Universidad Distrital Francisco José de Caldas
Facultad Tecnológica
Tecnología en Electrónica Industrial

Implementación del algoritmo de reconocimiento de objetos YOLOv8 para detección de peatones en la plataforma robótica Turtlebot3

Nicolas Rey Ducuara[1], Horacio Enrique Camayo [2]
Facultad Tecnológica, Universidad Distrital Francisco José de Caldas, Colombia

Palabras clave:
Detección de objetos, Procesamiento de imágenes, Robótica, Robots móviles, Visión artificial, ROS.

RESUMEN

En la robótica industrial uno de los aspectos clave es la detección de objetos, en especial para robots móviles, muchas áreas logísticas de la industria tienen un alto flujo de operarios, por lo que es fundamental que estos robots puedan detectarlos para garantizar su seguridad, por otro lado, la detección de peatones sigue siendo un reto debido a la cantidad de variables a analizar, la velocidad de inferencia, oclusión ambiental, entre otros, por tanto, en el siguiente trabajo se presenta la implementación del algoritmo YOLOv8 en la plataforma robótica Turtlebot3 para la detección de peatones y de la señal de tráfico STOP en un entorno controlado, que puede ser aplicado en robots móviles de carga, así como robots multipropósito a nivel logístico. Se obtuvo un sistema autónomo que detecta en tiempo real peatones en un entorno controlado, así como una señal de tráfico de "stop", se implementó además la librería Nav de ROS para la navegación autónoma del robot capaz de fijar destinos, trazar rutas y esquivar obstáculos. Se seleccionó el algoritmo YOLOv8 para la detección dado su velocidad de inferencia y precisión contando con 5 distintos modelos de predicción [1], se ofrece una explicación detallada del proceso de implementación y desarrollo a través de la plataforma robótica y el uso del entorno ROS. El sistema logra una detección en tiempo real con video de 15 FPS a 480p y una velocidad de inferencia de 10ms con el modelo Yolov8n.

Contacto Autores:

[1] Nicolás Rey Ducuara
E-mail: nreyd@udistrital.edu.co

[2] Horacio Enrique Camayo Guegue
E-mail: hecamayog@udistrital,.edu.co

1. INTRODUCCIÓN

La industria de la robótica experimenta su mayor crecimiento en Asia, Europa y América del Norte, siendo Asia el mercado líder a nivel mundial con una participación del 71% en la utilización de robots industriales, así mismo, en países como Colombia la implementación de sistemas autónomos se ha convertido en una necesidad prioritaria para aumentar la productividad y convertirse en un país competitivo en la nueva Industria 4.0 [2], los sistemas robóticos autónomos a través del uso de algoritmos de IA pueden realizar tareas repetitivas y de gran volumen, además un robot móvil debe ser capaz de detectar los objetos a su alrededor y en especial a las personas, así mismo, los algoritmos basados en visión por computadora son utilizados en el diseño de vehículos inteligentes y autónomos [3, 4, 5], por lo cual, existe la necesidad de mejorar constantemente los algoritmos de procesamiento de imágenes para el reconocimiento de objetos [6]. En este sentido, algunas investigaciones proponen la combinación de algoritmos para mejorar la eficiencia en la detección [7, 8], de esta manera se han diseñado y aplicado redes neuronales convolucionales complejas para lograr mayor precisión en tiempo real [9], implementando enfoques que fusionan y dividen los objetos en sub-imágenes que cumplen con las restricciones de tamaño y forma asociadas a los peatones, estas sub-imágenes son entradas para la red neuronal, entrenada en el reconocimiento de peatones [10]. Por otro lado, el algoritmo YOLO ha generado interés debido a la eficiencia en el reconocimiento de peatones en tiempo real de 84.76% de precisión en entornos de tráfico [11], cabe aclarar que el rendimiento del algoritmo depende de la base de datos utilizada para su entrenamiento [12], cualidad especialmente valiosa dado que entre mayor sea la cantidad de imágenes de una persona es posible reconocer extremidades individuales como indicativo de la presencia

de un peatón [13], concluyendo que la velocidad de inferencia de la serie de algoritmos YOLO es más rápida en comparación con otros algoritmos de detección de objetos [14].

La detección de objetos y peatones está ligada al desarrollo de vehículos inteligentes y autónomos, es así que este proyecto está orientado a la industria logística donde se evidencia un gran potencial de automatización, por ejemplo, en los almacenes donde se requieren máquinas de transporte de mercancías la implementación de robots móviles en estos entornos podría mejorar la eficiencia y optimizar procesos que son repetitivos y que no requieren de supervisión constantemente, siendo posible establecer un robot móvil que reemplace la maquinaria de carga, para ello se deben tener en cuenta algunos aspectos importantes como la capacidad de detectar objetos a su alrededor, en especial a peatones, de modo que se garantice su seguridad en el mismo entorno, sin embargo, existen múltiples limitaciones en los sistemas de detección de peatones que podrían afectar el rendimiento y efectividad de la detección, estos sistemas son altamente dependientes de las condiciones del entorno (neblina, oscuridad, opacidad), los obstáculos presentes en el entorno y la capacidad de procesamiento de grandes cantidades de datos por segundo, entre otros factores [15].

Por lo anterior, se implementó el algoritmo de reconocimiento de objetos YOLOv8 en la plataforma robótica Turtlebot3 a través de ROS para la detección de peatones y el reconocimiento de señal de tránsito “Stop” en un entorno controlado, además de mapear el entorno se incorpora con ROS Nav la navegación autónoma, capaz de fijar destinos, trazar rutas y esquivar obstáculos. Estas contribuciones se explican detalladamente en las siguientes secciones que se organizan de esta manera: Sección 2 y 3 se presentan la metodología de investigación, el desarrollo e implementación y en la sección 4 se presentan los resultados obtenidos.

2. METODOLOGÍA DE INVESTIGACIÓN

El aporte del proyecto se basa en el funcionamiento conjunto del algoritmo de reconocimiento de objetos YOLOv8, el sistema de frenado y navegación desarrollado a través de Python y ROS para la detección de peatones y la señal de tránsito “stop” en la plataforma robótica turtlebot3, lo anterior probado en un entorno controlado. A continuación, se presenta una descripción de cada uno de los elementos utilizados.

2.1 Plataforma Robótica Turtlebot

La robótica como campo multidisciplinario involucra características como creación, diseño y operación de plataformas robóticas, refiriéndose al uso mecánico, electrónico y software para realizar tareas de forma autónoma, estos conceptos son parte fundamental en el desarrollo y evolución que han tenido la robótica a través de los años, para el desarrollo de este proyecto se utiliza la plataforma turtlebot3 modelo Burger; siendo este es un robot móvil diferencial que integra para su funcionamiento el entorno de ROS (Robot Operating System) [16, 17].

Tabla 1: Características de la plataforma turtlebot3 modelo Burger

Ítems	Modelo Burger
Velocidad máxima de traslación	0.22 m/s
Velocidad máxima de rotación	2.84 rad/s (162.72 grados/s)
Carga máxima	15kg
Tamaño (largo x ancho x alto)	138mm x 178 mm x 192 mm
Peso (SBC + Batería + sensores)	1kg
Tiempo de operación esperado	2h 30m
SBC (ordenadores de placa única)	Raspberry pi 3
LDS (sensor de distancia laser)	Sensor Láser LDS 01 360 con rango de 3.5 metros
IMU	Giroscopio 3 ejes, Acelerómetro 3 ejes
Periféricos	UART x3, puede x1, SPI x1, I2C x1, ADC x5, 5 pines OLLO x4
Botones e interruptores	Botones pulsadores x2, botón de reinicio, interruptor DIP x2
Batería	Polímero Litio 11.1V, 1800 mAh/19.98Wh 5C
Pines de expansión	GPIO 18 pines, Arduino 32 pies

La tabla 1 muestra las características y parámetros de la plataforma robótica turtlebot3, que se tuvieron en cuenta al momento de realizar el sistema. Destacando parámetros como el rango sensor laser LDS, velocidad máxima de traslación y velocidad máxima de rotación.

2.2 ROS (Robot Operating System)

Es una plataforma con herramientas, bibliotecas, servicios de abstracción de hardware, control de dispositivos de bajo nivel, mensajes entre procesos y mantenimiento de paquetes para el desarrollo de software de robots, es de código libre en su mayor porcentaje orientado para sistemas UNIX (Linux en su distribución de Ubuntu) [18], para el desarrollo del proyecto se pueden resaltar las siguientes características:

- Nodo principal de coordinación, comunicación entre nodos que permite un eficiente funcionamiento de las diferentes partes del robot pueden trabajar de forma independiente a través de una eficaz distribución y recolección de datos por medio de sus interfaces, para el desarrollo del proyecto se usa una computadora con Ubuntu 20.04 donde se ejecuta el nodo central y una raspberry pi 4 con Ubuntu server 20.04 que actúa nodo auxiliar y coordinador de los datos sensados del robot.
- Publicación y subscripción de mensajes y flujos de datos, métodos por el cual se comunican los diferentes actuadores para recolectar o mostrar información, el robot cuenta con múltiples subscriptores y publicadores, entre estos, Twist que controla el movimiento de los motores, Scan que obtiene los datos del sensor láser LDS, además se crea un publicador y subscriptor para la obtención de la imagen de la cámara y la publicación de inferencias del algoritmo.
- Creación, visualización y destrucción de nodos, que permite la recolección de datos y ejecución de procesos en tiempo real, facilitando un mejor entendimiento de cómo funcionan los sistemas robóticos y así poder resolver problemas. Para este sistema se crearon 2 nodos principales: para el sistema de frenado del robot y para detectar peatón y señal de stop, publican datos visualizados por los tópicos de nodos de ROS.
- Nodos multiprocesos, permite un manejo simultaneo de los diferentes procesos ya que se pueden ejecutar los nodos de forma independiente para cada una de las tareas que se desee realizar, de la tal manera que trabajan al mismo tiempo, por ejemplo, si alguno de ellos no se está ejecutando el sistema puede trabajar de manera parcial, esto es conveniente para realizar testeos y pruebas de manera que se facilite el entorno de trabajo.

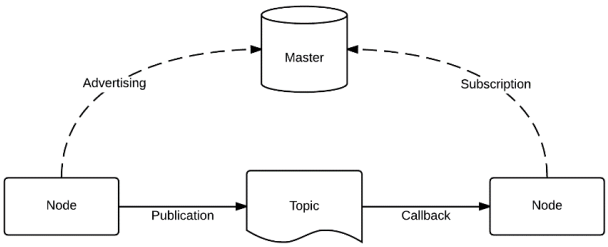


Figura 1. Estructura funcionamiento de ROS tomado de Wikipedia Commons: https://upload.wikimedia.org/wikipedia/commons/e/e7/ROS-master-node-topic.png

En la Figura 1 se observa el diagrama general de los nodos básicos generados por ROS y la forma en que se comunican mediante los tópicos. Cuando se ejecuta un nodo, este puede crear subscriptores y publicadores que son los que envían o reciben datos mediante funciones de Callback, estos datos se actualizan en los tópicos que son accesibles desde cualquier otro nodo.

- Navegación autónoma, ROS cuenta con un paquete de librerías para la navegación autónoma de robots diferenciales, en este caso para Turtlebot3. Se mapea el entorno controlado a través de ROS Slam y se usa ROS Nav para la navegación autónoma en 2D tomando datos de odometría y el sensor láser, permitiendo además fijar objetivos, esquivar obstáculos y trazar rutas de destino, se configuran los

parámetros de local map, global map, local costmap, global costmap y local planner para la versión Burger de Turtlebot3 con el fin de obtener el mejor rendimiento.

- Simulaciones, está plataforma es muy popular en el sentido de que se integra de manera precisa y eficiente con los diferentes simuladores, entre los más conocidos y utilizados RVIZ y GAZEBO permitiendo a un desarrollador realizar pruebas y depurar código de forma virtual antes de implementar de manera física [19].

2.3 Algoritmo de detección de objetos YOLO v8 (You Only Look Once):

Es la versión más reciente de YOLO (You Only Look Once) desarrollado por Ultralytics, es un modelo de detección de objetos en tiempo real y segmentación de imágenes. Esta nueva versión se basa en los avances de las versiones anteriores y presenta mejoras significativas en el rendimiento y la flexibilidad en diversas tareas de visión artificial, como detección, segmentación, estimación de poses, seguimiento y clasificación, Yolov8 tiene una alta tasa de precisión medida por COCO y Roboflow 100, logra una precisión en su modelo YOLOv8m de 50.2% de mAP (values are for single-model single-scale) [20].

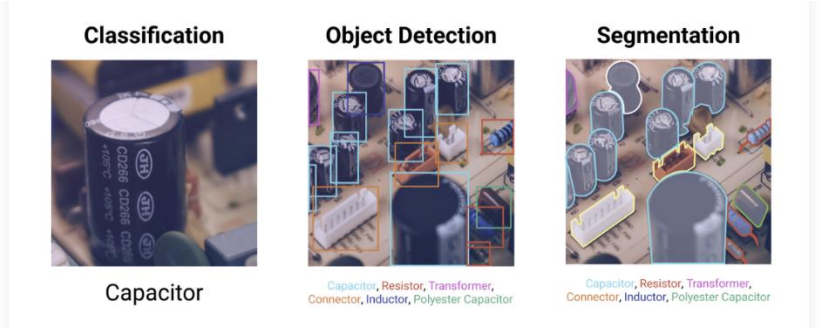


Figura 2. Funciones del algoritmo Yolov8 tomada de Augmented Startup Computer Vision, AI, Drones: https://kajabi-storefronts-production.kajabi-cdn.com/kajabi-storefronts-production/file-uploads/blogs/22606/images/a035457-538d-83df-e6f4-2a7502ced5e_1673353303009.png, (a) Clasificación de clases de objetos, (b) Predicción y detección de objetos, (c) Segmentación de imágenes.

En la Figura 2 se observa las diferentes tareas que realiza el algoritmo Yolov8, (a) La red neuronal dedicada para la clasificación de imágenes que determina una clase de objeto en una imagen y devuelve su nombre y la probabilidad de esta predicción, por otro lado (b), la red neuronal dedicada a la detección de objetos devuelve la probabilidad de la predicción, clase de objeto y coordenadas del recuadro que encapsula al objeto (top, bottom, right, left), por último (c), en la segmentación se detecta el contorno de los objetos. Para el desarrollo del sistema de detección de peatones se utiliza la red neuronal específica de detección y predicción de objetos, que además puede detectar múltiples objetos al mismo tiempo.

3. DESARROLLO E IMPLEMENTACIÓN

El sistema implementado hace uso de la plataforma turtlebot3 y del algoritmo de reconocimiento de objetos YOLOv8, además de las librerías de navegación de ROS, la implementación de este algoritmo se realizó siguiendo la estructura de ROS, creando los nodos, tópicos, publicadores y subscriptores necesarios. YOLO detecta, evalúa y establece la probabilidad de coincidencia basándose en el dataset COCO que cuenta con 80 diferentes clases de objetos, obtenidas las inferencias del algoritmo, se desarrollaron los sistemas de detección de peatones y la señal de tránsito “Stop”, el sistema de frenado y de navegación sobre la plataforma robótica. El diagrama del sistema desarrollado se presenta a continuación:

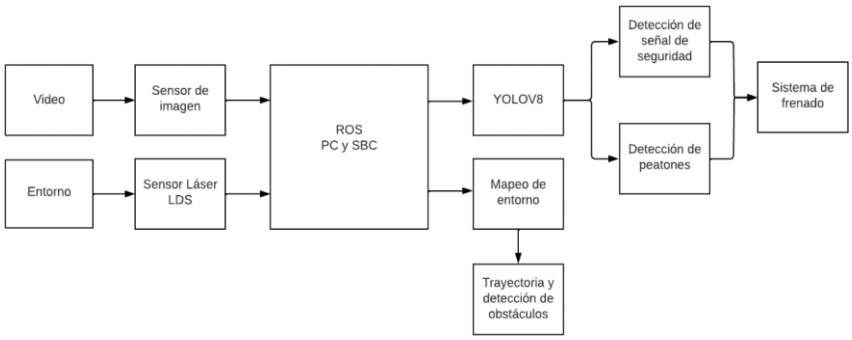


Figura 3. Diagrama de bloques del sistema.

3.1 Detección de objetos, peatones y señal de "Stop"

La detección de peatones y de señales de seguridad es unos de los factores más importantes en un sistema robótico autónomo, por lo que se utilizó una cámara que transmite imágenes en tiempo real que son procesadas por el algoritmo implementado emitiendo inferencias cada 10ms usando el modelo Yolov8n, el framerate del vídeo de 15 FPS permite disminuir la latencia y el tiempo de retraso en el proceso de envío de las imágenes desde las raspberry hasta el PC. Para la publicación de las inferencias se crearon 2 objetos de la librería msg de ROS uno para la publicación de la clase de objeto y sus coordenadas y otro para la publicación de imagen, de esta manera se crean los tópicos accesibles por los demás nodos que se estén ejecutando, además de los publicadores de cada tópico.

Algoritmo 1. Detección de objetos implementando Yolov8 en ROS:

Creación de subscriptores, publicadores y variables iniciales:

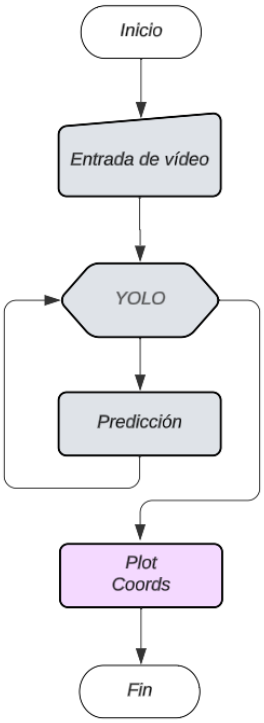
```
def __init__(self):
    self.model = YOLO('yolov8n.pt')
    self.inference = Inference()
    self.inference_result = InferenceResult()
```

Función de predicción del modelo Yolov8n:

```
def camera_callback(self, data):
    img = CvBridge.imgmsg_to_cv2(data, "bgr8")
    self.results = self.model.predict(img)
    for r in self.results:
        boxes = r.boxes
        for box in boxes:
            b = box.xyxy[0]
            c = box.cls
            class_name = self.model.names[int(c)]
            if class_name  == 'stop sign' or class_name == 'person':
                self.inference_result.class_name
                self.inference_result.left = int(b[0])
                self.inference_result.top = int(b[1])
                self.inference_result.right = int(b[2])
                self.inference_result.bottom = int(b[3])
```

Publicación de inferencias e imágenes con los objetos encapsulados:

```
annotated_frame = self.results[0].plot()
    img_msg = bridge.cv2_to_imgmsg(annotated_frame)
    self.img_pub.publish(img_msg)
    self.yolov8_inference.yolov8_inference.append(self.inference_result)
    self.yolov8_pub.publish(self.yolov8_inference)
    self.yolov8_inference.yolov8_inference.clear()
```



En el algoritmo 1 muestra el funcionamiento del modelo Yolov8n implementado, en gris se encuentra la función de predicción del modelo, que realiza la iteración en bucles anidados de las imágenes de entrada: El primero para la obtención de los resultados de predicción del modelo y el segundo que obtiene las coordenadas y la clase de objeto en una lista. En lila se observa la publicación de los resultados en formato de texto (Publicador InferenceResult) e imagen (Publicador Inference).

3.2 Sistema de frenado

En el sistema de frenado se utilizaron los tópicos cmd_vel de *Twist* y scan de *LaserScan*, además de los publicadores creados para las inferencias de YOLOv8, su funcionamiento es el siguiente: El subscriptor de YOLO recibe las inferencias del algoritmo directamente del publicador InfereceResult del sistema de detección de objetos; si se detecta un peatón el robot determina la distancia a la que se encuentra (scan obtiene un vector de 360 posiciones cada segundo con los datos de distance del sensor láser), si es inferior a 0.5 metros se detiene hasta que el peatón se retire, por otro lado, si se reconoce la señal de stop, se calcula la distancia a la que se encuentra utilizando las coordenadas (top, bottom, right, left) y haciendo una aproximación del grado sexagesimal del sensor para obtener la distancia de la señal de stop, si la señal está a 0.75 metros el robot continúa su marcha hasta estar próximo a la señal, de tal forma que el robot frene durante 5 segundos y continúe su camino si no detecta un peatón.

Algoritmo 2. Sistema de frenado

Creación de subscriptores, publicadores y variables iniciales:

```
def __init__(self):
    self.publisher_control    (Twist)
    self.subscriber_laser     (Scan)
    self.subscriber_yolo      (YOLO)
    self.stop = 1
```

Función de frenado:

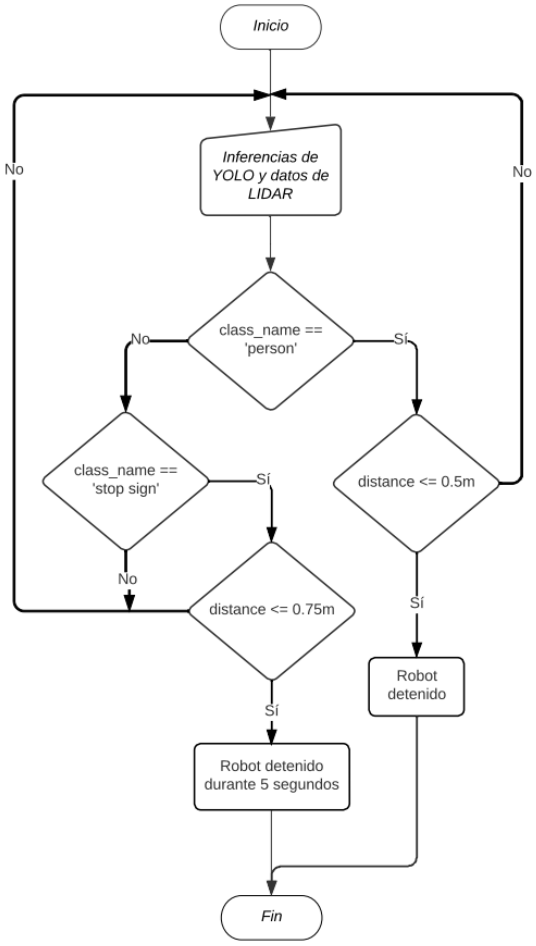
```
def stoprb3(self):
    twist = Twist()
    twist.linear = 0.0
    twist.angular = 0.0
    self.publisher_control(twist)
```

Función distancia de objeto:

```
def object_distance(self):
    laserscan = [10,5,0,359,354,349]
    for i in laserscan:
        distance = self.scan_ranges[i]
        self.objetct_detect(distance)
```

Función de detección de objetos:

```
def object_detect(self, distance):
    object = self.class_name
    if object == 'person':
      if 0.0 < distance <= 0.5:
          self.stoptb3()
    elif object = 'stop signal':
      if 0.0 < distance <= 0.75:
          if self.stop == 1:
             time_init = time.time()
             while True:
                self.stoptb3()
                self.person_detect()
                time_end = time.time()
                if time_end - time_init >= 5:
                   break
             self.stop = 0
    else:
      self.stop = 1
```



3.3 Navegación autónoma

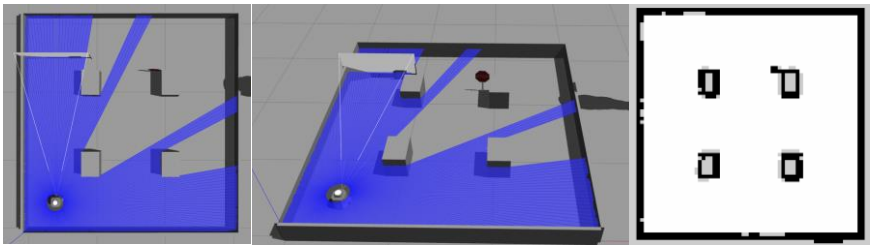
La navegación es el núcleo central para un robot móvil autónomo, fijar un destino, trazar rutas y esquivar obstáculos son funciones que brinda ROS con su paquete de librerías NAV, para la implementación de la navegación autónoma se siguió el siguiente procedimiento: En primer lugar, se mapeó el entorno previamente realizado con ROS Slam, el mapa se muestra en el simulador RVIZ una vez se ejecuta el nodo de la Navegación y sirve como guía para la posición inicial y final del robot, además RVIZ dispone de una interfaz que facilita la navegación visualizando el entorno y controlando la posición del robot en tiempo real. Algunos

parámetros de Nav2 se modificaron para obtener mejor rendimiento en la navegación, como la velocidad de desplazamiento, la velocidad angular y la actualización de lectura de datos, pues se presentaron algunos inconvenientes por pérdida de datos que ocurrían por factores como la velocidad de conexión a internet y la distancia del robot al computador. Los parámetros modificados fueron los siguientes: dwa_local_planner_params_burger, global_costmap_params y local_costmap_params.

Dwa_local_planner_params	**Local y global costmap params**
max_vel_x: 0.1 m/s min_vel_x: -0.1 m/s max_vel_y: 0.0 min_vel_y: 0.0 acc_lim_x: 1.0 m/s2 acc_lim_y: 0.0 m/s2 max_vel_trans: 0.1 m/s min_vel_trans: 0.1 m/s max_vel_theta: 1.0 rad/s min_vel_theta: 0.5 rad/s acc_lim_theta: 1.0 rad/s2	update_frequency: 15.0 Hz publish_frequency: 15.0 Hz transform_tolerance: 1.0 seconds

3.4 Simulación

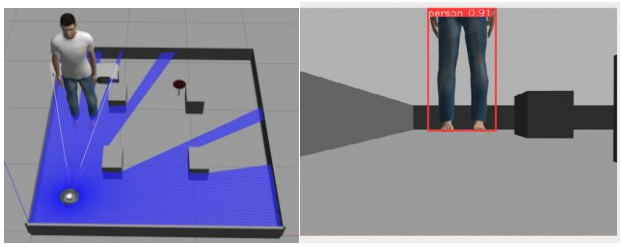
Robotics ofrece paquetes de simulación de las plataformas robóticas que ha lanzado al mercado, incluyendo Turtlebot3 Burger, con GAZEBO y RVIZ se crea un entorno controlado, se mapea usando ROS Slam y se ejecutan los nodos del sistema de detección y del sistema de frenado realizando algunas modificaciones para la fuente de video que se provee de un pluggin de simulación de cámara para Gazebo, además se importan los modelos en 3D de una persona y una señal de stop en Gazezo:



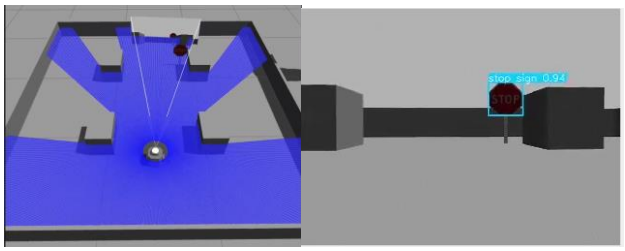
(a) (b)

Figura 4. (a) Entorno creado en Gazebo y (b) mapa obtenido con ROS Slam.

En la Figura 4 (a), se observa el diseño del entorno en Gazebo con dimensiones de 3m x 3m, 4 obstáculos con dimensiones 25cm de alto, 35cm de largo y 25 cm de ancho, en (b) se observa el mapa obtenido después de mapear en entorno con ROS Slam, para ello se recorrió el mapa en su totalidad con el robot utilizando el nodo de teleoperación para manejarlo con el teclado, los tópicos de Scan (Datos del sensor láser), TF (Transformación de datos de odometría) y Map (Asigna un valor de 0 a 100 a los bloques de espacio que identifican si existe o no algo en ese espacio, siendo 0 el color negro lo que indica que hay algún objeto y 100 el color blanco que indica que este espacio está vació) generan un mapa en 2D que sirve como referencia para el desplazamiento y ubicación del robot.



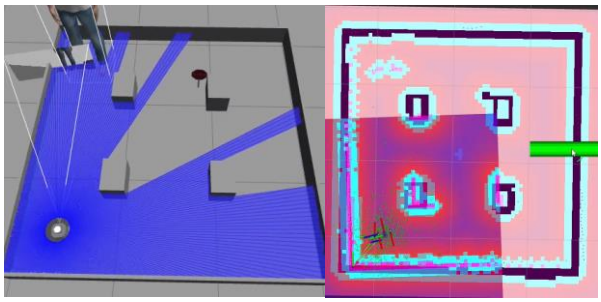
(a) (b)



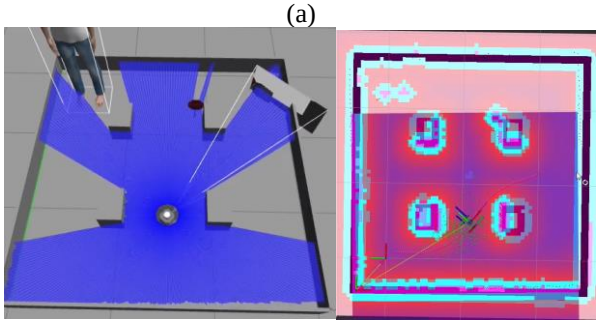
(c) (d)

Figura 5. (a) Persona en el entorno controlado en Gazebo, (b) Detección de persona YOLOv8, (c) Señal de Stop en el entorno en Gazebo y (d) Detección de señal de Stop YOLOv8.

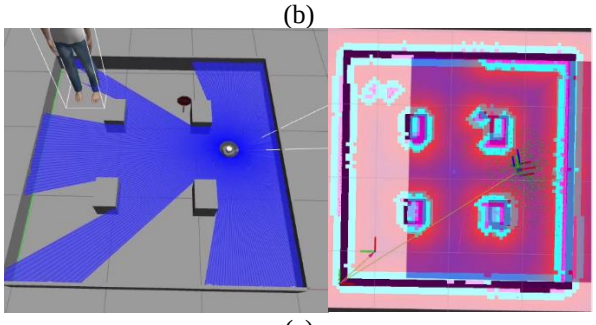
La figura 5(a) muestra el entorno de simulación con un peatón a 1.5 metros del robot, en (b) se muestra la detección del peatón por medio de la cámara del robot, cabe destacar que el reconocimiento del peatón se realizó con solo las extremidades, un punto muy fuerte del algoritmo, ya que no es necesario mostrar al peatón en su totalidad, en (c) se muestra el entorno de simulación con la señal de tránsito "STOP" a 1.5 metros de distancia del robot y en (d) se muestra la detección de la señal de tránsito "STOP" por medio de la cámara del robot.



(a)



(b)



(c)

Figura 6. Navegación autónoma: (a) Posición inicial y punto de destino (flecha verde), (b) Robot a medio camino, (c) Robot en el destino.

En la figura 6 se realiza un trazado de ruta para observar la navegación fijando un punto de destino y visualizando el desplazamiento del robot: En (a) se observa en entorno completo y con el entorno de RVIZ se fija un destino (flecha verde)**,** una vez se fija el destino ROS Nav traza la ruta más cercana al punto final e

inmediatamente comienza a desplazarse el robot, en (b) el robot está a medio camino y en (c) el robot llega a su destino tomando la dirección de la flecha verde inicial.

4. RESULTADOS Y ANÁLISIS

El sistema implementado sobre la plataforma robótica turtlebot3 modelo Burger comprendido por la detección de peatones y la detección de la señal de "stop", sistema de frenado y la navegación autónoma fue probado en el entorno controlado construido de dimensiones 3x3 metros, a continuación, se presentan los resultados obtenidos durante las pruebas:

4.1 Detección de objetos, peatones y señal de "Stop"

Con base en el diseño del entorno de Gazebo (Figura 4) se realizó a escala un entorno controlado hecho con cartón y la señal de stop con madera:

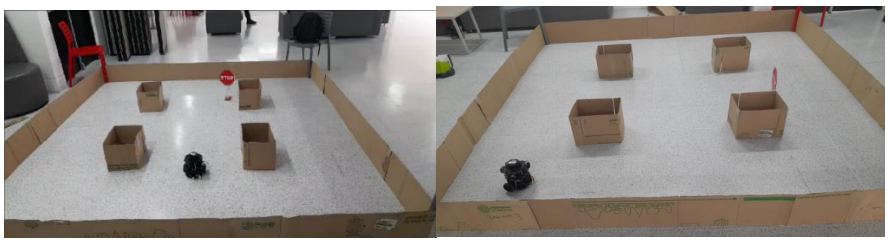
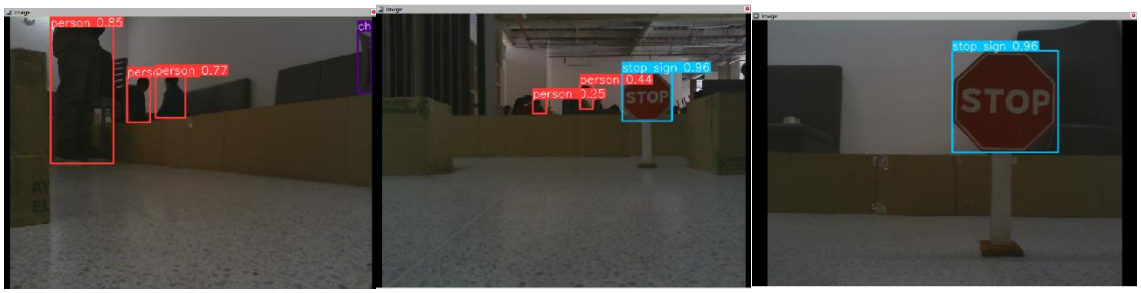


Figura 7. Entorno controlado realizado.

La figura 7 muestra el entorno construido con dimensiones 3x3 metros y 4 obstáculos con dimensiones 25cm de alto, 35cm de largo y 25 cm de ancho, todo esto realizado con cartón, además la señal de STOP a escala del robot hecha con madera.

Con el entorno finalizado, se realizaron pruebas del sistema de detección, observando y publicando las inferencias del algoritmo:



(a) (b) (c)

Figura 8. (a) Detección de varias personas, (b) Detección de personas y señal de stop, (c) Detección de señal de stop.

La figura 8(a) muestra la inferencia del algoritmo YOLOv8 en presencia de varias personas arrojando la probabilidad de acierto y un recuadro en cada una de ellas, obsérvese que YOLO arroja múltiples inferencias por fotograma, además véase que la iluminación es baja en esta imagen, por lo que YOLOv8 tiene una alta capacidad predecir la presencia de una clase de objeto incluso con factores ambientales adversos como la oclusión ambiental y poca iluminación, en (b) se observa las 2 clases de objetos que se están trabajando para este proyecto detectándose al mismo tiempo, YOLO puede detectar las clases de objetos con las que está preentrenado el modelo (COCO cuenta con 80 clases diferentes) todas al mismo tiempo en una imagen, por otro lado YOLO puede predecir la clase de objeto solo con algunos pixeles como se observa en detección de las personas y (c) se observa la detección de la señal de stop en primer plano.

4.2 Sistema de frenado

En este sistema se publican las inferencias de YOLOv8 y se obtiene la distancia a la que se encuentra la clase detectada: si es un peatón, cuando la distancia es inferior a 0.5 metros el robot que está en marcha se detiene, cuando detecta la señal de STOP a 0.75 metros, continúa el robot su marcha y cuando se acerca a la señal se detiene durante 5 segundos.

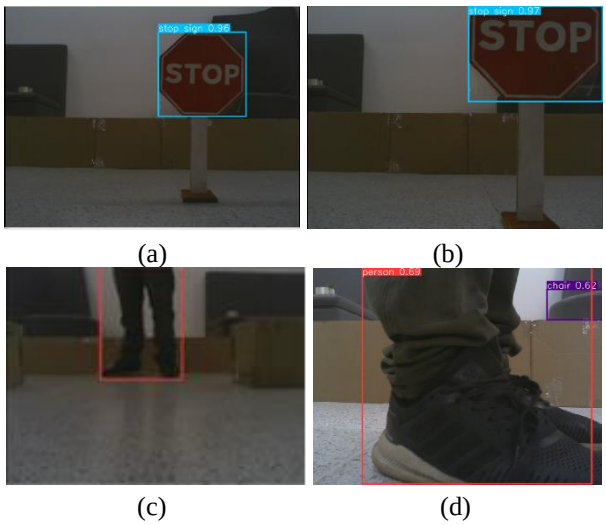


Figura 9. (a) Señal de stop detectada a 0.75 metros, (b) señal de stop a 0.56 metros. (c) Persona detectada a 0.85 metros, (d) persona detectada a 0.48 metros.

En la Figura 9: (a) se observa la detección de la señal stop a 0.75 metros, el robot está en marcha todavía, hasta que en (b) la distancia a la señal de stop es de 0.58 metros, momento en el que se detiene durante 5 segundos, obsérvese que YOLO detecta la señal incluso sin estar completa, es decir, el algoritmo es capaz de predecir la clase de objeto observando sólo la mitad del mismo, en (c) la persona es detectada a 0.85 metros, el robot sigue su camino hasta que en (d) la persona es detectada a 0.48 metros, el robot se detiene inmediatamente, obsérvese también que en este caso se YOLO predice la presencia de una persona sólo observando la parte inferior de las extremidades inferiores, esto demuestra una vez más lo potente y preciso que es este algoritmo.

4.3 Navegación autónoma

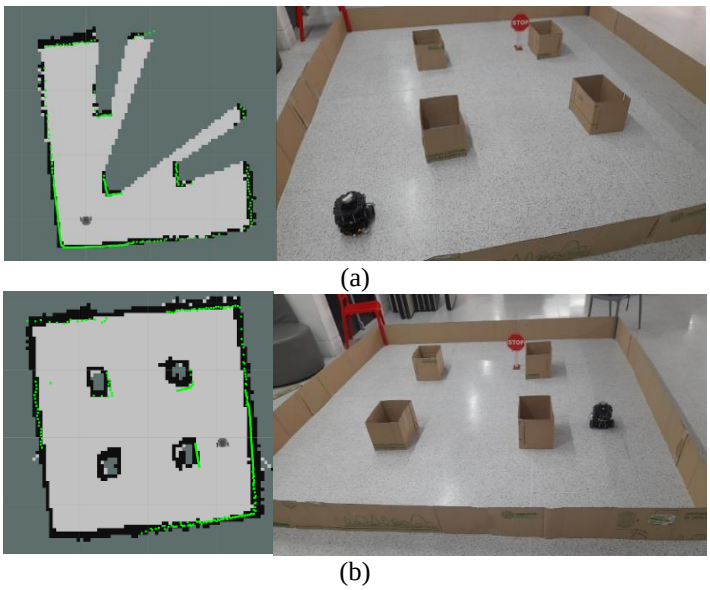
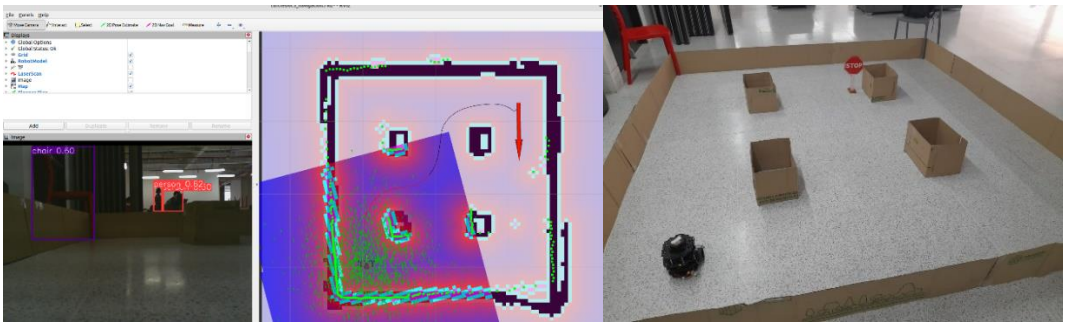


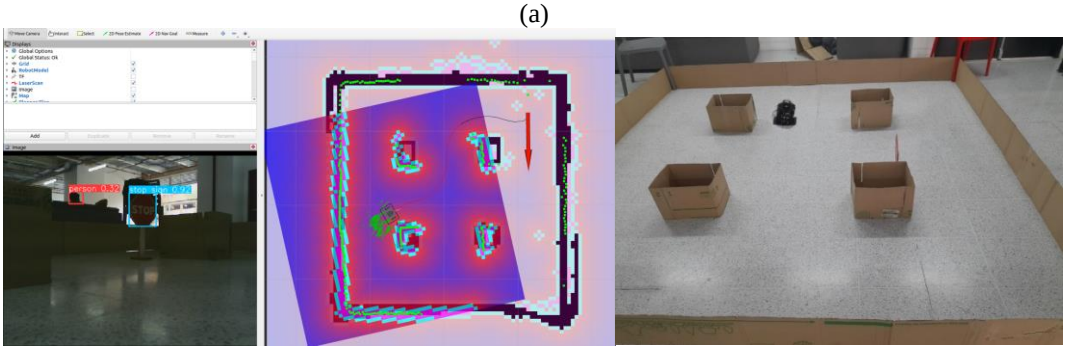
Figura 10. Mapeo del entorno con ROS SLAM.

En la Figura 10 (a) se observa el proceso de mapeo inicialmente, se utilizó un nodo de teleoperación dispuesto para controlar el robot utilizando ROS Slam y RVIZ para recorrer cada esquina del entorno, en (b) se observa el mapa generado a través de Slam en 2D usando el sensor láser LIDAR y odometría, este se muestra en RVIZ y permite el control de la posición del robot en la navegación autónoma, sirviendo como guía.

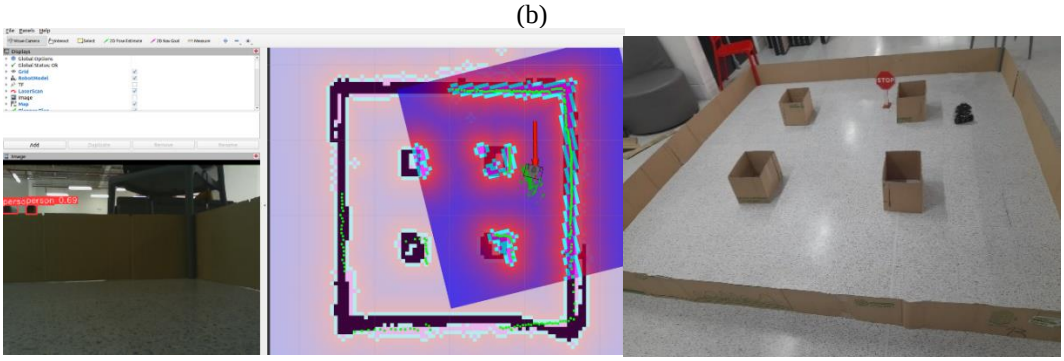
Una vez se obtuvo el mapeo se configura ROS Nav, en esta interfaz se puede fijar un punto de destino (flecha roja), se traza una ruta e inmediatamente el robot comienza su marcha.



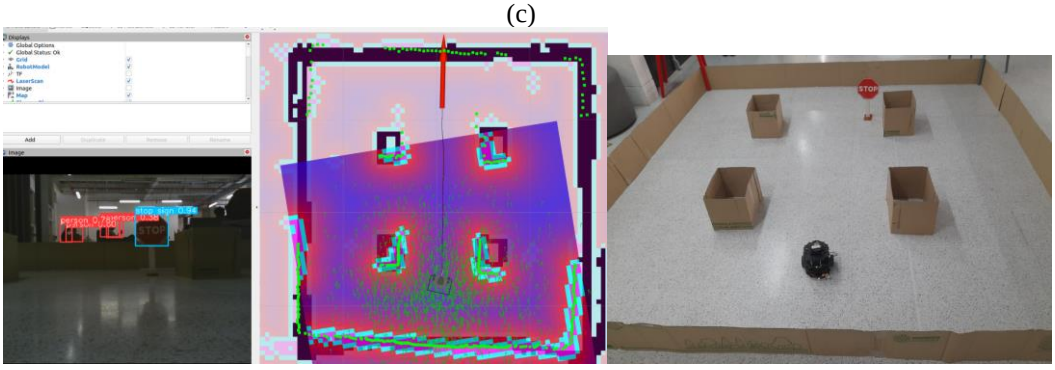
(a)



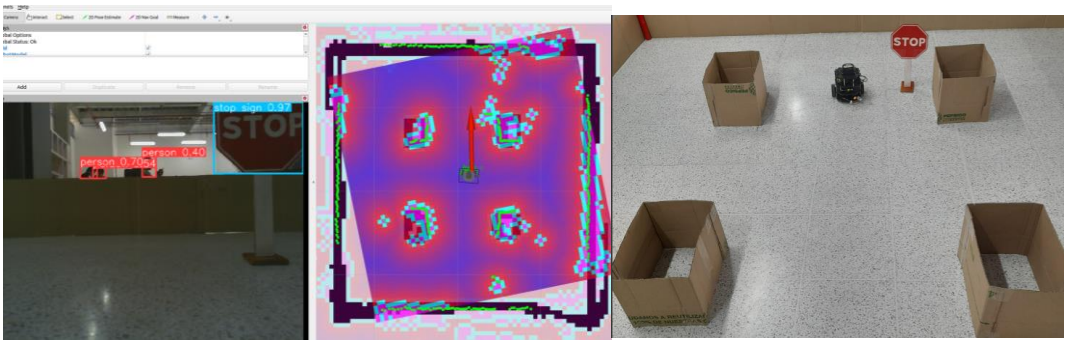
(b)



(c)



(d)



(e)

Figura 11. Navegación de ROS Nav: (a) Punto de Inicio con destino Fijado (flecha roja), (b) Robot en el destino.

En la figura 11 (a) se observa al robot en la posición inicial, el RVIZ se fija el destino al que se va a dirigir el robot (flecha roja), Ros Nav realiza el trazado de ruta y el robot comienza su marcha, en (b) se muestra el desplazamiento del robot siguiendo la ruta trazada inicialmente, luego en (c) el robot llega a su destino posicionándose en dirección a la que apunta la flecha. Por otro lado, en (d) se establece otra posición inicial en el camino central y se fija el destino para que vaya en dirección recta para encontrarse con la señal de "stop", el (d) el robot detectó la señal de stop a 0.75 metros, se acerca un poco más y se detiene durante 5 segundos.

5. CONCLUSIONES

La implementación del sistema de frenado constituye un gran porcentaje del trabajo realizado dado que aquí se obtienen los datos de los diferentes tópicos como el LaserScan, Twist y las inferencias de YOLOv8 y, se controla el movimiento del robot con base en todos los datos anteriores.

ROS ofrece una amplia gama de paquetes de librerías para la simulación y creación de entornos permitiendo visualizar y ejecutar los nodos de reconocimiento y frenado, además de los nodos de Turtlebot3 en RVIZ y GAZEBO como se observa en las figuras 4, 5 y 6, obteniendo una simulación que ejecuta el sistema en su totalidad gracias al gran desarrollo de este apartado por la comunidad de ROS.

El algoritmo de detección de objetos YOLOv8n se adaptó para el reconocimiento de peatones y la señal de "stop" y gracias al dataset COCO con el cual esta preentrenado el algoritmo, se puede detectar personas desde casi cualquier ángulo, incluso con baja luminosidad, se destaca la característica de reconocer extremidades como indicativos de una persona y predice con muy alta eficiencia la clase de objeto, es decir, no fue necesario que la cámara captara en su totalidad al peatón ni la señal de stop para evaluar la condición de que el robot se detenga, como se observa en las figuras 8 y 9.

Como se puede observar en la figura 11 el sistema es capaz de trazar una ruta desde el punto inicial preestablecido hasta el destino que se desee, en este trayecto esquiva obstáculos y reconoce los peatones y la señal de "stop" cumpliendo las condiciones asignadas; los obstáculos los esquiva o frena si están muy cerca; cuando hay un peatón a menos de 0.5 metros el robot se detiene por completo y cuando hay una señal de tránsito el robot se detiene durante 5 segundos, si no hay peatones continua hasta llegar al destino fijado, demostrando así funcionamiento del sistema implementado.

REFERENCIAS

[1] Ultralytics, "Ultralytics Yolov8", [En línea], Disponible en: https://docs.ultralytics.com/
[2] IFR, "IFR presents World Robotics 2021 reports", [online], Disponible en: https://ifr.org/ifr-press-releases/news/robot-sales-rise-again
[3] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, y S. Tubaro, "Deep Convolutional Neural Networks for pedestrian detection", Signal Process. Image Commun., vol. 47, pp. 482-489, sep. 2016, doi: 10.1016/j.image.2016.05.007.
[4] D. Liu, S. Gao, W. Chi, y D. Fan, "Pedestrian detection algorithm based on improved SSD", Int. J. Comput. Appl. Technol., vol. 65, n.o 1, pp. 25-35, ene. 2021, doi: 10.1504/IJCAT.2021.113643.

[5] H. Elzein, S. Lakshmanan, y P. Watta, “A motion and shape-based pedestrian detection algorithm”, en IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683), jun. 2003, pp. 500-504. doi: 10.1109/IVS.2003.1212962.
[6] D. Liu, S. Gao, W. Chi, y D. Fan, “Pedestrian detection algorithm based on improved SSD”, Int. J. Comput. Appl. Technol., vol. 65, n.o 1, pp. 25-35, ene. 2021, doi: 10.1504/IJCAT.2021.113643.
[7] C. Ning, L. Menglu, Y. Hao, S. Xueping, y L. Yunhong, “Survey of pedestrian detection with occlusion”, Complex Intell. Syst., vol. 7, n.o 1, pp. 577-587, feb. 2021, doi: 10.1007/s40747-020-00206-8.
[8] H. Ramezani, H. ZakiDizaji, H. Masoudi, y G. Akbarizadeh, “A new DSWTS algorithm for real-time pedestrian detection in autonomous agricultural tractors as a computer vision system”, Measurement, vol. 93, pp. 126-134, nov. 2016, doi: 10.1016/j.measurement.2016.06.067.
[9] J. Cao et al., “Pedestrian Detection Algorithm for Intelligent Vehicles in Complex Scenarios”, Sensors, vol. 20, n.o 13, Art. n.o 13, ene. 2020, doi: 10.3390/s20133646.
[10] C.-W. Zhang, M.-Y. Yang, H.-J. Zeng, y J.-P. Wen, “Pedestrian detection based on improved LeNet-5 convolutional neural network”, J. Algorithms Comput. Technol., vol. 13, p. 1748302619873601, ene. 2019, doi: 10.1177/1748302619873601.
[11] L. Zhao y C. E. Thorpe, “Stereo- and neural network-based pedestrian detection”, IEEE Trans. Intell. Transp. Syst., vol. 1, n.o 3, pp. 148-154, sep. 2000, doi: 10.1109/6979.892151.
[12] K. Saleh, M. Hossny, y S. Nahavandi, “Real-time Intent Prediction of Pedestrians for Autonomous Ground Vehicles via Spatio-Temporal DenseNet”. arXiv, 22 de abril de 2019. doi: 10.48550/arXiv.1904.09862.
[13] B. Montenegro y M. Flores, “Detección de peatones en el día y en la noche usando YOLO-v52, Ingenius, n.o 27, Art. n.o 27, ene. 2022, doi: 10.17163/ings.n27.2022.08.
[14] D. Varytimidis, F. Alonso-Fernandez, B. Duran, y C. Englund, “Action and intention recognition of pedestrians in urban traffic”. arXiv, 23 de octubre de 2018. doi: 10.48550/arXiv.1810.09805.
[15] Y. Li, K. Yin, J. Liang, C. Wang, y G. Yin, “A Multi-task Joint Framework for Real-time Person Search”. arXiv, 11 de diciembre de 2020. doi: 10.48550/arXiv.2012.06418.
[16] K. Saleh, M. Hossny, y S. Nahavandi, “Real-time Intent Prediction of Pedestrians for Autonomous Ground Vehicles via Spatio-Temporal DenseNet”. arXiv, 22 de abril de 2019. doi: 10.48550/arXiv.1904.09862.
[17] Robotics e-Manual, “Manual Turtlebot3”, nombre de la editorial, 2017. [online]. Disponible en https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
[18] C. Capera Cuellar, “Manual de usuario del robot móvil turtlebot3 modelo Burger”. 2020. [online]. Disponible en: https://repository.udistrital.edu.co/bitstream/handle/11349/24883/CaperaCuellarCamila 2020.pdf?sequence=2&isAllowed=y
[19] Robot Operating System (ROS), “ROS: Home”. [online]. Disponible en https://www.ros.org/
[20] J. Solawetz y Francesco, “What is YOLOv8? The Ultimate Guide.”, 2023. [online]. Disponible en https://blog.roboflow.com/whats-new-in-yolov8/