencia como el conocimiento interno del robot sobre el estado del ambiente y la representan a través de distribuciones de probabilidad condicional.

Las distribuciones de *creencia* (bel) son probabilidades posteriores sobre variables de estado x_t en el tiempo t condicionadas a todas las mediciones sensoriales$z_1 : t$ y comandos de control $u_1 : t$ anteriores:

$$bel(x_t) = p(x_t \mid z_{1:t},\, u_{1:t})$$

En el presente trabajo, será de utilidad calcular una distribución posterior antes de incorporar z_t, justo después de ejecutar comandos de control u_t. Dado que las detecciones visuales pueden estar ausentes en ciertos fotogramas por oclusión, baja confianza o ausencia de puntos de referencia en el campo de visión, el filtro debe ser capaz de propagar la creencia únicamente a partir del modelo de movimiento.

$$\overline{bel}(x_t) = p(x_t \mid z_{1:t-1},\, u_{1:t})$$

Esta distribución de probabilidad se denomina *predicción* ($\overline{bel}(x_t)$) e indica el estado en el tiempo t a partir de la distribución posterior del estado anterior, antes de incorporar la medición en el tiempo t. El cálculo de $bel(x_t)$ a partir de $\overline{bel}(x_t)$ se denomina *corrección* [51].

El **Filtro de Bayes** proporciona una solución recursiva, es decir que la creencia $bel(x_t)$ en el tiempo t se calcula a partir de la creencia $bel(x_{t-1})$, y general a este problema en dos pasos alternados. El primero es la **predicción**, en la que se propaga la creencia sobre el estado x_t del robot basado en la creencia anterior del estado x_{t-1} y el control u_t:

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\, bel(x_{t-1})\, dx_{t-1}$$

El segundo es la **actualización de medición**, en la que la creencia predicha se corrige con la probabilidad de que la medición actual z_t haya sido observada bajo cada hipótesis de estado x_t:

$$bel(x_t) = \eta\, p(z_t \mid x_t)\, \overline{bel}(x_t)$$

donde η es un factor de normalización que garantiza que $bel(x_t)$ sea una distribución de probabilidad válida [51].

Las distintas implementaciones del filtro bayesiano difieren en la forma en que representan $bel(x_t)$ y en los supuestos sobre la linealidad y distribución del ruido de los modelos de movimiento y observación.

2.5.1. Filtro de Kalman extendido

El filtro de Kalman clásico es una técnica recursiva para estimar el estado de un sistema dinámico a partir de un modelo de transición lineal, una relación de medición lineal y ruido Gaussiano aditivo. Bajo estas condiciones, la creencia sobre el estado del sistema permanece Gaussiana en el tiempo y queda completamente caracterizada por su media μ_t y su covarianza Σ_t, lo que permite realizar la actualización de forma cerrada y eficiente [51]. Su principal limitación es que depende fuertemente de la linealidad de los modelos de movimiento y observación, por lo que su aplicación directa se restringe a problemas relativamente sencillos.

El Filtro de Kalman Extendido (EKF) generaliza esta idea para sistemas no lineales. Es una de las implementaciones más conocidas del filtro de Bayes para espacios continuos, donde la creencia se representa mediante una distribución Gaussiana unimodal parametrizada por un vector de media μ_t y una matriz de covarianza Σ_t [40]. Esta representación, también denominada *parametrización de momentos*, resulta adecuada cuando la incertidumbre del estado se concentra alrededor de una única hipótesis dominante. Sin embargo, el EKF no es tan adecuado para problemas de estimación global en donde puede haber múltiples hipótesis plausibles del estado, ya que sigue una distribución unimodal.

Para sistemas con modelos no lineales, como la cinemática bípeda, el EKF linealiza las funciones de transición de estado g y de observación h en torno al estado actual utilizando matrices Jacobianas ($G_t = \nabla g$ y $H_t = \nabla h$):

$$\bar{\mu}_t = g(u_t, \mu_{t-1}), \qquad \bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$$

donde R_t es la covarianza del ruido del proceso [18]. La actualización de la medición incorpora la observación z_t a través de la ganancia de Kalman K_t:

$$K_t = \bar{\Sigma}_t H_t^T \left(H_t \bar{\Sigma}_t H_t^T + Q_t \right)^{-1}$$

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)), \qquad \Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

siendo Q_t la covarianza del ruido de la medición [40]. Si bien el EKF es computacionalmente eficiente (su costo es $\mathcal{O}(n^2)$ en la dimensión del estado), su rendimiento se degrada severamente cuando los modelos presentan no linealidades fuertes o cuando la distribución de la creencia es multimodal, situaciones comunes en la localización global de robots humanoides que se inician desde una posición desconocida [51].

2.5.2. Filtro de partículas

El filtro de partículas es una implementación no paramétrica del filtro de Bayes que representa la distribución posterior del estado del robot, $bel(x_t)$,

mediante un conjunto finito de M muestras aleatorias denominadas **partículas** [51]. Cada partícula $x_t^{[m]}$ constituye una hipótesis completa sobre la pose del robot en el plano:

$$x_t^{[m]} = (x,\ y,\ \theta)^T, \quad m = 1, \ldots, M$$

A cada partícula se le asocia un peso de importancia $w_t^{[m]} \geq 0$, normalizado tal que $\sum_{m=1}^{M} w_t^{[m]} = 1$. El conjunto ponderado $\mathcal{X}_t = \{\langle x_t^{[m]}, w_t^{[m]} \rangle\}$ aproxima la distribución posterior real $p(x_t \mid z_{1:t}, u_{1:t})$, y su precisión mejora conforme M aumenta [51].

A diferencia del Filtro de Kalman Extendido, el filtro de partículas no impone ningún supuesto paramétrico sobre la forma de $bel(x_t)$, lo que le permite representar distribuciones multimodales con múltiples hipótesis activas de posición de forma simultánea. Esta flexibilidad es particularmente valiosa en el problema de localización global, donde el robot debe estimar su pose desde cero sin una estimación inicial confiable [16]. Un criterio ampliamente utilizado para diagnosticar la calidad del enjambre de partículas es el **tamaño efectivo de muestra** N_{eff}, que cuantifica el grado de concentración de la distribución de pesos [51]:

$$N_{\text{eff}} = \frac{1}{\sum_{m=1}^{M} \left(w_t^{[m]} \right)^2}$$

Cuando todos los pesos son iguales ($w_t^{[m]} = 1/M$), se obtiene el valor máximo $N_{\text{eff}} = M$, indicando que el enjambre representa bien la distribución posterior. Por el contrario, cuando unos pocos pesos dominan, $N_{\text{eff}} \ll M$, señalando que la mayoría de las partículas contribuyen marginalmente a la estimación y el remuestreo es necesario. Este criterio permite activar el remuestreo de forma adaptativa, solo cuando la diversidad del enjambre se ha degradado suficientemente, evitando así tanto la pérdida prematura de diversidad como el costo computacional de remuestrear en cada iteración [51]. El ciclo de operación del filtro de partículas sigue el esquema recursivo del filtro de Bayes en tres pasos:

1. **Predicción (muestreo del modelo de movimiento):** Cada partícula $x_{t-1}^{[m]}$ se propaga generando una nueva muestra a partir del modelo de movimiento estocástico:

   $$x_t^{[m]} \sim p(x_t \mid u_t,\ x_{t-1}^{[m]})$$

   Este modelo incorpora explícitamente el ruido inherente al sistema de locomoción (deslizamiento, error odométrico) mediante distribuciones de probabilidad sobre las primitivas de movimiento [51].

2. **Actualización (ponderación por observación):** A cada partícula propagada se le asigna un peso de importancia proporcional a la verosimilitud

de la observación actual z_t asumiendo que el robot se halla en la pose hipotética $x_t^{[m]}$:

$$w_t^{[m]} = p(z_t \mid x_t^{[m]}, \mathbf{m})$$

donde $\mathbf{m}$ es el mapa conocido del entorno. Bajo el supuesto de independencia condicional entre observaciones individuales (ec. 6.2, p. 152 de [51]):

$$p(z_t \mid x_t^{[m]}, \mathbf{m}) = \prod_{j=1}^{J} p\Big(z_t^j \mid x_t^{[m]}, \mathbf{m}\Big)$$

3. **Remuestreo:** Se genera un nuevo conjunto de M partículas muestreando con reemplazo del conjunto actual con probabilidad proporcional a los pesos $w_t^{[m]}$. Las partículas con pesos bajos (hipótesis improbables) son descartadas y reemplazadas por réplicas de las de mayor peso, concentrando el enjambre en las regiones de mayor verosimilitud [51].

2.5.3. Método de Monte Carlo

La Localización de Monte Carlo (MCL) es la aplicación del filtro de partículas al problema de autolocalización de robots móviles, y constituye el estándar de facto para sistemas de localización probabilística [16]. MCL resuelve tanto la **localización local** (seguimiento de pose con estimación inicial conocida) como la **localización global** (estimación de pose desde cero), siendo en este último caso donde el filtro de partículas exhibe una ventaja decisiva sobre el EKF al poder distribuir hipótesis uniformemente por todo el espacio de estados [52].

Para evitar la degeneración del enjambre por remuestreo excesivo, se emplea el algoritmo **Low Variance Sampler** (Tabla 4.4, p. 110 de [51]), que utiliza un único número aleatorio $r \sim \mathcal{U}[0, M^{-1})$ y genera los M puntos de selección de forma uniformemente espaciada:

$$U_m = r + (m-1) \cdot M^{-1}, \quad m = 1, \ldots, M$$

Este método preserva la diversidad del enjambre al garantizar una cobertura uniforme de la distribución de pesos, en contraste con el remuestreo independiente estocástico que puede producir degeneración severa por colapso de partículas [51].

2.6. Trabajo Relacionado

La localización basada en la visión se ha estudiado ampliamente en el contexto de los robots humanoides y móviles [47, 28]. Los enfoques clásicos se basan en la extracción de características geométricas a partir de las líneas de campo y las porterías, seguida de una estimación probabilística del estado mediante filtros de partículas [16, 52]. Estos métodos suelen suponer una segmentación limpia y una geometría de puntos de referencia bien definida, supuestos que rara vez se

cumplen en entornos de fútbol del mundo real [7]. Trabajos pioneros como el de Hong et al. [23]demostraron la viabilidad de la localización de Monte Carlo en robots humanoides de fútbol mediante el uso de características visuales simples [7], mientras que Nagi et al. [34] extendieron este enfoque a la liga *kid-size* del RoboCup, destacando los retos de la robustez ante cambios de iluminación y variaciones en la apariencia del entorno. Los recientes avances en el aprendizaje profundo han permitido una detección robusta de objetos en condiciones difíciles [15]. Los detectores de una sola etapa, como YOLO [25, 49, 45], se han adoptado ampliamente en la percepción robótica debido a su rendimiento en tiempo real [39]. Sin embargo, su dependencia de los rectángulos de delimitación basados en anclajes los hace sensibles a las variaciones de escala y a las inconsistencias en las anotaciones, especialmente en el caso de puntos de referencia alargados o no rígidos [22, 50]. Las redes neuronales convolucionales han dominado el reconocimiento visual en robótica durante la última década [35, 27, 26], aunque su sesgo inductivo de localidad las limita para capturar relaciones globales entre regiones distantes de la imagen [8].

Los detectores basados en transformadores, incluidos DETR [14] y sus variantes, abordan algunas de estas limitaciones formulando la detección como un problema de predicción de conjuntos mediante mecanismos de atención [55, 17]. En particular, RT-DETR [30] demostró que los transformadores pueden operar en tiempo real superando a los detectores YOLO en precisión. El Deformable DETR [57] mejora aún más la convergencia y la precisión de la localización al atender a un conjunto disperso de ubicaciones relevantes de la imagen, lo que lo hace especialmente adecuado para detectar puntos de referencia estructurados pero sin restricciones, como las marcas del campo. Esta propiedad es particularmente beneficiosa cuando los tamaños de las cajas delimitadoras varían ampliamente entre anotaciones, como ocurre con los conjuntos de datos de RoboCup [10]. La arquitectura Vision Transformer [17, 32] ha mostrado además capacidad para detectar objetos camuflados y de escala variable en condiciones no controladas [38], lo que refuerza su pertinencia en entornos de fútbol robótico.

A pesar de estos avances, la mayoría de los trabajos anteriores evalúan el rendimiento de la detección utilizando métricas basadas en el IoU [29], lo que puede no estar directamente relacionado con la precisión de la localización del robot [28]. Por el contrario, el presente trabajo evalúa explícitamente los modelos de detección desde la perspectiva de su contribución a la estimación de pose, integrando las detecciones como observaciones z_t dentro de un esquema de localización probabilística basado en filtros de partículas [52, 51].

Capítulo 3

Sistema de localización visual

En este capítulo se describe el sistema de localización visual desarrollado para un robot humanoide en el contexto de la liga *Humanoid Soccer* de la RoboCup. El sistema aborda el problema de estimar la pose del robot (posición (x, y) y orientación θ) dentro de un campo de juego estandarizado, a partir exclusivamente de una cámara RGB y sin acceso a sensores activos de posicionamiento.

El capítulo se organiza de la siguiente manera. En la Sección 3.1 se introduce el problema de la auto-localización en robots bípedos de fútbol y se describen las marcas visuales del campo empleadas como puntos de referencia geométricos. Las Secciones 3.2 y 3.3 presentan el conjunto de datos utilizado (TORSO-21 enriquecido con imágenes de RoboCup 2025) y el proceso de curaduría y etiquetado automático aplicado. En la Sección 3.4 se detallan los modelos de detección entrenados (YOLOv8, RT-DETR y Deformable DETR) y sus respectivas configuraciones de entrenamiento. Finalmente, la Sección 3.5 describe el filtro de partículas de Monte Carlo que integra las detecciones como observaciones visuales z_t, incluyendo el modelo de movimiento adaptado a la locomoción bípeda, el modelo de observación basado en rumbos angulares y el algoritmo de remuestreo de baja varianza.

3.1. Robots bípedos para jugar fútbol soccer

Las capacidades de alto nivel de un robot humanoide que tiene como objetivo desempeñarse de manera efectiva en un partido de fútbol consisten en la planificación de trayectorias, la toma de decisiones tácticas, la coordinación con otros robots en el campo y la ejecución de acciones motoras, como patear o posicionarse frente a una portería. Todas estas tareas dependen de una base fundamental; la capacidad del robot de estimar su posición y orientación dentro del campo en todo momento, conocida como *auto-localización*.

Al inicio de cada juego, el robot debe establecer una estimación inicial de su *pose*, es decir, su posición (x, y) y orientación θ respecto al sistema de coordenadas del campo. Durante el transcurso del juego, esta estimación debe actualizarse constantemente y en tiempo real, compensando los errores acumulados por la odometría y los efectos de colisiones u otras perturbaciones externas. Sin una localización confiable, acciones fundamentales para un juego de fútbol como orientarse hacia una portería, evitar el área propia, o ejecutar un pase dirigido resultan casi imposibles de realizar correctamente.

En la liga *Humanoid Adult Size* de la RoboCup, los robots no pueden tener sensores activos, disponen únicamente de una cámara RGB como sensor principal de percepción, lo que convierte la *localización visual* en el enfoque predominante. A diferencia de sistemas que utilizan LiDAR, Magnetómetros o GPS, la localización basada en visión debe inferir la pose del robot a partir de las características visuales del entorno, como las marcas del campo, los postes de la portería y las líneas del terreno de juego.

3.1.1. Marcas relevantes en el campo de juego

El campo en el que se implementa la *localización visual* está diseñado con un conjunto de marcas visuales que pueden ser utilizadas como referencias geométricas para la localización. Estas marcas son conocidas a priori por el robot, ya que las dimensiones del campo están estandarizadas por el reglamento de la liga.

Las marcas utilizadas para localización son:

- **Intersecciones en L**: se conforman de dos líneas perpendiculares que se unen en un extremo. Aparecen comúnmente en las esquinas del campo y del área de penalti.

- **Intersecciones en T**: se conforman por tres líneas que confluyen en un mismo punto, donde una línea termina perpendicular a las otras dos. Aparecen en varias secciones del campo, en los puntos medios y esquinas de las áreas de penalti y portería.

- **Intersecciones en X**: formadas por dos líneas que se cruzan perpendicularmente. Aparecen comúnmente en el centro del campo, donde el círculo central intersecta la línea de medio campo.

- **Postes de portería**: son estructuras verticales de color blanco ubicadas en posiciones fijas y conocidas. Su detección aporta una referencia espacial con alto alcance visible.

Cada tipo de intersección tiene una distribución espacial única dentro del campo, lo que permite al sistema de localización acortar la posición posible del robot cuando detecta múltiples de estas marcas simultáneamente. La combinación de detecciones reduce la ambigüedad inherente de un campo de fútbol debido a su simetría.

3.2. Dataset TORSO-21

El conjunto de datos base utilizado para el entrenamiento de los modelos de detección es del repositorio **TORSO-21** (*Typical Objects in RoboCup Soccer 2021*), publicado por Bestman et al. [10] y disponible públicamente en `https://github.com/bit-bots/TORSO_21_dataset`. TORSO-21 fue diseñado específicamente como benchmark para sistemas de visión en el dominio de humanoides jugadores de fútbol de RoboCup, y comprende imágenes del mundo real capturadas por distintos robots participantes de la Humanoid League y la Standard Platform League, así como imágenes generadas en el simulador Webots.

3.3. Curaduría de datos

La curaduría de datos comprendió un conjunto de decisiones orientadas a garantizar la calidad y pertinencia del material de entrenamiento para la tarea de detección de marcas visuales del campo de juego. A partir del repositorio público TORSO-21, se aplicaron criterios de filtrado sobre las imágenes y sus anotaciones, descartando aquellas instancias que no aportaban información relevante. Adicionalmente, el conjunto fue enriquecido con imágenes capturadas en condiciones reales de competencia durante RoboCup 2025. El resultado de este proceso es un dataset balanceado y representativo de los entornos reales en los que opera el robot.

3.3.1. Selección de imagenes

A partir de las imágenes recabadas del repositorio de TORSO-21, se realizó un proceso de curaduría orientado a conservar únicamente las imágenes y anotaciones pertinentes para la tarea de detección de marcas visuales del campo de juego. Se aplicaron los siguientes criterios de filtrado:

- **Solo imágenes del mundo real**: se descartaron íntegramente las imágenes del subconjunto simulado, conservando solamente las capturas obtenidas en entornos reales de competencia.
- **Selección de clases**: se conservaron únicamente las anotaciones correspondientes a cinco clases de interés para el sistema de localización y visión básico: balón (*ball*), portería (goal), intersección en L (L), intersección en T (T) e intersección en X (X).
- **Filtrado por contenido**: se mantuvieTron aquellas imágenes que contienen al menos una anotación válida de alguna de las cinco clases seleccionadas.

Con el objetivo de enriquecer el conjunto de datos, se incorporaron **300 imágenes adicionales** capturadas por el equipo de robótica **LIRA-UNAM** durante la competencia **RoboCup 2025**. Si bien se capturó un volumen mayor de imágenes durante el evento, la selección final se limitó a aquellas cuyas anotaciones automáticas, generadas mediante el sistema descrito en la Sección 3.3.2, fueron validadas satisfactoriamente tras una revisión manual. Dado que el modelo utilizado en el sistema de auto-etiquetado constituía una aproximación inicial (sujeta a errores en forma de falsos positivos y falsos negativos), la inspección humana resultó indispensable para garantizar la calidad de las anotaciones incorporadas al dataset de entrenamiento.
El conjunto de datos final quedó conformado por **8,900 imágenes**, distribuidas en **7,119 para entrenamiento** (80 %) y **1,781 para validación** (20 %).
Adicionalmente, se reservó un **conjunto de prueba independiente** compuesto por **90 imágenes** obtenidas también durante RoboCup 2025, distintas de las incorporadas al conjunto de entrenamiento. Dado que ninguna de estas imágenes fue vista por los modelos durante el entrenamiento ni la validación, este subconjunto permite medir la capacidad de generalización de cada arquitectura en condiciones reales.

3.3.2. Sistema de etiquetado automático

Para gestionar el etiquetado de las imágenes capturadas en RoboCup 2025 de manera eficiente, se desarrolló una herramienta de etiquetado automático basada en **YOLOv8** [25].
La herramienta opera de la siguiente manera:

1. Se carga una carpeta de imágenes y un modelo YOLOv8 previamente entrenado (`best.pt`).
2. El modelo genera automáticamente *bounding boxes* y etiquetas de clase para cada imagen.
3. Las anotaciones se exportan en formato compatible con **LabelMe** (archivos /ttt.json), para su revisión y corrección manual.
4. Las anotaciones revisadas se integraron al pipeline de entrenamiento.

El proceso de revisión manual fue indispensable dado que, si bien el modelo utilizado para incorporar las nuevas imágenes automatiza la mayor parte del etiquetado, no produce anotaciones perfectas en todos los casos. Por ello, cada imagen generada automáticamente fue inspeccionada visualmente, descartando o corrigiendo las anotaciones incorrectas antes de incorporarlas al conjunto de datos.

La interfaz de la herramienta, implementada con **Gradio** [3], permite visualizar los resultados del etiquetado automático en lotes de imágenes y ejecutar el etiquetado de una carpeta completa con un solo clic, reduciendo significativamente el tiempo de preparación del conjunto de datos respecto al etiquetado manual convencional.

Figura 3.1: Captura de pantalla de la interfaz gráfica.

3.4. Entrenamiento de modelos

En esta sección se describe el proceso de entrenamiento de los tres modelos de detección evaluados en este trabajo: YOLOv8, RT-DETR y Deformable DETR. Los tres modelos fueron entrenados sobre el mismo conjunto de datos descrito en la Sección 3.3, compuesto por imágenes reales del dominio RoboCup anotadas con cinco clases. Si bien los tres modelos comparten el mismo conjunto de datos y objetivo de detección, difieren en su arquitectura y herramienta de entrenamiento, por lo que cada uno se describe de forma independiente en las subsecciones siguientes.

3.4.1. Entrenamiento del modelo YOLOv8

Herramientas de Ultralytics

El entrenamiento se llevó a cabo mediante el *framework* **Ultralytics YOLO** [25], una biblioteca de código abierto desarrollada en Python que proporciona

una interfaz unificada para el entrenamiento, validación, inferencia y exportación de modelos de la familia YOLO. Su diseño modular permite configurar el proceso de entrenamiento a través de un archivo `.yaml` de dataset y parámetros de configuración mediante línea de comandos.

Entre las características de Ultralytics relevantes para este trabajo destacan la carga automática de pesos preentrenados (*pretrained weights*), una *pipeline* de aumentación de datos integrada aplicada en tiempo de entrenamiento, el registro automático de métricas y curvas de aprendizaje, y la gestión de *checkpoints* con guardado del mejor modelo observado durante la validación (`best.pt`).

Configuración del entrenamiento

El modelo base utilizado fue **YOLOv8s** (`yolov8s.pt`), inicializado con pesos preentrenados sobre un conjunto de datos en formato COCO [25]. Se realizó un ajuste fino completo (*full fine-tuning*) sobre el conjunto de entrenamiento descrito en la Sección 3.3. La configuración completa de hiperparámetros se detalla en el Cuadro 3.1.

Cuadro 3.1: Hiperparámetros de entrenamiento del modelo YOLOv8s.

Hiperparámetro	**Valor**
Modelo base	YOLOv8s
Épocas	150
Early stopping patience	100
Tamaño de lote	16
Resolución de entrada	640×640
Optimizador	Auto (SGD)
Tasa de aprendizaje inicial	0.01
Tasa de aprendizaje final	0.01
Congelamiento de capas	Ninguno

El punto de partida del entrenamiento es el *modelo base*. En lugar de inicializar los pesos al azar, se parte de un modelo con pesos inicializados preentrenados sobre el conjunto de datos COCO (Common Objects in COntext) que se conforma de 330,000 imágenes anotadas en 80 categorías de objetos [29].

Una *época* equivale a una iteración del modelo de pasar por todas las imágenes de entrenamiento. Con 150 épocas el modelo recorre esa cantidad de veces el conjunto de imágenes de entrenamiento o *dataset*, ajustando los pesos en cada iteración.

El *Early stopping patience* indica cuándo el modelo debe detenerse si no mejora la validación en ese punto, evitando así desperdiciar tiempo y previniendo sobreajuste.

El tamaño del lote o *batch* es el número de imágenes que se procesan simultáneamente durante el entrenamiento, se calcula el error promedio, y actualiza los pesos una vez. Lotes más grandes generan estimaciones más estables, pero ocupan más VRAM.

La *resolución de entrada* es el tamaño al que se redimensionan las imágenes antes de entrar al modelo.

El optimizador, seleccionado automáticamente por Ultralytics, es el algoritmo matemático que ajusta los pesos para reducir el error; en este caso *SDG* (Descenso del Gradiente Estocástico).

Las *tasas de aprendizaje* inicial y final se mantienen; indican el tamaño del paso al inicio y al final del entrenamiento. Debido a que este fue un caso de fine-tuning sobre un modelo ya preentrenado en COCO, los pesos inician en un buen punto de partida y no es necesaria una exploración agresiva. Se optó por un ajuste fino completo, *full fine-tunning*, sin congelamiento de capas dado que las imágenes del campo de fútbol son significativamente diferentes al conjunto de datos sobre el que el modelo fue preentrenado, lo que hace necesario adaptar también las capas de extracción de características de bajo nivel. Ultralytics aplica automáticamente una *pipeline* de aumentación de datos durante el entrenamiento.

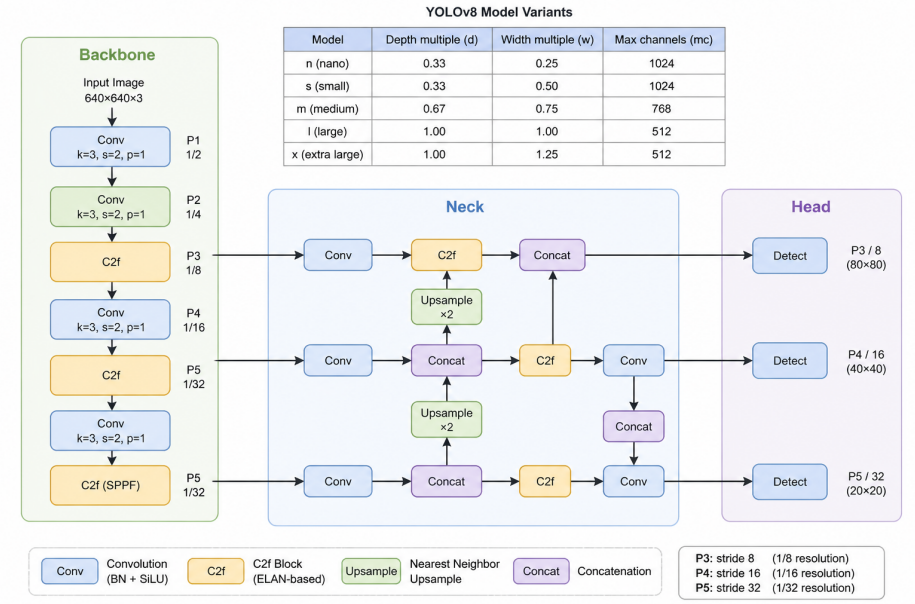


Figura 3.2: Arquitectura del modelo YOLOv8s adaptada de [22]

La Figura 3.2 muestra la arquitectura de YOLOv8 empleada como base en este trabajo. La variante del modelo se determina mediante los parámetros *depth_multiple*, *width_multiple* y *max_channels*, los cuales controlan la profundidad de los bloques C2f y el número de canales de salida en cada etapa [22].

La red inicia con la *etapa inicial* o *stem*, compuesta por dos bloques convolucionales con *stride* 2 y *kernel size* 3, cuya función es extraer las características básicas de la imagen y reducir su resolución espacial. Posteriormente, el *backbone* o *columna vertebral* utiliza bloques C2f en varias etapas, donde algunas

incluyen *shortcuts* o *conexiones residuales* para facilitar el flujo de información, mientras que en el *neck* o *cuello* estos bloques se emplean sin dichas conexiones, siguiendo la configuración descrita en el artículo de referencia [22].

Después del último bloque del *backbone*, se incorpora el módulo SPPF (*Spatial Pyramid Pooling Fast* o *agrupamiento espacial piramidal rápido*), el cual permite obtener una representación multiescala de los mapas de características mediante *pooling* o *agrupamiento* en diferentes escalas [22]. En el *neck*, las operaciones de *upsampling* o *aumento de resolución* y *concat* o *concatenación* permiten combinar características de distintas resoluciones, incrementando la información espacial disponible para la detección [22].

Finalmente, la *head* o *cabeza* del modelo produce tres salidas de detección asociadas a objetos pequeños, medianos y grandes, lo que permite realizar una detección multiescala en la imagen [22].

3.4.2. Entrenamiento del modelo RT-DETR

De forma similar al entrenamiento del modelo YOLOv8, el modelo RT-DETR cuenta con la interfaz de entrenamiento proporcionada por Ultralytics, que permite configurar el proceso directamente desde la línea de comandos [2]. La información detallada sobre el uso de la biblioteca se puede consultar en la documentación oficial [1].

Entre los hiperparámetros más relevantes para el entrenamiento se encuentran:

- **Tamaño de imagen de entrada** (`imgsz`): las imágenes se redimensionaron a 640×640 píxeles, que es la resolución estándar de entrada para RT-DETR en Ultralytics [2].

- **Trabajadores de carga de datos** (`workers`): se emplearon 8 *workers* paralelos para la carga de datos, lo que permite alimentar la GPU sin que el preprocesamiento de imágenes provoque en un cuello de botella.

- **Pesos preentrenados** (`pretrained=true`): el modelo se inicializó con pesos preentrenados en COCO, aplicando *transfer learning* para acelerar la convergencia.

- **Optimizador** (`optimizer=auto`): Ultralytics elige automáticamente entre *SGD* y AdamW según el modelo y la tarea.

[1] `https://docs.ultralytics.com/es/models/rtdetr`

Cuadro 3.2: Hiperparámetros principales del entrenamiento RT-DETR.

Parámetro	Valor	Descripción
`model`	`rtdetr-l.pt`	Variante *large* con pesos COCO
`epochs`	150	Épocas máximas de entrenamiento
`patience`	100	Épocas de parada temprana
`batch`	8	Imágenes por lote
`imgsz`	640	Resolución de entrada (píxeles)
`workers`	8	Hilos de carga de datos
`pretrained`	`true`	Transferencia de aprendizaje

En cuanto a las aumentaciones de datos aplicadas durante el entrenamiento, Ultralytics configura automáticamente las transformaciones para RT-DETR. Entre los parámetros activos en este entrenamiento se encuentran: volteo horizontal aleatorio (`fliplr=0.5`), escalado (`scale=0.5`), traslación (`translate=0.1`), variaciones en el espacio de color HSV (`hsv_h=0.015`, `hsv_s=0.7`, `hsv_v=0.4`), mosaico (`mosaic=1.0`) y borrado aleatorio (`erasing=0.4`). La política de aumentación automática empleada fue `randaugment` [1].

3.4.3. Entrenamiento del modelo Deformable-DETR

Herramienta de entrenamiento

El entrenamiento de Deformable DETR se realizó utilizando el repositorio oficial publicado por Zhu et al. [57], implementado en PyTorch. A diferencia del modelo RT-DETR, este entrenamiento no utilizó la API de Ultralytics, sino que se ejecutó directamente sobre el código fuente original, lo que permitió un mayor control sobre la configuración del proceso de optimización y el seguimiento detallado de las métricas de entrenamiento.

El modelo Deformable DETR combina un backbone convolucional con un transformer encoder-decoder basado en atención deformable de múltiples escalas. Su arquitectura consta de cuatro módulos principales:

- **Backbone (ResNet-50):** Extrae características visuales jerárquicas. Se utilizan las salidas de las etapas C_3, C_4 y C_5, con resoluciones reducidas progresivamente, además de una cuarta escala generada mediante una proyección adicional. Todos los canales se proyectan a 256 dimensiones.
- **Encoder deformable (6 capas):** Aplica atención deformable multiescala sobre las características del backbone. En lugar de atender a todos los píxeles, cada consulta muestrea un conjunto reducido de puntos de referencia, reduciendo la complejidad computacional de la atención estándar.
- **Decoder deformable (6 capas):** Recibe 300 *object queries* aprendibles como entrada. Cada capa refina iterativamente las predicciones de caja y clase mediante self-attention y cross-attention deformable hacia las características codificadas.

- **Cabeza de detección:** Produce las predicciones finales de forma *anchor-free*, generando directamente coordenadas de caja normalizadas y puntuaciones de clase para las cinco categorías del dataset: balón, portería, intersección-L, intersección-T e intersección-X.

El modelo fue inicializado con pesos preentrenados en MS-COCO 2017 y entrenado mediante *full fine-tuning* sobre el conjunto de datos combinado TORSO-21 + RoboCup 2025. El modelo completo cuenta con 39,847,265 parámetros entrenables, lo que refleja una capacidad de representación considerable para el dominio de localización visual en escenas deportivas.

Configuración del entrenamiento

La configuración del entrenamiento se ajustó manualmente a partir de los logs generados durante la ejecución. Se utilizaron 200 épocas de entrenamiento, con el mejor rendimiento observado alrededor de la época 192, donde el modelo alcanzó una mejora clara en la métrica de evaluación COCO BBox respecto a los primeros ciclos. El *learning rate* inicial se mantuvo en 1×10^{-4}, y se observó una reducción progresiva hasta valores del orden de 10^{-8} hacia el final del entrenamiento, lo que indica la aplicación de una programación decreciente de la tasa de aprendizaje.

Durante el entrenamiento, la pérdida total descendió hasta valores cercanos a 2.74 en las últimas iteraciones, mientras que la pérdida de validación se mantuvo alrededor de 3.46. A nivel de métricas de evaluación, el modelo mostró una mejora consistente en `testcocoevalbbox`, que corresponde a la métrica de evaluación COCO para cajas delimitadoras (*bounding boxes*), y resume la calidad con la que el detector localiza los objetos en el conjunto de prueba, alcanzando valores en el entorno de 0.64 al final del entrenamiento, frente a valores cercanos a 0.36-0.45 en las primeras épocas. Asimismo, la pérdida de clasificación de prueba (`testclasserror`) se redujo hasta valores cercanos a 1.22, lo que muestra una convergencia estable del detector.

Cuadro 3.3: Hiperparámetros principales del entrenamiento de Deformable DETR.

Parámetro	**Valor**	**Descripción**
`backbone`	ResNet-50	Extracción jerárquica de características
`encoder`	6 capas	Atención deformable multiescala
`decoder`	6 capas	Refinamiento iterativo de predicciones
`object queries`	300	Consultas aprendibles de entrada
`pretrained`	MS-COCO 2017	Inicialización de pesos
`fine-tuning`	full	Ajuste de todos los parámetros
`epochs`	200	Épocas de entrenamiento
`lr`	10^{-4} a 10^{-8}	Tasa de aprendizaje observada

3.5. Filtro de partículas de Monte Carlo

En este trabajo se usó específicamente la implementación del filtro de partículas de Monte Carlo descrito en las secciones 2.5.2 y 2.5.3, adaptado al contexto de un robot humanoide en un campo de fútbol estandarizado. El sistema mantiene un conjunto de $M = 700$ partículas $\mathcal{X}_t = \{x_t^{[1]}, \ldots, x_t^{[M]}\}$, donde cada partícula $x_t^{[m]} = (x, y, \theta)^T$ representa una hipótesis de la pose del robot. El ciclo de actualización sigue la estructura del Algoritmo MCL (Tabla 8.2 de [52]): en cada iteración se propagan las partículas mediante el modelo de movimiento, se ponderan con el modelo de observación y se remuestrea el conjunto para aproximar la distribución posterior $bel(x_t)$.

3.5.1. Distribución inicial

Al inicio de cada iteración, las partículas se distribuyen de forma uniforme sobre todo el campo de juego, ya que no se dispone de información previa sobre la pose del robot. Cada partícula se inicializa muestreando independientemente:

$$x \sim \mathcal{U}(-3{,}04,\ 3{,}04), \quad y \sim \mathcal{U}(-4{,}53,\ 4{,}53), \quad \theta \sim \mathcal{U}(-\pi,\ \pi)$$

donde x e y son las coordenadas de posición en metros respecto al centro del campo, θ es la orientación en radianes, y $\mathcal{U}(a, b)$ denota la distribución uniforme continua en el intervalo $[a, b]$.

Los límites $\pm 3{,}04$ m y $\pm 4{,}53$ m corresponden a las dimensiones del campo que se utilizó en las pruebas experimentales.

Todos los pesos se inicializan uniformemente como $w^{[m]} = 1/M$. Esta inicialización corresponde al problema de *localización global*, en el que el robot desconoce completamente su posición inicial [52].

3.5.2. Modelo de movimiento

El modelo de movimiento implementado corresponde al algoritmo de la Tabla 5.6 de [52]. El movimiento relativo entre dos poses consecutivas de odometría se descompone en tres componentes: una rotación inicial δ_{rot1}, una traslación δ_{trans} y una rotación final δ_{rot2}:

$$\delta_{rot1} = \operatorname{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta} \tag{3.1}$$

$$\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2} \tag{3.2}$$

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1} \tag{3.3}$$

A cada componente se le inyecta ruido gaussiano cuya varianza es proporcional a la magnitud del movimiento ejecutado, según los parámetros $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \alpha_3, \alpha_4]$, donde $\mathcal{N}(0, \sigma^2)$ denota una muestra de una distribución gaussiana de media cero y varianza σ^2. Los parámetros $\boldsymbol{\alpha}$ modelan cuatro fuentes de error de odometría [52]:

- α_1: ruido en la rotación causado por la rotación ejecutada.
- α_2: ruido en la rotación causado por la traslación ejecutada.
- α_3: ruido en la traslación causado por la traslación ejecutada.
- α_4: ruido en la traslación causado por la rotación ejecutada.

Los valores $\hat{\delta}_{rot1}$, $\hat{\delta}_{trans}$ y $\hat{\delta}_{rot2}$ son los componentes de movimiento **ruidosos** que se utilizan para propagar cada partícula, en contraste con los valores δ_{rot1}, δ_{trans} y δ_{rot2} que representan el movimiento **observado** por la odometría. La varianza de cada componente depende cuadráticamente de la magnitud del movimiento: por ejemplo, la varianza de $\hat{\delta}_{rot1}$ crece con el cuadrado de la propia rotación ($\alpha_1\delta_{rot1}^2$) y con el cuadrado de la traslación ($\alpha_2\delta_{trans}^2$), reflejando que ambos tipos de movimiento contribuyen a la incertidumbre angular [52].

$$\hat{\delta}_{rot1} = \delta_{rot1} - \mathcal{N}(0, \alpha_1\delta_{rot1}^2 + \alpha_2\delta_{trans}^2) \tag{3.4}$$

$$\hat{\delta}_{trans} = \delta_{trans} - \mathcal{N}(0, \alpha_3\delta_{trans}^2 + \alpha_4\delta_{rot1}^2 + \alpha_4\delta_{rot2}^2) \tag{3.5}$$

$$\hat{\delta}_{rot2} = \delta_{rot2} - \mathcal{N}(0, \alpha_1\delta_{rot2}^2 + \alpha_2\delta_{trans}^2) \tag{3.6}$$

La nueva pose de cada partícula se obtiene aplicando el movimiento ruidoso:

$$x_t = x_{t-1} + \hat{\delta}_{trans}\cos(\theta_{t-1} + \hat{\delta}_{rot1}) \tag{3.7}$$

$$y_t = y_{t-1} + \hat{\delta}_{trans}\sin(\theta_{t-1} + \hat{\delta}_{rot1}) \tag{3.8}$$

$$\theta_t = \theta_{t-1} + \hat{\delta}_{rot1} + \hat{\delta}_{rot2} \tag{3.9}$$

En la implementación, la actualización del modelo de movimiento se activa únicamente cuando se detecta un desplazamiento significativo ($\delta_{trans} > 0{,}2$ m) o una rotación significativa ($|\delta_{rot1}+\delta_{rot2}| > 0{,}08$ rad), evitando propagar ruido innecesario cuando el robot está estático. Adicionalmente, se aplica un *acotamiento* (clamping) para mantener todas las partículas dentro de los límites del campo. Los parámetros de ruido empleados fueron $\boldsymbol{\alpha} = [0{,}001, 0{,}0003, 0{,}04, 0{,}001]$, valores calibrados para reflejar la inestabilidad de la locomoción bípeda, especialmente la incertidumbre traslacional ($\alpha_3 = 0{,}04$) producida por el balanceo lateral de la marcha.

El movimiento del robot entre dos pasos de tiempo consecutivos se descompone en una rotación inicial, una traslación y una rotación final. Cada componente se ve afectado por ruido gaussiano cuya varianza depende de la magnitud del movimiento ejecutado.

Para analizar el impacto de la incertidumbre inducida por la locomoción en robots humanoides, se evaluaron dos modelos de movimiento basados en odometría: un modelo idealizado de bajo ruido y un modelo estocástico humanoide que tiene en cuenta el balanceo inducido por la marcha. Ambos modelos se derivan del modelo de movimiento basado en odometría descrito en [52], donde el movimiento relativo entre dos poses consecutivas se descompone en una rotación inicial δ_{rot1}, una traslación δ_{trans} y una rotación final δ_{rot2}.

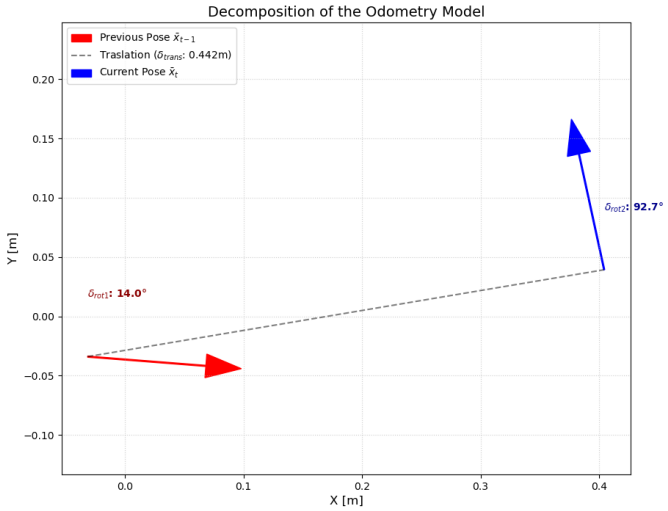


Figura 3.3: Movimiento detectado: Rot1: 0.2450, Trans: 0.4417, Rot2: 1.6182

Se inyecta ruido gaussiano en cada componente, con varianza proporcional a la magnitud del movimiento ejecutado. El modelo ideal asume parámetros de ruido mínimos $\boldsymbol{\alpha} = [0{,}005, 0{,}005, 0{,}005, 0{,}005]$, representando una plataforma robótica rígida y estable. En contraste, el modelo humanoide utiliza parámetros de ruido incrementados, particularmente en la componente traslacional, para reflejar el balanceo lateral y la inestabilidad inherente a la locomoción bípeda.

Como se muestra en la Figura.3.4, la nube de colo azul representa la distribución de la incertidumbre obtenida utilizando parámetros de bajo ruido, lo que corresponde a una plataforma rígida ideal. La nube de color naranja ilustra el aumento de la dispersión introducido para modelar el balanceo lateral y la inestabilidad inducida por la marcha observados en la odometría de humanoides reales.

La descomposición del movimiento en δ_{rot1}, δ_{trans}, y δ_{rot2} sigue el marco probabilístico basado en la odometría propuesto por Thrun et al. [52]. El modelo de movimiento ideal produce una distribución incierta estrecha alineada con la dirección del movimiento. Mientras este comportamiento es deseable para robots con ruedas o robots rígidos, falla para capturar la naturaleza de la locomoción de un humanoide.

El modelo estocástico del humanoide genera una nube de partículas más amplia en dirección lateral, lo cual refleja los efectos oscilatorios provocados por el paso, deslizamiento del pie y balanceo del torso. Aunque este aumento de la incertidumbre reduce precisión de la pose instantánea, mejora significativamente la robustez al garantizar que el estado real del robot se mantenga dentro de la estimación del filtro de partículas.

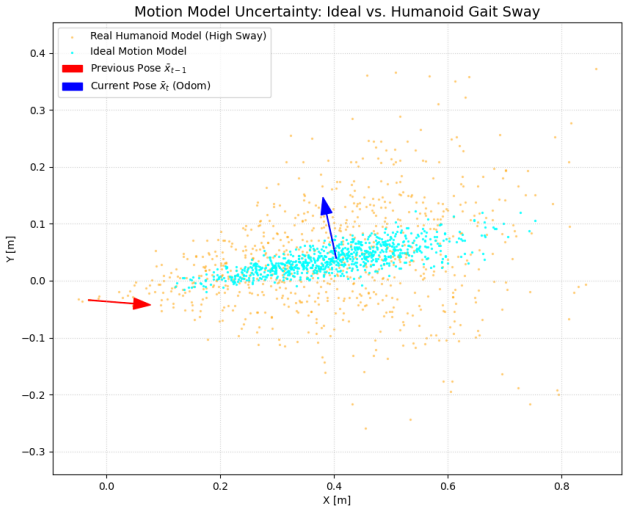


Figura 3.4: Comparación entre el modelo de movimiento ideal y el modelo de movimiento estocástico del humanoide.

Este equilibrio entre precisión y robustez es fundamental para evitar la divergencia del filtro en escenarios reales de localización de humanoides, en los que el ruido de la odometría es altamente no gaussiano y está fuertemente correlacionado con la dinámica de la marcha.

3.5.3. Modelo de observación

El modelo de observación implementa el paso de *actualización* del filtro de partículas, calculando el peso $w^{[m]}$ de cada partícula como la verosimilitud $p(z_t \mid x_t^{[m]}, m)$ de las detecciones visuales recibidas z_t, dado el estado hipotético representado por la partícula $x_t^{[m]}$ y el mapa conocido del campo $\mathbf{m}$ [52]. Las marcas del campo se modelan como *landmarks* o hitos geométricos, siguiendo el modelo de medición basado en marcas descrito por Thrun et al. [52]. Cada observación corresponde a una marca identificada por su clase y caracterizada por su rumbo angular relativo ϕ respecto al eje frontal del robot.

Conversión de detecciones a rumbos angulares

Las observaciones z_t son producidas por el detector de objetos, que entrega cajas delimitadoras en coordenadas de imagen. Para integrarlas en el filtro de partículas es necesario convertir cada detección en un rumbo angular relativo al robot.

Sea u la coordenada horizontal en píxeles del centro de la caja delimitadora de una marca detectada, y W el ancho de la imagen. La coordenada normalizada $u_{norm} \in [-1,\, 1]$ se obtiene como

$$u_{norm} = \frac{u - c_x}{c_x}, \qquad c_x = \frac{W}{2}. \tag{3.10}$$

El modelo estándar de cámara *pinhole* transforma coordenadas de píxel en ángulos mediante $\alpha = \arctan(u/f)$. Sin embargo, dado que el campo de visión horizontal de la cámara es $\mathrm{FOV}_h \approx 53{,}7$ (obtenido experimentalmente), la desviación angular máxima respecto al eje óptico es de $\pm 26{,}85$ ($\approx 0{,}47\,\mathrm{rad}$). En este rango la aproximación del ángulo pequeño $\tan\theta \approx \theta$ introduce un error despreciable frente a la incertidumbre inherente a la regresión de cajas delimitadoras, por lo que se emplea un mapeo lineal directo:

$$\alpha_{cam} = -\,u_{norm} \cdot \frac{\mathrm{FOV}_h}{2}. \tag{3.11}$$

El signo negativo refleja la convención de imagen donde el eje u crece hacia la derecha mientras los ángulos crecen en sentido antihorario. Finalmente, se suma el ángulo de *pan* de la cabeza del robot ψ_{head} para expresar el rumbo en el marco del torso:

$$\alpha_{robot} = \mathrm{wrap}_{[-\pi,\pi]}(\alpha_{cam} + \psi_{head})\,, \tag{3.12}$$

donde $\mathrm{wrap}_{[-\pi,\pi]}(\cdot)$ normaliza el ángulo al intervalo $[-\pi, \pi]$, evitando discontinuidades al cruzar los límites angulares.

Predicción de mediciones

Para cada partícula con pose hipotética (p_x, p_y, p_θ), se predicen los rumbos esperados hacia todas las marcas de la cancha que se encuentran dentro del campo de visión horizontal del robot. Dada una marca j con posición conocida (ℓ_{jx}, ℓ_{jy}), el ángulo relativo predicho es

$$\hat{\phi}_j = \mathrm{wrap}_{[-\pi,\pi]}(\mathrm{atan2}(L_{jy} - p_y,\; L_{jx} - p_x) - p_\theta) \tag{3.13}$$

Solo se consideran las marcas que satisfacen $|\hat{\phi}_j| \leq \mathrm{FOV}/2$, modelando la limitación del campo de visión de la cámara del robot. Esto resulta en un conjunto predicho $\mathcal{D} = \{(\mathrm{id}_j,\, \hat{\phi}_j)\}$, ordenado de menor a mayor por ángulo. El mapa empleado comprende 27 posiciones distribuidas en cuatro clases: postes de portería (clase 1), intersecciones en L (clase 3), intersecciones en T (clase 4) e intersecciones en X (clase 5).

Asociación de datos

Tanto las predicciones $\mathcal{D}$ de como las observaciones reales z_t se mantienen ordenadas por un ángulo ascendente. La correspondencia entre ambos conjuntos se establece mediante un recorrido con dos punteros i (sobre z_t) y j (sobre $\mathcal{D}$), inicializados en cero. En cada paso del recorrido:

- Si $z_t[i].\text{id} = \mathcal{D}[j].\text{id}$, se registra una correspondencia válida, se calcula el error angular
$$e_k = \text{wrap}_{[-\pi,\pi]}\left(\phi_k - \hat{\phi}_k\right), \tag{3.14}$$
y se avanza ambos punteros simultáneamente.
- Si los identificadores no coinciden, se avanza únicamente el puntero j. Esto permite que el filtro tolere marcas que la partícula predice ver pero el detector no reportó, sin penalizar por falsos negativos del detector.

Sea $\mathcal{E} = \{e_k\}$ el conjunto de errores de todas las correspondencias válidas, $|z_t|$ el número de observaciones reales y $|\mathcal{D}|$ el número de predicciones generadas por la partícula.

Cálculo del peso

El peso de cada partícula combina tres factores diseñados para equilibrar precisión y robustez ante las imperfecciones del detector.

Verosimilitud gaussiana. Siguiendo el modelo de landmarks de [52] (ec. 6.2), la consistencia de las correspondencias válidas se mide con una verosimilitud gaussiana sobre el error angular. Para evitar que partículas con mayor número de correspondencias dominen sistemáticamente, se emplea el promedio del log-likelihood:
$$q = \exp\left(\frac{-1}{2\,\sigma_\phi^2\,|\mathcal{E}|}\sum_{e_k\in\mathcal{E}} e_k^2\right), \tag{3.15}$$
donde $\sigma_\phi = 5$ es la desviación estándar del modelo de error angular. Este parámetro fue inicializado tomando como referencia el modelo propuesto en [52]. Después de experimentar con este valor, se comprobó que funciona en este caso para absorber la incertidumbre tanto de la regresión de cajas delimitadoras como de la linealización del modelo de cámara.

3.5.4. Remuestreo y el muestreo de baja varianza

Para evitar la privación de partículas, implementamos el algoritmo de Remuestreo de Baja Varianza [52]. Este método reduce la varianza del conjunto de partículas en comparación con el muestreo aleatorio simple, manteniendo una distribución de hipótesis más robusta [23]. Descarta hipótesis con pesos bajos y duplica los pesos de aquellas partículas que presentan mayor verosimilitud con respecto a las observaciones reales del robot. Sin embargo, un muestreo aleatorio simple e independiente aplicado a cada una de las partículas del conjunto puede generar el error de muestreo mencionado al inicio, la privación de partículas. Este fenómeno provoca que posibles posiciones correctas se pierdan de manera prematura, lo que lleva a la nube a colapsar prematuramente [52]. Para mitigar este conflicto, en este trabajo se implementó la estrategia de *Muestreo de Baja Varianza* propuesto por Thrun et al. [52]. La idea central radica en calcular un

único número aleatorio r acotado en el intervalo $[0, M^{-1})$ al inicio del proceso. A partir de este valor, el algoritmo selecciona la nueva nube de partículas de tamaño M de forma determinista y uniformemente espaciada. La selección se guía por la que es la variable de control U:

$$U_m = r + (m-1) \cdot M^{-1} \tag{3.16}$$

En un ejemplo práctico es visualizar este procedimiento como la colocación de una regla con M marcas distribuidas de manera uniforme sobre el vector acumulado de los pesos de las partículas. Cualquier punto de selección U que esté dentro del intervalo $[0, 1]$ apunta a un índice de partícula i dentro de la distribución de masa acumulada:

$$i = argmin \sum_{m=1}^{j} w_t^{[m]} \geq U \tag{3.17}$$

El algoritmo 4 descrito en la siguiente subsección ilustra este procedimiento. El ciclo *while* funge dos operaciones: calcula de forma iterativa la suma acumulada de los pesos normalizados (c) y localiza el índice exacto de la primer partícula cuya masa acumulada excede el umbral de prueba U.

3.6. Localización por Monte Carlo y marcas de la cancha

La localización se implementó siguiendo directamente el Algoritmo MCL (Tabla 8.2 de [52]), integrando el modelo de movimiento por odometría (Tabla 5.6) y el modelo de observación basado en rumbos angulares descritos en las secciones anteriores. El sistema mantiene $M = 700$ partículas sobre el campo, actualizadas de forma recursiva a partir de la odometría u_t y las detecciones visuales $z_t = \{(\mathrm{id}_k, \phi_k)\}$.

Algorithm 1 MCL — basado en Tabla 8.2 de [52]

Require: $\mathcal{X}_{t-1},\ u_t,\ z_t,\ \mathbf{m}$
Ensure: $\mathcal{X}_t$
$\bar{\mathcal{X}}_t \leftarrow \emptyset;\quad \mathcal{X}_t \leftarrow \emptyset$
for $m = 1$ **to** M **do**
  $x_t^{[m]} \leftarrow \text{SAMPLE\_MOTION\_MODEL}(u_t,\ x_{t-1}^{[m]})$
  $w_t^{[m]} \leftarrow \text{OBS\_MODEL}(z_t,\ x_t^{[m]},\ \mathbf{m})$
  $\bar{\mathcal{X}}_t \leftarrow \bar{\mathcal{X}}_t \cup \langle x_t^{[m]},\ w_t^{[m]} \rangle$
end for
if $\max_m w_t^{[m]} < 10^{-5}$ **then** ▷ *detección de divergencia*
  Reiniciar el 20 % de $\bar{\mathcal{X}}_t$ con poses $\sim \mathcal{U}(\text{campo})$
end if
$\tilde{w}_t^{[m]} \leftarrow w_t^{[m]} / \sum_{i=1}^{M} w_t^{[i]} \quad \forall\, m$ ▷ *normalización*
$N_{eff} \leftarrow 1 \,/\, \sum_{m=1}^{M} \big(\tilde{w}_t^{[m]}\big)^2$
if $N_{eff} < 0{,}4\,M$ **then**
  $\mathcal{X}_t \leftarrow \text{LOW\_VARIANCE\_SAMPLER}(\bar{\mathcal{X}}_t,\ \tilde{\mathcal{W}}_t)$ ▷ *Tabla 4.4*
else
  $\mathcal{X}_t \leftarrow \bar{\mathcal{X}}_t$
end if
return $\mathcal{X}_t$

Algorithm 2 MODELO_DE_MOVIMIENTO — basado en Tabla 5.6 de [52]

Require: $u_t = (\bar{x}_{t-1},\ \bar{x}_t),\quad x_{t-1} = (x,\ y,\ \theta)^T,\quad \boldsymbol{\alpha} = [\alpha_1, \alpha_2, \alpha_3, \alpha_4]$
Ensure: x_t
if $\delta_{trans} \le 0{,}08\,\text{m}$ **y** $|\delta_{rot1} + \delta_{rot2}| \le 0{,}08\,\text{rad}$ **then**
  return x_{t-1} ▷ *umbral de activación*
end if
$\delta_{rot1} \leftarrow \text{atan2}(\bar{y}' - \bar{y},\ \bar{x}' - \bar{x}) - \bar{\theta}$
$\delta_{trans} \leftarrow \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
$\delta_{rot2} \leftarrow \bar{\theta}' - \bar{\theta} - \delta_{rot1}$
$\hat{\delta}_{rot1} \leftarrow \delta_{rot1} - \mathcal{N}(0,\ \alpha_1\delta_{rot1}^2 + \alpha_2\delta_{trans}^2)$
$\hat{\delta}_{trans} \leftarrow \delta_{trans} - \mathcal{N}(0,\ \alpha_3\delta_{trans}^2 + \alpha_4\delta_{rot1}^2 + \alpha_4\delta_{rot2}^2)$
$\hat{\delta}_{rot2} \leftarrow \delta_{rot2} - \mathcal{N}(0,\ \alpha_1\delta_{rot2}^2 + \alpha_2\delta_{trans}^2)$
$x' \leftarrow x + \hat{\delta}_{trans}\cos(\theta + \hat{\delta}_{rot1})$
$y' \leftarrow y + \hat{\delta}_{trans}\sin(\theta + \hat{\delta}_{rot1})$
$\theta' \leftarrow \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$
$x' \leftarrow \text{clamp}(x',\ -3{,}04,\ 3{,}04);\quad y' \leftarrow \text{clamp}(y',\ -4{,}53,\ 4{,}53)$ ▷ *acotamiento al campo*
return $x_t = (x',\ y',\ \theta')^T$

Algorithm 3 OBS_MODEL — modelo de observación por rumbos angulares

Require: $x_t^{[m]} = (x, y, \theta)$, z_t, $\mathbf{m}$, σ_ϕ
Ensure: $w_t^{[m]}$
$\mathcal{D} \leftarrow \left\{ (\mathrm{id}_j,\ \hat{\phi}_j) \mid \ell_j \in \mathbf{m},\ |\hat{\phi}_j| \le \frac{\mathrm{FOV}}{2} \right\}$, donde $\hat{\phi}_j = \mathrm{wrap}(\mathrm{atan2}(\ell_{jy} - y,\, \ell_{jx} - x) - \theta)$
if $z_t = \emptyset$ **o** $\mathcal{D} = \emptyset$ **then**
  return ε
end if
Ordenar z_t y $\mathcal{D}$ por ángulo ascendente
$\mathcal{E} \leftarrow \emptyset$; $i \leftarrow 1$; $j \leftarrow 1$
while $i \le |z_t|$ **y** $j \le |\mathcal{D}|$ **do**
  if $z_t[i].\mathrm{id} = \mathcal{D}[j].\mathrm{id}$ **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathrm{wrap}(\phi_i - \hat{\phi}_j)\}$; i += 1; j += 1
  else
    j += 1 ▷ *tolera falsos negativos*
  end if
end while
if $|\mathcal{E}| = 0$ **then**
  return ε
end if
$w \leftarrow \exp\left(\frac{-1}{2\sigma_\phi^2 \, |\mathcal{E}|} \sum_{e_k \in \mathcal{E}} e_k^2 \right) \cdot \frac{|\mathcal{E}|}{|z_t|}$ ▷ *ec. 6.2 [52] × ratio de cobertura*
return $\mathrm{máx}\big(\mathrm{mín}(w,\ 1),\ \varepsilon\big)$

Algorithm 4 MUESTREO_DE_BAJA_VARIANZA — basado en Tabla 4.4 de [52]

Require: $\bar{\mathcal{X}}_t$,
Ensure: $\mathcal{X}_t$
$\mathcal{X}_t \leftarrow \emptyset$; $r = \mathrm{rand}(0,\ M^{-1})$; $c = w_t^{[1]}$; $i = 1$
for $m = 1$ **to** M **do**
  $U = r + (m-1) \cdot M^{-1}$
  while $U > c$ **do**
    $i = i + 1$; $c = c + w_t^{[i]}$
  end while
  add $x_t^{[i]}$ a $\bar{X}_t$
end for
return $\bar{X}_t$

Estimación de pose

La pose estimada se obtiene como la media del conjunto de partículas tras el remuestreo. La posición se promedia aritméticamente, y la orientación sobre el círculo unitario para evitar discontinuidades en $\pm\pi$:

$$\hat{x} = \frac{1}{M}\sum_{m=1}^{M} x^{[m]}, \qquad \hat{y} = \frac{1}{M}\sum_{m=1}^{M} y^{[m]}, \qquad \hat{\theta} = \operatorname{atan2}\left(\sum_{m=1}^{M} \sin\theta^{[m]},\ \sum_{m=1}^{M} \cos\theta^{[m]}\right). \tag{3.18}$$

En este capítulo se describió el sistema de localización visual desarrollado, desde la construcción y curaduría del conjunto de datos hasta el entrenamiento de los modelos de detección y el diseño del filtro de partículas de Monte Carlo con modelo de movimiento adaptado a la locomoción bípeda. En el siguiente capítulo se presentan los resultados obtenidos al integrar estos componentes sobre la plataforma comercial NAO de Aldebaran, evaluando tanto el desempeño de los detectores como el comportamiento del sistema de estimación de pose en condiciones reales de operación.

Capítulo 4

Resultados

En este capítulo se presentan los resultados obtenidos a partir de la implementación del sistema de localización visual desarrollado. Se describen las condiciones generales de ejecución, la plataforma ROS utilizada para la integración de los módulos de percepción y estimación, así como el comportamiento observado durante las pruebas realizadas en escenarios reales y simulados.

Asimismo, se analizan las principales medidas de desempeño del sistema, entre ellas la precisión de localización, la estabilidad de las estimaciones, el tiempo de respuesta y el grado de robustez ante variaciones en la escena. De manera complementaria, se comparan los resultados de los distintos modelos evaluados, con el fin de identificar sus ventajas y limitaciones dentro del contexto de aplicación.

Se incluyen las métricas más relevantes obtenidas durante el entrenamiento y la evaluación de los modelos de detección, así como una discusión sobre el impacto de estos resultados en el desempeño global del sistema de localización visual.

4.1. Plataforma ROS

ROS (Robot Operating System) es un ecosistema de código abierto que proporciona el marco de trabajo, las herramientas y las bibliotecas necesarias para construir, desplegar, ejecutar y mantener aplicaciones relacionadas con la robótica. ROS es usado en muchas áreas de la robótica, por ejemplo, en logística, ayuda a los robots a mover mercancías en almacenes al proporcionar navegación, mapeo, control de movimiento y coordinación entre múltiples robots. [31] En manufactura, permite tareas avanzadas como operaciones automatizadas de recolección y colocación mediante sistemas de visión para una manipulación precisa. En el ámbito de la salud, ROS da soporte a sistemas robóticos que asisten en la atención de pacientes y mejoran la eficiencia de los flujos de trabajo clínicos [31].

A partir de ROS 1, surgió la necesidad de evolucionar hacia una arquitectura

más robusta y flexible. ROS 2 fue concebido como una reingeniería de ROS 1 para responder a exigencias modernas de confiabilidad, rendimiento e interoperabilidad. En ROS 1, el uso de sockets TCP/IP podía generar variaciones en la latencia (*jitter*) en canalizaciones de alto rendimiento, como las basadas en LiDAR o visión por computadora. ROS 2 mitiga este problema mediante DDS, que incorpora políticas de tiempo real como entrega fiable o de mejor esfuerzo, control de plazos y monitoreo de actividad, permitiendo equilibrar el rendimiento con el determinismo [5].

Además ROS 2 incorpora mecanismos como la comunicación entre procesos mediante memoria compartida y la transferencia sin copia (*zero-copy*), los cuales reducen la sobrecarga de serialización y mejoran la latencia. Cuando estas capacidades se combinan con soporte de sistemas operativos en tiempo real, como PREEMPT_RT, el sistema puede acercarse a una ejecución casi en tiempo real en aplicaciones como control automotriz o robótica quirúrgica. En conjunto con la aplicación de QoS y el monitoreo de actividad, estas características proporcionan una base sólida para una ejecución predecible, aunque en implementaciones prácticas todavía pueden observarse variaciones bajo condiciones de alta carga [5].

4.2. Reconocimiento de marcas del campo de juego

En esta sección se evalúa el desempeño de los tres modelos de detección entrenados, YOLOv8, Deformable DETR y RT-DETR sobre el conjunto de prueba independiente de imágenes capturadas durante la competencia RoboCup 2025. Cada imagen fue anotada manualmente con cajas delimitadoras de referencia (*ground truth*) para las cinco clases de interés: balón, portería, intersección en T, intersección en L e intersección en X. Las métricas reportadas se calcularon por clase y por modelo a partir de la comparación entre las predicciones y las anotaciones de referencia.

4.2.1. Medidas de desempeño

La evaluación de sistemas de detección de objetos requiere métricas que capturen tanto la capacidad del modelo de encontrar los objetos de interés como la confiabilidad de sus predicciones. A continuación, se describen las medidas utilizadas en este trabajo.

Matriz de confusión

La *matriz de confusión* es una tabla que resume el desempeño de un modelo de detección comparando las etiquetas predichas contra las etiquetas reales. Para el caso binario (objeto presente o ausente), sus cuatro celdas son: Verdaderos Positivos (TP), cuando el modelo detecta correctamente un objeto presente; Falsos Positivos (FP), el caso donde el modelo detecta un objeto que no existe;

Falsos Negativos (FN) son si el modelo omite un objeto presente; y Verdaderos Negativos (TN), cuando el modelo detecta un objeto donde no lo hay. A partir de estas cuatro celdas se derivan todas las métricas de evaluación [37].

Precisión y Sensibilidad

La **Precisión** mide qué fracción de las detecciones realizadas por el modelo corresponden realmente a objetos presentes [37]:

$$\text{Precisión} = \frac{TP}{TP + FP} \tag{4.1}$$

Un valor de Precisión alto indica que cuando el modelo detecta una marca, es probable que esa detección sea correcta. Sin embargo, con la Precisión no se puede saber cuántas marcas reales fueron encontradas.

La **Sensibilidad** o *Recall* mide qué fracción de los objetos presentes fueron detectados por el modelo [37]:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.2}$$

Un valor de sensibilidad alto indica que el modelo tiene baja tasa de omisiones, lo cual es relevante para el sistema de localización debido a que una marca no detectada equivale a información perdida para el filtro de partículas. Como Powers explica [37], Precisión y Recall son métricas complementarias y ninguna es suficiente por sí sola, ya que la primera ignora los falsos negativos y la segunda ignora los falsos positivos.

Para que una detección se considere un verdadero positivo se requiere que la métrica de evaluación, que en la mayoría de los casos es la caja delimitadora predicha, tenga una intersección sobre la unión (IoU) con la caja de referencia mayor o igual a 0,5, umbral estándar en evaluación de detección de objetos.

Error de localización angular

Además de detectar que exista una marca, el sistema de localización necesita conocer con precisión el ángulo al que se encuentra. Por ello se reportan dos métricas adicionales sobre el error de localización del centro horizontal de la caja delimitadora, expresado en píxeles:

- $\boldsymbol{\mu}_{\text{err}}$: error medio (sesgo sistemático) entre el centro horizontal predicho $\hat{u}$ y el centro de referencia u^*, calculado sobre todos los verdaderos positivos de cada clase:

$$\mu_{\text{err}} = \frac{1}{|TP|} \sum_{i \in TP} |\hat{u}_i - u_i^*| \tag{4.3}$$

- $\boldsymbol{\sigma}$: desviación estándar del mismo error, que cuantifica la dispersión de la localización horizontal alrededor del valor medio:

$$\sigma = \sqrt{\frac{1}{|TP|} \sum_{i \in TP} \left(|\hat{u}_i - u_i^*| - \mu_{\text{err}}\right)^2} \tag{4.4}$$

Ambas métricas se expresan en píxeles. Dado que el ángulo de rumbo se calcula mediante el mapeo lineal descrito en la Sección 3.5, un error horizontal de Δu píxeles se traduce directamente en un error angular de $\Delta\phi = \Delta u \cdot \mathrm{FOV}_h / W$ radianes, donde W es el ancho de la imagen en píxeles. Para la cámara utilizada ($W = 640\,\mathrm{px}$, $\mathrm{FOV}_h \approx 53{,}7$), un error de $5\,\mathrm{px}$ corresponde aproximadamente a 0,42, valor dentro del margen de $\sigma_\phi = 5$ asumido por el modelo de observación.

Resultados por clase y modelo

Para contextualizar el rendimiento de las arquitecturas, es indispensable analizar primero la composición del conjunto de datos de entrenamiento (constituido por 8,900 imágenes reales). La distribución de instancias anotadas por clase presenta una variabilidad que influye directamente en la capacidad de generalización de los modelos: la clase intersección en L cuenta con el mayor volumen de ejemplos (7,085 instancias), seguida por T (5,926), *goalposts* (5,423), X (4,907) y, finalmente, *ball* (4,138).

El Cuadro 4.1 presenta las métricas obtenidas por cada modelo sobre el conjunto de prueba independiente. Los valores de Precisión y Recall se calcularon con umbral de IoU = 0,5; μ_{err} y σ se calcularon únicamente sobre los verdaderos positivos de cada clase.

Cuadro 4.1: Métricas de detección por clase y modelo sobre el conjunto de prueba (90 imágenes, RoboCup 2025). μ_{err} y σ en píxeles.

Clase	YOLOv8			
	Precisión	Sensibilidad	σ	μ_{err}
Ball	0.996	0.882	2.48	3.47
Goal	0.897	0.625	3.76	5.41
T	0.799	0.459	3.73	4.77
L	0.814	0.591	3.74	4.72
X	0.781	0.843	4.21	5.81
Clase	Deformable DETR			
	Precisión	Sensibilidad	σ	μ_{err}
Ball	0.996	0.958	2.60	2.44
Goal	0.960	0.833	4.49	3.11
T	0.948	0.639	1.74	0.47
L	0.851	0.893	2.29	0.58
X	0.961	0.827	2.07	0.46
Clase	RT-DETR			
	Precisión	Sensibilidad	σ	μ_{err}
Ball	0.948	0.985	2.56	3.56
Goal	0.926	0.825	4.15	6.00
T	0.588	0.661	4.03	5.02
L	0.569	0.903	3.71	5.22
X	0.810	0.900	4.09	5.38

Los resultados muestran que Deformable DETR obtiene los errores de localización más bajos en las clases de intersecciones (T, L, X), con $\mu_{\text{err}} < 1\,\text{px}$ en los tres tipos, lo que se traduce en errores angulares menores a 0,08. YOLOv8 presenta la Precisión más alta en balón (0.996) pero el Recall más bajo en intersecciones en T (0.459), lo que implica una tasa elevada de omisiones para esa clase. RT-DETR alcanza el Recall más alto en balón (0.985) y en intersecciones en L (0.903), aunque con Precisión más baja que los otros dos modelos en las clases de intersecciones, indicando mayor tasa de falsas detecciones. Para el sistema de localización, el Recall tiene mayor peso que la Precisión: una marca no detectada no contribuye a la actualización del filtro, mientras que una falsa detección es parcialmente absorbida por las penalizaciones suaves del modelo de observación (Sección 3.6).

Adicionalmente, la dinámica del aprendizaje y la estabilidad de las arquitecturas se evaluaron mediante las curvas de convergencia obtenidas de los datos del entrenamiento (Figura 4.1).

Figura 4.1: Curvas de convergencia comparativas que muestran la evolución de las funciones de pérdida (loss) y el comportamiento del mAP a lo largo de las épocas de entrenamiento para YOLOv8, RT-DETR y Deformable DETR.

Es importante destacar que el entrenamiento se dividió en dos configuraciones de hardware distintas. Los modelos YOLOv8 y RT-DETR fueron entrenados utilizando una unidad de procesamiento gráfico (GPU) NVIDIA GeForce RTX 3070 Ti con 8 GB de memoria VRAM. Por otro lado, Deformable DETR se procesó empleando una GPU NVIDIA GeForce RTX 4090 con 24 GB de memoria VRAM.

Como se observa en la Figura 4.1, YOLOv8 exhibe una convergencia acelerada, reduciendo drásticamente sus pérdidas de caja y clasificación en las primeras épocas de entrenamiento debido a la ligereza y optimización estructural de sus

capas convolucionales. En contraste, las arquitecturas basadas en Transformers muestran curvas con una pendiente inicial más suave. RT-DETR presenta una evolución constante y sostenida de la métrica mAP, estabilizándose hacia el final de sus ciclos de procesamiento. Por su parte, Deformable DETR, a pesar de manejar un espacio computacional más complejo debido a sus mapas de atención dispersos, estabiliza sus pérdidas de manera óptima en las épocas tardías. Esta maduración progresiva del proceso de atención global explica por qué, a pesar de un entrenamiento más pausado, Deformable DETR alcanza la mayor consistencia geométrica y los errores de regresión de cajas más bajos (μ_{err}) reportados en el Cuadro 4.1.

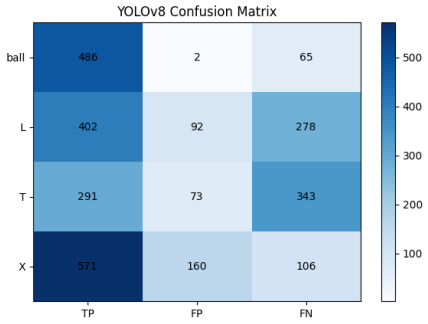


Figura 4.2: Matriz de confusión del modelo: yolov8

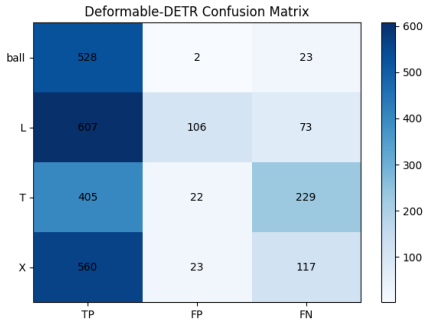


Figura 4.3: Matriz de confusión del modelo: Deformable-DETR

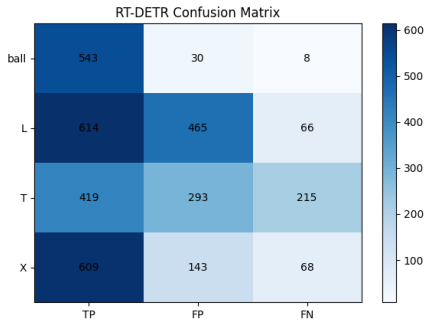


Figura 4.4: Matriz de confusión del modelo: RT-DETR

4.3. Estimación de posición

Las pruebas de estimación de posición se realizaron integrando el módulo de detección con el filtro de partículas de Monte Carlo descrito en la Sección 3.6, ejecutado sobre la plataforma ROS 2. El sistema recibe como entrada las observaciones $z_t = \{(\mathrm{id}_k, \varphi_k)\}$ producidas por el detector y las fusiona con la odometría u_t para mantener una estimación continua de la pose $\hat{x}_t = (\hat{x}, \hat{y}, \hat{\theta})^T$ del robot sobre el campo de juego. La pose estimada se publica en el tópico `/estimated_pose` y se visualiza en RViz junto con el enjambre de partículas sobre el mapa del campo.

4.3.1. Problema del robot secuestrado

El *problema del robot secuestrado* (*kidnapped robot problem*) es un caso de prueba clásico para sistemas de localización probabilística, descrito por Thrun et al. [52]. Consiste en reubicar al robot de forma abrupta e inesperada en una posición arbitraria del entorno, sin notificarle del cambio. Un filtro de partículas que haya convergido a una hipótesis de pose concentrará su enjambre en una región del espacio de estados que deja de ser válida tras el secuestro, por lo que debe ser capaz de detectar la divergencia y redistribuir las partículas para recuperar la localización desde cero [52].

Este escenario evalúa dos capacidades críticas del sistema: la detección de divergencia severa y la recuperación autónoma de la estimación de pose. En la implementación desarrollada, la divergencia se detecta monitoreando el peso máximo del enjambre: cuando $\max_m w_t^{[m]} < 10^{-5}$, se considera que ninguna partícula explica satisfactoriamente las observaciones actuales, lo que indica que la creencia ha colapsado. Ante esta condición, el sistema inyecta automáticamente el 20 % de las partículas como muestras uniformes sobre el campo, preservando el 80 % restante para no descartar hipótesis que pudieran aún ser válidas (Algoritmo 0, líneas 7–9).

La prueba consistió en inicializar el filtro en una posición conocida del campo

y permitir que convergiera hasta obtener una estimación estable; posteriormente, el robot fue reubicado manualmente en una posición distinta, simulando el evento de secuestro. Se observó que, al perder consistencia entre las observaciones visuales y las predicciones del enjambre concentrado, el indicador N_{eff} descendió por debajo del umbral $0{,}4\,M$ y el peso máximo cayó por debajo de 10^{-5}, activando el mecanismo de recuperación. En las iteraciones siguientes, la combinación de la inyección de partículas aleatorias y la ponderación por el modelo de observación permitió que el enjambre se redistribuyera progresivamente hacia la nueva posición real del robot, recuperando una estimación coherente con las detecciones visuales.

El resultado confirma que el filtro de partículas, en combinación con el mecanismo de detección de divergencia, cumple con el requisito de robustez ante perturbaciones abruptas de la pose, comportamiento documentado como una de las ventajas fundamentales del método de Monte Carlo frente al Filtro de Kalman Extendido en el contexto de localización global [52, 16].

4.3.2. Parámetros utilizados

El Cuadro 4.2 resume los parámetros del filtro de partículas empleados durante las pruebas de estimación de posición. Los valores fueron determinados a partir de las características observadas en la odometría del robot y de los resultados de detección descritos en la Sección 4.1.

Cuadro 4.2: Parámetros del filtro de partículas de Monte Carlo.

Parámetro	**Valor**	**Descripción**
M	700	Número de partículas
$[\alpha_1, \alpha_2, \alpha_3, \alpha_4]$	[0,001, 0,0003, 0,04, 0,001]	Ruido del modelo de movimiento por odometría (Tabla 5.6 de [52])
σ_φ	5 (0,087 rad)	Desviación estándar del modelo de error angular en la función de similitud
FOV_h	53,7	Campo de visión horizontal de la cámara (obtenido experimentalmente)
Umbral de activación de movimiento	$\delta_{\text{trans}} > 0{,}02\,\text{m}$ ó $\lvert\delta_{\text{rot}}\rvert > 0{,}02\,\text{rad}$	Condición para propagar el modelo de movimiento
Umbral de divergencia	$\text{máx}_m\, w^{[m]} < 10^{-5}$	Condición para inyectar partículas aleatorias
Fracción de reinicio	20 %	Proporción de partículas reemplazadas tras divergencia
Umbral de remuestreo	$N_{\text{eff}} < 0{,}4\,M$	Condición para ejecutar el muestreo de baja varianza (Tabla 4.4 de [52])
Jitter estático (σ_{xy}, σ_θ)	0,002 m, 0,002 rad	Ruido añadido durante el remuestreo cuando el robot está estático, para preservar diversidad
Límites del campo (x, y)	[±3,04 m, ±4,53 m]	Dimensiones del campo utilizadas en las pruebas

Los parámetros de ruido del modelo de movimiento $\alpha = [\alpha_1, \alpha_2, \alpha_3, \alpha_4]$ modelan cuatro fuentes de error odométrico según el marco probabilístico de [52]. El valor elevado de $\alpha_3 = 0{,}04$, que controla el ruido en la traslación causado por la traslación ejecutada, refleja la inestabilidad característica de la locomoción bípeda: el balanceo lateral de la marcha introduce una dispersión lateral significativa en la trayectoria real del robot respecto a la reportada por la odometría, tal como se ilustró en la Figura 3.4. El resto de parámetros toman valores bajos, dado que los errores rotacionales puros son relativamente pequeños en comparación con la incertidumbre traslacional.

Capítulo 5

Conclusiones

Los experimentos que se realizaron en este trabajo dieron como resultado que el error de localización basado en el centro es una métrica que refleja con mayor fidelidad la calidad para detectar objetos sin restricciones espaciales (como las marcas de un campo de fútbol) que las métricas convencionales de Intersección sobre Unión (IoU). Las marcas de campo de este trabajo, que fueron intersecciones en T, X y L, no corresponden a objetos con geometría rígida, cerrada y definida. Esta ausencia de fronteras físicas introduce una ambigüedad inherente en el etiquetado mediante cajas delimitadoras (*bounding boxes*), lo cual suele penalizar injustamente y confundir a los optimizadores basados en regiones.

La raíz de este problema radica en que una intersección es, vectorialmente, una característica puntual o un punto de cruce; sin embargo, al forzar su delimitación dentro de una caja de dimensiones específicas, las dimensiones y la relación de aspecto de la caja quedan sujetas a criterios arbitrarios de anotación y a las distancias de captura, es decir que tan lejos o cerca está el robot de la marca. Esto confunde al modelo, ya que lo obliga a predecir el tamaño de un rectángulo que cambia constantemente por la perspectiva, incluso a una distancia fija de la marca, en lugar de centrarse en ubicar el punto central. Por este motivo, evaluar la precisión con base en el error del centro mitiga el ruido geométrico de la caja y aporta una métrica metodológicamente coherente con las necesidades del filtro de partículas.

El modelo Deformable-DETR produce las observaciones más precisas para el sistema de localización. En las clases de intersecciones obtuvo errores de localización menores a 1 px ($\mu_{err} < 1$) px en los tres tipos de intersección, así como una sensibilidad (*Recall*) superior a la de YOLOv8 en intersecciones L y X. Sin embargo, YOLOv8 presenta ventajas de desarrollo por su amplia documentación y uso de bibliotecas estandarizadas de Ultralytics contrario a Deformable-DETR que no es *out of the box* y requiere mayor tiempo de desarrollo.

RT-DETR alcanzó la sensibilidad (*Recall*) más alta en el balón (0.985) e intersecciones en L (0.903), pero con una Precisión notablemente inferior, lo que introduce falsas observaciones en el filtro.

La combinación de detección de marcas con transformadores visuales y fil-

tros de partículas demostró ser una configuración adecuada para localizar un robot en un campo de fútbol. La posición estimada, aunque presenta errores relativamente grandes, es suficiente para usarse en la toma de decisiones en aplicaciones de fútbol para robots bípedos autónomos.

Resulta ventajoso utilizar un sistema de localización visual basado en marcas naturales y rumbos angulares, dado que esta implementación utiliza exclusivamente una cámara RGB que es un sensor pasivo y de bajo costo, en lugar de recurrir a sensores activos como el LiDAR que es el enfoque predominante en localización robótica móvil.

5.1. Trabajo futuro

Como una extensión del sistema que se desarrolló en este trabajo se plantean las siguientes lineas de investigación:

Las pruebas realizadas en este trabajo se llevaron a cabo sobre una plataforma específica en condiciones de laboratorio. Una línea de trabajo relevante consiste en trasladar el sistema a otras plataformas humanoides comerciales, como el Booster T1 o el Unitree G1, cuyas diferencias en altura, campo de visión de la cámara, dinámica de marcha y capacidad de cómputo embebido introducirían condiciones distintas de operación. Esto permitiría evaluar la generalidad del enfoque propuesto y calibrar los parámetros del modelo de movimiento en función de las características cinemáticas de cada plataforma. En la implementación actual, el procesamiento se ejecuta fuera del robot. Un objetivo prioritario es migrar las tareas de percepción y estimación para ejecutarse íntegramente en el hardware embebido de la plataforma robótica, como el Booster T1.

Se planea probar el sistema de localización en competencia y probar su funcionalidad bajo condiciones de oclusión de las marcas del campo.

Bibliografía

[1] Configuration — ultralytics docs. `https://docs.ultralytics.com/usage/cfg`, 2023. Ultralytics Documentation.

[2] RT-DETR: Baidu's real-time object detector. `https://docs.ultralytics.com/es/models/rtdetr`, 2023. Ultralytics Documentation.

[3] Abubakar Abidali, Ali Abdalla, Ahmed Abdalla, Ayyaz Ali, and et al. Gradio: Build and share delightful machine learning apps in Python, 2019. Accessed: 2024-05-22.

[4] AI Wiki. Nao (robot), 2025.

[5] Abdulrahman Al-Batati, Anis Koubaa, Khaled Gabr, Mohamed Abdelkader, and Hamad Aloqaily. Ros 2 in a nutshell: A survey. *ACM Computing Surveys*, 2024.

[6] Ahmad Jobran Al-Mahasneh, Sreenatha G Anavatti, and Matthew A Garratt. The development of neural networks applications from perceptron to deep learning. In *2017 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA)*, pages 1–6. IEEE, 2017.

[7] Aislan C Almeida, Anna HR Costa, and Reinaldo AC Bianchi. Vision-based monte-carlo localization for humanoid soccer robots. In *2017 Latin American robotics symposium (LARS) and 2017 Brazilian symposium on robotics (SBR)*, pages 1–6. IEEE, 2017.

[8] Ayoub Benali Amjoud and Mustapha Amrouch. Object detection using deep learning, cnns and vision transformers: A review. *IEEE Access*, 11:35479–35516, 2023.

[9] Awesome Robots. Booster Robotics Robots. `https://www.awesomerobots.xyz/brands/booster-robotics`, 2025. Accedido: 2025.

[10] Marc Bestmann, Timon Engelke, Niklas Fiedler, Jasper Güldenstein, Jan Gutsche, Jonas Hagge, and Florian Vahl. TORSO-21 Dataset: Typical Objects in RoboCup Soccer 2021. In *RoboCup 2021: Robot World Cup XXIV*, 2021.

[11] Hans-Dieter Block. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123, 1962.

[12] Booster Robotics. Booster Robotics — Official Website. `https://www.booster.tech`, 2025. Accedido: 2025.

[13] Booster Robotics. Booster T1 — Made for Developers. `https://www.booster.tech/booster-t1/`, 2025. Accedido: 2025.

[14] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.

[15] Bingqi Chen and Siyao Chen. Deep learning. In *Machine Vision Technology*, pages 273–297. Springer, 2026.

[16] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. *ICRA*, 1999.

[17] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[18] Keisuke Fujii. Extended kalman filter. *Refernce Manual*, 14(41):2, 2013.

[19] Tetsuharu Fukaushima, Yoshihiro Kuroki, and Tatsuzou Ishida. Development of a new actuator for a small biped walking entertainment robot. In *Second IEE International Conference on Power Electronics, Machines and Drives*, pages v1–126. IET, 2004.

[20] Marylou Gabrié, Surya Ganguli, Carlo Lucibello, and Riccardo Zecchina. Neural networks: From the perceptron to deep nets. In *Spin Glass Theory and Far Beyond: Replica Symmetry Breaking After 40 Years*, pages 477–497. World Scientific, 2023.

[21] D Gouaillier, V Hugel, P Blazevic, C Kilner, J Monceaux, P Lafourcade, B Marnier, J Serre, and B Maisonnier. The nao humanoid: A combination of performance and affordability. arxiv. *arXiv preprint arXiv:0807.3223*, 2008.

[22] Priyanto Hidayatullah, Nurjannah Syakrani, Muhammad Rizqi Sholahuddin, Trisna Gelar, and Refdinal Tubagus. Yolov8 to yolo11: A comprehensive architecture in-depth comparative review. *arXiv preprint arXiv:2501.13400*, 2025.

[23] Wei Hong, Changjiu Zhou, and Yantao Tian. Robust monte carlo localization for humanoid soccer robot. In *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 934–939. IEEE, 2009.

[24] Humanoids Daily. Booster Robotics Launches K1: A RoboCup-Winning Humanoid for Education and Research. `https://www.humanoidsdaily.com/news/booster-robotics-launches-k1-robocup-champion-platform`, 2025. Accedido: 2025.

[25] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2023.

[26] Nikhil Ketkar and Jojo Moolayil. Convolutional neural networks. In *Deep learning with Python: learn best practices of deep learning models with PyTorch*, pages 197–242. Springer, 2021.

[27] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.

[28] Pedro Lima et al. Localization in robotic soccer: A review. In *RoboCup 2009: Robot Soccer World Cup XIII*. Springer, 2010.

[29] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.

[30] Wenyu Lv, Shangliang Xu, Yian Zhao, Guanzhong Wang, Jinman Wei, Cheng Cui, Yuning Du, Qingqing Dang, and Yi Liu. Detrs beat yolos on real-time object detection, 2023.

[31] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[32] Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12309–12318, 2022.

[33] Kenichiro Nagasaka. Sony qrio. *Humanoid Robotics: A Reference*, pages 187–200, 2019.

[34] Imre Nagi, Widyawardana Adiprawita, and Kusprasapta Mutijarsa. Vision-based monte carlo localization for robocup humanoid kid-size league. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 1433–1438. IEEE, 2014.

[35] Keiron O'shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[36] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS transactions on circuits and systems*, 8(7):579–588, 2009.

[37] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[38] Rohan Putatunda, Md Azim Khan, Aryya Gangopadhyay, Jianwu Wang, Carl Busart, and Robert F Erbacher. Vision transformer-based real-time camouflaged object detection system at edge. In *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 90–97. IEEE, 2023.

[39] Diogo Reis, Victor J. Silva, Nuno Lau, and Luis Paulo Reis. Deep learning for soccer robotics. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, 2019.

[40] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741, 2004.

[41] ROBOTIS. Robotis op3 introduction, 2024.

[42] Robots Guide. Unitree g1, 2024.

[43] RobotShop. Booster T1 Humanoid Robot (Standard). `https://www.robotshop.com/products/booster-t1-humanoid-robot-standard`, 2025. Accedido: 2025.

[44] Frank Rosenblatt et al. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, volume 55. Spartan books Washington, DC, 1962.

[45] Ranjan Sapkota and Manoj Karkee. Ultralytics yolo evolution: An overview of yolo26, yolo11, yolov8 and yolov5 object detectors for computer vision and pattern recognition. *arXiv preprint arXiv:2510.09653*, 2025.

[46] Satoshi Shigemi, Ambarish Goswami, and Prahlad Vadakkepat. Asimo and humanoid robot research at honda. *Humanoid robotics: A reference*, 55:90, 2018.

[47] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[48] Manuel F Silva and JA Tenreiro Machado. A historical perspective of legged robots. *Journal of Vibration and Control*, 13(9-10):1447–1486, 2007.

[49] Mupparaju Sohan, Thotakura Sai Ram, and Ch Venkata Rami Reddy. A review on yolov8 and its advancements. In *International conference on data intelligence and cognitive informatics*, pages 529–545. Springer, 2024.

[50] Moahaimen Talib, Ahmed HY Al-Noori, and Jameelah Suad. Yolov8-cab: Improved yolov8 for real-time object detection. *Karbala International Journal of Modern Science*, 10(1):5, 2024.

[51] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[52] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial intelligence*, 128(1-2):99–141, 2001.

[53] Top 3D Shop. Unitree G1 Humanoid Robot by Unitree Robotics. `https://top3dshop.com/product/unitree-robotics-g1-humanoid-robot`, 2025. Accedido: 2025.

[54] S Vaishnavi, R Lalitha, and BS Ashwin Prabhakar. The humanoid robot (darwin-op).

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[56] Xinhua News Agency. Chinese team wins RoboCup Humanoid League in AdultSize category. `https://en.people.cn/n3/2025/0724/c90000-20344358.html`, July 2025. Accedido: 2025.

[57] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.