UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Análisis e integración de algoritmos de visión artificial con Turtlebot4 y ROS2

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Armero Martínez, Esmeralda

Tutor/a: Ivorra Martínez, Eugenio

CURSO ACADÉMICO: 2023/2024

Resumen

Los algoritmos de visión por computadora, especialmente las redes neuronales convencionales (CNN), son herramientas fundamentales que permiten a los robots procesar, analizar e interpretar información visual a partir de imágenes o vídeos. Utilizando estas técnicas inspiradas en la estructura y función del sistema visual biológico, los robots pueden comprender y actuar en entornos visuales de manera autónoma.

En el presente trabajo, se realizará un análisis de los algoritmos de visión utilizando el robot Turtlebot4 y ROS2 como plataforma de desarrollo. A través de este proyecto, se aplicarán conocimientos de visión como el reconocimiento de patrones o el procesamiento de imágenes en tiempo real, además de profundizar en el uso de la plataforma ROS (*Robot Operating System*).

Palabras clave: Turtlebot, ROS, oak-d, visión.

Resum

Els algoritmes de visió per computadora, especialment les xarxes neuronals convencionals (CNN), són ferramentes fonamentals que permeten als robots processar, analitzar i interpretar informació visual a partir d'imatges o vídeos. Utilitzant estes tècniques inspirades en l'estructura i funció del sistema visual biològic, els robots poden comprendre i actuar en entorns visuals de manera autònoma.

En el present treball, es realitzarà una anàlisi dels algoritmes de visió utilitzant el robot Turtlebot4 i ROS2 com a plataforma de desenvolupament. A través d'aquest projecte, s'aplicaran coneixements de visió com el reconeixement de patrons o el processament d'imatges en temps real, a més d'aprofundir en l'ús de la plataforma ROS (*Robot Operating System*).

Paraules clau: Turtlebot, ROS, oak-d, visió.

Abstract

Computer vision algorithms, especially Convolutional Neural Networks (CNNs), are fundamental tools that enable robots to process, analyze, and interpret visual information from images or videos. By utilizing these techniques inspired by the structure and function of the biological visual system, robots can comprehend and act in visual environments autonomously.

In this work, an analysis of vision algorithms will be conducted using the Turtlebot4 robot and ROS2 as the development platform. Through this project, knowledge of vision, such as pattern recognition or real-time image processing, will be applied, in addition to delving into the use of the ROS platform (*Robot Operating System*).

Keywords: Turtlebot, ROS, oak-d, vision.

Índice de contenidos

Capítulo 1: Memoria

Capítulo 4: Presupuesto

Anexo A

Anexo B

Anexo C

Índice de figuras

Índice de tablas

Capítulo 1
Memoria

1. Introducción

Los avances en el campo de la visión artificial han permitido el desarrollo de tecnologías que capacitan a los robots para entender y actuar en entornos complejos. Esto ha sido crucial para dotar a los robots de la capacidad de interpretar información visual, utilizando técnicas avanzadas como las redes neuronales convolucionales (CNN). Estas técnicas permiten a los robots realizar tareas de manera autónoma al emular ciertas capacidades del sistema visual humano.

El presente Trabajo de Fin de Grado propone investigar y aplicar algoritmos de visión artificial en un TurtleBot 4. Este proyecto tiene como objetivo explorar cómo los algoritmos de visión, en combinación con la plataforma ROS2 (*Robot Operating System*), pueden mejorar la capacidad del robot para interactuar con su entorno.

1.1. Objeto

El objetivo principal de este proyecto es desarrollar e implementar un sistema autónomo para el seguimiento de personas utilizando el robot TurtleBot 4. Para logarlo, se empleará la cámara OAK-D integrada en el robot, que permite capturar imágenes y ejecutar algoritmos de visión artificial. Este sistema de visión detectará y seguirá a personas de manera precisa.

1.2. Estructura del proyecto

La estructura del proyecto empleada asegura una progresión lógica y coherente del desarrollo del proyecto, desde la teoría hasta la práctica y la evaluación. Y se organiza de la siguiente manera:

En la **primera fase**, se establecen los antecedentes teóricos y técnicos, donde se exploran conceptos fundamentales de visión artificial, el Deep Learning, y la historia de los robots con capacidad de visión. Esta base teórica es crucial para comprender las tecnologías y métodos que se utilizarán en el proyecto.

En la **fase de diseño y planificación**, se definen los requerimientos del proyecto y se evalúan las diferentes alternativas de diseño. Este análisis permite seleccionar las opciones más adecuadas para cumplir con los objetivos del proyecto, teniendo en cuenta las especificaciones necesarias y las posibles limitaciones.

Finalmente, en **la fase de implementación y pruebas**, se desarrolla la solución adoptada. Esto incluye la puesta en marcha del TurtleBot 4, la configuración de la cámara y la implementación de algoritmos de detección y seguimiento. Se realizan pruebas para validar la funcionalidad de los diseños y la eficiencia del sistema, y se analizan los resultados obtenidos para identificar posibles mejoras.

2. Antecedentes

2.1. Introducción a los Sistemas de Visión Artificial

La visión artificial es una disciplina científica que se dedica al desarrollo de sistemas capaces de interpretar y comprender imágenes y vídeos del mundo real, de manera similar a como lo hace el ojo humano. Estos sistemas, conocidos como Sistemas de Visión Artificial, se basan en el procesamiento y análisis de imágenes digitales adquiridas por cámaras con el fin de producir información que pueda ser tratada por algoritmos complejos.

La conceptualización inicial de estos sistemas de visión está influenciada por la literatura de ciencia ficción. Obras como *"R.U.R."* de Karel Čapek o *"I, Robot"* de Isaac Asimov plantearon ideas sobre la capacidad de los "robots" para percibir y comprender su entorno, lo que influyó significativamente en los investigadores que buscaban dotar a las máquinas de inteligencia.

Los primeros esfuerzos académicos para desarrollar la visión artificial comenzaron a tomar forma en la década de los sesenta. En 1963, Laurence G. Roberts exploró en su tesis *"Machine perception of Three-Dimensional Solids"* cómo las máquinas podían percibir y comprender objetos tridimensionales. Roberts sentó los principios fundamentales para la visión por computadora.

En 1966, Seymour Papert lideró *"The Summer Vision Project",* cuyo objetivo era replicar el sistema de visión humano en máquinas. Este proyector fue trascendente, ya que establece las bases de la segmentación del fondo en imágenes reales, una técnica crucial para el reconocimiento de patrones.

Otro hito importante ocurrió en 1971 con el trabajo *"On Seeing Things"* de Max Clowes, que se enfocó en la interpretación de imágenes 3D y en el etiquetado de líneas. Este trabajo contribuyó a desarrollar técnicas que permiten a las computadoras interpretar el mundo tridimensional.

Finalmente, David Marr publicó *"Vision: A Computational Investigation into the Human Representation and Processing of Visual Information"* en 1982. Propuso un modelo informático de visión artificial que opera siguiendo los procedimientos del sistema visual humano. Además, introdujo la idea de descripciones simbólicas de los objetos que componen las escenas.

Todos estos esfuerzos y avances académicos han establecido las bases para la tecnología moderna, la cual se encuentra omnipresente en muchos ámbitos y va ligada a la inteligencia artificial y a las redes neuronales profundas. En definitiva, la visión artificial se ha convertido en una tecnología esencial en muchos sectores. Desde el industrial para el control de calidad y automatización, hasta en la medicina para el diagnóstico por imágenes y en la cirugía asistida por robots. Sin embargo, a pesar de sus avances, también existen preocupaciones éticas relacionadas con la privacidad y la recopilación de datos, como el uso de tecnologías de reconocimiento facial.

2.2. Diseño de los Sistemas de Visión Artificial

Dependiendo de cada aplicación de Visión Artificial y de sus especificaciones, existen unas etapas a seguir a la hora de diseñar ese sistema. No necesariamente se deben de aplicar todas, ya que hay veces que solo se tiene en cuenta un subconjunto de fases. Sin embargo, el esquema general de etapas de diseño de un Sistema de Visión Artificial sigue la distribución de la Figura 1. Se debe tener en cuenta que, aunque en el esquema aparezca un encadenamiento entre etapas, en la práctica siempre existe realimentación entre las distintas fases.

Las etapas de diseño son:

- **Adquisición**: proceso de adquirir, transmitir y convertir las imágenes.

- **Procesamiento**:
    - **Preproceso**: preparación de la imagen adquirida antes de su procesamiento principal.
    - **Segmentación**: separación el fondo de los objetos de interés en la imagen.

- **Modelos de conocimiento**:
    - **Descripción**: etiquetado los objetos con información útil (características) para identificarlos y distinguirlos. Esto requiere un proceso de aprendizaje para determinar las características más discriminantes o relevantes para la aplicación.
    - **Reconocimiento**: clasificación de los objetos previamente etiquetados.
    - **Resultados y decisiones**: toma de decisiones basadas en la identificación de los objetos.

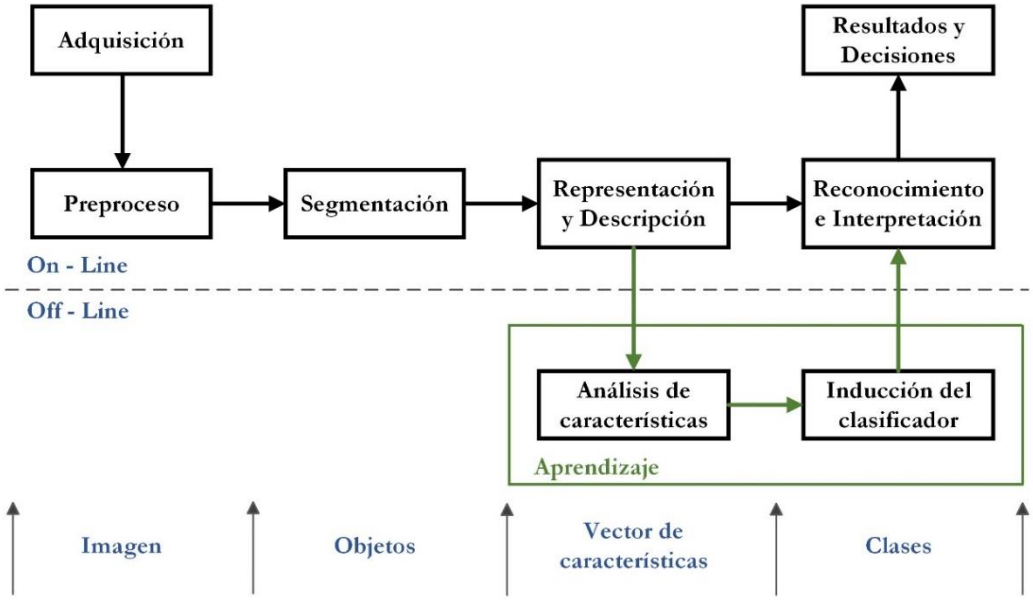


Figura 1 - Etapas del diseño de un Sistema de Visión Artificial

2.2.1. Adquisición de imágenes

La adquisición de imágenes es el primer paso crucial en cualquier sistema de visión artificial. Este proceso implica capturar, transmitir y convertir imágenes de alta calidad para su análisis posterior. A continuación, se detallan los componentes y consideraciones necesarios para una adquisición de imágenes efectiva:

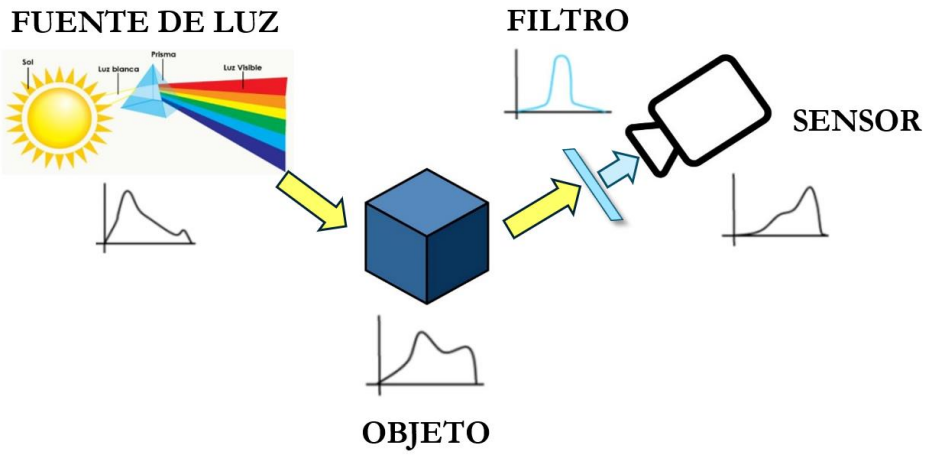


Figura 2 - Esquema general del proceso de adquisición

Iluminación

La iluminación es un factor determinante en la calidad y efectividad de las imágenes capturadas. Su objetivo es optimizar el contraste, reducir las sombras no deseadas y el ruido, y resaltar características de interés del objeto. Asegura que las imágenes sean claras, detalladas y consistentes facilitando su procesamiento posterior.

Dependiendo de la aplicación del sistema, se deben de seleccionar los tipos (led, incandescentes, láser…) y técnicas de iluminación (frontal, lateral, contraluz…) correctos, ya que optimizan las capturas de las imágenes.

Por otro lado, se pueden acoplar los filtros ópticos. Estos se utilizan para separar unas determinadas longitudes de ondas, transmiten la longitud de onda deseada y absorben las restantes.

Tipos de cámaras

Las cámaras capturan las imágenes y pueden variar en función del tipo de óptica (enfoque, diafragma, distancia focal…), del tipo de sensor y del tiempo de apertura del obturador. Existen cámaras CCD (dispositivo de transferencia de carga) y CMOS (*complementay metal-oxide-semiconductor*), son dos tipos de tecnologías de sensores de imagen utilizados en cámaras digitales.

Las cámaras CCD utilizan un método de transferencia de carga para mover la carga eléctrica de un píxel a otro y finalmente convertirla en voltaje en único conversor analógico-digital. Proporcionan imágenes de alta calidad con menos ruido, especialmente en condiciones de poca luz. Sin embargo, su procesamiento de imágenes es más lento debido a la transferencia secuencial de carga y esto provoca un mayor consumo.

Las cámaras CMOS, utilizan transistores en cada píxel para convertir la luz en carga eléctrica y luego en voltaje, cada píxel tiene su proprio amplificador y conversor. Pueden procesar imágenes más rápido debido a que cada píxel puede ser accedido y procesado individualmente, esto provoca un menor consumo de energía. Tradicionalmente, estos sensores generaban más ruido que los CCD, aunque las nuevas tecnologías han mitigado este problema.

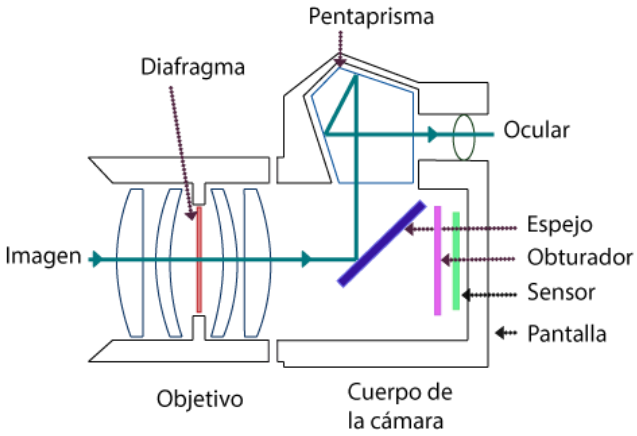


Figura 3 - Esquema de las partes de una cámara

Modelo de Cámara Pin-hole

Una de las bases teóricas de la adquisición de imágenes en 3D es el modelo de Cámara Pin-hole (estenopeica). Este modelo se basa en una cámara sin lentes, donde la luz entra por un pequeño orificio (*pinhole*) y proyecta una imagen invertida sobre un plano de proyección en el que inciden los rayos de luz procedentes del objeto. En definitiva, este modelo describe la relación matemática entre las coordenadas de un punto en el espacio tridimensional y su proyección en el plano de la imagen.

Para conseguir una imagen idéntica a la generada en el plano de la imagen, pero sin estar invertida se crea el plano de imagen virtual. Este plano situado entre el orificio de la cámara y el objeto (Figura 4) simplifica la modelización de la cámara.

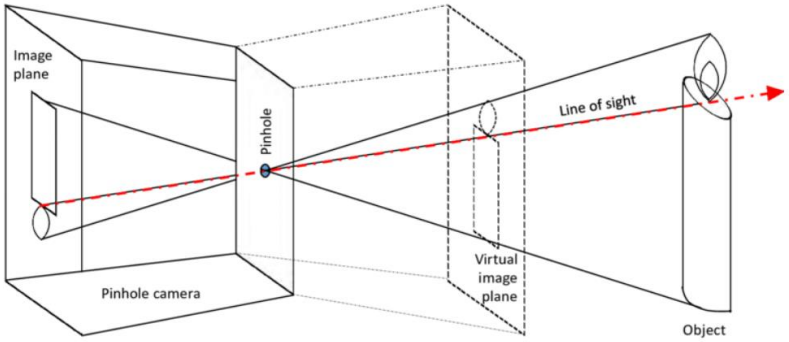


Figura 4 - Esquema plano imagen virtual

Proyección de la perspectiva

Para calcular las coordenadas de un objeto en la imagen en función de las coordenadas en el mundo real, se deben de establecer los sistemas de referencias y sus relaciones. Para ello, el modelo proyectivo de la cámara relaciona los puntos del plano de la imagen (mundo p) respecto al sistema de coordenadas de la escena 3D donde se encuentran los objetos (*world* w).

El modelo proyectivo se descompone en tres matrices de transformación: la matriz de parámetros extrínsecos (T), la matriz de proyección de perspectiva (P) y la matriz de parámetros intrínsecos (K).

$$\begin{bmatrix} u' \\ v' \\ s \end{bmatrix} = \lambda \cdot K \cdot T \cdot P \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u' \\ v' \\ s \end{bmatrix} = \lambda \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Donde:

- $\alpha_x = f_x/d_x$ distancia focal vertical en píxeles.
- $\alpha_y = f_y/d_y$ distancia focal horizontal en píxeles.
- d_x, d_y: Dimensiones del píxel en mm.
- $[x_0,\ y_0]^T$: Coordenadas del punto principal en píxeles.

Calibración de la cámara

El proceso de calibración de la cámara permite realizar una estimación de los parámetros intrínsecos y extrínsecos de la cámara a partir de una imagen. Requiere de un patrón externo cuya estructura y dimensiones sean conocidas a priori, como el patrón de calibración 2D de Zhang. Mediante este proceso, se pueden corregir distorsiones y mejorar la precisión de las mediciones y la proyección de puntos en el espacio 3D.

Proyección inversa

La proyección inversa es el proceso de recuperar las coordenadas tridimensionales de un punto en el espacio a partir de sus coordenadas bidimensionales de una imagen. Para solucionar el problema de pérdida de información al proyectar sobre un plano 2D surgen técnicas de visión 3D como la estereoscopía. Esta técnica utiliza dos cámaras iguales, cuyos ejes ópticos son paralelos y la traslación entre sus centros de proyección es paralela a las filas de la imagen, es decir, esta configuración es similar a la de los ojos humanos.

La reconstrucción de imágenes en configuración de cámara en paralelos (Figura 5) consiste en capturar dos imágenes de la misma escena desde diferentes ángulos y sabiendo la posición geométrica relativa de las cámaras y comparando las dos imágenes, se puede identificar la disparidad entre ambas. A partir del mapa de disparidades, se obtiene el mapa de profundidades, lo cual permite la reconstrucción de una imagen tridimensional.

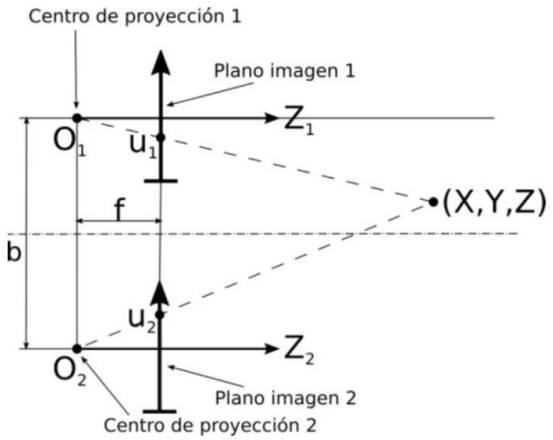


Figura 5 - Esquema configuración en paralelo

2.2.2. Procesamiento de imágenes

El procesamiento de imágenes abarca una serie de técnicas y métodos que transforman las imágenes capturadas en datos útiles. El objetivo principal es aumentar la calidad de las imágenes y extraer la información relevante que se necesite para las diferentes aplicaciones de los sistemas de visión artificial.

La fase inicial de esta etapa es el **preprocesamiento** de imágenes. Esta fase prepara las imágenes para un correcto análisis posterior, existen distintos métodos de preproceso como:

- **Operaciones entre imágenes:** combinaciones matemáticas de dos o más imágenes para obtener una nueva imagen deseada. Por ejemplo: la resta de imágenes se utiliza para eliminar el fondo o para la detección de objetos, el promediado de imágenes consigue eliminar el ruido, la normalización de la iluminación por división, etc.

- **Correcciones geométricas:** transforman las imágenes modificando la posición de los píxeles. Su objetivo es transformar los valores de una imagen, tal y como podrían observarse desde otro punto de vista, corrigen las distorsiones causadas por la perspectiva o por las propiedades de la óptica.

- **Filtros puntuales:** afectan cada píxel de la imagen de manera independiente. Por ejemplo: el realce del contraste para dar más profundidad y definir más las líneas y bordes, la compresión del contraste consigue que áreas menos visibles queden más claras, la normalización y ecualización del histograma para mejorar el contraste, etc.

- **Filtros espaciales:** consideran el valor del píxel de interés y sus vecinos. Estos filtros se utilizan para suavizar (paso bajo), agudizar, o detectar bordes (paso alto) en la imagen. Algunos ejemplos son: el Gaussiano y el Sobel.

Después de preparar la imagen según las necesidades del sistema de visión artificial, se deben de separar los objetos de interés de la imagen. Para ello, la **segmentación** es un proceso clave que divide una imagen en regiones significativas para facilitar su análisis. Se realiza binarizando la imagen, asignando a los píxeles de los objetos de interés y a los pertenecientes al fondo un valor distinto.

Existen métodos basados en el umbral, donde los píxeles con un valor de intensidad superior se consideran de interés, y métodos más avanzados como la segmentación basada en regiones o la adaptativa, que consideran también la textura.

Además, existen operaciones con imágenes binarias como las operaciones aritméticas y lógicas (AND, OR, XOR, NOT), operaciones morfológicas (Figura 6) que realizan modificaciones sobre la forma o estructura de una imagen binaria, etc.

En resumen, el procesamiento es una etapa fundamental para preparar y analizar datos visuales de manera efectiva. Las técnicas de corrección geométrica, filtrado y segmentación permiten mejorar la calidad de las imágenes y extraer información relevante para una amplia gama de aplicaciones tecnológicas y científicas.

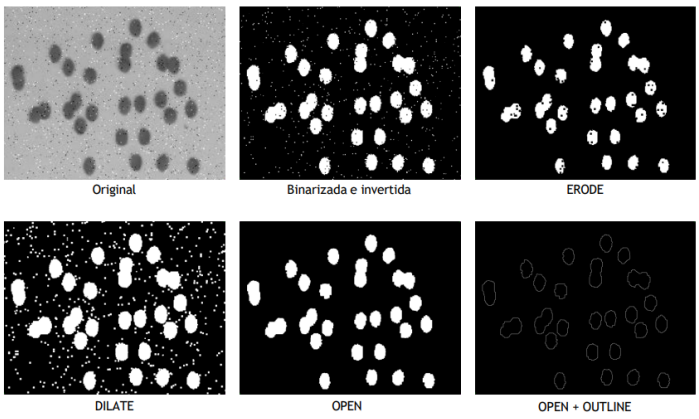


Figura 6 - Operaciones morfológicas

2.2.3. Modelos de conocimiento

El uso de modelos de conocimiento en visión artificial es esencial para interpretar y comprender toda la información visual que se ha obtenido. Estos modelos permiten transformar los datos u objetos de las imágenes en conocimiento útil, facilitando así la toma de decisiones en diversas aplicaciones tecnológicas.

Representación y descripción

La representación y descripción permite transformar los datos visuales en formas reconocibles y utilizables por algoritmos de análisis, esto es gracias a distintos métodos y técnicas como el etiquetado que consiste en asignar a todos los píxeles de cada componente conexa un mismo valor, la "Etiqueta" o "Categoría", que será única en la imagen. Además, incluye la identificación de bordes, texturas, formas y colores que pueden ser utilizados para diferenciar entre diferentes clases de objetos.

Reconocimiento e interpretación

El reconocimiento consiste en clasificar los objetos dentro de una imagen basándose en las características extraídas. Esto implica necesariamente un proceso de aprendizaje, a partir de un conjunto de datos.

Existen dos tipos de aprendizaje:

- **Aprendizaje supervisado:** dispone de un conjunto de datos y sus etiquetas. El objetivo es conseguir un modelo (clasificador) que pueda etiquetar automáticamente nuevos datos que no se hayan empleado en el proceso de aprendizaje. Algunos clasificadores supervisados más comunes son:
    - Árboles de decisión: modelos que utilizan un conjunto de reglas de decisión basadas en las características de los datos para clasificar observaciones.
    - K-vecinos más próximos: clasificadores que utilizan la geometría de las características para separar las clases.
    - Clasificador Bayesiano Ingenuo: utilizan la probabilidad y la estadística para hacer predicciones basadas en la distribución de las características.

- Regresión logística: un método de clasificación que utiliza una función logística para modelar la probabilidad de una clase dada.
- Máquinas de vectores de soporte (SVM): modelos que buscan un hiperplano óptimo que maximice la separación entre clases.
- Redes neuronales: algoritmos inspirados en el funcionamiento del cerebro humano que aprenden patrones complejos a través de múltiples capas de nodos interconectados.

- **Aprendizaje no supervisado:** solo se dispone de los datos sin las etiquetas de clase. Trata de modelar los datos en si mismos, descubriendo grupos de datos similares en el conjunto de datos, y qué características los hacen similares dentro del grupo y diferentes del resto. Un ejemplo de aprendizaje no supervisado es el agrupamiento (*Clustering*).

Evaluación del modelo

La evaluación del modelo permite determinar la efectividad y la precisión de un modelo de aprendizaje de un sistema de visión artificial para realizar la tarea para la cual fue diseñado. Este proceso asegura que el modelo funcione correctamente tanto con los datos de entrenamiento como con los datos nuevos.

Existen métodos de validación que evalúan la capacidad de generalización de un modelo utilizando datos que no fueron utilizados durante el entrenamiento. Existe técnicas como la validación simple y validación y prueba, pero presentan algunos inconvenientes ya que las clases pueden ser no balanceadas, llegando incluso a que solo haya representantes de una clase determinada en cada grupo. Sin embargo, la validación cruzada trata de resolver estos problemas, ya que se dividen los datos en varios subconjuntos y el modelo se entrena varias veces para asegurar que el rendimiento no dependa de una sola partición de datos.

Otro tipo de evaluación del modelo es la matriz de confusión (

Figura 7), que permite visualizar la bondad de un modelo basado en un clasificador supervisado binario, mostrando la relación entre las predicciones del modelo y las etiquetas reales. Esta matriz ayuda a identificar los tipos de errores como los falsos positivos y falsos negativos.

		Condition (as determined by "Gold standard")	
	Total population	Condition positive	Condition negative
Test outcome	Test outcome positive	**True positive**	**False positive** (Type I error)
	Test outcome negative	**False negative** (Type II error)	**True negative**

Figura 7 - Matriz de confusión

2.3. Deep Learning aplicado a la Visión Artificial

2.3.1. Introducción al aprendizaje automático

El aprendizaje es una condición inherente al ser humano. Sin embargo, el desarrollo de la **Inteligencia Artificial** (IA) ha logrado que las máquinas sean capaces de aprender y mejorar su propio desempeño. Este amplísimo campo, que se inspira en sistemas inteligentes naturales, surgió del famoso artículo de Alan Turing *"Computing Machinery and Intelligence"* (1950), donde presentó el Test de Turing como criterio para determinar la inteligencia de una máquina.

Dentro de la Inteligencia Artificial se encuentra el **Machine Learning**, que es la práctica donde las máquinas toman decisiones sin ser programadas. El objetivo es hacer máquinas que aprendan por medio de datos, construyendo modelos que identifican patrones en los datos y se utilizan para hacer predicciones.

A su vez dentro del Machine Learning (Figura 8), se desarrolla el ***Deep Learning*** (aprendizaje profundo) que trabaja con el concepto de malla, generando redes neuronales artificiales que emulan las redes neuronales del cerebro humano. El objetivo es construir redes neuronales que automáticamente descubran patrones para la detección de características. Además, el *Deep Learning* destaca en los procesos de análisis, reconocimiento y clasificación de imágenes, por ello se emplea comúnmente para la Visión Artificial.

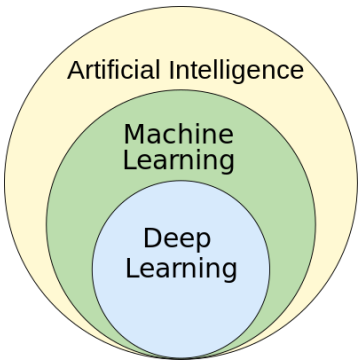


Figura 8 - Conjunto de la Inteligencia Artificial

2.3.2. Redes Neuronales Convolucionales

Las redes neuronales artificiales están formadas por un conjunto de nodos conocidos como neuronas artificiales que están conectadas y transmiten señales entre sí. Las redes reciben una serie de valores de entrada y cada una de estas entradas llega a una neurona. Cada neurona de la red posee un peso, un valor numérico, con el que modifica la entrada recibida. Los nuevos valores salen de las neuronas y continúan navegando por la red. Una vez alcanzado el final de la red se obtiene una salida que será la predicción calculada por la red (Figura 9). Cuantas más capas posea la red y más compleja sea, también serán más complejas las funciones que pueda realizar.

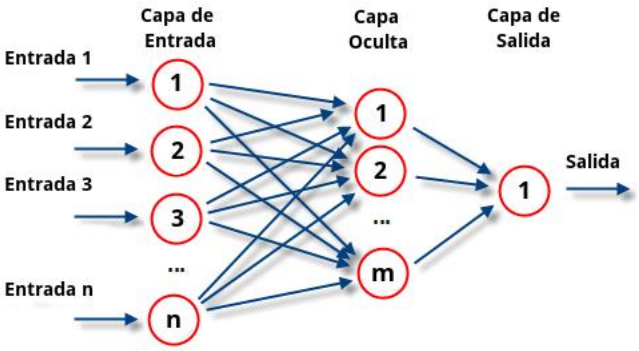


Figura 9 - Funcionamiento redes neuronales

A diferencia de las redes neuronales convencionales, las Redes Neuronales Convolucionales (CNN) procesan la información de forma más rápida y sencilla. Son un tipo de red neuronal que trata de emular el comportamiento del sistema visual humano, por ese motivo son especialmente efectivas para el desarrollo de proyectos de Visión Artificial que requieren un tipo de inteligencia.

Su funcionamiento consiste en localizar patrones en áreas reducidas de la imagen y no en su conjunto. Para ello, el algoritmo toma un pequeño cuadro de píxeles (entrada), al cual se le aplican filtros según vaya pasando por las distintas capas ocultas (neuronas). Estas capas están organizadas gradualmente según su analítica, las primeras capas detectan patrones básicos como líneas o bordes, cuanto más se profundiza en la red son capaces de reconocer formas complejas como personas o vehículos. Por tanto, a mayor número de capas mejor será la capacidad de visionado de la red y más certera será su salida.

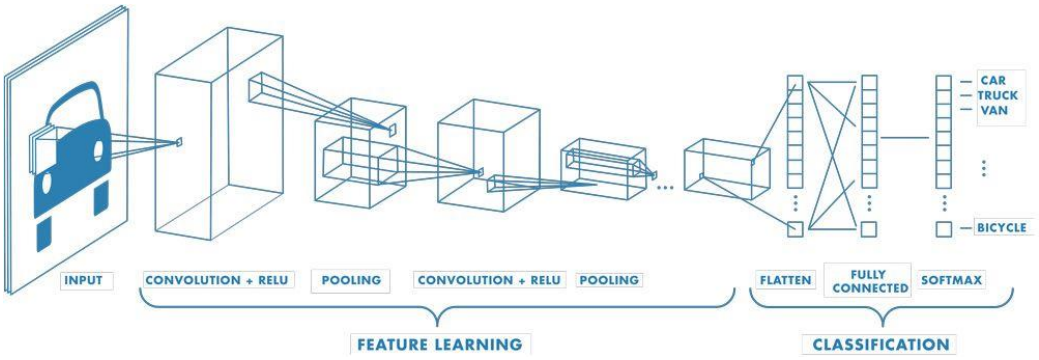


Figura 10 - Redes Neuronales Convolucionales

En definitiva, el proceso de aprendizaje de una red neuronal convolucional implica la interconexión de características en cada capa para mejorar su capacidad de reconocimiento, llegando a identificar formas complejas en etapas avanzadas. Al entrenarla con un amplio espectro de imágenes, la red genera su propia memoria y, una vez completada esta fase, el modelo de Deep Learning puede clasificar nuevas imágenes sin necesidad de clasificación previa.

2.3.3. Modelos de detección de objetos

La detección de imágenes puede parecer una tarea sencilla, ya que los humanos somos capaces de ver una imagen y reconocer inmediatamente cualquier objeto y clasificarlo. Sin embargo, la detección de objetos en imágenes mediante algoritmos de *Deep Learning* que implican una red neuronal convolucional (CNN) es bastante compleja y moderna.

En 2014 surge uno de los primeros y más conocidos modelos, el **R-CNN** (*Region-Based Convolutional Neural Network*). Este modelo se basa en: primero determinar las regiones de interés mediante un algoritmo de búsqueda selectiva, y luego, aplica una CNN a cada una de estas regiones para clasificar los objetos y ajustar las cajas delimitadoras. Aunque R-CNN fue revolucionario en su momento, tenía limitaciones en términos de velocidad debido a la necesidad de procesar cada región de interés por separado.

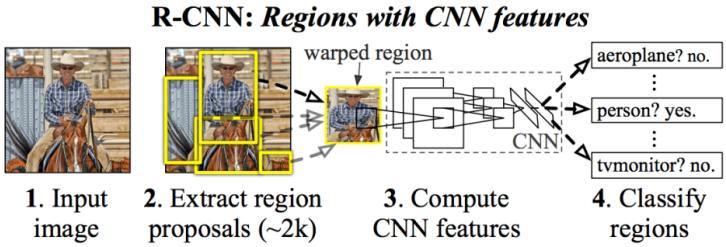


Figura 11 - Modelo de detección de objetos R-CNN

Por ello, para intentar mejorar el tiempo de detección surgen variante como Fast R-CNN y Faster R-CNN. Fast R-CNN mejora el algoritmo inicial haciendo reutilización de algunos recursos como el de las *features* extraídas por la CNN agilizando el entreno y detección de las imágenes. Faster R-CNN, por otro lado, introduce una red adicional llamada Red de Propuestas de Región (RPN) que genera regiones de interés directamente desde las características extraídas, reduciendo aún más el tiempo de procesamiento y mejorando la precisión.

En 2016 nace **YOLO** (*You Only Look Once*), una red que adopta una estrategia de detección en una sola etapa, haciendo una única pasada a la CNN y detectando todos los objetos para los que ha sido entrenada para clasificar. Este enfoque permite una detección en tiempo real debido a su alta velocidad, aunque inicialmente sacrificaba algo de precisión en comparación con los métodos de dos etapas. Las versiones más recientes de YOLO han mejorado significativamente tanto en precisión como en velocidad.

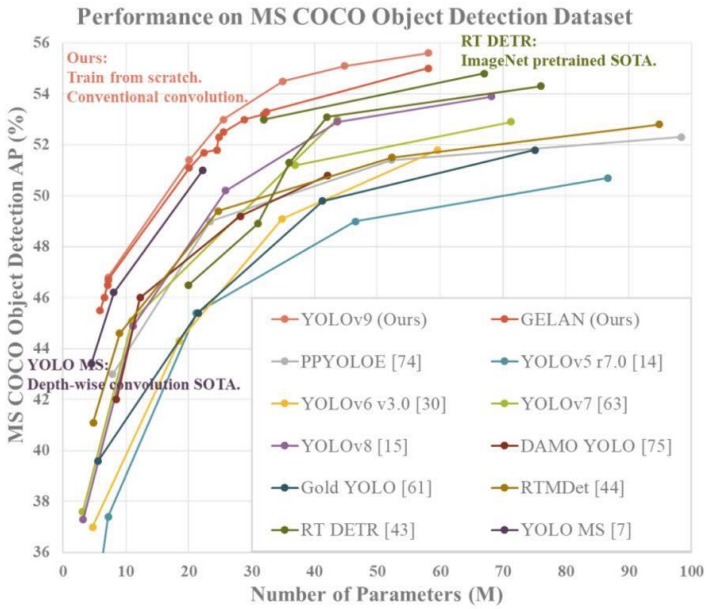


Figura 12 - Comparación de las versiones de YOLO. Fuente: visionplatform.ai

Otro modelo de detección de objetos en una sola etapa es **SSD** (*Single Shot MultiBox Detector*). Este modelo divide la imagen en una serie de mapas de características y, similar a YOLO, realiza predicciones directamente desde estos mapas. La ventaja de SSD es que puede detectar objetos grandes y pequeños gracia a su estructura piramidal en su CNN en la que las capas van disminuyendo gradualmente.

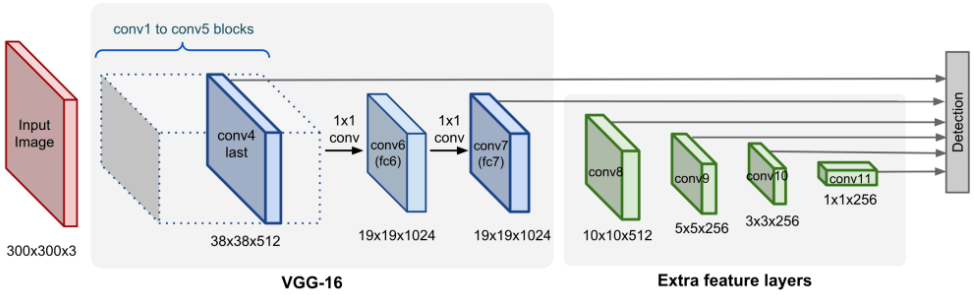


Figura 13 - Arquitectura modelo SSD

Al pasar lo años van surgiendo nuevos modelos de detección de objetos como el RetinaNet, Google Spinet, Facebook DETR, entre otros… En definitiva, el futuro de los modelos de detección de objetos promete ser muy interesante debido a la tendencia hacia modelos más pequeños y eficientes, la colaboración entre humanos y sistemas de IA, el desarrollo de hardware especializado, como unidades de procesamiento neuronal (NPU) y mejoras en GPUs, etc.

2.4. Historia de los robots con visión

En las décadas de 1960 y 1970, surgieron los primeros intentos de robots autónomos equipados con cámaras. Un ejemplo destacado es el robot **SHAKEY**, desarrollado por el *Stanford Research Institute*. Este robot es considerado uno de los primeros robots móviles autónomos y estaba equipado con una cámara de televisión y sensores táctiles. Podía desplazarse por el suelo gracias a dos computadores (uno a bordo y otro en remoto) que estaban conectados por radio.

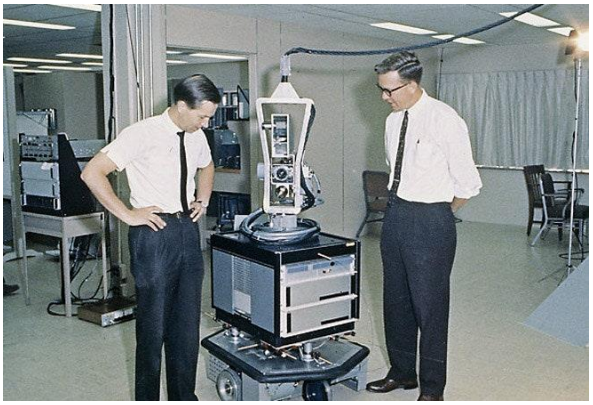


Figura 14 - Robot SHAKEY

Otro ejemplo de la década de los 70 fue el robot **MARS-ROVER**. Fue una plataforma que integraba un brazo mecánico, sensores de proximidad, un dispositivo telemétrico láser y cámaras estéreo. Fue desarrollado por la NASA para explorar terrenos hostiles o desconocidos.

En la década de 1980, se lograron mejoras significativas en la percepción y la navegación de los robots. El **Stanford Cart** (1980) fue uno de los primeros en utilizar una cámara para la navegación autónoma, capaz de seguir una línea blanca en el suelo y evitar obstáculos simples gracias a las coordenadas cartesianas en sus vértices y a la visión por computadora.

La década de 1990 trajo avances en autonomía y capacidad de procesamiento. El **Sojourner Rover** (1997), parte de la misión Mars Pathfinder de la NASA, fue el primer robot con cámara en explorar otro planeta. Equipado con cámaras para tomar imágenes del terreno marciano, envió valiosos datos y fotos a la Tierra.

Figura 15 - Robot Sojourner Rover

La década de 2000 vio la aparición de robots de servicio y exploración más avanzados. Por ejemplo, el **Roomba** (2002) de iRobot utilizó cámaras y sensores para mapear habitaciones y evitar obstáculos, revolucionando el mercado de los robots de limpieza doméstica.

En la década pasada (2010), la robótica avanzó hacia aplicaciones más especializadas y complejas. Por ejemplo, la NASA continuó la exploración científica de Marte, incluyendo la capacidad de tomar selfies y panorámicas del paisaje marciano mediante el robot **Curiosity Rover.** En el ámbito educativo y de investigación, la serie de robots **TurtleBot** permitió experimentar con algoritmos de navegación y visión por computadora.

Por otro lado, en paralelo a esta evolución de los robots con visión, se comenzaron a integrar sistemas de visión industrial en procesos de fabricación, desde la inspección de piezas y productos en líneas de ensamblaje hasta realizar tareas de ensamblaje, soldadura y manipulación de materiales con exactitud. Los robots con visión juegan un papel crucial en el control de calidad de productos.

En definitiva, los robots con cámaras han evolucionado desde simples dispositivos de navegación hasta sofisticados robots que gracias al Deep Learning realizan tareas complejas, como la exploración espacial, la asistencia en el hogar y la industria, control de calidad exhaustivo, etc.

2.5. Plataforma de desarrollo del proyecto

Para llevar a cabo este proyecto, es necesario un framework y hardware específicos. En este apartado se detalla el contexto tecnológico requerido para conseguir el correcto funcionamiento del proyecto y los resultados esperados.

2.5.1. ROS2

Este proyecto necesita el framework. Esto es una plataforma robótica que proporciona un conjunto de herramientas, bibliotecas y buenas prácticas. Es decir, es una estructura estandarizada que permite a los desarrolladores centrarse en la lógica específica de su aplicación, ya que ofrece funcionalidades predefinidas como manejo de base de datos, seguridad, gestión de usuarios, etc.

En este caso, se utiliza el framework de ROS 2 (*Robot Operating System*). Es un sistema operativo de robots de código abierto que proporciona servicios como la abstracción de hardware, control de dispositivos,

ROS2 es un sistema operativo de robots de código abierto que proporciona servicios similares a un sistema operativo, como abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comunes, paso de mensajes entre procesos y gestión de paquetes.

La arquitectura de ROS 2 está diseñada para ser flexible, modular y adecuada para aplicaciones distribuidas. A continuación, se detalla su arquitectura (Figura 16) y funcionamiento:

- ***Nodes***: son los componentes básicos de ROS 2. Cada uno es un proceso independiente que realiza una función específica dentro de un sistema robótico.

- ***Middleware* DDS** (*Data Distribution Service*): su comunicaicón se basa en DDS que facilita la transferencia de datos entre nodos. Este middleware proporciona mecanismos para garantizar la entrega fiable y de calidad de los mensajes.

- ***Topics***: los nodos se comunican publicando y suscribiéndose a los topics. Un topic es un canal de comunicación con un tipo de datos específico.

- ***Services***: los servicios permiten la comunicación síncrona entre nodos.

- ***Actions***: son similares a los servicios, pero están diseñadas para tareas que pueden durar un tiempo indeterminado. Permiten una comunicación asíncrona con retroalimentación continua sobre el progreso de la tarea.

- **Librerías y herramientas**: facilitan el desarrollo y la gestión de sistemas robóticos.

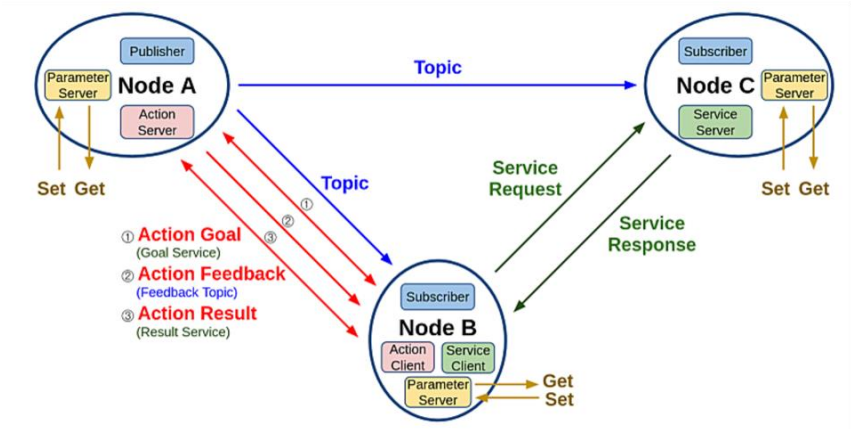


Figura 16 - Arquitectura ROS 2.

En definitiva, ROS 2 ofrece muchas ventajas como plataforma de desarrollo de aplicaciones robóticas:

- **Soporte multiplataforma**: está diseñado para ser compatible con múltiples sistemas operativos, incluyendo Ubuntu, Windows, macOS... Esto facilita el desarrollo de aplicaciones robóticas en diferentes entornos y hardware.

- **Comunicaciones en tiempo real**: ROS2 introduce características que permiten un mejor soporte para aplicaciones en tiempo real.

- **Estructura modular y escalable**: la arquitectura de ROS2 permite un mayor modularidad, facilitando la creación de sistemas escalables y distribuibles.

- **Mayor flexibilidad de lenguaje**: soporta una variedad de lenguajes de programación, incluyendo C++, Python, y otros.

2.5.2. TurtleBot 4

TurtleBot es una plataforma robótica con software abierto basada en ROS (*Robot Operating System)* que se utiliza para la investigación y la educación en robótica, programación y navegación autónoma.

TurtleBot fue creado en Willow Garage por Melonee Wise y Tully Foote en 2010. La visión detrás de TurtleBot era proporcionar una plataforma de hardware asequible y accesible para facilitar el aprendizaje y la investigación en robótica, especialmente en la navegación autónoma y el desarrollo de algoritmos de mapeo y localización.

Con los pasos de los años, el TurtleBot fue actualizándose y creando nuevas versiones (Figura 17). Estas nuevas versiones fueron mejorando el hardware, la estructura siendo más avanzada y modular, integrando mejores sensores y añadiendo nuevas funcionalidades para los usuarios.

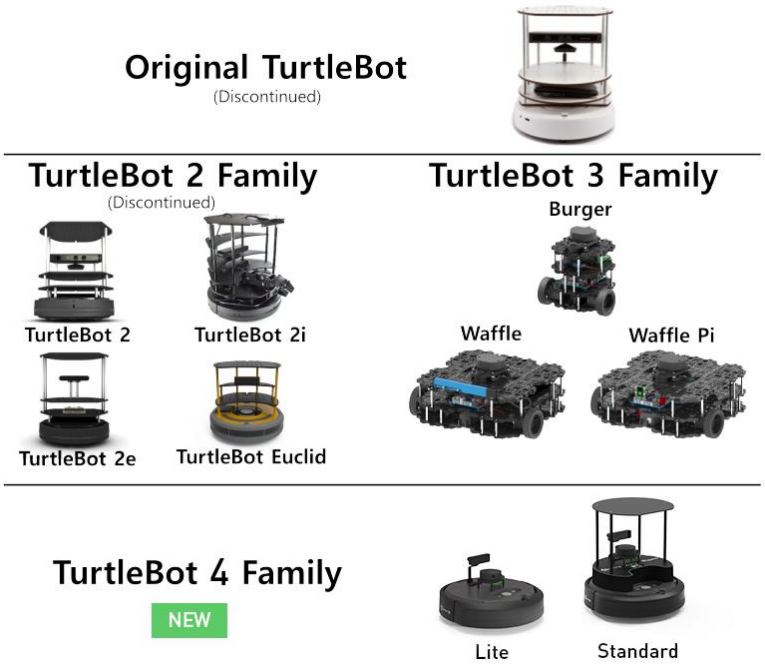


Figura 17 - Historia de los robots TurtleBot. *Fuente: TurtleBot.*

TurtleBot 4 es la última versión de esta serie, lanzada por *Open Robotics y Clearpath Robotics*. Es un robot móvil basado en ROS 2 destinado a la educación y la investigación. TurtleBot 4 es capaz de mapear el entorno del robot, navegar de forma autónoma, ejecutar modelos de IA en su cámara y más.

Existen dos modelos: TurtleBot 4 Standard y TurtleBot 4 Lite. Sin embargo, en este proyecto se utilizará el TurtleBot 4 (Figura 18), que a continuación se detallan sus componentes que se han integrado a la perfección para ofrecer una plataforma de desarrollo y aprendizaje lista para usar.

- Consta de dos unidades informáticas: Create® 3 y Raspberry Pi, y se conectan entre sí a través de un cable USB-C, que se utiliza tanto para alimentar a Raspberry Pi como para establecer una conexión Ethernet entre las unidades. Ambas unidades también tienen tarjetas Wi-Fi.

- Tiene una placa de interfaz de usuarios que ofrece indicadores LED de estado, botones de usuario y una pequeña pantalla. Además, tiene 4 puertos USB 3.0 (tipo C), así como puertos de alimentación adicionales y algunos pines de Raspberry Pi para el usuario.
- En la parte superior de la placa de interfaz de usuario se encuentra el RPLIDAR A1M8 de 360 grados y una cámara OAK-D-Pro.

Figura 18 - TurtleBot 4. Fuente: TurtleBot.

2.5.3. Cámara OAK-D-Pro

El TurtleBot 4 integra en su diseño la cámara OAK-D-Pro. Esta cámara es una versión avanzada de la serie de cámaras OpenCV AI Kit (OAK), desarrollada por Luxonis. Este tipo de cámaras están diseñadas para aplicaciones de inteligencia y visión artificial, ofreciendo capacidades avanzadas de percepción 3D y procesamiento en tiempo real.

La cámara OAK-D-Pro de Luxonis utiliza un sensor de color IMX214 4K junto con un par de sensores estéreo OV9282, que producen imágenes de color y profundidad de alta calidad y resolución. Además, también incorpora un proyector de puntos láser IR y un LED de iluminación IR, esto permite que la cámara cree imágenes de profundidad de mayor calidad y funcione mejor en entornos con poca luz.

Por otro lado, la unidad de procesamiento visual Myriad X le otorga a la cámara la capacidad de ejecutar aplicaciones de visión artificial, seguimiento de objetos y modelos de IA.

Figura 19 - Cámara OAK-D-Pro y sus funcionalidades

El software de la cámara proporciona interfaces de programación (APIs). La API de DepthAI permite a los usuarios desarrollar e implementar canales de visión que se ejecutan en el propio hardware de la cámara. En definitiva, esto permite a los usuarios empezar rápidamente con aplicaciones básicas de visión artificial.

Componentes de DepthAI (Figura 20)

- **Nodos**: representan elementos funcionales dentro del *pipeline*. Estos puede ser sensores, hardware acelerado o funciones computacionales.
- ***Pipeline***: es una serie de nodos conectados entre sí que se implementan en el dispositivo. Define el flujo de datos y el procesamiento que se realiza en los nodos. Una vez definido, el pipeline se despliega en el dispositivo y se ejecuta en bloques de hardware acelerados.
- **Mensajes**: son los medios de comunicación entre los nodos dentro del pipeline. Contienen tanto datos (imágenes) como metadatos (información adicional sobre los datos).
- **Dispositivo**: se refiere al hardware, en este caso a la cámara OAK-D-Pro. Se encarga de la conectividad y la comunicación con el entorno externo.
- ***Bootloader***: maneja la lógica de inicio del dispositivo. Se encarga de que el dispositivo sea accesible para la conexión y que esté listo para ejecutar el pipeline definido.

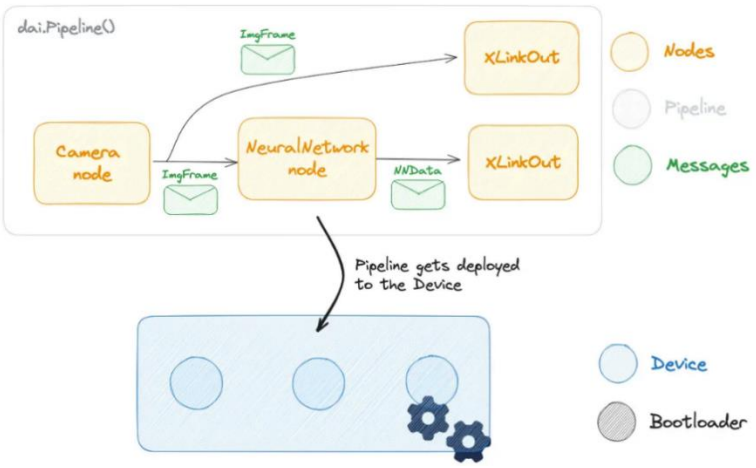


Figura 20 - Componentes de DepthAI

Respecto a los nodos, son elementos básicos y cada uno proporciona una funcionalidad específica en DepthAI, un conjunto de propiedades configurables y entradas/salidas (Figura 21). La salida de cada nodo tiene su propio pool, un bloque en la RAM donde se almacenan los mensajes y cada entrada tiene una cola para mensajes.

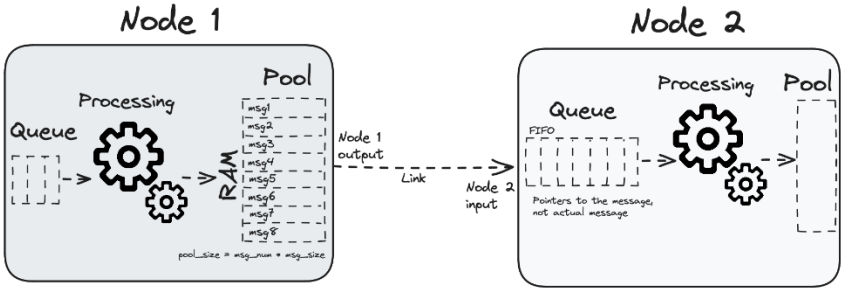


Figura 21 - Entradas y salidas de los nodos

3. Requerimientos del proyecto

3.1. Especificaciones del proyecto

Este proyecto propone desarrollar un sistema de seguimiento de personas utilizando el robot TurtleBot4 y ROS2. Para llevar a cabo este proyecto que combina la robótica, la Visión Artificial y técnicas de Deep Learning, es necesario considerar una serie de requisitos y factores que aseguran el éxito y la correcta implementación del proyecto. A continuación, se detallan los principales requisitos y factores a considerar.

3.1.1. Requerimientos de la implementación

- La configuración inicial y de red del TurtleBot4 se implementará adecuadamente para este proyecto, garantizando que no se presenten problemas a posteriori.

- El sistema ROS2-Humble se instalará y configurará correctamente en el TurtleBot4. Esto incluye la instalación de todos los paquetes necesarios para la comunicación con la cámara OAK-D Pro y el procesamiento de datos de visión.

- Tanto la plataforma del robot, iRobot® Create® 3, como ROS2-Humble serán actualizados al firmware más reciente. Esto hará al sistema compatible con futuras versiones, asegurando su longevidad y utilidad.

- Se empleará un adaptador USB Wi-Fi 5 GHz para proporcionar al robot una red local propia e independiente, asegurando un rendimiento óptimo del robot.

- El diseño 3D de la pinza para el adaptador USB Wi-Fi se realizará utilizando el programa *SolidWorks*, teniendo en cuenta las dimensiones y características físicas del adaptador.

- Se implementará un diseño de sistema modular que permitirá futuras funcionalidades al robot, logrando así un robot inteligente y multidisciplinar.

- Se verificarán las distintas funcionalidades de la cámara antes de diseñar el sistema de seguimiento de personas, asegurando su correcto funcionamiento y detectando posibles errores.

- Todos los nodos diseñados serán comprobados y validados individualmente para garantizar su correcto funcionamiento.

- Se programará un comando global, un *launch* o lanzamiento de ROS2, que permitirá a cualquier usuario utilizar esta aplicación sin ninguna dificultad.

- Las pruebas de validación del algoritmo de visión y del robot se realizarán en entornos controlados.

3.1.2. Limitaciones del proyecto

- El robot TurtleBot4 operará únicamente en superficies uniformes. Esto limita su capacidad para funcionar en entornos con superficies irregulares o terrenos accidentados, restringiendo su uso en interiores o áreas con pisos planos.

- La implementación puede tener limitaciones en la precisión del seguimiento debido al ruido de la cámara. Factores como la variable de iluminación, el contraste del entorno y el rápido movimiento de la persona puede afectar la precisión y confiabilidad del sistema de visión.

- La autonomía del robot estará limitada por la duración de su batería. El uso prolongado del robot y sobre todo de su cámara puede agotar la batería en un par de horas, requiriendo recargas frecuentes y limitando el tiempo operativo continuo.

- El consumo de potencia de la Raspberry Pi, especialmente con los cuatro puertos USB en uso y la cámara que ejecuta los algoritmos de visión, puede ser significativo y demandante. Esto provocará problema de rendimiento, interfiriendo en el funcionamiento de la cámara.

3.2. Normativa

Para realizar un proyecto completo y que esté sea garantizado, se deben tener en cuenta varias normativas y regulaciones aplicables, tanto en términos de seguridad, uso de tecnologías específicas, sistemas electrónicos, etc. A continuación, se detallan las normativas generales y específicas aplicables al proyecto.

Normativa	**Descripción**
Normativa General	
Ley 12/1986, del 1 de abril	Define las atribuciones de los Ingenieros Técnicos.
Real Decreto 1215/97, del 18 de julio	Normativa sobre la seguridad y salud en el trabajo.

Normativa Específica	
ISO 8373	Definición de términos relacionados con robots.
ISO 9787	Sistemas de coordenadas para robots y movimientos básicos.
UNE-EN ISO 13482	Especificaciones para el diseño de robots no industriales.
IEC 61508	Seguridad funcional de equipos eléctricos, electrónicos y electrónicos programables.
ISO/ASTM 52900:2021	Fabricación aditiva. Principios generales. Fundamentos y vocabulario.
Directiva (UE) 2019/904	Reducción del impacto de determinados productos de plástico en el medio ambiente
Dibujo Técnico y Planos	
UNE-EN ISO 5455	Definición y tipos de escalas.
UNE 1-032-82	Clases de líneas, grosores.
UNE 1-035-95	Cajetines y cuadros de rotulación
UNE 1-027-95	Plegado de planos

Tabla 1 - Normativa

4. Planteamiento de alternativas

4.1. Alternativas de configuración de red

4.1.1. Selección de red

El TurtleBot 4 realizará un seguimiento de personas e implementará algoritmos de visión que se tendrán que ejecutar en la cámara, pero se visualizarán a través de la conexión con el PC. Considerando esto, la elección y configuración de red será fundamental para garantizar la transmisión de datos de manera eficiente y estable. A continuación, se comparan las posibles configuraciones de red, evaluando sus ventajas y desventajas.

Una de las configuraciones consiste en conectar el TurtleBot 4 a un **punto de acceso Wi-Fi creado por un móvil**, el cuál dispondría de acceso a internet. Esta configuración facilita las actualizaciones, instalación de paquetes y acceso a servicios en la nube. Sin embargo, la red móvil suele tener una latencia alta, lo cual puede afectar la transmisión de datos de la cámara.

Una red local sin acceso a internet proporcionará una menor latencia, siendo una red más estable y de mayor alcance. Generalmente, esta configuración ofrece un mejor rendimiento en términos de velocidad y latencia, lo cual es crucial para la transmisión de imágenes en tiempo real. Para esta configuración se pensó en tres posibles opciones.

La primera opción es crear un **hotspot desde el PC**. Esta configuración permite un control completo sobre la red y también ofrece la flexibilidad de tener o no acceso a internet según sea necesario. Sin embargo, el alcance está limitado al rango de la señal Wi-FI del PC, lo que limita al robot a moverse en distancias largas.

La segunda opción es utilizar un **router Wi-Fi**. Este router crearía una red local estable y no tendría limitaciones de rango de señal ya que se puede colocar en la superficie superior del robot. No obstante, esta configuración requiere una alimentación elevada que el TurtleBot no es capaz de ofrecer (Tabla 3). Esta opción se basó en el enrutador RT-AC51U de ASUS.

Figura 22 - Enrutador RT-AC51U ASUS

Ventajas y desventajas enrutador RT-AC51U ASUS	
Ventajas	**Desventajas**
Banda dual de 2.4 GHz y de 5Ghz.	Alimentación CC de +12 V y 1.5 A. La cuál el TurtleBot 4 no puede administrar.
Se podría colocar encima del robot, no tendría limitaciones de rango de señal.	Estéticamente no sería lo correcto, ya que es un aparato bastante grande.
Alto rendimiento en términos de velocidad y latencia.	Para integrarlo en el robot haría falta un diseño elaborado para su sujeción.
Fácil configuración.	Si no está colocado en el TurtleBot 4, tendría las mismas limitaciones que un hotspot desde el PC.

Tabla 2 - Ventajas y desventajas enrutador RT-AC51U ASUS

Pinout	Source	Max current output (mA)	Fuse Hold at (mA)
1	VBAT	300	350
2	12V	300	350
3	GND		
4	5V	500	500
5	3V3	250	300
6	GND		

Tabla 3 - Distribución de pines del puerto de alimentación del TurtleBot 4

La tercera y última opción es crear un hotspot desde el TurtleBot 4 mediante un **adaptador USB Wi-Fi**. Esta configuración convierte al TurtleBot 4 en independiente, ya que es una solución completamente autónoma y portátil. Además de cumplir todas las premisas de estabilidad y latencia requeridas. El adaptador seleccionado es el AC650 de BrosTrend.

Ventajas y desventajas adaptador Wi-Fi USB AC650 BrosTrend	
Ventajas	**Desventajas**
Banda dual de 2.4 GHz y de 5Ghz.	Para integrarlo en el robot haría falta un diseño sencillo para su sujeción.
Compatible con Ubuntu 22.04 y Raspberry Pi 4B+.	Instalación de drivers en la Raspberry Pi.
Antena Wi-Fi de largo alcance de 5 dBi.	
Alimentación y conexión sencilla por puerto USB.	
Estética integrada en el robot.	

Tabla 4 - Ventajas y desventajas adaptador Wi-Fi USB AC650 BrosTrend.

4.1.2. Configuración de red TurtleBot 4

Para la red ROS 2, el TurtleBot 4 puede funcionar con dos configuraciones diferentes: *Simple Discovery* o *Discovery Server*.

Por un lado, **Simple Discovery** es el protocolo determinado de ROS 2 y utiliza multidifusión para permitir que un host envíe paquetes a varios destinatarios en la red simultáneamente.

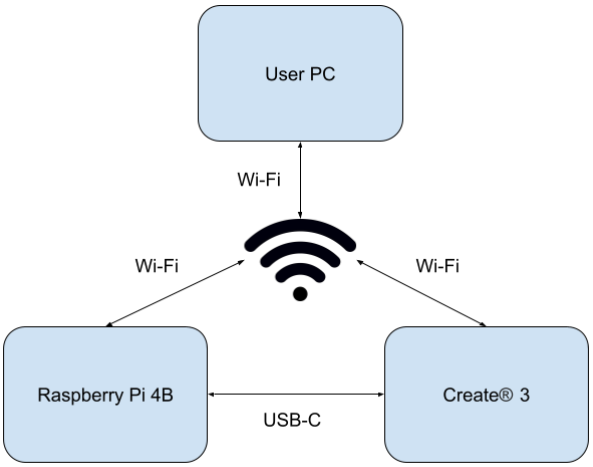


Figura 23 - Configuración Simple Discovery TurtleBot 4

Ventajas y desventajas Simple Discovery	
Ventajas	**Desventajas**
Configuración más sencilla y rápida.	Requiere capacidades de multidifusión lo que puede resultar problemático con algunas redes Wi-Fi.
No requiere instalación adicional.	No se escala de manera eficiente, ya que la cantidad de paquetes intercambiados aumenta considerablemente a medida que se agregan nuevos nodos.
Compatible con CycloneDDS y FastDDS.	

Tabla 5 - Ventajas y desventajas Simple Discovery.

Por otro lado, el servidor **Discovery Server** permite que un dispositivo de la red actúe como servidor de detección, mientras que el resto de los dispositivos se convierten en clientes de detección. Cada cliente comparte su información con el servidor Discovery Server y recibe la información de descubrimiento de este.

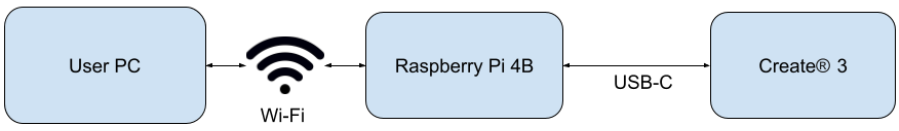


Figura 24 - Configuración Discovery Server TurtleBot 4

Ventajas y desventajas Discovery Server	
Ventajas	**Desventajas**
Evita problemas de multicast. El tráfico de red se reduce considerablemente.	Requiere configuración adicional en los dispositivos.
Permite que TurtleBot 4 opere solo con Wi-Fi de la Raspberry Pi.	Solo es compatible con FastDDS como middleware.
Mejora la conectividad y robustez en redes complejas.	

Tabla 6 - Ventajas y desventajas Discovery Server

4.2. Alternativas de repositorio cámara OAK-D

Al abordar el desarrollo de este proyecto utilizando la cámara OAK-D Pro, se presentan dos alternativas principales: el uso del repositorio de ROS de DepthAI o la biblioteca de Python de DepthAI. Ambas opciones tienen ventajas y desventajas que deben considerarse para determinar cuál es la más adecuada según las necesidades.

El TurtleBot 4 ya va integrado con ROS2 y ya tiene un launch predeterminado para iniciar la cámara "oakd.launch.py". Este launch es lanzado cada vez que el robot se inicia. Sin embargo, para obtener más funcionalidades y ejemplos de IA de la cámara se necesita un el **repositorio DepthAI ROS** disponible en Github.

Este repositorio ofrece una integración directa con ROS2, lo cual facilita la comunicación y gestión de nodos del TurtleBot4. Sin embargo, no está completamente diseñado para utilizarlo en la Raspberry Pi, ya que la visualización desde rviz2 en la Raspberry es muy costosa con relación al procesamiento.

Ventajas y desventajas repositorio ROS DepthAI	
Ventajas	**Desventajas**
Integración directa con el ecosistema ROS2.	Poca adaptabilidad a Raspberry Pi, el rendimiento no es del todo óptimo.
Facilita la comunicación y gestión de nodos.	Problemas de visualización en rviz2.
	Estructura del código muy compacta y entrelazada, dificulta modificaciones y personalizaciones necesarias.

Tabla 7 - Ventajas y desventajas repositorio ROS DepthAI

Por otro lado, la **biblioteca de Python DepthAI** ofrece una mayor simplicidad y flexibilidad. Esto facilita la realización de modificaciones y personalizaciones rápidas, lo cuál es muy beneficios durante la fase de desarrollo y pruebas. En términos de rendimiento, Python ofrece ventajas significativas em hardware limitado como la Raspberry Pi. Sin embargo, sería necesario integrar las funciones de Python en ROS2, creando así manualmente los nodos y los topics.

Ventajas y desventajas biblioteca Python DepthAI	
Ventajas	**Desventajas**
Mayor simplicidad y flexibilidad.	Integrar manualmente el código con ROS2, crear los topics y los nodos.
Ayuda a comprender su funcionamiento y poder aplicar posibles modificaciones.	
Ofrece un mejor rendimiento en Raspberry Pi, permite optimizaciones más eficientes.	
Facilidad de depuración.	
Mejor visualización de imágenes en OpenCV	

Tabla 8 - Ventajas y desventajas biblioteca Python DepthAI

4.3. Alternativas de modelos de detección de objetos

En este proyecto la detección de objetos, en concreto de personas, es el punto clave ya que determinará el correcto seguimiento de personas. Por tanto, es crucial elegir el modelo de detección adecuado que tenga en cuenta la precisión, el tiempo de ejecución, la eficiencia computacional y la arquitectura. A continuación, se presenta una descripción detallada de los tres modelos de detección de objetos principales.

En primer lugar, **Faster R-CNN** (*Region-based Convolutional Neural Network*) es un modelo de detección de objetos en dos etapas que primero propone regiones de interés y luego las clasifica. Introduce la Red de Propuestas de Región (RPN) para generar propuestas de regiones directamente dentro del modelo, eliminando la necesidad de procesos externos de selección de regiones.

Ventajas y desventajas Faster R-CNN	
Ventajas	**Desventajas**
Altamente preciso en la detección de objetos pequeños.	Necesita más recursos computacionales, puede ser un desafío para la Raspberry Pi.
Se adapta a distintos tipos de aplicaciones, como la detección de personas.	Su velocidad se ve comprometida debido al proceso de dos etapas.
Muy efectivo en escenarios con mucho ruido o múltiples objetos.	Más complejo de implementar y ajustar.

Tabla 9 - Ventajas y desventajas Faster R-CNN

YOLO (*You Only Look Once*) es un modelo de detección de objetos que predice probabilidades de clase y cuadros delimitadores en una sola etapa. Los modelos YOLO son conocidos por su velocidad excepcional, capaces de realizar inferencias en tiempo real, pero pueden sacrificar algo de precisión en comparación con otros modelos.

Ventajas y desventajas YOLOv8	
Ventajas	**Desventajas**
Es capaz de procesar imágenes a una alta tasa de FPS.	Sacrifica algo de precisión en comparación con otros modelos.
Utiliza la arquitectura DarkNet que permite identificar y detectar rápidamente objetos en las imágenes.	Requerimientos de hardware altos.
Combina una alta velocidad con precisión notable, ideal para aplicaciones en tiempo real.	

Tabla 10 - Ventajas y desventajas YOLOv8

MobileNet-SSD es una combinación de MobileNet como red base para la extracción de características y SSD para la detección de objetos. Este modelo de detección de objetos de disparo único (Single Shot Detector, SSD) utiliza MobileNet como base y puede lograr una detección optimizada para dispositivos móviles.

Ventajas y desventajas MobileNet-SSD	
Ventajas	**Desventajas**
Utiliza la arquitectura SSD, que está diseñada para lograr eficiencia sin comprometer la precisión.	Puede ser menos preciso en la detección de objetos pequeños en comparación con modelos más complejos.
Funciona bien en hardware de gama baja, ya su algoritmo es simplificado en comparación con otros modelos.	Puede tener dificultades en la detección de objetos en escenas muy complejas o con mucho ruido.
Ofrece un equilibrio entre velocidad y precisión.	

Tabla 11 - Ventajas y desventajas MobileNet-SSD

4.4. Alternativas de control

Para el seguimiento de personas con el TurtleBot 4 es crucial elegir el controlador adecuado para garantizar un rendimiento eficiente y preciso. Asumiendo que tenemos un robot con alta precisión en sus movimientos, consideraremos diferentes tipos de controladores. A continuación, se detalla una comparativa basada en sus características, ventajas y desventajas.

El **controlador proporcional (P)** ajusta la salida en función del error actual. El error es la diferencia entre la posición deseada (la posición de la persona) y la posición actual del robot. Este controlador es ideal por su simplicidad y efectividad en entornos donde el seguimiento preciso y rápido es fundamental. Sin embargo, puede dejar un pequeño error residual u oscilación.

Ventajas y desventajas control proporcional (P)	
Ventajas	**Desventajas**
Simplicidad. Es fácil de implementar y ajustar.	Puede dejar un error residual constante ya que no corrige errores acumulados.
Proporciona una respuesta inmediata al error actual.	Si la ganancia proporcional es demasiado alta, puede causar oscilaciones alrededor de la persona.
Requiere menos recursos computacionales.	

Tabla 12 - Ventajas y desventajas control proporcional (P)

El **controlador proporcional variable (PV)** es muy similar al anterior, pero ajusta la ganancia proporcional en función de la magnitud del error. Esto permite una respuesta más rápida cuando el error es grande y una respuesta más suave cuando el error es pequeño. En definitiva, proporciona un seguimiento más eficiente y adaptativo.

Ventajas y desventajas control proporcional variable (PV)	
Ventajas	**Desventajas**
Mejora la respuesta del sistema al ajustar la ganancia proporcional dinámicamente.	Requiere definir adecuadamente la función de variación de la ganancia proporcional.
Proporciona un movimiento más suave y eficiente al robot.	Quizás algo ineficiente, ya que el TurtleBot 4 tiene una velocidad máxima.
	Más complicado de implementar que un controlador P simple.

Tabla 13 - Ventajas y desventajas control proporcional variable (PV)

Existen otros tipos de controladores, pero se cree que para el seguimiento de personas con el TurtleBot 4 no son necesarios. Ya que, por ejemplo, el controlador PD puede amplificar el ruido, lo que podría causar movimientos inestables; el controlador PI puede introducir retardos y oscilaciones debido a su componente integral; y el controlador PID, aunque proporciona el control más completo, es excesivamente complicado de ajustar. En definitiva, no ofrecen beneficios significativos en este contexto, ya que el propio TurtleBot 4 ya tiene una alta precisión en sus movimientos

5. Solución adoptada

Una vez expuestas las distintas alternativas de diseño del proyecto, se ha realizado un exhaustivo análisis en los distintos ámbitos y se ha determinado la solución adoptada para realizar correctamente el desarrollo del sistema de seguimiento de personas utilizando TurtleBot 4 y ROS 2.

En primer lugar, se ha optado por la creación de un hotspot 5 GHz desde el TurtleBot4 mediante el **adaptador Wi-Fi USB** AC650. Esto se debe a la independencia y portabilidad que ofrece esta configuración, permitiendo que el TurtleBot 4 sea autónomo para la conectividad, sin depender de ningún rango de señal como sucede con el hotspot creado desde el PC. Además, al ser una red sin acceso a internet, facilita un entorno de pruebas más confiable y adaptado a las necesidades del proyecto, teniendo una conexión estable para la transmisión de datos en tiempo real.

Cabe destacar que, en determinados momentos como la configuración inicial, instalación de paquetes, dependencias, drivers, es necesario tener acceso a internet. Para ello, inicialmente el robot se conectará a internet mediante el punto de acceso del móvil, se instalará todo lo necesario para el proyecto y seguidamente se cambiará de configuración de red mediante el adaptador Wi-Fi USB. En el hipotético caso que se necesite alguna instalación a mitad de proyecto, se podría conectar a la Raspberry Pi un cable Ethernet con internet.

Siguiendo con la configuración de red, se ha elegido la configuración **Discovery Server** del TurtleBot 4. Esta configuración ofrece ventajas significativas, en comparación con el Simple Discovery, especialmente en redes Wi-Fi. Aunque requiere una configuración adicional y es exclusiva para FastDDS, mejora la conectividad y evita los problemas de multidifusión. Además, permite que el TurtleBot 4 funcione utilizando solo la conexión Wi-Fi de la Raspberry Pi, sin necesidad de que el Create® 3 esté conectado a la red Wi-Fi, ofreciendo una solución más robusta y confiable para entornos controlados.

Respecto a los repositorios que se van a utilizar, se han probado ambos y teniendo en cuenta sus ventajas y desventajas se ha decidido utilizar un enfoque híbrido para maximizar los beneficios. Dado que la **biblioteca Python DepthAI** ha demostrado ser más funcional, el desarrollo del proyecto va a comenzar probando todas las funcionalidades clave del proyecto y de la cámara en Python. Este desarrollo inicial permitirá optimizar el rendimiento y depurar posibles problemas.

Una vez que las funcionalidades necesarias estén bien desarrolladas, se procederá a la integración con ROS2, para esto se utilizará rospy para integrar el código Python con nodos ROS2. Se implementarán nodos ROS2 que llamarán a scripts Python, esto permitirá mantener la modularidad y escalabilidad futura del proyecto.

Respecto a la selección del modelo de detección de objetos, se han tenido muchas dudas respecto a la elección entre YOLOv8 o MobileNet-SSD, ya que ambos modelos presentan características muy similares. Ambos modelos se han probado en este proyecto y se comprobó que a nivel de precisión y velocidad ambos son iguales.

Sin embargo, se acabó eligiendo el **MobileNet-SSD** por tres razones:

- Tiene menos clases de objetos a detectar. Esto facilita la única detección que tiene que hacer la cámara, la de las personas.

- La visualización de las imágenes junto con las detecciones era mejor procesa por la Raspberry Pi, ya que era una visualización rápida y sin ser pixelada.
- La arquitectura ligera y eficiente que proporciona un equilibrio ideal entre precisión y rendimiento.

Terminando con la solución adoptada, se ha optado tanto por el controlador proporcional (P) como por el controlador proporcional variable. Ambos son similares, y la simplicidad y efectividad de estos controladores son suficientes para lograr un seguimiento suave y eficiente. En el desarrollo del proyecto se diseñarán los dos y se compararán para elegir el más preciso.

Finalmente, el presente proyecto contribuye con los siguientes Objetivos de Desarrollo Sostenible, tal y como indica la ONU en la agenda 2030 (anexo X).

Figura 25 - Objetivos de Desarrollo Sostenible (ODS)

5.1. Diagrama

Para poder justificar de adecuadamente la solución adoptada de este proyecto, se realiza una representación gráfica que muestra la estructura organizativa. Este diagrama es muy útil, ya que proporciona una visión clara de las partes y componentes que intervienen en el proyecto y como se relacionan entre sí.

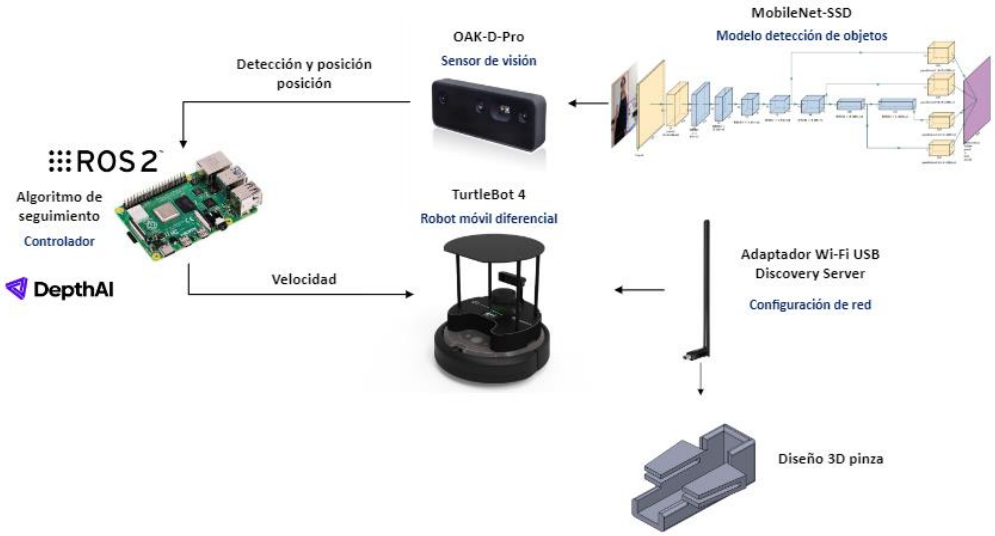


Figura 26 - Diagrama solución adoptada

5.2. Puesta en marcha TurtleBot4

En este apartado se detalla la puesta en marcha inicial del TurtleBot4.

5.2.1. Acceder al TurtleBot 4

En primer lugar, para interactuar con el robot Turtlebot4, el PC debe tener el sistema operativo Ubuntu 22.04 versión Humble. Además, se debe tener instalado ROS2 y el turtlebot4_desktop.

```
sudo apt update && sudo apt install ros-humble-turtlebot4-desktop
```

A continuación, se enciende el robot colocándolo en su base. Cuando el LED circular se ilumine en verde durante unos segundos y suene un sonido, querrá decir que el Create®3 se ha encendido, solo queda esperar un par de minutos para que el robot se acabe de iniciar. Para asegurarse de un correcto inicio se encenderán los cinco LEDS verdes del robot.

La primera vez que se hace la configuración, la Raspberry Pi crea un punto de acceso que permite conectarse al Turtlebot4 a través de Wi-Fi. Desde la configuración Wi-Fi del PC, se conecta a la red con las siguientes características:

- **Nombre**: Tutrlebot4
- **Tipo de conexión**: 5 GHz (el PC debe admitir este tipo de Wi-Fi)
- **Contraseña**: Turtlebot4

Cuando el PC ya se haya conectado al Wi-Fi, se accede a la Raspberry Pi mediante SSH. Para ello, se abre una terminal en el PC y se ingresa la siguiente línea de comando:

```
ssh ubuntu@10.42.0.1
```

La contraseña al iniciar sesión es turtlebot4.

Si no es la primera vez que se realiza la configuración inicial del robot, el robot ya tendrá modificada su conexión Wi-Fi y pueden suceder dos casos: que el robot siga conectado a esa red Wi-Fi o que esa conexión se haya perdido.

En el primer caso, es tan sencillo como fijarse en la IP que aparece en la pantalla del robot y acceder a ella mediante SSH. En el segundo caso, no se puede acceder a la Raspberry Pi a través de Wi-Fi y para recuperar esto, se realiza una conexión por cable Ethernet a la Raspberry Pi. Para ello, en configuración por cable del PC, se debe marcar la opción de IPv4 Manual y rellenar los siguientes campos:

- **Dirección**: 192.168.185.5
- ***Netmask***: 255.255.255.0
- ***Gateway***: 192.168.185.1

Ahora ya se puede acceder mediante SSH al robot:

```
ssh ubuntu@192.168.185.3
```

5.2.2. Conexión a la red y actualización del TurtleBot 4

Una vez iniciada la sesión SSH, se puede conectar la Raspberry Pi a la red que se necesite, la primera vez se recomienda utilizar una red que tenga conexión a internet, ya que se deben de aplicar las nuevas actualizaciones. Además, también se recomienda que sea una red Wi-Fi de 5 GHz para obtener un rendimiento óptimo.

En primer lugar, se ejecuta la herramienta de configuración Turtlebot 4

```
turtlebot4-setup
```

Y en "Wi-Fi Setup" se configura la conexión. En este caso, la conexión será en modo cliente ya que la Raspberry Pi se conecta al punto de acceso del móvil, que es una red ya existente. La IP se dejará vacía y el DHCP activado, los campos serán completado según cada red. Por último, se guarda y se aplica la configuración.

La Raspberry Pi se reiniciará y se conectará a la nueva red. Como el DHCP se activó, este le asignará una nueva dirección IP. Desde el TurtleBot 4, esta nueva IP se muestra en la pantalla del robot y ya se podrá acceder a él mediante SSH.

```
ssh ubuntu@172.20.10.3
```

Una vez que se haya accedido al robot, se deben aplicar las nuevas actualizaciones del Create®3 y de la Raspberry Pi. En primer lugar, se accede al servidor web del Create® 3 desde un navegador web e ingresando la IP de la Raspberry con el puerto 8080 (172.20.10.3:8080).

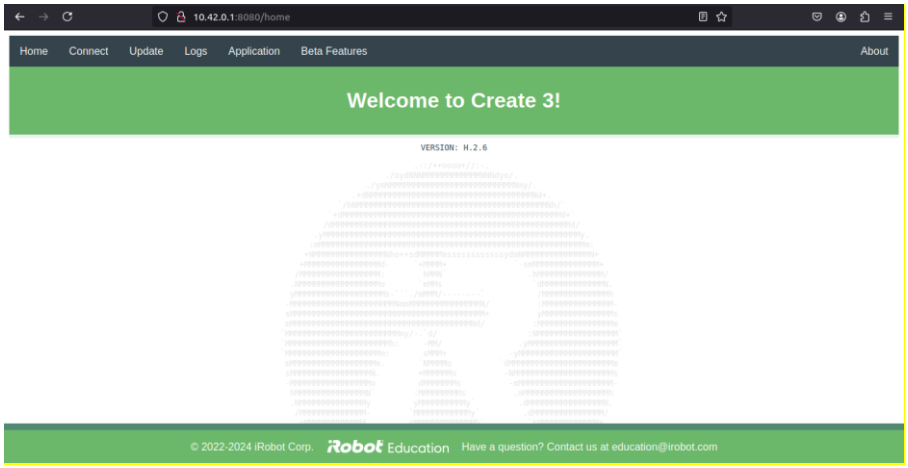


Figura 27 - Servidor web Create® 3

En el servidor web se puede consultar la versión de firmware que tiene el Create® 3 del TurtleBot 4. Una vez conocida la versión, se debe consultar las versiones de software de Create® 3 e instalar la versión más reciente. Para ello, se descarga el nuevo firmware desde la web anterior y en el servidor web, en la pestaña de actualizar, se carga el archivo y se actualiza el robot.

Respecto a las actualizaciones de todos los paquetes de la Raspberry Pi, se accede mediante SSH y se llama al siguiente comando:

```
sudo apt update && sudo apt upgrade
```

5.3. Creación punto de acceso Wi-Fi con adaptador USB

En este proyecto se ha decidido que el robot TurtleBot 4 se conecte a una red local sin conexión a Internet, ya que asegura un rendimiento de red más estable y predecible para las comunicaciones internas y el control del robot. Además, la comunicación dentro de una red local tiende a tener menor latencia, algo determinante para obtener respuestas rápidas y precisas del TurtleBot 4.

Por otro lado, el TurtleBot 4 trabajará de manera autónoma y continua sin interrupciones externas de Internet. Esto facilita el desarrollo y pruebas locales de software y algoritmos, lo cual es beneficioso y conveniente para investigaciones en entornos controlados.

5.3.1. Instalación de controladores

En primer lugar, este modelo de adaptador Wi-Fi necesita unos controladores determinados, los cuales se deben de instalar. Para ello, se seguirán los siguientes pasos:

1. Se debe tener en cuenta que se requiere de una conexión a Internet en el momento de la instalación. En este caso, la Raspberry Pi ya está conectada a una red Wi-Fi que dispone de Internet.
2. Se recomienda actualizar y reiniciar la Raspberry Pi antes de ejecutar el instalador del controlador.
3. Una vez iniciada la Raspberry Pi, se inserta el adaptador Wi-Fi en la ranura USB 2.0.
4. Mediante conexión SSH con la Raspberry Pi, se copia en la terminal el siguiente comando:

```
sh -c 'wget linux.brostrend.com/install -O /tmp/install && sh /tmp/install'
```

5. Este comando descarga, instala y carga automáticamente el controlador corrector para el adaptador.
6. En algunos casos, puede ser necesario reiniciar la Raspberry Pi.

Para saber si el controlador ha sido correctamente instalado, implementando el comando ip a debe aparecer algo similar a la Figura 28. El adaptador USB Wi-Fi es wlx347de44fbb54, este identificador será necesario para crear el punto de acceso.

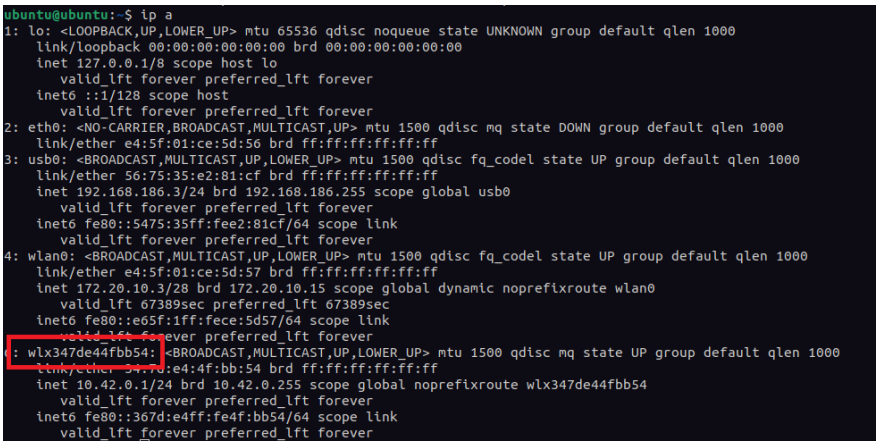


Figura 28 - Comando ip a

Antes de continuar con la configuración del punto de acceso Wi-Fi y desconectar al robot de internet se recomienda instalar la biblioteca de la cámara OAK-D Pro (5.5.1. Configuración inicial).

5.3.2. Configuración punto de acceso Wi-Fi

La creación del punto de acceso Wi-Fi (hotspot) se realiza utilizando el NetworkManager, que es una herramienta de gestión de redes para sistemas operativos Linux, que facilita la configuración y administración de conexiones de red. Además, se debe de configurar la ejecución automática al inicio del sistema. Esto consigue que cada vez que se encienda el robot cree automáticamente el punto de acceso Wi-Fi de 5 GHz.

Los pasos a seguir son:

1. Crear el script *accespoint.sh* que creará el punto de acceso.

   sudo nano accesspoint.sh

   El script contendrá el comando nmcli, que es una herramienta de línea de comandos para gestionar conexiones de red.

```
#!/bin/bash

# Eliminar cualquier conexión existente con el mismo nombre
nmcli connection delete Hotspot

# Crear una nueva conexión Wi-Fi configurada como punto de acceso
nmcli connection add type wifi ifname wlx347de44fbb54 con-name Hotspot
ssid Turtlebot4AP mode ap

# Modificar la conexión para usar la banda de 5 GHz y otros parámetros
nmcli connection modify Hotspot 802-11-wireless.band a
nmcli connection modify Hotspot 802-11-wireless.channel 36
nmcli connection modify Hotspot 802-11-wireless-security.key-mgmt wpa-psk
nmcli connection modify Hotspot 802-11-wireless-security.psk UPV12345
nmcli connection modify Hotspot ipv4.method shared
```

```
# Activar la conexión
nmcli connection up Hotspot
```

2. Hacer ejecutable el script.
   ```
   sudo chmod +x /home/ubuntu/accesspoint.sh
   ```

3. Crear el servicio de inicio automático.
   ```
   sudo nano /etc/systemd/system/rc-local.service
   ```

   Este archivo configura systemd para ejecutar /etc/rc.local al inicio del sistema y el contenido de este es:

   ```
   [Unit]
   Description=/etc/rc.local Compatibility
   ConditionPathExists=/etc/rc.local

   [Service]
   Type=forking
   ExecStart=/etc/rc.local start
   TimeoutSec=0
   StandardOutput=tty
   RemainAfterExit=yes
   SysVStartPriority=99

   [Install]
   WantedBy=multi-user.target
   ```

4. Configurar el script de inicio
   ```
   sudo nano /etc/rc.local
   ```

   El contenido de este script ejecuta el script accespoint.sh al inicio del sistema:

   ```
   #!/bin/bash

   # Ejecutar el script para configurar el hotspot
   bash /home/ubuntu/accesspoint.sh

   exit 0
   ```

5. Hacer ejecutable rc.local y activar el servicio.
   ```
   sudo chmod +x /etc/rc.local
   sudo systemctl enable rc-local
   sudo systemctl start rc-local
   ```

6. Verificar el estado de rc-local para asegurar que está activo.
   ```
   sudo systemctl status rc-local
   ```

7. Se recomiendo reiniciar el sistema para aplicar todos los cambios.

La IP que suele llevar el hotspot es 10.42.0.1, sin embargo, se puede comprar utilizando el comando ip a (Figura 29).

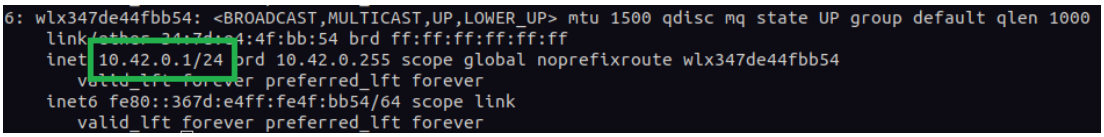


Figura 29 - Comando *ip a* con IP hotspot

Una vez creado el hotspot, la Raspberry Pi se tiene que conectar a este nuevo acceso Wi-Fi. Para ello, siguiendo las indicaciones del apartado 5.2.2. se cambiará la configuración Wi-Fi quedando como en la Figura 30.

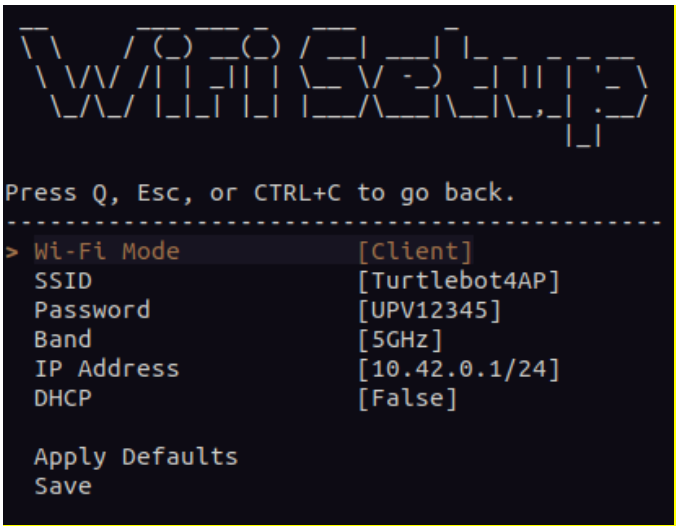


Figura 30 - Wi-Fi Setup

5.3.3. Diseño 3D pinza adaptador Wi-Fi USB

El adaptador Wi-Fi USB al ir conectado directamente a la ranura USB de la Raspberry, puede interferir en el mapeo del LiDAR, además de romper la estética del robot. Por ello, se diseña en SolidWorks un prototipo de pinza (Figura 31) para enganchar el adaptador en la parte superior del robot (en el techo del robot), este diseño evitará los posibles problemas que se han comentado anteriormente.

Por otro lado, se necesita un cable alargador USB 2.0 para que el adaptador Wi-Fi quede enganchado en la pinza y a la vez conectado a la ranura USB de la Raspberry Pi.

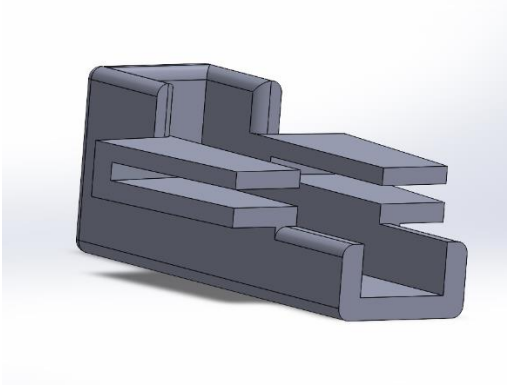


Figura 31 - Diseño 3D adaptador Wi-Fi USB

Figura 32 - Pinza para el adaptador USB

5.4. Configuración Discovery Server

Para configurar el Discovery Server, el Create® 3 debe actualizarse al firmware más reciente y, una vez actualizado, se realiza un restablecimiento de fábrica para desconectar el Create® 3 de cualquier red Wi-Fi. Este restablecimiento se realiza desde su servidor web, en la parte final de *"about"*.

Respecto a la Raspberry Pi se debe de configurar como Discovery Server. Para ello, desde la herramienta turtlebot4-setup, se ingresa al menú Discovery Server a través de ROS Setup y se guarda la configuración de la Figura 33.

Figura 33 - Configuración Discovery Server

Una vez guardada, se sale de la herramienta de configuración y en la terminal se ejecuta el comando turtlebot4-source. A continuación, se reinicia ROS 2 daemon (turtlebot4-daemon-restart) para aplicar la nueva configuración. Cuando suene el timbre del Create® 3, se comprueba la configuración llamando a ros2 topic list para ver los topics del robot.

Para terminar la configuración del Discovery Server, desde el PC se descargar y se ejecuta el script de configuración llamando a:

```
wget -qO - https://raw.githubusercontent.com/turtlebot/turtlebot4_setup/humble/turtlebot4_discovery/configure_discovery.sh | bash <(cat) </dev/tty
```

Donde se indica la IP de la Raspberry Pi (la que aparece por pantalla) y el resto de cosas aparecen por defecto entre corchetes, únicamente hay que ir pasando con el enter y se configuran por defecto. Cuando el script se haya ejecutado, se realiza un source para aplicar la nueva configuración.

```
source ~/.bashrc
```

Para comprobar que el script se ha ejecutado correctamente, se ingresa el comando ip route. Donde debe de aparecer una entrada como la resaltada de la Figura 34, la IP que aparezca será la que se haya configurado en el script (la IP de la Raspberry Pi).

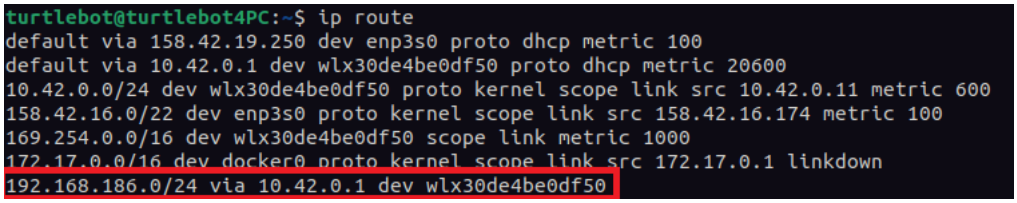


Figura 34 - Comando *ip route*

En este caso con el adaptado USB Wi-Fi, el script no funcionaba correctamente, ya que la ruta IP no aparecía. Para resolver este problema, se debe de ingresar la ruta manualmente con `sudo ip route add 192.168.186.0/24 via 10.42.0.1` reemplazando la IP por la de la Raspberry Pi. Hay que tener en cuenta que, al hacerlo manual, cada vez que se apague la Raspberry Pi esta configuración desaparecerá.

Después de esto reinicia ROS daemon:

```
ros2 daemon stop; ros2 daemon start
```

Para comprobar el correcto funcionamiento del Discovery Server, al ejecutar ros2 topic list, deben aparecer todos los temas de la Raspberry y Create® 3. Si no aparecen estos topics, también se recomienda que el servidor web del Create® 3 tenga la configuración de la Figura 35.

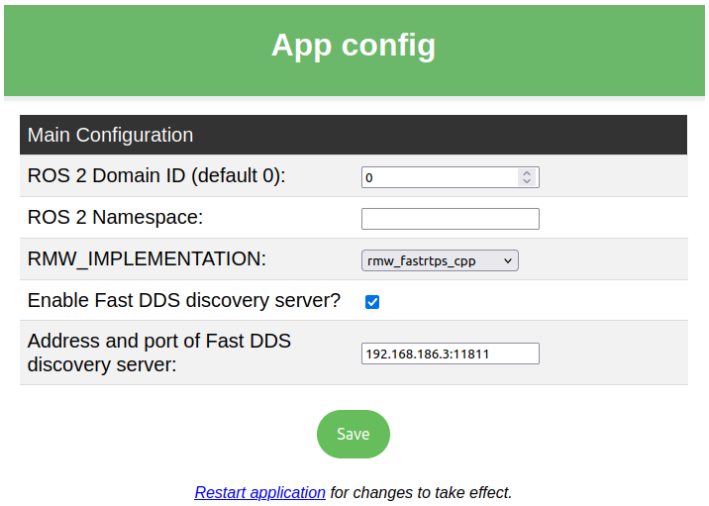


Figura 35 - Configuración servidor web Create® 3

Tanto en la Raspberry Pi como en el PC, cuando se produce un cambio en la red o se agrega un nuevo nodo, es posible que deba reiniciar el demonio ros2. También es posible que deba llamar ros2 topic list dos veces para obtener una lista completa de temas.

5.5. Cámara OAK-D Pro

Teniendo en cuenta el hardware, software y funcionamiento de la cámara OAK-D-Pro (2.5.3. Cámara OAK-D-Pro), se procede a configurarla en el TurtleBot 4 y probar algunos de sus algoritmos y funcionamientos.

5.5.1. Configuración inicial

Como se ha detallado al inicio de la solución adoptada (5. Solución adoptada), se ha decidido utilizar la biblioteca Python DepthAI como base para el proyecto. Como al iniciar el TurtleBot 4 se lanza un nodo que inicia la cámara, se tendrá que desactivar este lanzamiento desde turtlebot4_bringup. Esto es necesario ya que la cámara no puede estar siendo utilizada por más de un lanzamiento o nodo, no es capaz de ejecutar más de un algoritmo a la vez.

Para desactivar esta launch, se seguirán los siguientes pasos:

1. Editar el archivo standard.launch.py. Se recomienda hacer una copia del original.
   sudo nano
   /opt/ros/humble/share/turtlebot4_bringup/launch/standard.launch.py

2. Comentar las líneas que hagan referencia al oakd, quedando como las Figura 36 y Figura 37

```
# Launch files
turtlebot4_robot_launch_file = PathJoinSubstitution(
    [pkg_turtlebot4_bringup, 'launch', 'robot.launch.py'])
joy_teleop_launch_file = PathJoinSubstitution(
    [pkg_turtlebot4_bringup, 'launch', 'joy_teleop.launch.py'])
diagnostics_launch_file = PathJoinSubstitution(
    [pkg_turtlebot4_diagnostics, 'launch', 'diagnostics.launch.py'])
rplidar_launch_file = PathJoinSubstitution(
    [pkg_turtlebot4_bringup, 'launch', 'rplidar.launch.py'])
#oakd_launch_file = PathJoinSubstitution(
#    [pkg_turtlebot4_bringup, 'launch', 'oakd.launch.py'])
description_launch_file = PathJoinSubstitution(
    [pkg_turtlebot4_description, 'launch', 'robot_description.launch.py']
)
```

Figura 36 - Desactivar launch inicial de la cámara OAK-D

```
actions = [
        PushRosNamespace(namespace),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([turtlebot4_robot_launch_file]),
            launch_arguments=[('model', 'standard'),
                              ('param_file', namespaced_param_file)]),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([joy_teleop_launch_file]),
            launch_arguments=[('namespace', namespace)]),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([rplidar_launch_file])),

        #IncludeLaunchDescription(
        #    PythonLaunchDescriptionSource([oakd_launch_file]),
        #    launch_arguments=[('camera', 'oakd_pro')]),

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([description_launch_file]),
            launch_arguments=[('model', 'standard')]),
    ]
```

Figura 37 - Desactivar launch inicial de la cámara OAK-D

3. Reiniciar el servicio del turtlebot4.
   systemctl restart turtlebot4.service
4. Comprobar el estado del servicio del turtlebot4, para verificar la correcta desactivación del launch oakd. Tiene que aparecer activo.
   systemctl status turtlebot4.service

Ya desactivado el launch inicial de la cámara, solo queda instalar la biblioteca Python DepthAI. Se seguirán los pasos del manual de instalación de Luxonis:

1. Instalación de dependencia. Ejecutar el siguiente comando:
   sudo wget -qO- https://docs.luxonis.com/install_dependencies.sh | bash

   - Instala Python3, Pip3, Cmake, Git, Udev (si aún no está instalado).
   - Instala otras dependencias como libusb, libudev y otras.
   - Si OpenCV falla, se agrega:
     echo "export OPENBLAS_CORETYPE=ARMV8" >> ~/.bashrc
     source ~/.bashrc
2. Instalación de DepthAI, ejecutar:
   python3 -m pip install depthai
3. Instalación de prueba:
   - Clonar el repositorio depthai.python e instalar los requisitos.

```
git clone https://github.com/luxonis/depthai-python.git
cd depthai-python
cd examples
python3 install_requirements.py
```

- Ejecutar un script de prueba para asegurar de que todo esté funcionando:
  python3 ColorCamera/rgb_preview.py

5.5.2. Ejemplos de algoritmos de la cámara

Este apartado se ha detallado para conocer mejor las funcionalidades de la cámara y sus nodos. Además, sirve para familiarizarse con su programación de nodos, ya que esto permitirá hacer un correcto uso de la cámara en nuestro proyecto de seguimiento de personas.

El nodo **EdgeDetector** permite detectar los bordes de la imagen utilizando el filtro Sobel para crear una nueva imagen que enfatiza los bordes (Figura 38). Tiene como entradas la propia imagen y la configuración de detección de los bordes, y como salida la nueva imagen con los bordes.

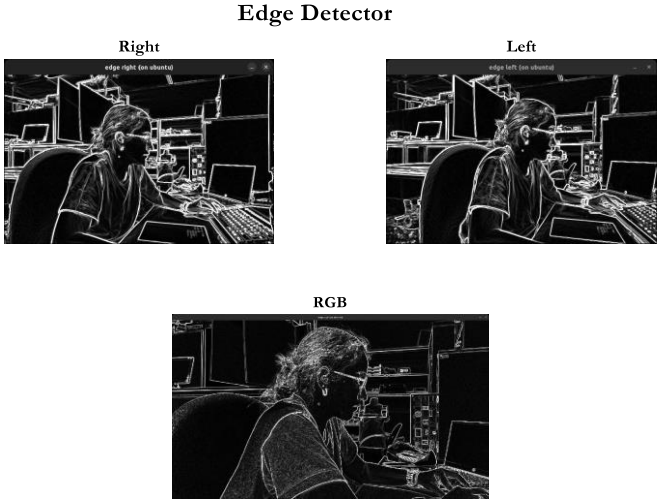


Figura 38 - Edge detector

El nodo **FeatureTracker** detecta punto clave (características) en un fotograma y los rastrea en el siguiente fotograma (Figura 39). Con este rastreo permite saber si los puntos clave se han desplazado a la izquierda-derecha o arriba-abajo. Las características válidas se obtienen a partir de la puntuación de Harris o Shi-Tomasi.

Figura 39 - Feature Detector

Los nodos anteriores son bastante sencillos, sin embargo, nodos como el **StereoDepth** son más complejos. Este nodo calcula la disparidad (Adquisición de imágenes) o profundidad del par de cámaras estéreo. No es de importancia en este proyecto la programación interna de este nodo, pero en la siguiente figura se muestra el diagrama de bloques internos del nodo StereoDepth (Figura 40). Un ejemplo de resultado de este nodo es un mapa de colores según la profundidad (Figura 41).

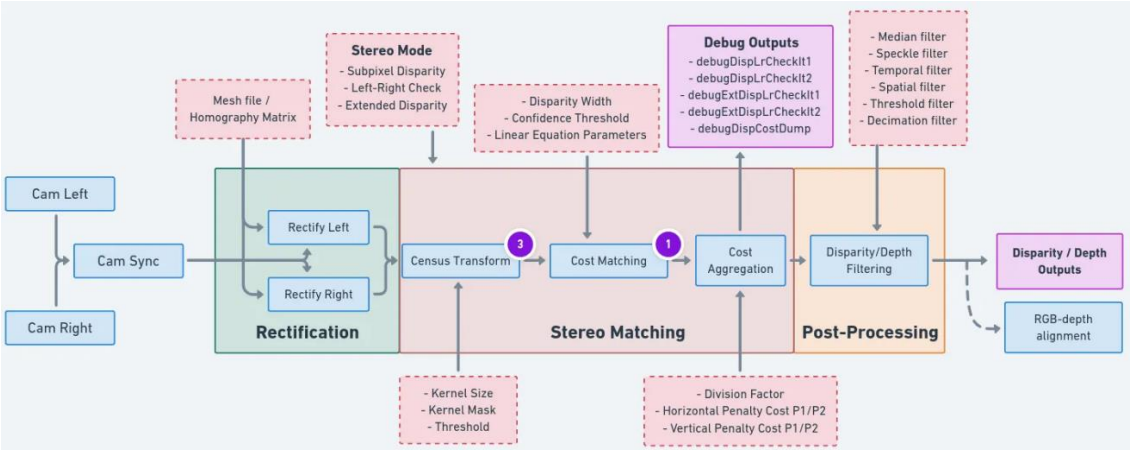


Figura 40 - Diagrama de bloques internos del nodo StereoDepth

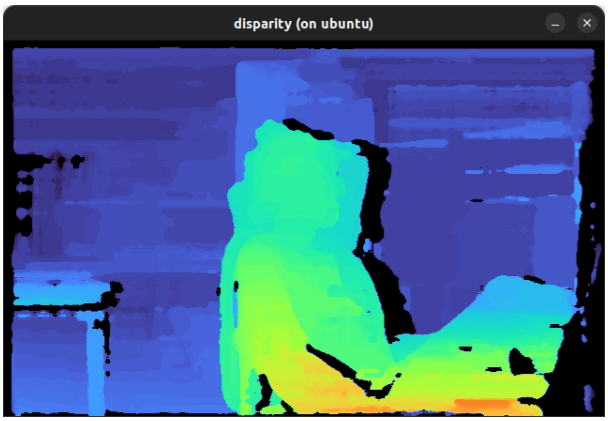


Figura 41 - Mapa de color según la profundidad

5.6. Ejecución del nodo de detección espacial MobileNet

La cámara proporciona un nodo llamado MobileNetSpatialDetectionNetwork, el cuál consigue una detección espacial, teniendo en cuenta la profundidad, utilizando la NN MobileNet.

Este nodo combina varios nodos como: StereoDepth, MobileNetDetectionNetwork y SpatialLocationCalculator. El proceso que sigue este nodo es el siguiente (Figura 42):

1. **Detección**: la red neuronal (MobileNet) se encarga de detectar objetos en la imagen de entrada y genera una lista de objetos detectados, cada uno representado por un cuadro delimitador (*bounding box*), una etiqueta (ID) que identifica el tipo de objeto y un puntuaje de confianza de la detección (*score*).
2. **Alineación**: se ajusta el mapa de profundidad, que muestra la distancia del objeto al sensor, para que coincida con el marco de entrada donde se realizó la detección. Esto es necesario porque la detección se realiza respecto a la imagen de entrada y el cálculo de profundidad opera en el mapa de profundidad.
3. **Escalado de BBOX**: el cuadro delimitador de la red neuronal se envía a SpatialLocationCalculator y se escala de acuerdo con BoundingBoxScaleFactor. Esto se hace para garantizar que incluya todo el objeto. Luego, el cuadro delimitador se utiliza junto con la profundidad para calcular las coordenadas espaciales del objeto.
4. **Cálculo de coordenadas espaciales**:
    - Las coordenadas X e Y**:** Se obtienen a partir del centro del cuadro delimitador, ajustadas por el desplazamiento desde el centro del marco.
    - La profundidad (coordenada Z): se tiene en cuenta cada píxel dentro del cuadro delimitador escalado (ROI). Esto da un conjunto de valores de profundidad, que luego se promedian para obtener el valor de profundidad final.
5. Devuelve la imagen con la detección, la cual va detallada con las coordenadas, la bbox, la clase de objeto y el porcentaje de certeza de la detección.

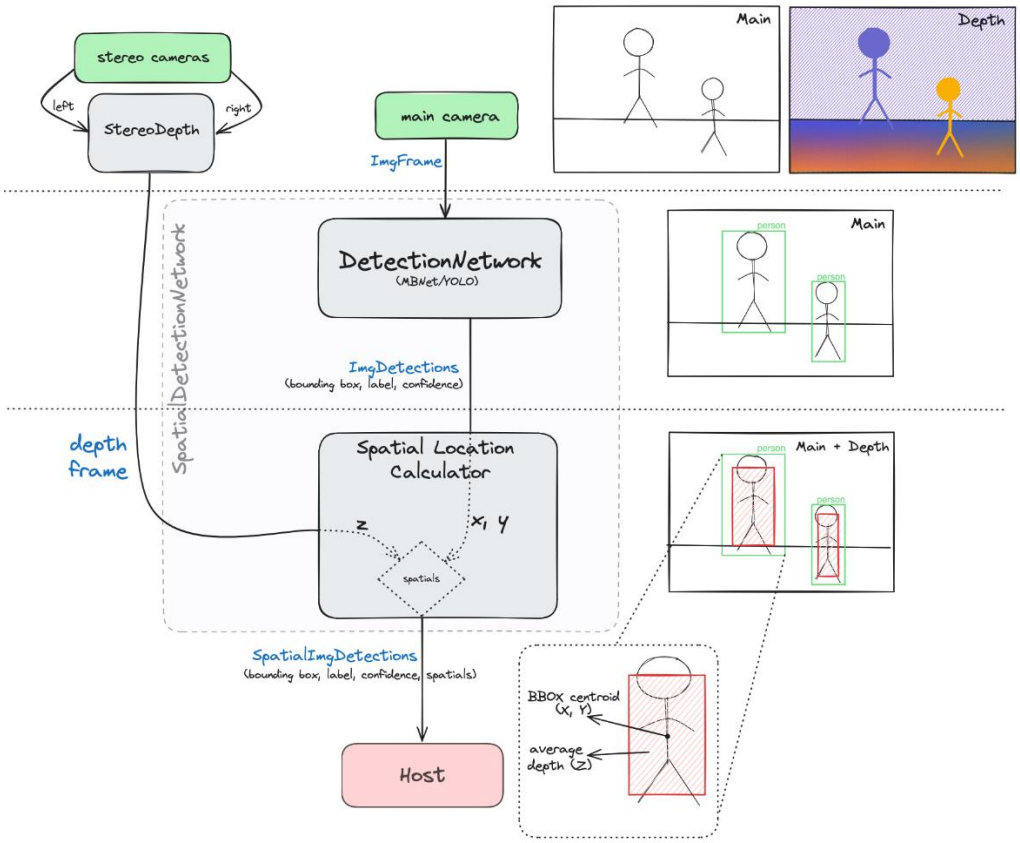


Figura 42 – Proceso detección espacial MobileNet

Una vez consolidado el funcionamiento del algoritmo de detección espacial MobileNet de la cámara, se desarrolla un nodo en ROS 2 para que la cámara OAK-D Pro funcione en el robot TurtleBot 4 y cumpla su objetivo de realizar la detección de personas como parte del proyecto de seguimiento de personas.

Este nodo de ROS 2 se llama “mobilenet” y primero configura el pipeline de DepthAI, que incluye cámaras monocromáticas para la percepción de profundidad, una cámara RGB para la captura de imágenes a color, y una red neuronal MobileNet-SSD para la detección de objetos.

Cada vez que el nodo recibe una nueva imagen y sus correspondientes datos de detección y profundidad, procesa esta información para identificar personas. Calcula las coordenadas y dimensiones de las cajas delimitadoras (bbox) alrededor de las personas detectadas y las convierte en mensajes de ROS 2 para su publicación en los tópicos 'camera/detections' (Figura 43) y 'camera/image'. Las detecciones incluyen información sobre la posición en el espacio tridimensional de la persona detectada (Figura 44), lo cual es crucial para el seguimiento.

```
---
header:
  stamp:
    sec: 1717521357
    nanosec: 893976283
  frame_id: oak_rgb_camera_optical_frame
detections:
- results:
  - class_id: '16'
    score: 0.0
  bbox:
    center:
      position:
        x: 269.5
        y: 307.0
      theta: 0.0
    size_x: 143.0
    size_y: 144.0
  position:
    x: 0.1263636201620102
    y: -0.20328056812286377
    z: 1.2243582010269165
  is_tracking: false
  tracking_id: ''
---
```


Figura 43 - Topic camera/detections

Figura 44 - Imagen con la detección de la persona

5.7. Ejecución nodos de las transformadas

5.7.1. Nodo *broadcaster*

Como se puede comprobar en la Figura 45, la cámara (*oakd_rgb_camera_optical_frame*) y el robot TurtleBot 4 (*base_link*) no tienen el mismo sistema de coordenadas (*frame*). Sin embargo, para el correcto seguimiento de personas es necesario que el sistema de coordenadas de la persona sea el mismo que el robot, pero con la altura de la cámara.

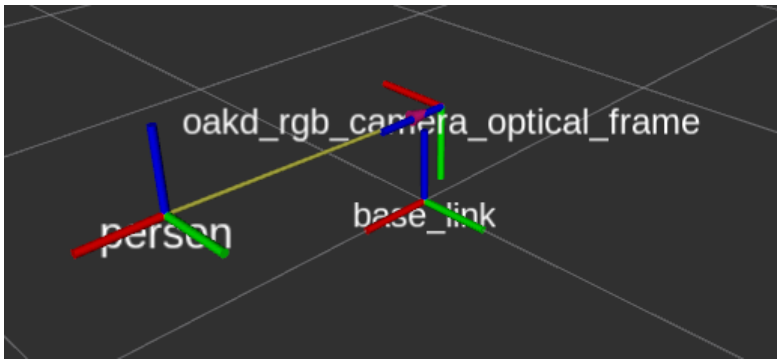


Figura 45 - TF cámara y TF robot

Por tanto, para obtener la transformación completa desde el frame de la cámara (*oakd_rgb_camera_optical_frame*) hasta el frame de la persona (*person*), se combinan las transformaciones de traslación y rotación.

La matriz de transformación homogénea completa del frame de la cámara al frame de la persona T_{oakd}^{person} y que combina la rotación y la traslación es:

$$T_{oakd}^{person} = \begin{bmatrix} R_{oakd}^{person} & t_{oakd}^{person} \\ 0 & 1 \end{bmatrix}$$

Donde:

- R_{oakd}^{person}: es la matriz de rotación desde el frame de la cámara al frame de la persona.
- t_{oakd}^{person}: es el vector de traslación desde el frame de la cámara al frame de la persona.

La matriz de traslación t_{oakd}^{person} viene definida por:

$$t_{oakd}^{person} = [x \quad y \quad z]^T$$

Entrando en detalle en la matriz de rotación R_{oakd}^{person} esta se compone de dos rotaciones. Estas rotaciones son representadas por cuaterniones y luego combinadas:

1. Rotación 1: 90º en el eje X. El cuaternión para esta rotación es:

$$q_x = \left(\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right), 0, 0\right) = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0, 0\right)$$

2. Rotación 2: 90º en el eje Z. El cuaternión para esta rotación es:

$$q_z = \left(\cos\left(\frac{\pi}{4}\right), 0, 0, \sin\left(\frac{\pi}{4}\right)\right) = \left(\frac{\sqrt{2}}{2}, 0, 0, \frac{\sqrt{2}}{2}\right)$$

3. Combinación de rotaciones multiplicando los cuaterniones:

$$q_{combinado} = q_x \times q_z = \left(\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}, \frac{1}{2}\right)$$

4. Convertir el cuaternión a matriz de rotación:

$$R_{oakd}^{person} = \begin{bmatrix} 1-2(y^2+z^2) & 2(xy-zw) & 2(xz+yw) \\ 2(xy+zw) & 1-2(x^2+z^2) & 2(yz-xw) \\ 2(xz-yw) & 2(yz+xw) & 1-2(x^2+y^2) \end{bmatrix}$$

Sustituyendo los valores del cuaternión combinado, la matriz de rotación es:

$$R_{oakd}^{person} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

En definitiva, para transformar la posición de la persona que es recibida en el frame de la cámara como un punto $p_{oakd} = (x, y, z, 1)$ al frame de la persona, se utiliza la matriz de transformación homogénea de la siguiente forma:

$$p_{person} = T_{oakd}^{person} \cdot p_{oakd} = \begin{bmatrix} 0 & 0 & 1 & x \\ 1 & 0 & 0 & y \\ 0 & 1 & 0 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Para llevar a cabo este desarrollo en el TurtleBot 4, se crea un nodo llamado que va a funcionar como *TransformBroadcaster* (transmisor). Esto es una herramienta utilizada en ROS para publicar transformaciones entre diferentes frames de referencia. Estas transformaciones son esenciales para permitir que distintos componentes de un sistema robótico comprendan la relación espacial entre ellos.

Este nodo se llama "tf_persona_camara" el cuál recibe la posición de la persona detectada por la cámara y transforma esa posición al frame de referencia. En primer lugar, se suscribe al topic /PuntoPersona, el cual contiene la posición real de la persona en coordenadas 3D, y almacena esta posición. A continuación, *TransformBroadcaster* se encarga de realizar las transformaciones (rotaciones y traslaciones) y de publicar la transformación de las coordenadas, permitiendo que otros nodos del sistema ROS puedan conocer la posición y orientación de la persona respecto a la cámara.

En resumen, este nodo de ROS2 se encarga de recibir la posición de la persona detectada por la cámara OAK-D-Pro, transformar esa posición al frame de referencia de la cámara, y luego publicar esta transformación para que otros nodos en el sistema puedan utilizar esta información para sus propios propósitos.

5.7.2. Nodo *listener*

Una vez hechas las transformaciones se crea un nodo que funciona como *'TransformListener'* (oyente) que permite recibir y almacenar las transformaciones entre diferentes frames de referencia en un sistema robótico como el TurtleBot 4. El *TransformListener* escucha las publicaciones del nodo anterior (*broadcaster*)y las almacena en un

buffer de transformaciones, permitiendo que las transformaciones se consulten posteriormente.

Este nodo se llama 'tf_camara_odom' y tiene como objetivo principal obtener y publicar las posiciones de la persona y las del robot en el frame de referencia del sistema de odometría. En primer lugar, escucha y almacena las transformaciones publicadas en el sistema ROS, esto lo hace mediante el Buffer y el TransformListener. A continuación, con la función lookup_transform obtiene las transformaciones actuales desde el frame 'odom' hacia los frames 'person' y 'base_link'. Por último, publica las posiciones transformadas en sus topics correspondientes /PersonPositionInOdom y /BaseLinkPositionInOdom.

5.8. Ejecución nodo controlador

Para poder realizar el control del seguimiento de las personas del TurtleBot 4, es necesario entender cómo funciona su modelo cinemático diferencial.

El modelo cinemático diferencial del Turtlebot4 (Figura) se basa en el principio de control diferencial, donde el movimiento del robot es determinado por la velocidad de sus dos ruedas motrices laterales. Cada rueda puede girar a diferentes velocidades, lo que permite al robot moverse hacia adelante, hacia atrás, y girar sobre su eje.

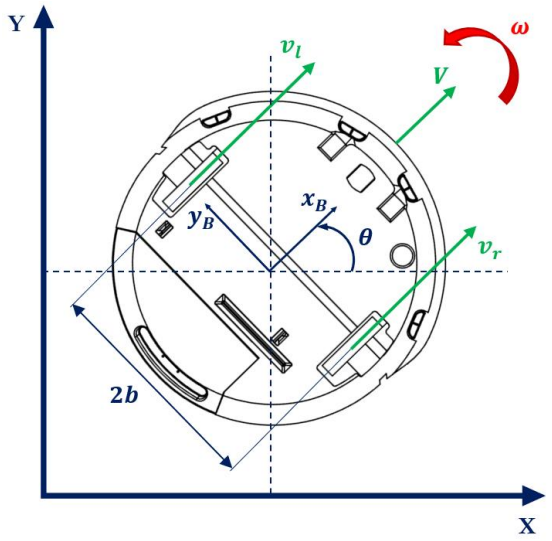


Figura 46 - Modelo cinemático diferencial

Ecuaciones del modelo cinemático diferencial

La velocidad lineal y angular del robot se pueden determinar a partir de las velocidades de sus ruedas.

$$V = \frac{v_l + v_r}{2}$$

$$\omega = \frac{v_l - v_r}{2b}$$

Donde:

- v_l: velocidad lineal de la rueda izquierda.
- v_r: velocidad lineal de la rueda derecha.
- V: velocidad lineal del centro del eje de las ruedas (velocidad del robot).
- ω: velocidad angular del robot (alrededor del eje vertical).
- $2b$: distancia total entre las ruedas izquierda y derecha.

El movimiento del robot en el plano XY se describe mediante las siguientes ecuaciones:

$$\dot{x}_B = V \cdot \cos(\theta)$$

$$\dot{y}_B = V \cdot \sin(\theta)$$

$$\dot{\theta} = \omega$$

Donde:

- $\dot{x}_B$ e $\dot{y}_B$ son las componentes de la velocidad del robot en las direcciones x e y, respectivamente.
- $\dot{\theta}$: es la tasa de cambio de la orientación del robot.

Controlador proporcional

El diseño del controlador proporcional se basa en como se quiere que el robot actúe, en este caso el TurtleBot 4 va a actuar de la siguiente manera cuando siga a una persona:

- El robot nunca va a llegar a chocar con la persona. Esto se tiene en cuenta en el error lineal del controlador, ya que habrá que restarle esta mínima distancia.
- El robot siempre va a buscar que la persona se encuentre en el centro de la imagen, esto quiere decir que al error angular no se le restara nada ya que la mínima distancia horizontal es cero.
- El robot siempre va a buscar a la persona. Muchas veces la persona se puede salir del rango de cámara, es decir, no aparece en el plano de la imagen. Sin embargo, el TurtleBot 4 girará en el sentido donde esté la persona. Ya que almacena la última posición horizontal de la persona antes de salir del plano.

Los errores son:

$$e_{lineal} = x - x_{min}$$

$$e_{angular} = y$$

Donde:

- x: es la distancia de la persona detectada.
- y: es la posición horizontal de la persona detectada.
- x_{min}: es la distancia mínima deseada del robot a la persona. Su valor es 0.55 m.

Las ecuaciones del controlador proporcional son:

$$v = K_{lineal} \cdot e_{lineal}$$

$$\omega = K_{angular} \cdot e_{angular}$$

Donde:

- K_{lineal}: es la ganancia proporcional para el control lineal. Su valor es 0.17.
- $K_{angular}$: es la ganancia proporcional para el control angular. Su valor es 0.35.

Por último, este controlador va programado en el nodo "mobilenet_Pcontrol". El cuál recibe la posición de la persona detectada, aplica el control y manda estas nuevas velocidades al topic /cmd_vel.

Este topic en el contexto de ROS es un canal de comunicación estándar utilizado para enviar comando de velocidad a los robots. Este tópico es generalmente de tipo '*geometry_msgs/Twist*', que contiene dos vectores: uno para las velocidades lineales y otro para las velocidades angulares.

Controlador proporcional variable

Para hacer que el robot se acerque más rápido a la distancia mínima deseada utilizando un controlador proporcional (P), se puede modificar la ganancia proporcional de manera dinámica en función de la distancia actual a la persona. Este controlador proporcional variable solo sería necesario en la velocidad lineal del TurtleBot4.

Para su implementación:

1. Se define las distancias (errores) mínimas y máximas, y a su vez, se definen las ganancias proporcionales máximas y mínimas:

$$x_{min} = 0.20\ m \qquad x_{max} = 5.0\ m$$

$$K_{lineal\_min} = 0.1 \qquad K_{lineal\_max} = 0.5$$

2. Cálculo de la ganancia variable: se define una función que ajusta la K_{lineal} basándose en el error actual.

$$K_{lineal} = a \cdot x + b$$

Donde:

$$a = \frac{K_{lineal\_max} - K_{lineal\_min}}{x_{max} - x_{min}}, \qquad b = K_{lineal\_min} - a \cdot x_{min}$$

Los errores de este controlador proporcional variable son:

$$e_{lineal} = x - x_{min}$$

$$e_{angular} = y$$

Y las ecuaciones del controlador:

$$v = K_{lineal} \cdot e_{lineal} = (a \cdot x + b) \cdot e_{lineal}$$

$$\omega = K_{angular} \cdot e_{angular} = 0.35 \cdot e_{angular}$$

Por último, este controlador va programado en el nodo "mobilenet_PVcontrol". El cuál recibe la posición de la persona detectada, aplica el control y manda estas nuevas velocidades al topic /cmd_vel.

6. Análisis de resultados y conclusiones

6.1. Modelos de detección de objetos

Aunque en el diseño inicial se optó por MobileNet-SSD, se quiso comparar este modelo con la segunda opción tenida en cuenta, YOLOv8. Como se puede visualizar en las Figura 47 y Figura 48, ambas redes neuronales convolucionales (CNN) detectan perfectamente todos los objetos. Incluso en las situaciones más difíciles de interpretar.

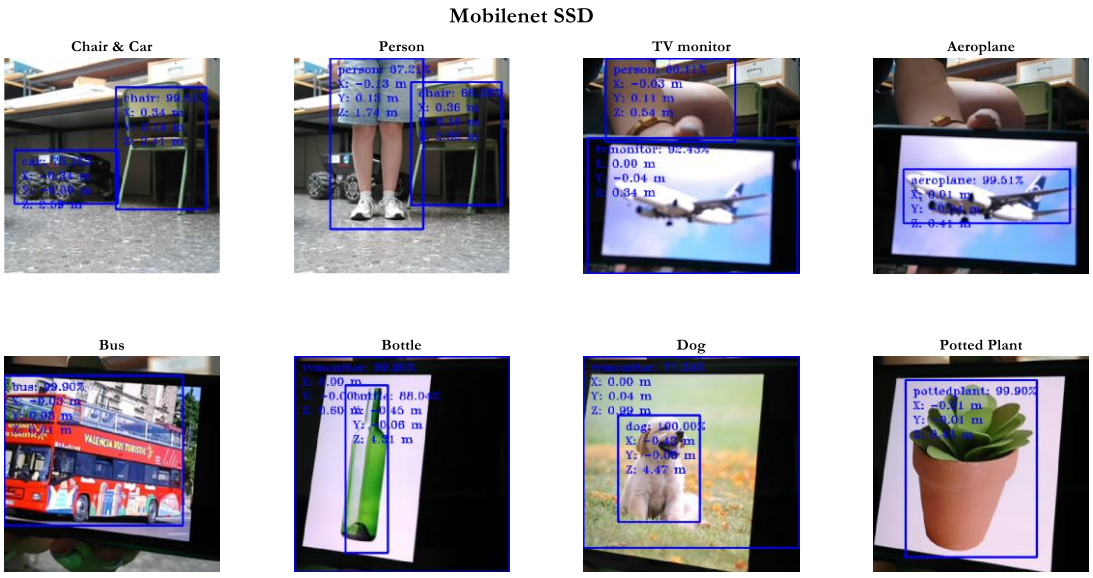


Figura 47 - Detecciones MobileNet SSD

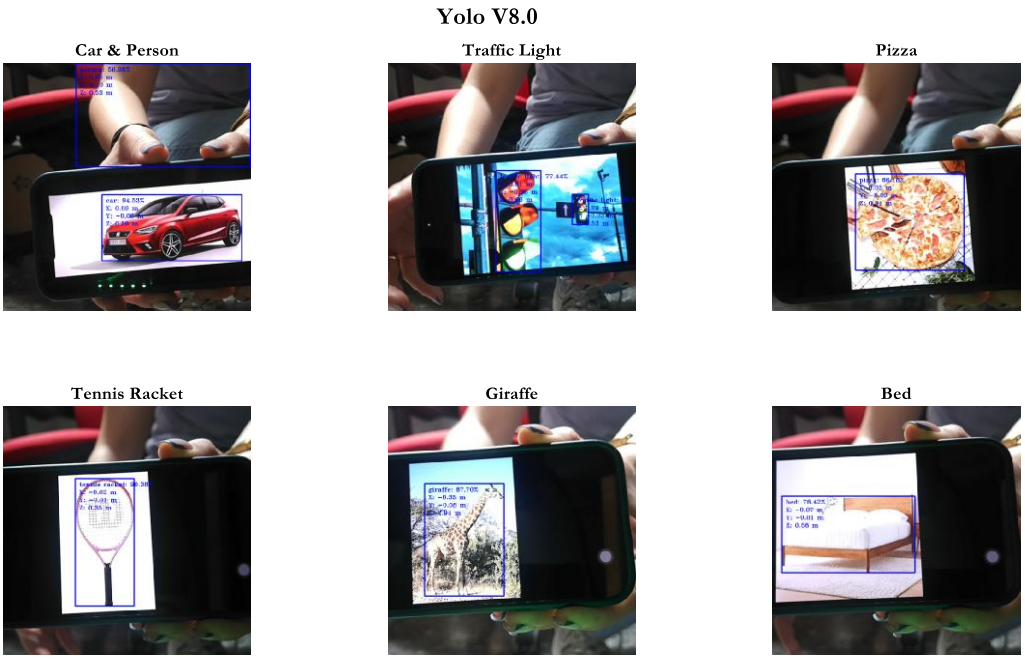


Figura 48 - Detecciones YOLOv8

Hay que destacar que YOLO tiene un conjunto de clases de objetos más grande que MobileNet-SSD. Eso puede ser interesante en otras aplicaciones, sin embargo, en el seguimiento de personas solo interesa que detecte a personas. Y ambos modelos detectan a la perfección a las personas.

Por otro lado, en esta aplicación es muy importante el rendimiento y la visualización de la imagen desde el PC, el cuál accede a la Raspberry por ssh. Por ello, YOLO es conocido por ser un modelo de detección más completo, capaz de detectar múltiples objetos en una sola imagen. Sin embargo, requiere más poder de procesamiento, lo cual puede ser una desventaja a la hora de visualizar las imágenes. Cabe destacar, que la detección no sería con retroceso, ya que este algoritmo lo emplea la cámara, sino que la visualización a través de la Raspberry sería de peor calidad.

Por tanto, para este proyecto de seguimiento de personas mediante TurtleBot 4, MobileNet SSD es la opción elegida debido a su eficiencia, a su simplicidad del modelo y su precisión suficiente.

6.2. Trayectorias simples

Se han realizado dos trayectorias simples, una circular y otra cuadrada. Estas trayectorias se basan únicamente en el tiempo y en velocidades constantes, sin ninguna retroalimentación de sensores o corrección de errores en la trayectoria. A continuación, se describen las partes del código que realizan las trayectorias para corroborar que no se utiliza ningún tipo de controlador.

```
    def timer_callback(self):
        msg = Twist()
        current_time = self.get_clock().now()

        # velocidad lineal y angular
        msg.linear.x = 0.2  # m/s
        msg.angular.z = msg.linear.x / 1.0  # rad/s (radio = 1.00 m)

        # el tiempo que tarda en completar el circulo
        if (current_time - self.start_time).nanoseconds > 32.9 * 1e9:
            msg.linear.x = 0.0
            msg.angular.z = 0.0
            self.publisher_.publish(msg)
            self.get_logger().info('Trayctoria circular completada.')
            rclpy.shutdown()

        self.publisher_.publish(msg)
```

Figura 49 - Código trayectoria circular sin control

```
    def timer_callback(self):
        msg = Twist()
        current_time = self.get_clock().now()
        elapsed_time = (current_time - self.start_time).nanoseconds / 1e9

        if self.state == 'move':
            msg.linear.x = self.linear_speed
            msg.angular.z = 0.0
            self.distance_travelled = self.linear_speed * elapsed_time

            if self.distance_travelled >= self.side_length:
                self.start_time = current_time
                self.state = 'turn'
                self.distance_travelled = 0.0

        elif self.state == 'turn':
            msg.linear.x = 0.0
            msg.angular.z = self.angular_speed
            self.angle_turned = self.angular_speed * elapsed_time

            if self.angle_turned >= self.turn_angle:
                self.start_time = current_time
                self.state = 'move'
                self.angle_turned = 0.0

        self.publisher_.publish(msg)

        if self.state == 'move' and elapsed_time >= 4 * (self.side_length /
self.linear_speed + self.turn_angle / self.angular_speed):
```

```
        msg.linear.x = 0.0
        msg.angular.z = 0.0
        self.publisher_.publish(msg)
        self.get_logger().info('Trayectoria cuadrada completada.')
        rclpy.shutdown()
```

Figura 50 -Código trayectoria cuadrada sin control

Estas trayectorias simples permiten familiarizarse con el control básico del robot y es esencial para comprender las limitaciones y comportamientos del robot, lo cual es importante al desarrollar proyectos más complejos como el seguimiento de personas usando la cámara del TurtleBot 4. Al comprobar como el robot maneja estos movimientos simples programados, se identifican las necesidades específicas de un controlador avanzado para mejorar la precisión y adaptabilidad de este proyecto.

Las pruebas de estas trayectorias se han realizado en el LabLeni de la Universitat Politècnica de València (UPV), concretamente en la sala donde se encuentra el sistema OptiTrack. Es un sistema avanzado de captura de movimiento (*motion capture*) utilizado para rastrear y registrar el movimiento de objetos y sujetos en tiempo real con alta precisión. La tecnología se basa en varias cámaras de alta velocidad calibradas y sincronizadas entre sí, que gracias a los infrarrojos que inciden en los objetos (previamente marcados con unas bolitas de un material especial que reflejan muy bien los infrarrojos) definen un volumen de captura en el cual se registran los movimientos de dichos objetos, en este caso del TurtleBot 4.

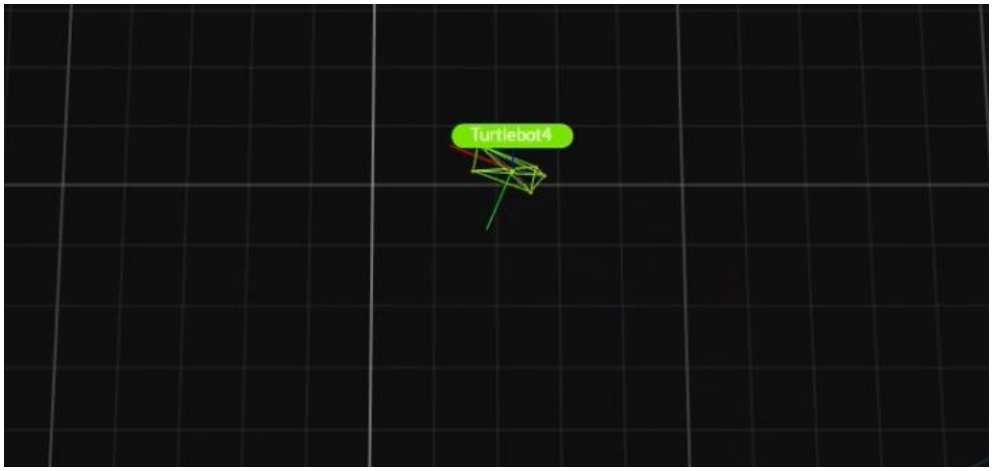


Figura 51 - TurtleBot 4 capturado desde OptiTrack

Observando los resultados de las trayectorias circular de dos metros de diámetro (Figura 52) y la trayectoria cuadrada de dos metros de lado (Figura 53), se puede notar que el enfoque basado en tiempo es efectivo para movimiento simples y predefinidos, y que ajustando los tiempos correctamente se puede conseguir las trayectorias deseadas sin implementar controladores avanzados. Por tanto, estas pruebas de trayectorias demuestran que el robot es bastante fiel a los comandos, velocidades constantes y tiempos que se le aplican.

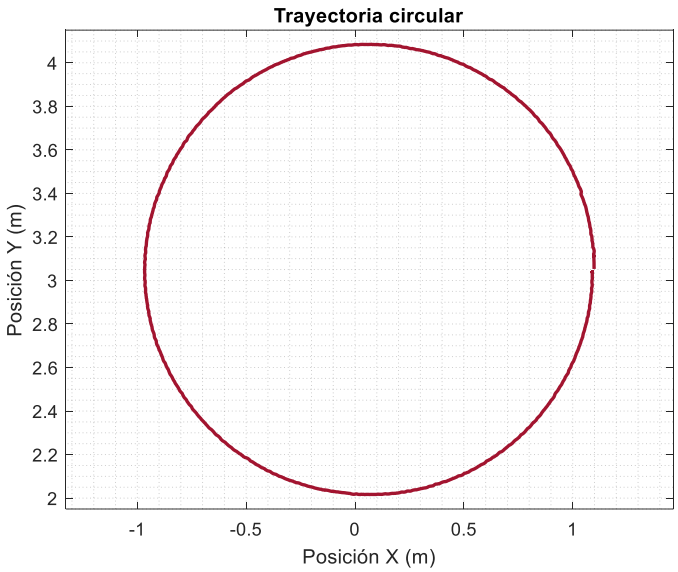


Figura 52 - Trayectoria circular

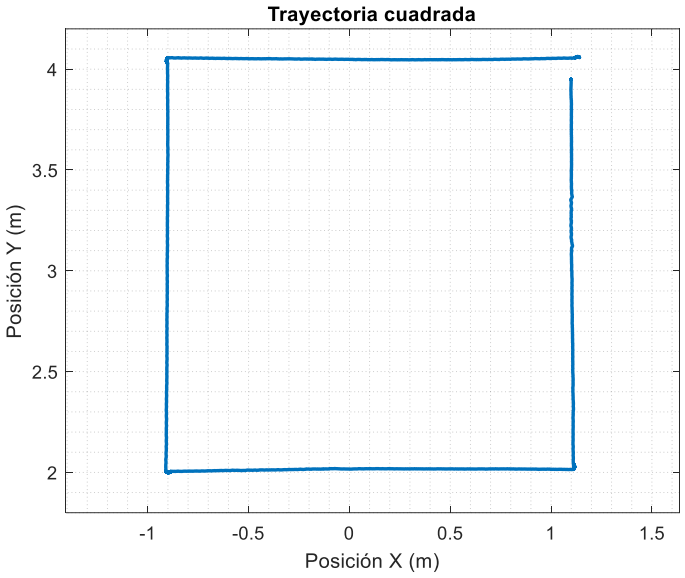


Figura 53 - Trayectoria cuadrada

6.3. Controladores

Teniendo en cuenta los buenos resultados de las trayectorias simples, se determina que un controlador proporcional (P) o proporcional variable (PV) pueden ser una solución adecuada debido a su simplicidad y efectividad. En este apartado, se comparan ambos controladores en el mismo escenario.

Las pruebas se realizan con el OptiTrack, este sistema capta los movimientos tanto de la persona como del robot, y los registra. Sin embargo, comparar dos controladores es difícil cuando la trayectoria está definida por el seguimiento del robot a la persona. Esto se debe a que las personas no siguen trayectorias exactas y predefinidas, sus movimientos pueden variar al igual que su velocidad, ya que no son constantes.

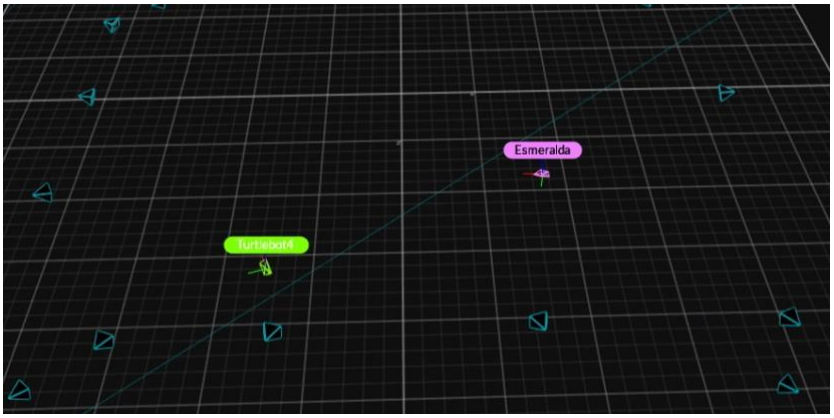


Figura 54 - Persona y TurtleBot 4 desde Optitrack

Para conseguir una correcta comparación de controladores, se ha establecido unos patrones y pasos a seguir en la trayectoria de la persona (figura X):

1. **Posición inicial:** el robot comienza estando de lado a la persona a una distancia aproximada de 2 metros. La persona se encuentra estática, al igual que el robot.
2. **Giro del robot:** el robot gira hacia la izquierda hasta encontrar a la persona.
3. **Avance del robot:** una vez detecta a la persona, empieza a seguirla y a acercarse.
4. **El robot alcanza a la persona:** cuando llega a la distancia mínima de 0.55 metros, el robot para.
5. **Movimiento de la persona:** la persona comienza a caminar perpendicularmente, contando un segundo por cada paso para intentar mantener una velocidad constante. El robot va girando para no perder a la persona.
6. **Parada de la persona:** la persona se detiene después de avanzar aproximadamente tres metros y espera al robot, manteniéndose estática. El robot mientras sigue avanzando.
7. **El robot alcanza a la persona:** cuando llega a la distancia mínima de 0.55 metros, el robot para.
8. **Movimiento de la persona:** la persona comienza a caminar perpendicularmente, contando un segundo por cada paso para intentar mantener una velocidad constante. El robot va girando para no perder a la persona.
9. **Parada de la persona:** la persona se detiene después de avanzar aproximadamente tres metros y espera al robot, manteniéndose estática. El robot mientras sigue avanzando.
10. **Parada final:** cuando llega a la distancia mínima de 0.55 metros, el robot se detiene.

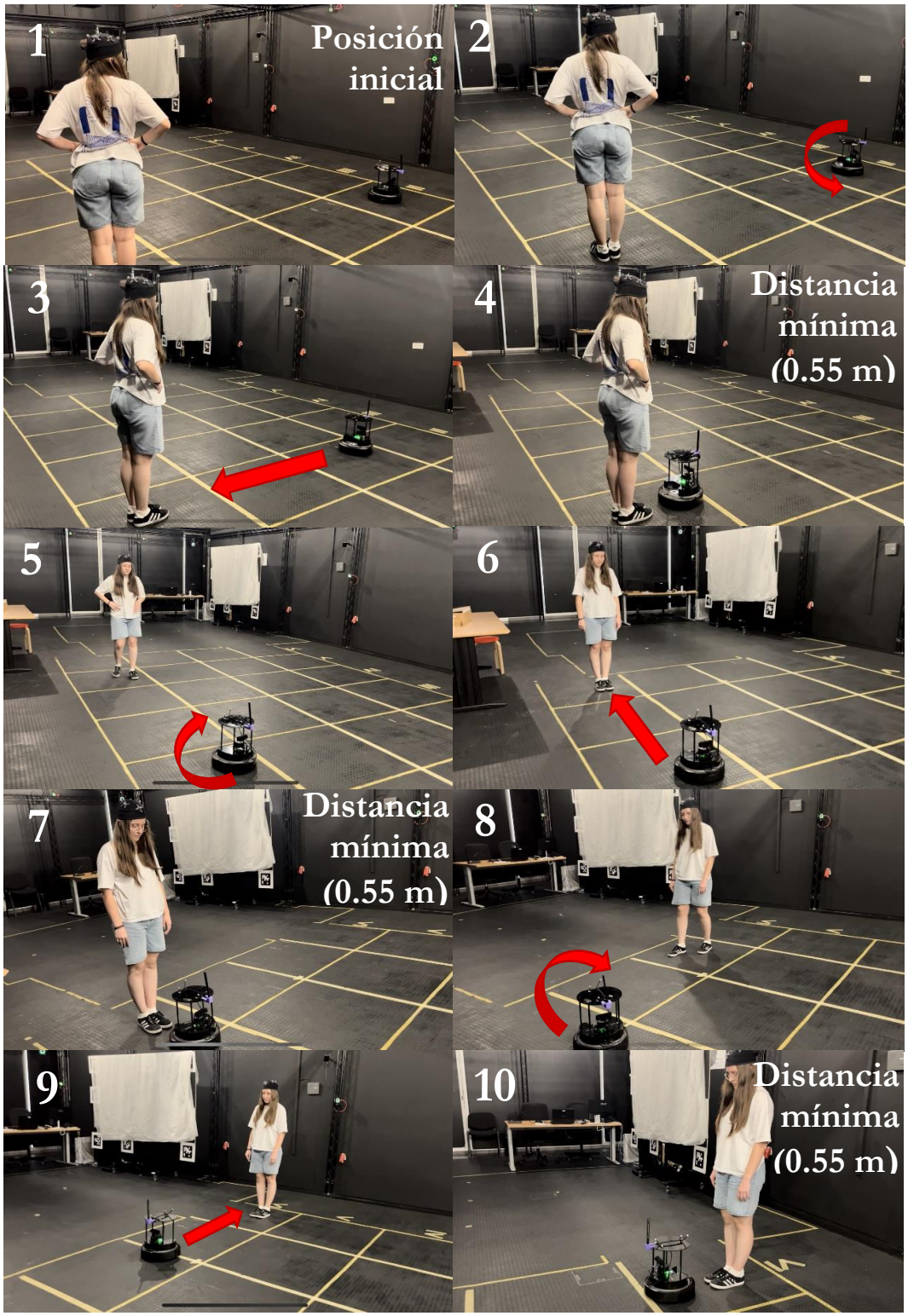


Figura 55 - Pasos a seguir en la prueba de los controladores

Una vez establecida la metodología a seguir en las pruebas de los dos controladores, se procede a analizarlos. A simple vista (Figura 56 y Figura 57) ambos controladores siguen fielmente el seguimiento de la persona y se aprecian todos los pasos o movimientos detallados anteriormente, todas las paradas, giros, avances, etc.

Inicialmente se propuso el controlador proporcional ya que cumplía con el funcionamiento, precisión, simplicidad y eficiencia espera, y como se puedo comprobar en la Figura 56 el diseño inicial fue el correcto ya que cumple con todo ello. Sin embargo, como alternativa se propuso el controlador proporcional variable (Figura 57), que cumplía con todos los requisitos y, además, podía ofrecer una ganancia adaptativa según la circunstancia del robot y persona. Adaptando su velocidad según se fuera acercando o alejando de la persona.

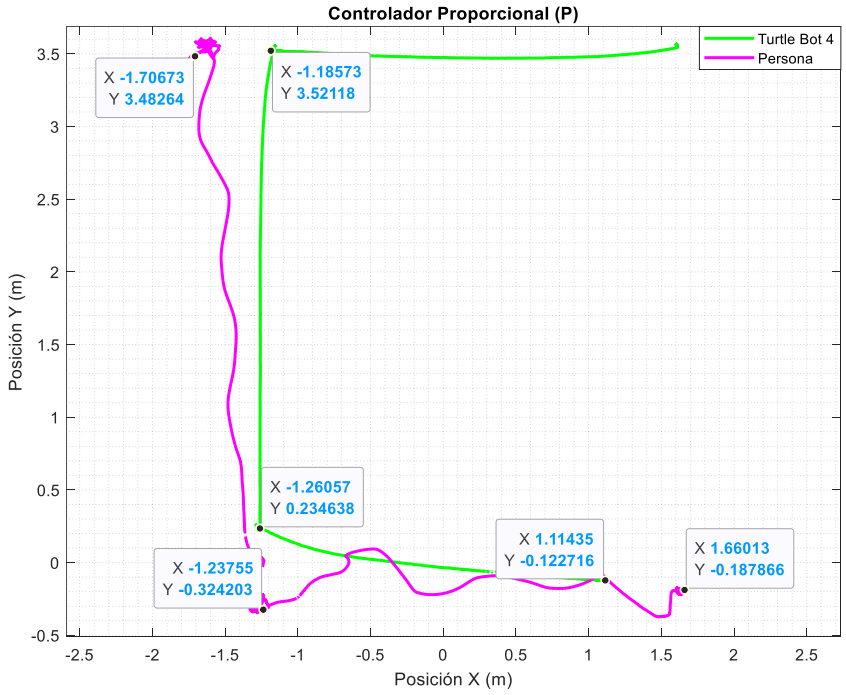


Figura 56 - Controlador Proporcional

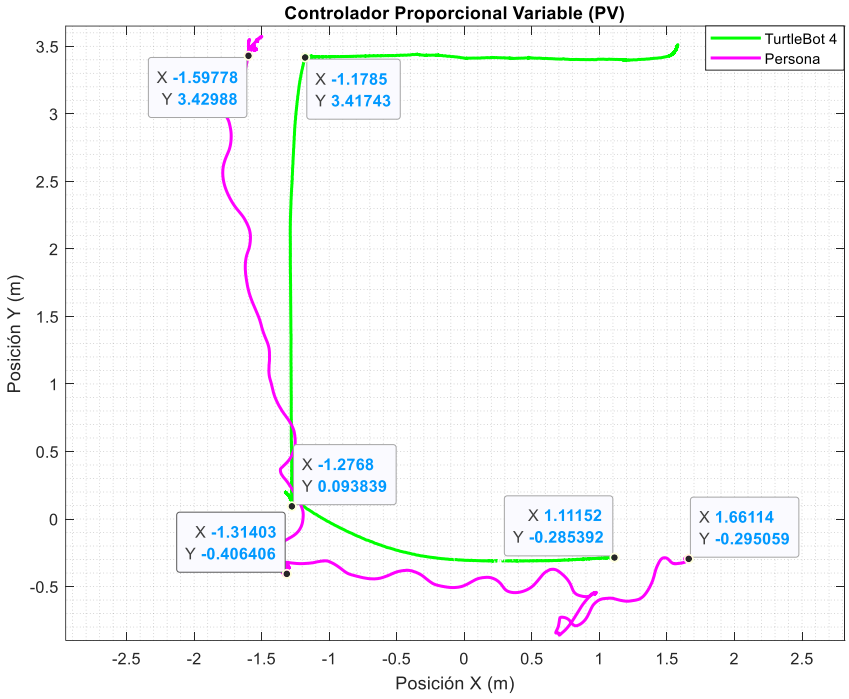


Figura 57 - Controlador Proporcional Variable

Por tanto, se procede a comparar numéricamente el tiempo transcurrido entre un controlador y otro. Además, de corroborar que ambos cumplen con la distancia mínima establecida entre la persona y el TurtleBot 4.

El OptiTrack recoge los datos en un Excel, donde a parte de almacenar la posición en 3D de los objetos, también registra la variable del tiempo. Sabiendo donde empieza la trayectoria y donde acaba se obtiene fácilmente el tiempo transcurrido de las trayectorias. Por tanto, se obtiene que el controlador proporcional (P) es menos de un segundo más rápido. Estos resultados se dan como iguales, por que como se ha comentado anteriormente una trayectoria descrita por una persona es muy difícil que sea exactamente igual en ambos casos.

$$t_P = 98.333\ seg$$

$$t_{PV} = 99.167\ seg$$

Por otro lado, se ha demostrados también que el controlador proporcional variable (PV) no aumenta significativamente las velocidades, ya que a velocidades altas (lejos de la persona) el TurtleBot 4 tiene un límite establecido de fábrica y velocidades bajas (cerca de la persona) no puedo aumentar más la velocidad ya que sino el controlador perdería su efecto y no mantendría la distancia mínima ya que se pasaría. A continuación, esto se verificará.

Respecto al cumplimiento de la distancia mínima, ambas figuras se muestran con etiquetas los puntos (posiciones) del robot y de la persona. Estas posiciones son exactamente en los pasos (pasos 4, 7 y 10) donde el robot alcanza a la persona y debe mantener la distancia

mínima. Con estos datos de las posiciones se realiza la siguiente tabla donde se comparan los resultados.

Comparativa de la distancia mínima entre los controladores P y PV				
Pasos	Controlador P		Controlador PV	
4	$x_t = -1.185$ $x_p = -1.733$	$\boldsymbol{d_{min} = 0,548\ m}$	$x_t = -1.178$ $x_p = -1.597$	$\boldsymbol{d_{min} = 0,419\ m}$
7	$y_t = 0.234$ $y_p = -0.324$	$\boldsymbol{d_{min} = 0,558\ m}$	$y_t = 0.093$ $y_p = -0.407$	$\boldsymbol{d_{min} = 0,500\ m}$
10	$x_t = 1.116$ $x_p = 1.661$	$\boldsymbol{d_{min} = 0,545\ m}$	$x_t = 1.111$ $x_p = 1.661$	$\boldsymbol{d_{min} = 0,550\ m}$

Tabla 14 - Comparativa de la distancia mínima entre los controladores P y PV

Donde:

- x_t: es la posición del TurtleBot 4 en el eje X.
- y_t: es la posición del TurtleBot 4 en el eje Y.
- x_p: es la posición de la persona en el eje X.
- y_p: es la posición de la persona en el eje Y.
- d_{min}: es la distancia mínima entre el TurtleBot 4 y la persona.

A simple vista es fácil saber cuál es el más preciso, pero se van a realizar las respectivas desviaciones promedio:

$$D_{\bar{x}} = \frac{\sum_{i=1}^{N} |x_i - \bar{x}|}{N}$$

$$D_{\bar{x},P} = \frac{|0.548 - 0.55| + |0.558 - 0.55| + |0.545 - 0.55|}{3} = 0.005$$

$$D_{\bar{x},PV} = \frac{|0.419 - 0.55| + |0.500 - 0.55| + |0.550 - 0.55|}{3} = 0.060$$

Como resultado, el controlador proporcional P tiene una desviación promedio menor que el controlador proporcional variable. Esto demuestra que es más preciso ya que tiene mejores resultados consiguiendo la distancia mínima entre el TurtleBot 4 y la persona.

6.4. Trayectorias recogidas por los topics

Mediante los topics /PersonPositionInOdom y /BaseLinkPositionInOdom, se pueden representar las trayectorias de la persona y del TurtleBot 4, respectivamente. En esos topics se publican las posiciones en 2D (x, y) de cada uno.

El primer topic, el que publica la posición de la persona se obtiene mediante la posición que detecta la cámara sobre la persona y mediante los nodos de las transformadas (tf) se obtiene la verdadera posición de la persona respecto a odom. Sin embargo, el segundo topic publica la posición del robot debido a la transformada de base link (robot) respecto de odom.

Para aclarar, el término odom se refiere a la odometría, un concepto crucial en robótica para el seguimiento del movimiento de un robot. La odometría proporciona datos sobre la posición y orientación de un robot con el tiempo, basándose en las mediciones de los sensores de movimiento, como los encoders en las ruedas del robot. En ROS 2, los datos de odometría se publican en un topic llamado /odom.

El topic /odom se debe de resetear a coordenadas iniciales (0,0), ya que guarda su última odometría. Existe un comando que resetea la pose que tenga guardada pero no el sistema de ejes y sus rotaciones. Eso se puede comparar fácilmente en las dos Figura 58 y Figura 59, empiezan en (0,0), pero van en diferente dirección, aunque la persona está realizando el mismo recorrido.

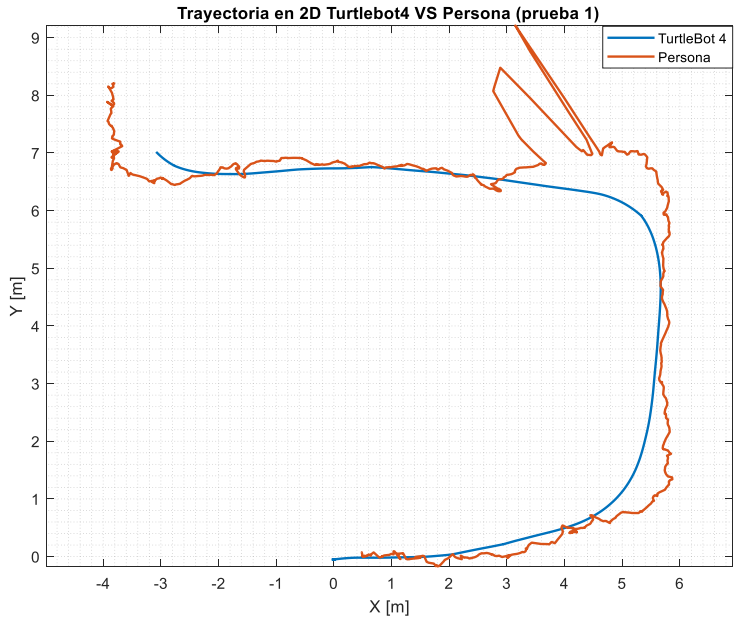


Figura 58 - Trayectoria TurtleBot 4 VS Persona mediante topics (prueba 1)

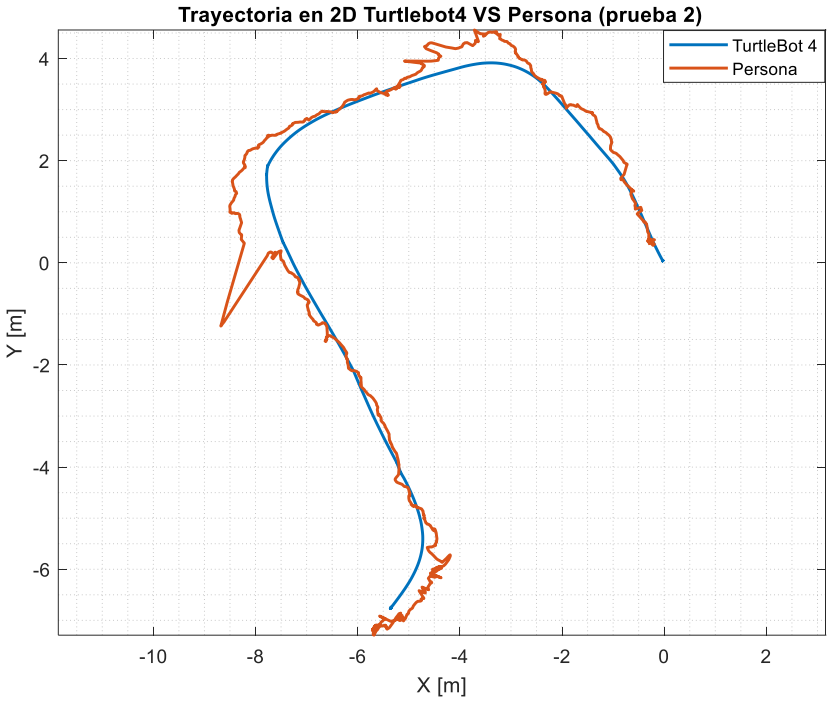


Figura 59 -Trayectoria TurtleBot 4 VS Persona mediante topics (prueba 2)

Por otro lado, en ambas figuras se observa como la trayectoria de la persona tiene ciertas oscilaciones, ya que la trayectoria se obtiene a través de lo que recoge la cámara y esta tiene cierta sensibilidad. Además, en los momentos más críticos como las curvas cerradas (esquinas), se muestran esos picos que significan que la persona se ha salido del rango de la cámara.

7. Conclusiones

El presente proyecto ha cumplido satisfactoriamente todos los objetivos planteados inicialmente. A lo largo del proyecto, se ha logrado integrar con éxito algoritmos de visión artificial avanzados y un sistema de navegación robusto, lo que permite al robot detectar y seguir a individuos de manera precisa y eficiente.

En primer lugar, se ha integrado de manera efectiva los componentes principales del sistema, como el TurtleBot 4 y la cámara OAK-D-Pro, creando una plataforma robusta para la implementación de algoritmos de visión artificial. Se ha detallado la configuración de red y la puesta en marcha del sistema, asegurando su correcta operación y facilidad de uso. Además, se han explorado y evaluado diversos métodos Deep Learning que implementa la cámara, comparando los modelos de detección de personas, como MobileNet y YOLOv8

Las pruebas realizadas han confirmado que el TurtleBot 4, equipado con el adaptador Wi-Fi USB, mantiene una conexión estable y una comunicación fluida con los sistemas de control. Los ajustes y la elección de los algoritmos de control y detección de personas han

permitido mejorar la respuesta del robot a las mediciones de la cámara, asegurando un funcionamiento confiable.

En resumen, este proyecto no solo ha alcanzado los objetivos establecidos inicialmente, sino que también ha proporcionado valiosos conocimientos y experiencia en el desarrollo de sistemas robóticos basados en ROS 2 y en la visión artificial. Las conclusiones obtenidas servirán como base para futuros trabajos y mejoras, incluyendo la exploración de nuevas técnicas de inteligencia artificial y la integración de estos sistemas en la vida cotidiana.

7.1. Mejoras futuras

Sin embargo, se han identificado áreas de mejora o de posibles trabajos futuros para completar el presente proyecto:

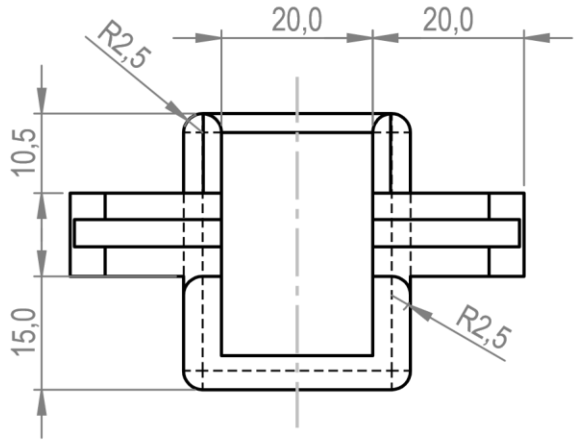
- **Evitación de obstáculos integrando el LiDAR:** en este proyecto se intentó desarrollar. Sin embargo, el consumo de la cámara junto al adaptador Wi-Fi USB y el LiDAR era muy elevado. A la hora de utilizar los tres componentes a la vez, se perdía la conexión y Wi-Fi y la cámara no arrancaba correctamente. Por ello, un trabajo futuro sería implementar la capacidad del que robot evitara obstáculos.
- **Reconocimiento de gestos:** añadir capacidades de reconocimiento de gestos para que el robot pueda responder a comandos gestuales de la persona que sigue. Esto sería muy aplicable a situaciones de la vida cotidiana.
- **Aplicaciones Prácticas:**
    - **Asistencia Personal:** explorar aplicaciones en asistencia personal para personas mayores o con discapacidades, donde el robot pueda seguir y asistir al usuario en tareas diarias.
    - **Seguridad y Vigilancia:** investigar el uso del Turtlebot4 en aplicaciones de seguridad, donde el robot pueda seguir a personas sospechosas en un entorno vigilado.
- **Mejorar la interfaz con el usuario:** desarrollar una interfaz de usuario más intuitiva y funcional para el control y monitoreo del sistema. Esto podría incluir opciones para ajustar parámetros del sistema en tiempo real, visualizar datos de sensores y realizar diagnósticos remotos.

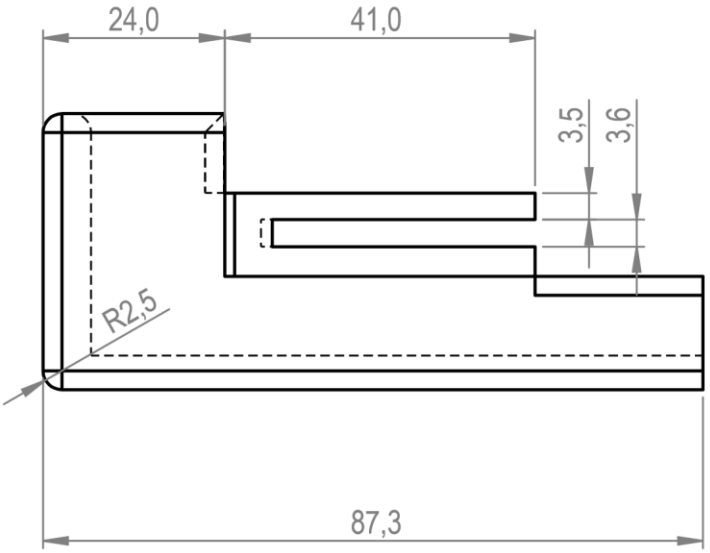
8. Bibliografía

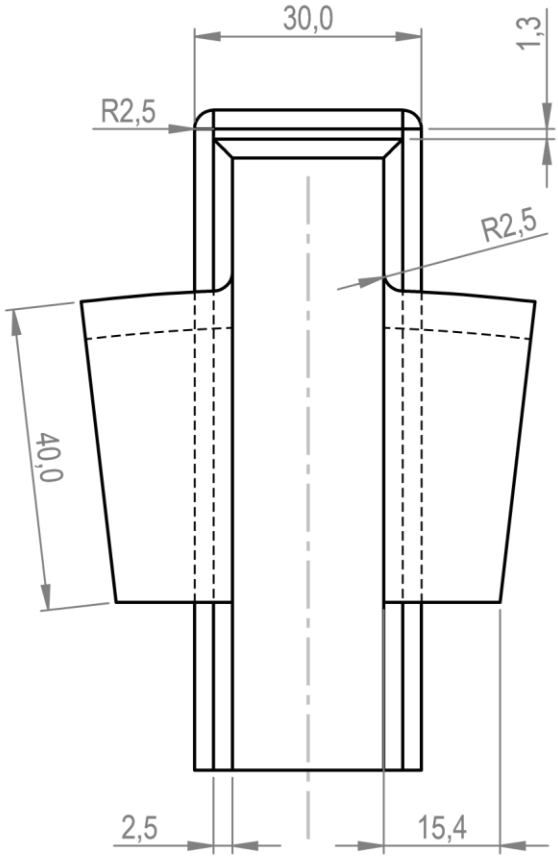
1. Keylabs AI. (sf). *YOLOv8 vs SSD: Cómo elegir el modelo de detección de objetos adecuado*. https://keylabs.ai/blog/yolov8-vs-ssd-cómo-elegir-el-modelo-de-detección-de-objetos-adecuado/
2. Aprende Aprendizaje Automático. (sf). *Modelos de detección de objetos* https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/
3. Luxonis. (sf). *Componentes de DepthAI: canalización*. https://docs.luxonis.com/software/ Depthai- components/pipeline/
4. Clearpath Robotics. (sf). *TurtleBot 4: plataforma robótica de código abierto*. https://clearpathrobotics.com/turtlebot-4/
5. Documentación de ROS2. (sf). *Documentación del sistema operativo robótico 2*. Recuperado de https://docs.ros.org/en/humble/index.html
6. Redmon, J. y Farhadi, A. (2018). *YOLOv3: una mejora incremental*. Preimpresión de arXiv arXiv:1804.02767. https://arxiv.org/abs/1804.02767
7. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, CY y Berg, AC (2016). *SSD: Single Shot MultiBox Detector*. En European Conference on Computer Vision (pp. 21-37). Springer, Cham. https://arxiv.org/abs/1512.02325
8. Luxonis. (sf). *Especificaciones de OAK-D-Pro* . https://docs.luxonis.com/hardware/oak/d/
9. Quigley, M., Gerkey, B. y Smart, WD (2015). *Programación de robots con ROS: una introducción práctica al sistema operativo de robots* . O'Reilly Media.
10. Ying, H. y Santoro, M. (2020). *Entrenamiento avanzado en ROS2: implementación de robots autónomos* . Packt Publishing.

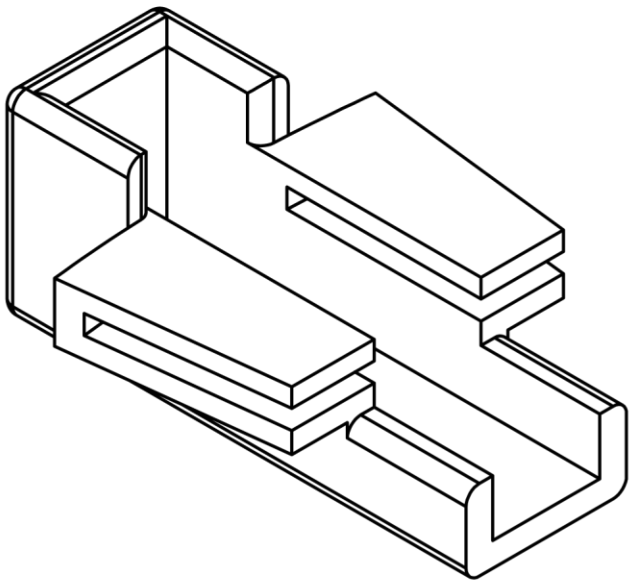
Capítulo 2

Planos

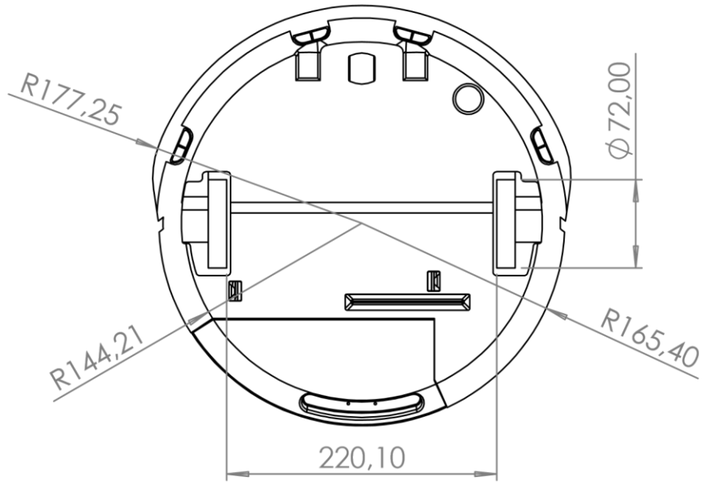


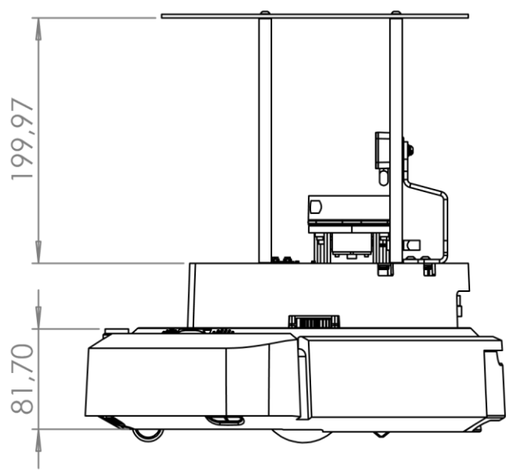


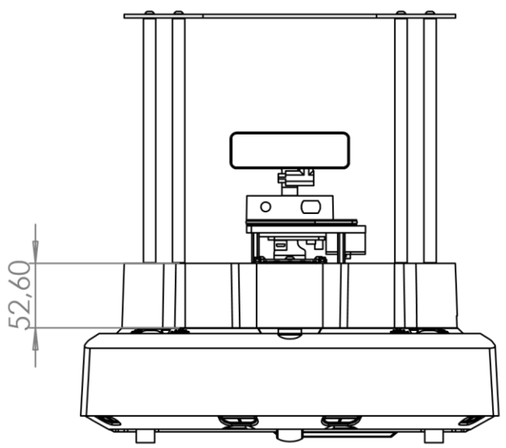


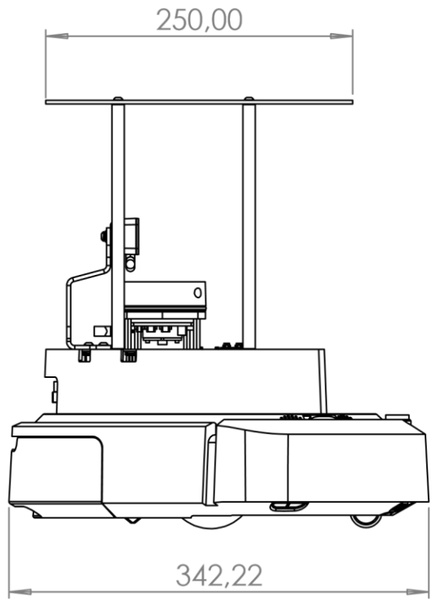


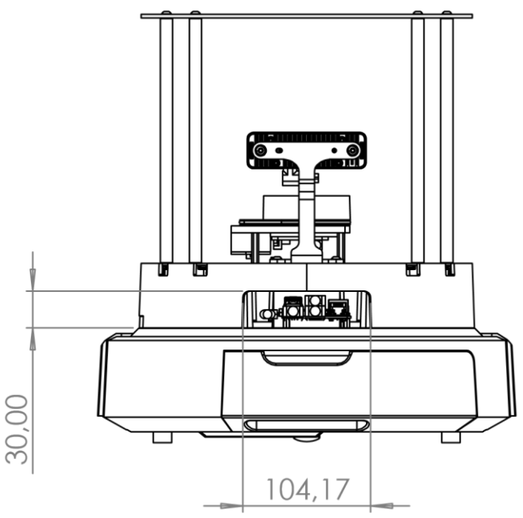
	Fecha	Nombre	Título del proyecto	
Dibujado:	9/04/2024	Esmeralda Armero	Análisis de algoritmos de visión con Turtlebot4 y ROS2	ETSI Aeroespacial y Diseño Industrial
Comprobado:	2/05/2024	Esmeralda Armero		
Escala 1:1	Unidades mm	Plano 76 Pinza adaptador USB Wi-Fi	Número 1	UNIVERSITAT POLITÈCNICA DE VALÈNCIA

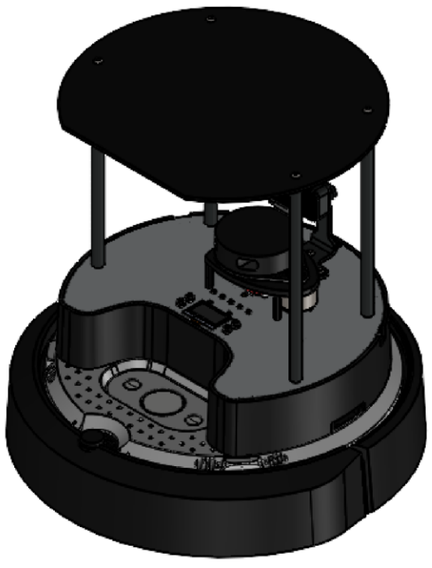


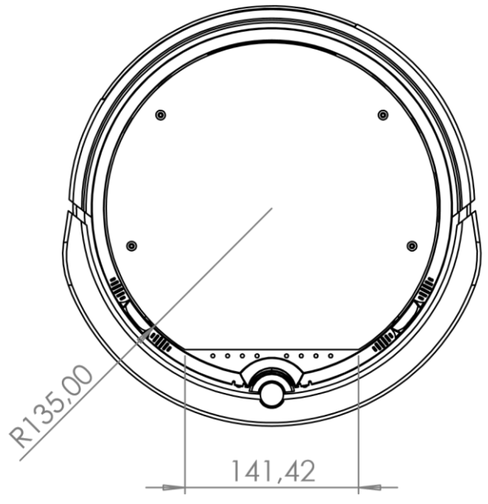












77

	Fecha	Nombre	Título del proyecto	
Dibujado:	13/05/2024	Esmeralda Armero	Análisis e integración de algoritmos de visión con Turtlebot4 y ROS2	ETSI Aeroespacial y Diseño Industrial
Comprobado:	20/05/2024	Esmeralda Armero		
Escala 1:5	Unidades mm	Plano TurtleBot 4	Número 2	UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Capítulo 3

Pliego de condiciones

1. Objeto

El presente pliego de condiciones tiene como finalidad establecer las especificaciones técnicas y operativas para la implementación y prestación del servicio de seguimiento autónomo de personas utilizando el robot TurtleBot 4 equipado con una cámara OAK-D. Este servicio está diseñado para aplicaciones en diversos entornos, tales como seguridad, asistencia en eventos, y soporte a personas con necesidades especiales.

2. Condiciones de los materiales

2.1. TurtleBot 4

Descripción

Se utilizará el robot móvil autónomo TurtleBot 4, proporcionado por *Open Robotics* y *Clearpath Robotics*. Este robot está diseñado como una plataforma de desarrollo accesible y versátil, con capacidades avanzadas para la navegación autónoma y la integración de múltiples sensores. Sus dimensiones, especificaciones y toda la información relevante del producto se encuentran en la página oficial del fabricante.

Control de calidad

El TurtleBot 4 debe cumplir con los estándares de calidad establecidos por el fabricante, incluyendo pruebas de funcionamiento y verificación de componentes antes de su entrega. Además, se verificará que todos los elementos del robot, como motores, sensores y sistemas de control, estén en perfecto estado de funcionamiento y libres de defectos.

2.2. Cámara OAK-D-Pro

Descripción

Se utilizará la cámara OAK-D-Pro es una cámara inteligente que combina capacidades de captura de imágenes en 3D con procesamiento de visión artificial en tiempo real. Será proporcionada por *Luxonis* y sus dimensiones, especificaciones e información del producto se encuentran en la página oficial del fabricante.

Control de calidad

El proveedor deberá garantizar que la cámara OAK-D-Pro cuenta con las siguientes certificaciones:

- **Certificado CE:** Asegura que el producto cumple con los requisitos esenciales de las Directivas europeas aplicables, relacionadas con la seguridad, salud y protección del medio ambiente.
- **Certificado de conformidad con RoHS3:** Acredita que el producto cumple con la Directiva 2011/65/UE de la Unión Europea, restringiendo el uso de ciertas sustancias peligrosas en equipos eléctricos y electrónicos, como plomo, mercurio y cadmio.

2.3. Adaptador USB Wi-Fi AC650

Descripción

Se utilizará el adaptador USB Wi-Fi AC650 de *Brostrend* para proporcionar conectividad inalámbrica al TurtleBot 4. Este adaptador es conocido por su alta velocidad de transmisión y estabilidad, lo que es crucial para la comunicación en tiempo real entre el robot y otros sistemas. Sus dimensiones, especificaciones e información del producto se encuentran en la página oficial del fabricante.

Control de calidad

- **Especificaciones técnicas**: El adaptador *Brostrend* deberá cumplir con las especificaciones técnicas proporcionadas por el fabricante, incluyendo soporte para

estándares Wi-Fi como 802.11ac, y garantizará una conexión de alta velocidad y baja latencia.
- **Certificaciones**: El adaptador deberá contar con las certificaciones necesarias, como el Certificado CE, el FCC y el Certificado de conformidad con RoHS3, que aseguran su seguridad y compatibilidad con los estándares ambientales y de seguridad europeos.

2.4. Pinza para el Adaptador Wi-Fi

Descripción

La pinza será diseñada para sujetar firmemente un adaptador Wi-Fi USB al TurtleBot 4. Este componente será fabricado mediante impresión 3D, utilizando materiales resistentes que aseguren una sujeción estable y una buena conectividad inalámbrica.

Control de calidad

- **Diseño y materiales:** La pinza deberá ser diseñada con precisión para garantizar la compatibilidad con el adaptador Wi-Fi y el TurtleBot 4. Los materiales utilizados deben ser duraderos y adecuados para su uso en entornos operativos normales.
- **Pruebas de ajuste y funcionalidad:** Se realizarán pruebas para asegurar que la pinza fija correctamente el adaptador y que no afectará negativamente al rendimiento del dispositivo.

3. Condiciones de ejecución

3.1. Estructura del robot

Descripción

El montaje del TurtleBot 4 incluirá la instalación y aseguramiento de todos los componentes mecánicos y electrónicos, garantizando la correcta integración del adaptador USB Wi-Fi *Brostrend*. La pinza diseñada mediante impresión 3D será utilizada para fijar el adaptador Wi-Fi de manera segura, evitando interferencias con otros componentes del robot.

Control de calidad

Se verificará que todos los componentes estén correctamente instalados y funcionales. Se realizarán pruebas de ensamblaje para asegurar que la estructura del robot es estable y que los sensores y la cámara están posicionados adecuadamente para la captura de datos óptimos.

3.2. Programación

Descripción

La programación del TurtleBot 4 incluirá el desarrollo del software necesario para la operación autónoma del robot, utilizando ROS 2 como plataforma base. Se implementarán algoritmos de visión artificial para la detección y seguimiento de personas, así como algoritmos de control. Estos algoritmos se basarán en técnicas de aprendizaje profundo, como redes neuronales convolucionales (CNN), y se optimizarán para procesamiento en tiempo real.

Control de calidad

Después de la programación, se llevarán a cabo pruebas exhaustivas para verificar el correcto funcionamiento del software en diversas condiciones operativas. Esto incluirá pruebas de los algoritmos de detección y seguimiento, así como de los controladores. Se corregirán cualquier error o anomalía identificada durante las pruebas, garantizando que el robot pueda seguir a las personas de manera autónoma y responder adecuadamente a las variaciones en el entorno y a las entradas del usuario.

4. Prueba de servicio

La prueba de servicio se llevará a cabo mediante una serie de validaciones destinadas a verificar la eficacia y precisión del TurtleBot 4 en el seguimiento autónomo de personas. Durante estas pruebas, se evaluarán aspectos críticos como la capacidad del robot para detectar y seguir a una persona, la efectividad de los distintos modos de control implementados, y la estabilidad de la conexión con la cámara OAK-D-Pro y el adaptador Wi-Fi *Brostrend*.

Se realizarán ajustes y optimizaciones según sea necesario para mejorar el rendimiento del sistema, garantizando su fiabilidad en una variedad de condiciones operativas, como cambios en la iluminación, la presencia de múltiples personas o la aparición de obstáculos imprevistos. Este proceso incluirá la calibración de sensores, la optimización de algoritmos de visión artificial y la revisión de los parámetros de control.

Los resultados de estas pruebas serán fundamentales para identificar áreas de mejora en el diseño y funcionalidad del sistema, asegurando que el robot opere con la máxima eficacia y eficiencia en aplicaciones reales.

Capítulo 4

Presupuesto

1. Cuadro de precios unitarios

Cuadro de materiales			
Ref.	**Ud.**	**Descripción**	**Precio (€)**
E1	ud.	Robot móvil autónomo TurtleBot 4	2191.44
E2	ud.	Adaptador USB Wi-Fi AC650 BrosTrend	20.99
E3	g	Filamento PLA	0.03
E4	ud.	Alargador USB 2.0 de nylon de 0.5 m	5.49
E5	ud.	SanDisk Ultra 32GB microSD Memory Card	10.24

Tabla 15 - Cuadro de materiales

Cuadro de maquinaria			
Ref.	**Ud.**	**Descripción**	**Precio (€)**
MQ1	h	Equipo informático	0.075
MQ2	h	Impresora 3D Bambulab X1C	1.62
MQ3	h	Licencia de SOLIDWORKS	0.783
MQ4	h	Licencia MATLAB	0.432

Tabla 16 - Cuadro de maquinaria

Cuadro de mano de obra			
Ref.	**Ud.**	**Descripción**	**Precio (€)**
MO1	h	Ingeniero en Electrónica Industrial y Automática	18.00

Tabla 17 - Cuadro de mano de obra

2. Cuadro de precios descompuestos

Ref.	Ud.	Descripción	Precio (€)	Cant.	Total
d1	ud.	Pinza adaptador USB Wi-Fi			
E3	g	Filamento PLA	0.03	49.11	1.47
MQ1	h	Equipo informático	0.075	20	1.5
MQ2	h	Impresora 3D Bambulab X1C	1.62	4	6.48
MQ3	h	Licencia de SOLIDWORKS	0.783	20	15.66
MO1	h	Ingeniero en Electrónica Industrial y Automática	18.00	20	360
%		Mano de obra sobre costes directos	385.11	2%	7,70
				Precio total por ud.	**392.81**

Tabla 18 - Partida 1 de precios descompuestos

Ref.	Ud.	Descripción	Precio (€)	Cant.	Total
d2	ud.	Implementación del proyecto: diseño y programación.			
E1	ud.	Robot móvil autónomo TurtleBot 4	2191.44	1	2191.44
E2	ud.	Adaptador USB Wi-Fi AC650 BrosTrend	20.99	1	20.99
E4	ud.	Alargador USB 2.0 de nylon de 0.5 m	5.49	1	5.49
E5	ud.	SanDisk Ultra 32GB microSD Memory Card	10.24	1	10.24
MQ1	h	Equipo informático	0.075	300	22.5
MQ3	h	Licencia de SOLIDWORKS	0.783	10	7.83
MQ4	h	Licencia MATLAB	0.432	20	8.64
MO1	h	Ingeniero en Electrónica Industrial y Automática	18.00	300	5400
%		Mano de obra sobre costes directos	7667,13	2%	153.34
				Precio total por ud.	**7820.47**

Tabla 19 - Partida 2 de precios descompuestos

3. Cuadro de mediciones y ejecución material

Ref.	Ud.	Descripción	Precio (€)	Cant.	Total
d1	ud.	Pinza adaptador USB Wi-Fi	392.81	1	392.81
d2	ud.	Implementación del proyecto: diseño y programación.	7820.47	1	7820.47
				Precio total de ejecución material:	**8213.28**

Tabla 20 - Cuadro de mediciones

4. Resumen general del presupuesto

Resumen general del presupuesto			
Partidas		**Descripción**	**Importe (€)**
d1	ud.	Pinza adaptador USB Wi-Fi	392.81
d2	ud.	Implementación del proyecto: diseño y programación.	7820.47
		Presupuesto de ejecución material	**8213.28**
13% de gastos generales			1067.72
6% de beneficio industrial			492.79
Suma parcial			9773.79
21% IVA			2052.49
		Presupuesto de ejecución por contrata	**11.826.28**
Honorarios de Ingeniero			
5% sobre presupuesto de ejecución material			410.66
21% IVA sobre honorarios proyecto			86.24
		Total honorarios de Ingeniero	496.90
		TOTAL PRESUPUESTO GENERAL	**12323.18**

Tabla 21 - Cuadro resumen general presupuesto

Anexo A

Especificaciones de hardware TurtleBot 4	
Tamaño (L x W x H)	342 x 339 x 351 mm
Peso	3945 g
Plataforma base	iRobot® Create® 3
Ruedas (Diámetro)	72 mm
Claridad del piso	4.5 mm
Computadora de a bordo	Raspberry Pi 4B 4GB
Velocidad lineal máxima	0,31 m/s en modo seguro, 0,46 m/s sin modo seguro
Velocidad angular máxima	1.90 rad/s
Carga útil máxima	9 kg
Tiempo de operación	2h 30m - 4h dependiendo de la carga
Tiempo de carga	2h 30m
Controlador Bluetooth	Controlador TurtleBot 4
LiDAR	RPLIDAR A1M8
Cámara	OAK-D-Pro
Potencia para el usuario	VBAT @ 300 mA 12V @ 300 mA 5V @ 500 mA 3.3v @ 250 mA
USB	USB 2.0 (Type A) x2 USB 3.0 (Type A) x1 USB 3.0 (Type C) x4
LEDs programables	Anillo de luz LED Create® 3 User LED x2
LEDs de estado	Power LED Motors LED WiFi LED Comms LED Battery LED
Botones e interruptores	Botones de usuario Create® 3 x2 Botón de encendido Create® 3 x1 Botones de usuario x4
Batería	26 Wh iones de litio (14.4V nominal)
Muelle de carga	Incluido

Tabla 22 - Especificaciones hardware TurtleBot 4

Especificaciones eléctricas de la cámara OAK-D-Pro				
Absolute Maximum Ratings				
SYMBOL	**RATINGS**	**MIN**	**MAX**	**UNIT**
$\mathbf{V}_{BUS}$	USB input supply voltage range.[2]	3.5	5.5	V
$\mathbf{I}_{VBUS}$	Maximum input current requirement		2	A
$\mathbf{T}_{stq}$	Ambient temperature	0	60	C

Tabla 23 - Rangos máximos absolutos eléctricos OKA-D-Pro

Especificaciones eléctricas de la cámara OAK-D-Pro					
Recommended Operating Conditions					
SYMBOL	**RATINGS**	**MIN**	**TYP**	**MAX**	**UNIT**
$\mathbf{V}_{BUS}$	VBUS input supply voltage		5V	5.25	V
P	Power consumption requirement	4	6	7.5	W
$\mathbf{P}_{IDLE}$	VBUS idle power draw (Myriad X booted)		2.5		W
$\mathbf{T}_{A}$	Ambient operating temperature			50	°C

Tabla 24 - Condiciones de operación recomendadas OAK-D-Pro

Especificaciones sensores de la cámara OAK-D-Pro	
Center Color Camera	
Parameter	**Value**
Image sensor	Sony IMX378
Active pixels	4056x3040@60fps
Output video format	RAW12/10/8
Focus type	Auto Focus 8cm - ∞ / Fixed Focus 50cm- ∞
FOV	78°
Shutter Type	Rolling shutter
IR sensitive	No

Tabla 25 - Especificaciones sensor del centro color OAK-D-Pro

Especificaciones sensores de la cámara OAK-D-Pro	
Stereo vision grayscale camera	
Parameter	**Value**
Image sensor	OmniVision OV9282
Active pixels	1280x800@120FPS
Output video format	8/10-bit RAW
Focus type	Fixed Focus 19.6cm - ∞
FOV	89.5°
Shutter Type	Global shutter

Tabla 26 - Especificaciones sensor de visión estéreo de escala de grises OAK-D-Pro

Especificaciones iluminación activa de la cámara OAK-D-Pro	
IR dot projector	
Parameter	**Value**
Projector	Dot-Pattern Infrared Illuminator
Projector type	VCSEL (vertical cavity surface emitting laser)
Wavelength	940nm
Control	Using strobe signal from the left stereo camera (PWM)
Compliance	Class 1, IEC 60825-1:2014 Edition 3
FOI (Field of illumination)	HFOI: 61°+/-4°, VFOI: 78°+/-4°
Projector	Dot-Pattern Infrared Illuminator

Tabla 27 - IR dot projector OAK-D-Pro

Especificaciones iluminación activa de la cámara OAK-D-Pro	
IR flood illumination LED	
Parameter	**Value**
Projector	IR Light Emitting Diode
Wavelength	940 nm
Control	Using strobe signal from the left stereo camera (PWM)
Compliance	IEC 62471:2006
FOI (Field of illumination)	FOI: 80°

Tabla 28 - IR flood illumination LED OAK-D-Pro

Especificaciones del adaptador Wi-Fi USB Linux AC650	
Estándares inalámbricos	IEEE 802.11ac, IEEE 802.11a, IEEE 802.11n, IEEE 802.11g, IEEE 802.11b
Frecuencia	5 GHz 2,4 GHz
Velocidad de señal	5 GHz 11ac: hasta 433 Mbps (dinámico) 11n: hasta 150 Mbps (dinámico) 11a: hasta 54 Mbps (dinámico) 2,4 GHz 11n: hasta 150 Mbps (dinámico) 11g: hasta 54 Mbps (dinámico) 11b: hasta 11 Mbps (dinámico)
Sensibilidad de recepción	5 GHz 11a 6 Mbps: -94 dBm 11a 54 Mbps: -78 dBm 11n HT20 MCS0: -94 dBm 11n HT20 MCS0: -77 dBm 11n HT40 MCS0: -92 dBm 11n HT40 MCS7: -74 dBm 11ac VHT80 MCS0: -89 dBm 11ac VHT80 MCS9: -64 dBm 2,4 GHz 11b 1 Mbps: -99 dBm 11b 11 Mbps: -91 dBm 11g 6 Mbps: -94 dBm 11g 54 Mbps: -77 dBm 11n HT20 MCS0: -95 dBm 11n HT20 MCS7: -76 dBm 11n HT40 MCS0: -92 dBm 11n HT40 MCS7: -73 dBm
Modos inalámbricos	Modo Ad-Hoc/Infraestructura
Seguridad inalámbrica	Admite WEP de 64/128 bits, WPA-PSK/WPA2-PSK, 802.1x
Tecnología de modulación	DBPSK, DQPSK, CCK, OFDM, 16-QAM, 64-QAM, 256-QAM
Potencia de transmisión	<20 dBm (PIRE)

Tabla 29 - Especificaciones adptador Wi-Fi USB AC650

Anexo B

Detección de personas: mobilenet.py

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from vision_msgs.msg import Detection2DArray, Detection2D,
ObjectHypothesisWithPose, BoundingBox2D
import depthai as dai
import numpy as np
import cv2
from pathlib import Path
from std_msgs.msg import Header

class mobilenet(Node):
    def __init__(self):
        super().__init__('mobilenet')
        # topics
        self.detection_pub = self.create_publisher(Detection2DArray,
'camera/detections', 10)
        self.image_pub = self.create_publisher(Image, 'camera/image', 10)

        # archivo NN
        nnPath = str(Path.home() / 'ros2_ws/src/oakd_pkg/models/mobilenet-
ssd_openvino_2021.4_6shave.blob')

        # crear pipeline para configurar la cámara
        pipeline = dai.Pipeline()

        # configuración de la cámara
        camRgb = pipeline.create(dai.node.ColorCamera)
        spatialDetectionNetwork =
pipeline.create(dai.node.MobileNetSpatialDetectionNetwork)
        monoLeft = pipeline.create(dai.node.MonoCamera)
        monoRight = pipeline.create(dai.node.MonoCamera)
        stereo = pipeline.create(dai.node.StereoDepth)

        xoutRgb = pipeline.create(dai.node.XLinkOut)
        xoutNN = pipeline.create(dai.node.XLinkOut)
        xoutDepth = pipeline.create(dai.node.XLinkOut)

        xoutRgb.setStreamName("rgb")
        xoutNN.setStreamName("detections")
        xoutDepth.setStreamName("depth")

        # parametros de la cámara
```

```
        camRgb.setPreviewSize(300, 300)
        camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1
080_P)
        camRgb.setInterleaved(False)
        camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)

        monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_
400_P)
        monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
        monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE
_400_P)
        monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)

        stereo.setDefaultProfilePreset(dai.node.StereoDepth.PresetMode.HIGH_D
ENSITY)
        stereo.setDepthAlign(dai.CameraBoardSocket.RGB)
        stereo.setOutputSize(monoLeft.getResolutionWidth(),
monoLeft.getResolutionHeight())
        stereo.setSubpixel(True)

        spatialDetectionNetwork.setBlobPath(nnPath)
        spatialDetectionNetwork.setConfidenceThreshold(0.5)
        spatialDetectionNetwork.input.setBlocking(False)
        spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
        spatialDetectionNetwork.setDepthLowerThreshold(100)
        spatialDetectionNetwork.setDepthUpperThreshold(5000)

        monoLeft.out.link(stereo.left)
        monoRight.out.link(stereo.right)

        camRgb.preview.link(spatialDetectionNetwork.input)
        spatialDetectionNetwork.passthrough.link(xoutRgb.input)
        spatialDetectionNetwork.out.link(xoutNN.input)
        stereo.depth.link(spatialDetectionNetwork.inputDepth)
        spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)

        self.pipeline = pipeline
        self.device = dai.Device(self.pipeline)

        self.qRgb = self.device.getOutputQueue(name="rgb", maxSize=4,
blocking=False)
        self.qDet = self.device.getOutputQueue(name="detections", maxSize=4,
blocking=False)
        self.qDepth = self.device.getOutputQueue(name="depth", maxSize=4,
blocking=False)

        # clases de objetos
        self.labelMap = [
```

```
            "background", "aeroplane", "bicycle", "bird", "boat", "bottle",
"bus", "car", "cat", "chair", "cow",
            "diningtable", "dog", "horse", "motorbike", "person",
"pottedplant", "sheep", "sofa", "train", "tvmonitor"
        ]

        self.timer = self.create_timer(1.0 / 40, self.run_detection)  # 40
FPS

    def run_detection(self):
        inRgb = self.qRgb.tryGet()
        inDet = self.qDet.tryGet()
        inDepth = self.qDepth.tryGet()

        detection_msg = Detection2DArray()
        detection_msg.header = Header()
        detection_msg.header.stamp = self.get_clock().now().to_msg()
        detection_msg.header.frame_id = "oakd_rgb_camera_optical_frame"

        if inRgb is not None and inDet is not None and inDepth is not None:
            frame = inRgb.getCvFrame()
            depthFrame = inDepth.getFrame()

            detections = inDet.detections
            for detection in detections:
                # Solo procesa detecciones de personas
                if detection.label == 15:  # Label 15 es "person" en
MobileNet-SSD
                    bbox = self.frame_norm(frame, (detection.xmin,
detection.ymin, detection.xmax, detection.ymax))
                    if bbox is not None:
                        det = Detection2D()
                        det.bbox = BoundingBox2D()
                        det.bbox.center.position.x = (bbox[0] + bbox[2]) /
2.0
                        det.bbox.center.position.y = (bbox[1] + bbox[3]) /
2.0
                        det.bbox.center.theta = 0.0
                        det.bbox.size_x = float(bbox[2] - bbox[0])
                        det.bbox.size_y = float(bbox[3] - bbox[1])

                        result = ObjectHypothesisWithPose()
                        result.hypothesis.class_id = str(detection.label)
                        result.hypothesis.score = detection.confidence * 100
                        result.pose.pose.position.x =
detection.spatialCoordinates.x / 1000.0  # de mm a m
                        result.pose.pose.position.y =
detection.spatialCoordinates.y / 1000.0  # de mm a m
```

```
                        result.pose.pose.position.z =
detection.spatialCoordinates.z / 1000.0  # de mm a m
                        det.results.append(result)

                        detection_msg.detections.append(det)

                        # dibujar bounding box y label de la detección
                        color = (255, 0, 0)
                        label = self.labelMap[detection.label]
                        cv2.putText(frame, f"{label}:
{result.hypothesis.score:.2f}%",
                                    (bbox[0] + 10, bbox[1] + 20),
cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
                        cv2.putText(frame, f"X:
{result.pose.pose.position.x:.2f} m",
                                    (bbox[0] + 10, bbox[1] + 40),
cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
                        cv2.putText(frame, f"Y:
{result.pose.pose.position.y:.2f} m",
                                    (bbox[0] + 10, bbox[1] + 60),
cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
                        cv2.putText(frame, f"Z:
{result.pose.pose.position.z:.2f} m",
                                    (bbox[0] + 10, bbox[1] + 80),
cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
                        cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2],
bbox[3]), color, 2)

            self.detection_pub.publish(detection_msg)

            # crear topic de la imagen
            image_msg = self.convert_frame_to_image_msg(frame)
            self.image_pub.publish(image_msg)

            # mostrar imagen junto a las detecciones
            cv2.imshow("frame", frame)
            if cv2.waitKey(1) == ord('q'):
                rclpy.shutdown()

    def convert_frame_to_image_msg(self, frame):
        msg = Image()
        msg.header.stamp = self.get_clock().now().to_msg()
        msg.height, msg.width, _ = frame.shape
        msg.encoding = 'bgr8'
        msg.is_bigendian = False
        msg.step = 3 * msg.width
        msg.data = frame.tobytes()
        return msg
```

```
    def frame_norm(self, frame, bbox):
        norm_vals = np.full(len(bbox), frame.shape[0])
        norm_vals[::2] = frame.shape[1]
        return (np.clip(np.array(bbox), 0, 1) * norm_vals).astype(int)

def main(args=None):
    rclpy.init()
    node = mobilenet()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Controlador proporcional (P): mobilenet_Pcontrol.py

```
import rclpy
from rclpy.node import Node
from vision_msgs.msg import Detection2DArray
from geometry_msgs.msg import Twist, Point

class MobilenetPcontrol(Node):

    def __init__(self):
        super().__init__('mobilenet_Pcontrol')

        # Suscripción a mensajes de la cámara
        self.yolo_pose_subscription =
self.create_subscription(Detection2DArray, '/camera/detections',
self.yolo_control_callback, 10)

        # Publicación en cmd_vel del Turtlebot4
        self.cmd_vel_publisher = self.create_publisher(Twist, '/cmd_vel', 10)

        # Publicación topic /PuntoPersona
        self.punto_persona_publisher = self.create_publisher(Point,
'PuntoPersona', 10)

        # Inicialización de variables
        self.x = 0.0
        self.z = 0.0
        self.error = 0.0
        self.id = -1
        self.last_x = None  # Última posición horizontal registrada
        self.detecting = False  # Indicador de detección
```

```
    # Callback cuando llega un mensaje sobre el tema '/camera/detections'
    def yolo_control_callback(self, msg):
        if len(msg.detections) > 0:
            detection = msg.detections[0]
            self.x = detection.results[0].pose.pose.position.x
            self.z = detection.results[0].pose.pose.position.z
            self.id = int(detection.results[0].hypothesis.class_id)

            self.detecting = True
            self.last_x = -self.x  # Registrar la última posición horizontal

            self.get_logger().info(f'X: {self.x}, Z: {self.z}, ID:
{self.id}')
            self.publish_velocity()
            self.publish_punto_persona()
        else:
            self.detecting = False
            self.get_logger().info('No detection')

            # Si no está detectando, girar según el último valor de x
            self.search_for_person()

    def publish_velocity(self):
        msg1 = Twist()
        if self.id == 15:  # persona
            min_distancia = 0.55
            error_lineal = self.z - min_distancia
            error_angular = -self.x

            K_angular = 0.35
            K_lineal = 0.17

            msg1.angular.z = K_angular * error_angular
            msg1.linear.x = K_lineal * error_lineal
        else:
            msg1.angular.z = 0.0
            msg1.linear.x = 0.0

        self.cmd_vel_publisher.publish(msg1)
        self.get_logger().info(f'Published velocities - Linear:
{msg1.linear.x}, Angular: {msg1.angular.z}')

    def publish_punto_persona(self):
        punto_msg = Point()
        punto_msg.x = self.x
        punto_msg.z = self.z
        self.punto_persona_publisher.publish(punto_msg)

    def search_for_person(self):
```

```
        msg1 = Twist()
        if self.last_x is not None:
            # Girar en la dirección del último valor de x registrado
            K_angular = 0.4
            msg1.angular.z = K_angular * self.last_x
        else:
            msg1.angular.z = 0.2  # Gira lentamente a la derecha por defecto

        self.cmd_vel_publisher.publish(msg1)
        self.get_logger().info(f'Searching for person - Angular:
{msg1.angular.z}')

def main(args=None):
    rclpy.init(args=args)
    node = MobilenetPcontrol()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Controlador proporcional variable (PV): mobilenet_PVcontrol.py

```
import rclpy
from rclpy.node import Node
from vision_msgs.msg import Detection2DArray
from geometry_msgs.msg import Twist, Point

class MobilenetPcontrol(Node):

    def __init__(self):
        super().__init__('mobilenet_Pcontrol')

        # Suscripción a mensajes de la cámara
        self.yolo_pose_subscription =
self.create_subscription(Detection2DArray, '/camera/detections',
self.yolo_control_callback, 10)

        # Publicación en cmd_vel del Turtlebot4
        self.cmd_vel_publisher = self.create_publisher(Twist, '/cmd_vel', 10)

        # Publicación TOPIC /PuntoPersona
        self.punto_persona_publisher = self.create_publisher(Point,
'PuntoPersona', 10)

        # Inicialización de variables
```

```
        self.x = 0.0
        self.z = 0.0
        self.error = 0.0
        self.id = -1
        self.last_x = None  # Última posición horizontal registrada
        self.detecting = False  # Indicador de detección

    # Función para calcular K_lineal variable
    def calcular_k_lineal(self, z):
        z_min = 0.20 # distancia mínima
        z_max = 5.0 # distancia máxima
        K_lineal_min = 0.1
        K_lineal_max = 0.5
        a = (K_lineal_max - K_lineal_min) / (z_max - z_min)
        b = K_lineal_min - a * z_min
        return a * z + b

    # Callback cuando llega un mensaje sobre el tema '/camera/detections'
    def yolo_control_callback(self, msg):
        if len(msg.detections) > 0:
            detection = msg.detections[0]
            self.x = detection.results[0].pose.pose.position.x
            self.z = detection.results[0].pose.pose.position.z
            self.id = int(detection.results[0].hypothesis.class_id)

            self.detecting = True
            self.last_x = -self.x  # Registrar la última posición horizontal

            self.get_logger().info(f'X: {self.x}, Z: {self.z}, ID:
{self.id}')
            self.publish_velocity()
            self.publish_punto_persona()
        else:
            self.detecting = False
            self.get_logger().info('No detection')

            # Si no está detectando, girar según el último valor de x
            self.search_for_person()

    def publish_velocity(self):
        msg1 = Twist()
        if self.id == 15:  # persona
            min_distancia = 0.55
            error_lineal = self.z - min_distancia
            error_angular = -self.x

            K_lineal = self.calcular_k_lineal(self.z)
            K_angular = 0.35  # K_angular constante
```

```
            msg1.angular.z = K_angular * error_angular
            msg1.linear.x = K_lineal * error_lineal
        else:
            msg1.angular.z = 0.0
            msg1.linear.x = 0.0

        self.cmd_vel_publisher.publish(msg1)
        self.get_logger().info(f'Published velocities - Linear:
{msg1.linear.x}, Angular: {msg1.angular.z}')

    def publish_punto_persona(self):
        punto_msg = Point()
        punto_msg.x = self.x
        punto_msg.z = self.z
        self.punto_persona_publisher.publish(punto_msg)

    def search_for_person(self):
        msg1 = Twist()
        if self.last_x is not None:
            # Girar en la dirección del último valor de x registrado
            K_angular = 0.35
            msg1.angular.z = K_angular * self.last_x
        else:
            msg1.angular.z = 0.2  # Gira lentamente a la derecha por defecto

        self.cmd_vel_publisher.publish(msg1)
        self.get_logger().info(f'Searching for person - Angular:
{msg1.angular.z}')

def main(args=None):
    rclpy.init(args=args)
    node = MobilenetPcontrol()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Transformada Broadcaster: tf_persona_camara.py

```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import TransformStamped, Point
from tf2_ros import TransformBroadcaster
import numpy as np
import transforms3d.euler as euler
```

```
import transforms3d.quaternions as quaternions

class tf_person_camera(Node):
    def __init__(self):
        super().__init__('tf_person_camera')
        self.tf_broadcaster = TransformBroadcaster(self)
        self.subscription = self.create_subscription(
            Point,
            'PuntoPersona',
            self.listener_callback,
            10
        )
        self.person_position = None

    def listener_callback(self, msg):
        # La posición de la persona en el frame de la cámara
        self.person_position = np.array([
            # coordenadas de un punto (persona) visto desde la cámara
            msg.x, # x distancia horizontal
            0.0,   # y distancia vertical
            msg.z  # z distancia de profundidad
        ])
        self.broadcast_person_transform()

    def broadcast_person_transform(self):
        if self.person_position is None:
            return

        # Rotación en el eje X y luego en el eje Z
        rotation_x = euler.euler2quat(np.radians(90), 0, 0, axes='sxyz')  #
90º en X
        rotation_z = euler.euler2quat(0, 0, np.radians(90), axes='sxyz')  #
90º en Z

        # Combinando las rotaciones
        combined_rotation = quaternions.qmult(rotation_x, rotation_z)

        t = TransformStamped()
        t.header.stamp = self.get_clock().now().to_msg()
        t.header.frame_id = 'oakd_rgb_camera_optical_frame'
        t.child_frame_id = 'person'
        t.transform.translation.x = self.person_position[0]
        t.transform.translation.y = self.person_position[1]
        t.transform.translation.z = self.person_position[2]
        t.transform.rotation.x = combined_rotation[1]
        t.transform.rotation.y = combined_rotation[2]
        t.transform.rotation.z = combined_rotation[3]
        t.transform.rotation.w = combined_rotation[0]
```

```
        self.tf_broadcaster.sendTransform(t)

def main(args=None):
    rclpy.init(args=args)
    node = tf_person_camera()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Transformada Listener: tf _camara_odom.py

```
import rclpy
from rclpy.node import Node
from tf2_ros import TransformListener, Buffer
from geometry_msgs.msg import PointStamped
import tf2_ros

class tf_camara_odom(Node):
    def __init__(self):
        super().__init__('tf_camara_odom')
        self.tf_buffer = Buffer()
        self.tf_listener = TransformListener(self.tf_buffer, self)
        self.timer = self.create_timer(0.05, self.on_timer)

        # topics
        self.person_position_in_odom_publisher =
self.create_publisher(PointStamped, 'PersonPositionInOdom', 10)
        self.baselink_position_in_odom_publisher =
self.create_publisher(PointStamped, 'BaseLinkPositionInOdom', 10)

    def on_timer(self):
        try:
            tf_person = self.tf_buffer.lookup_transform('odom', 'person',
rclpy.time.Time())
            self.publish_transform(tf_person, 'odom', 'person')

            tf_baselink = self.tf_buffer.lookup_transform('odom',
'base_link', rclpy.time.Time())
            self.publish_transform(tf_baselink, 'odom', 'base_link')
        except Exception as e:
            self.get_logger().warn(f'Could not lookup transform: {str(e)}')

    def publish_transform(self, transform, from_frame, to_frame):
        try:
```

```
            point = PointStamped()
            point.header.stamp = transform.header.stamp
            point.header.frame_id = transform.header.frame_id
            point.point.x = transform.transform.translation.x
            point.point.y = transform.transform.translation.y
            point.point.z = transform.transform.translation.z

            self.get_logger().info(f'{from_frame} position in {to_frame}:
{point.point}')

            if from_frame == 'odom' and to_frame == 'person':
                self.person_position_in_odom_publisher.publish(point)
            elif from_frame == 'odom' and to_frame == 'base_link':
                self.baselink_position_in_odom_publisher.publish(point)

        except Exception as e:
            self.get_logger().warn(f'Could not transform {from_frame}
position: {str(e)}')

def main(args=None):
    rclpy.init(args=args)
    node = tf_camara_odom()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Anexo C

En este anexo se detallan los **Objetivos de Desarrollo Sostenible** (ODS) que cumple este proyecto, en línea con la iniciativa global promovida por las Naciones Unidas para abordar los desafíos más apremiantes que enfrenta la humanidad.

Los **Objetivos de Desarrollo Sostenible** constituyen un conjunto de 17 objetivos interconectados diseñados para erradicar la pobreza, proteger el planeta y asegurar la prosperidad para todos. Cada objetivo tiene metas específicas que deben alcanzarse por todos los Estados Miembros de las Naciones Unidas para 2030, y su cumplimiento requiere la participación de gobiernos, empresas, y sociedad civil.

La **Agenda 2030** es un plan de acción para las personas, el planeta y la prosperidad, que también busca fortalecer la paz universal y el acceso a la justicia. La Agenda se basa en 17 ODS y 169 metas, que abarcan una amplia gama de temas que son fundamentales para garantizar un futuro sostenible. La Agenda 2030 hace un llamado a la acción por parte de todos los países, independientemente de su nivel de desarrollo económico, para trabajar en colaboración y construir un futuro más equitativo y sostenible.

Figura 60 - Objetivos de Desarrollo Sostenible. Fuente: ONU

A continuación, se describen los ODS específicos a los que contribuye el proyecto y cómo lo hace.

ODS 4: Educación de Calidad

El desarrollo y la implementación de sistemas de visión artificial con robots como el TurtleBot 4 en un entorno educativo o de investigación fomentan el aprendizaje de tecnologías avanzadas, como la inteligencia artificial y la robótica. Esto mejora la calidad educativa al proporcionar a los estudiantes y profesionales habilidades relevantes para el futuro laboral y académico.

ODS 9: Industria, Innovación e Infraestructura

Este proyecto contribuye al ODS 9 al promover la innovación en el campo de la robótica y la inteligencia artificial. Al desarrollar nuevas aplicaciones de visión artificial y optimizar el uso de hardware y software, como la cámara OAK-D y ROS2, el proyecto fomenta avances tecnológicos que pueden ser aplicados en diversas industrias, mejorando la infraestructura tecnológica y promoviendo el crecimiento económico.

ODS 11: Ciudades y Comunidades Sostenibles

La robótica y la visión artificial tienen un gran potencial para mejorar la calidad de vida en las ciudades, desde la seguridad hasta la eficiencia en el transporte. Un sistema de seguimiento de personas autónomo, como el que se desarrolla en este proyecto, puede tener aplicaciones en monitoreo de seguridad, asistencia a personas con discapacidades o en la gestión de multitudes, contribuyendo a crear comunidades más seguras y sostenibles.

ODS /Valoración	Nada apreciable	Poco apreciable	Efecto apreciable	Notable-mente apreciable	Muy apreciable
1: Fin de la pobreza	X				
2: Hambre cero	X				
3: Salud y Bienestar	X				
4: Educación de calidad					X
5: Igualdad de género		X			
6: Agua y saneamiento	X				
7: Energía Asequible y no contaminante		X			
8: Trabajo decente y crecimiento económico			X		
9: Industria, Innovación e infraestructura				X	
10: Reducción de las desigualdades	X				
11. Ciudades y comunidades sostenibles				X	
12: Producción y consumo responsable		X			
13: Acción por el clima	X				
14: Vida submarina	X				
15: Vida de ecosistemas terrestres	X				
16: Paz, justicia e instituciones sólidas	X				
17: Alianzas para lograr los objetivos	X				

Tabla 30 - ODS reflejados en el proyecto

-