

# **Instituto Tecnológico de Estudios Superiores de Monterrey**



## **Implementación de robótica inteligente & Manchester Robotics**

### **Manchester Robotics: Challenge 4**

#### **Profesores:**

Rigoberto Cerino Jiménez

Dr. Mario Martinez

#### **Integrantes**

Daniel Castillo López A01737357

Emmanuel Lechuga Arreola A01736241

Paola Rojas Domínguez A01737136

29 de Mayo de 2025

# Índice

<b>Índice.....</b>	<b>1</b>
<b>Resumen.....</b>	<b>2</b>
<b>Objetivos.....</b>	<b>3</b>
<b>Introducción.....</b>	<b>4</b>
Procesamiento de imágenes en una tarjeta embebida.....	4
Interconexión entre jetson y la cámara.....	4
Métodos de visión por computadora.....	5
Detección de los colores de un semáforo.....	5
Detección de línea.....	6
Robustez en sistemas de procesamiento de imágenes.....	6
Detección de los colores de un semáforo.....	6
Detección de línea.....	7
<b>Solución del problema.....</b>	<b>8</b>
<b>Resultados.....</b>	<b>11</b>
<b>Conclusiones.....</b>	<b>14</b>
<b>Referencias.....</b>	<b>15</b>

# Resumen

El resumen presenta una síntesis de los aspectos más relevantes del proceso de investigación y desarrollo llevado a cabo, destacando los objetivos, la metodología empleada y los principales resultados obtenidos.

El seguimiento de líneas utilizando visión por computadora es una tarea común en la robótica móvil, y representa un paso importante hacia la autonomía total de un robot. En este reto, se trabajó con el Puzzlebot, un robot móvil diferencial, para que fuera capaz de detectar una línea en el suelo y seguirla, al mismo tiempo que interpreta las luces de colores de un semáforo a escala.

Para ello, fue necesario aplicar conocimientos adquiridos en retos anteriores, como la navegación punto a punto y la implementación de capas de decisión. En esta ocasión se añadió una nueva capa: la percepción visual. El reto consistió en diseñar un sistema que combine estos elementos para obtener un comportamiento autónomo y fluido del robot, que pueda interpretar su entorno visual y actuar en consecuencia.

Además del seguimiento de línea, el robot debía responder a señales visuales específicas. El color rojo en un semáforo indica que debe detenerse; el amarillo, que debe disminuir su velocidad; y el verde, que puede continuar. Todo esto debía ejecutarse sin intervención humana, asegurando que el sistema sea completamente autónomo y capaz de enfrentar condiciones reales como variaciones de luz o perturbaciones físicas.

El sistema debía ser robusto, por lo que se consideraron estrategias para mejorar su estabilidad, como el ajuste de parámetros del controlador y el filtrado de imágenes. Este reto permitió poner en práctica múltiples conceptos de robótica y visión artificial, integrando diferentes componentes en una solución funcional.

# Objetivos

En esta sección se presentan el objetivo general y los objetivos particulares del reto, los cuales están enfocados en la implementación de un seguidor de línea en un Puzzlebot.

**Objetivo general:** Diseñar e implementar un sistema de seguimiento de línea para un robot móvil diferencial utilizando visión por computadora, integrándose con el sistema de control y navegación previamente desarrollado, y respondiendo adecuadamente a señales visuales que simulan un sistema de semáforo.

## Objetivos específicos

- Desarrollar un algoritmo de visión artificial que permita detectar una línea en el suelo utilizando la cámara del robot.
- Integrar la detección visual con el sistema de navegación autónoma del Puzzlebot.
- Incorporar el reconocimiento de señales visuales (semáforo) para modificar el comportamiento del robot según las condiciones del entorno.
- Diseñar un circuito de tres líneas según las especificaciones dadas, y probar el sistema en este entorno.
- Implementar técnicas de control robusto que tomen en cuenta perturbaciones, cambios en la iluminación y ruido en las imágenes.
- Validar el sistema mediante simulaciones y pruebas, asegurando que cumpla con el comportamiento esperado.

# Introducción

La introducción presenta el tema de investigación, proporcionando el contexto necesario para comprender la relevancia del reto a realizar. Se ofrece una visión general que facilita la construcción de un esquema mental sobre las metodologías utilizadas, además de los conceptos y tecnologías que se abordarán en el desarrollo del reporte.

## **Procesamiento de imágenes en una tarjeta embebida**

El procesamiento de imágenes en sistemas embebidos consiste en realizar operaciones de visión por computadora directamente en un dispositivo con recursos limitados, como una tarjeta Jetson Nano. A diferencia de un computador convencional, este tipo de hardware está optimizado para tareas específicas, con bajo consumo energético y alto rendimiento en operaciones paralelas gracias a su GPU integrada.

La Jetson Nano de NVIDIA, utilizada en este reto, incluye una GPU Maxwell de 128 núcleos y un procesador ARM Cortex-A57 de cuatro núcleos, lo que permite ejecutar algoritmos en tiempo real como filtrado de imágenes, segmentación por color y detección de objetos. En este proyecto, el procesamiento incluye conversión de color BGR a HSV, segmentación por umbral (thresholding), filtrado morfológico, y publicación del resultado mediante ROS 2.

La ventaja principal de realizar el procesamiento en la Jetson, y no en un nodo remoto, es la baja latencia y la posibilidad de ejecutar tareas de percepción de forma local, liberando ancho de banda de red y evitando cuellos de botella.

De acuerdo con el material proporcionado por Manchester Robotics, la imagen del sistema para la Jetson ya incluye ROS 2, OpenCV y los nodos necesarios para operar con cámaras tipo CSI (Raspberry Pi Camera), lo cual permite un desarrollo ágil y directo sobre la tarjeta embebida.

## **Interconexión entre jetson y la cámara**

La Jetson Nano se conecta directamente a la cámara Raspberry Pi mediante una interfaz CSI-2 (Camera Serial Interface), lo que permite una alta velocidad de transferencia de video con bajo consumo de CPU. Este tipo de conexión es preferido en sistemas embebidos porque está optimizado para cámaras integradas y soportado nativamente por el sistema operativo de Jetson.

Para facilitar su uso en ROS 2, se emplean nodos preinstalados como `video_source.ros2.launch` y `video_viewer.ros2.launch`, proporcionados por NVIDIA, que permiten publicar las imágenes capturadas en un tópico de ROS sin necesidad de configurar manualmente el pipeline de captura.

En este proyecto, se utiliza un nodo personalizado que lanza `video_source` para publicar las imágenes crudas en el tópico `/video_source/raw`, lo cual sirve de entrada para el nodo de

procesamiento de color. Este nodo convierte las imágenes, aplica filtros y publica un resultado numérico interpretado por el controlador.

El uso de esta arquitectura modular facilita el aislamiento de errores, la reutilización de código y el procesamiento concurrente, lo cual es clave en aplicaciones de robótica autónoma.

## **Métodos de visión por computadora**

La visión por computadora es una disciplina fundamental en robótica que permite a los sistemas interpretar su entorno visual a través del análisis automático de imágenes o secuencias de vídeo. Su aplicación abarca desde el reconocimiento de objetos hasta la navegación autónoma, y en este proyecto, tuvo dos funciones primordiales, identificar el color de un semáforo para modificar el comportamiento del robot de acuerdo con una máquina de estados y la detección de una línea negra para poder implementar un seguidor de línea.

### **Detección de los colores de un semáforo**

El procesamiento se inicia con la adquisición de imágenes mediante una cámara montada en el robot. Estas imágenes, obtenidas en formato BGR, se convierten al espacio de color HSV (Hue, Saturation, Value). Esta conversión no es trivial: el espacio HSV ofrece una mayor capacidad de segmentación basada en características perceptuales humanas, especialmente el matiz del color, que resulta más estable ante variaciones de iluminación que el modelo BGR. Por esta razón, HSV es ampliamente utilizado en aplicaciones de detección de color.

Una vez convertida la imagen, se aplica un filtrado por umbral para detectar regiones correspondientes a colores específicos. En este proyecto, se segmentaron los colores rojo, amarillo y verde utilizando rangos HSV calibrados experimentalmente. La detección del color rojo requirió dos rangos distintos, debido a que el matiz en HSV para este color se encuentra dividido en ambos extremos de la escala circular de 0 a 180 grados.

Para reducir el impacto del ruido visual se emplearon operaciones morfológicas como la erosión y la dilatación. Estas transformaciones, implementadas mediante filtros estructurantes, permiten eliminar objetos pequeños e inconsistentes, suavizando los contornos y mejorando la integridad de las regiones detectadas.

Posteriormente, se combinaron las máscaras de color para formar una imagen compuesta, y se realizó un conteo de píxeles para determinar cuál de los colores tenía mayor presencia en la escena. Esta estrategia permitió convertir una imagen compleja en una decisión numérica sencilla, que se publica como mensaje en un tópico de ROS 2 para ser interpretado por el controlador.

## **Detección de línea**

Para que el robot pueda seguir una línea negra centrada en la pista, se implementaron distintas estrategias de visión por computadora que combinan preprocesamiento, análisis estructural de la imagen y técnicas basadas en geometría y derivadas.

La primera etapa consistió en definir una Región de Interés (ROI) en la parte inferior de la imagen, donde es más probable encontrar la línea. Sobre esta ROI se aplicó conversión de grises, umbralado binario inverso, y limpieza morfológica para aislar las regiones negras del fondo más claro.

Luego, se utilizaron contornos (`cv2.findContours`) para identificar las regiones principales. Una vez aislado el contorno de la línea principal, se dividió verticalmente la ROI en tres bandas y, para cada una, se calcularon los momentos de imagen (`cv2.moments`) a fin de obtener el centroide ( $cx$ ,  $cy$ ). Estos centroides permiten estimar la posición central de la línea detectada.

El error de posición se calcula como la diferencia entre la coordenada  $x$  del centroide central (banda media) y el centro horizontal de la imagen, y se normaliza en el rango  $[-1, 1]$  para facilitar su uso por el controlador.

Este enfoque de visión por computadora es eficiente, de bajo consumo computacional y apropiado para plataformas embebidas como la Jetson Nano. A pesar de su simplicidad, ofrece una base sólida para implementar sistemas reactivos robustos en robótica móvil.

## **Robustez en sistemas de procesamiento de imágenes**

La robustez es una cualidad crítica en los sistemas de percepción visual utilizados en robótica. En este contexto, se define como la capacidad del sistema de mantener un comportamiento funcional y confiable a pesar de variaciones en las condiciones del entorno o la presencia de ruido en los datos de entrada. La naturaleza del entorno real, caracterizado por iluminación cambiante, superficies reflectantes o interferencias visuales, hace que la robustez sea un atributo indispensable para sistemas autónomos.

## **Detección de los colores de un semáforo**

El sistema de procesamiento de imágenes desarrollado en este reto fue diseñado para ser robusto mediante la adopción de varias estrategias. En primer lugar, la conversión de imágenes al espacio HSV proporciona una base sólida, ya que permite separar el componente de color del brillo. Esto reduce la sensibilidad a sombras o cambios leves en la intensidad de luz, permitiendo que el robot pueda detectar colores de forma más consistente en diferentes condiciones.

Otra fuente de robustez proviene de la calibración manual de los rangos HSV para cada color de interés. Estos rangos fueron definidos empíricamente utilizando una herramienta interactiva de selección de color, garantizando que correspondan a los valores reales

observados por la cámara del robot. Esta calibración fina permite evitar tanto falsas detecciones como omisiones, mejorando la fiabilidad del sistema.

Asimismo, la aplicación de operadores morfológicos contribuye a la estabilidad del sistema. Estas operaciones permiten limpiar la imagen de regiones pequeñas o inconsistentes que podrían activar falsamente la lógica de detección. De esta forma, se atenúan los efectos del ruido inherente a las cámaras o al ambiente.

Finalmente, la decisión sobre el color detectado no se basa en una única lectura puntual, sino en el análisis del número de píxeles pertenecientes a cada máscara de color. Este enfoque por mayoría mejora la resistencia del sistema a falsas detecciones causadas por elementos aislados o reflejos.

### **Detección de línea**

El sistema de seguimiento de línea fue diseñado para priorizar su estabilidad en variaciones o perturbaciones en condiciones reales, como cambios en la iluminación o interrupciones parciales de las líneas.

Se utilizó una estrategia multibanda: la imagen binarizada se dividió en tres regiones horizontales, lo que permite calcular centroides en diferentes alturas de la pista. Esto aporta redundancia vertical que estabiliza la detección.

Además, se implementó una validación temporal que compara los centroides detectados en el frame actual con los del frame anterior. Si se detecta un cambio abrupto, se descarta la lectura actual y se mantiene la última válida. Esta técnica filtra falsos positivos producidos por reflejos, sombras o líneas ajenas al recorrido.

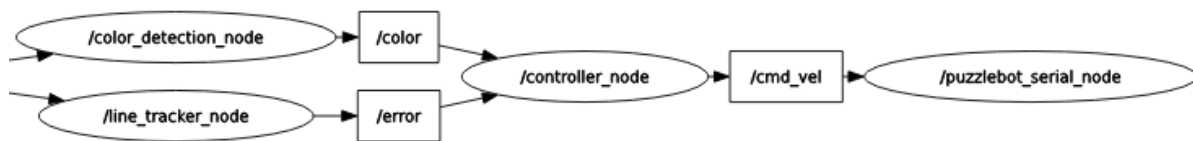
El cálculo de momentos también añade robustez, ya que estima el centro de masa de una región, lo cual es menos sensible a irregularidades en los bordes.

En conjunto, estas estrategias aseguran que el sistema de visión mantenga una detección fiable y estable incluso en condiciones adversas, reforzando el control general del robot y su capacidad de responder correctamente ante señales visuales del entorno.



# Solución del problema

En esta parte desarrollaremos una solución para nuestro reto, en el cual consiste en poder tener una distinción en la visión de computadora, para poder tener el seguidor de línea y el análisis de colores del semáforo y con estos dos conceptos poder tener una consecuencia directa al controlador del puzzlebot. Con el objetivo ya planteado, podemos realizar la siguiente conexión de Nodos en la Figura 1 para tener orden en la ejecución de nuestro problema.



**Figura 1:** Diagrama de conexión de nodos

## Nodo de seguidor de línea (line\_tracker):

Este nodo se presenta para poder hacer el procesamiento de imagen, se encarga de poder detectar una línea central en nuestra cámara para poder calcular el error de alineación respecto al centro de nuestra imagen y con eso poder publicar nuestro error para el nodo de “controller”.

- Primero crearemos un suscriptor en el tópico `/video_source/raw`, que transmite las imágenes de la cámara CSI, que creará una cola para tener un flujo continuo
- Creamos dos publicadores, el primero de estos es el `/error` que se encargará de enviar el mensaje al nodo controller, el segundo de estos creará un `/debug_image` que hará que podamos ver como se muestra el resultado de nuestro análisis
- Establecemos variables que serán para nuestro centroide y un vector para guardar los últimos valores de los tres centroides.
- Cada ROI procesada se almacena en un video AVI de nombre “line\_vid” con 20 fps y su resolución 1200 x 250
- Las imágenes recibidas serán convertidas en formato BGR mediante Bridge.Y será rotado en 180° y redimensionadas a 1200 x 480 pixeles para definir nuestra área de trabajo.
- El código plantea convertir la imagen en escala en grises y se binariza con un umbral fijo. Con esto implementamos invertir la imagen para así poder resaltar la línea negra de un fondo claro
- Aplicando una operación de apertura morfológica de kernel 3 x 3 con dos iteraciones para eliminar ruido y objetos no deseados.

- Después aplicamos una detección de contornos en la imagen binaria, seleccionando siempre el contorno mayor a 500 píxeles
- Partiendo desde la imagen binaria se aplican tres bandas horizontales. Para detectar la sección, si hay los suficiente píxeles se calcula el centroide utilizando momentos y se dibujarán los centroides y sus conexiones dentro de la línea sobre la imagen de salida.
- Ahora aplicando una condición si los tres centroides están presentes, se evalúa si el cambio respecto al cuadro anterior es aceptable, teniendo a un umbral de 50 píxeles. Si el cambio es muy abrupto se reutilizaran los centroides del cuadro anterior para mantener estabilidad.
- Definimos el cálculo del error como la diferencia horizontal del cálculo del promedio ponderado del centroide dos más el centroide 3, además el centro de nuestra imagen, normalizando por el ancho, estableciendo de -1 a 1. Este valor se publica en el tópico /error.

### **Detección de color para semáforo (color\_detection.py):**

El nodo recibe imágenes de una cámara, detecta el color del semáforo y publica un estado numérico en el topic /color.

- Primero, definimos los rangos de colores en HSV (Hue, Saturation, Value), con los siguientes colores: rojo, que se define mediante dos rangos en HSV debido a su ubicación en los extremos del círculo de tono; amarillo, con un único rango en HSV; y verde, también con un solo rango. Esto permite segmentar los colores típicos de los semáforos.
- Suscribimos el nodo al flujo del video /video\_source/raw que transmite las imágenes en formato sensor\_msgs/Image
- Y utilizamos el CvBridge para convertir imágenes ROS a formato BGR de OPenCV para su procesamiento óptimo.
- Establecemos un temporizador para procesamiento periódico cada 0.2 segundo (5 fps), reduciendo así la carga computacional.
- La imagen se convierte al espacio de color HSV, mejor adaptado para la detección de colores específicos.
- Hacemos la creación de máscaras binarias por color:
  - **Cv.inRange()**: Se genera una máscara para los colores del semáforo
  - **Cv.bitwise\_or()**: Para el rojo se combinan ambas máscaras
- Establecemos un filtrado morfológico de ruido, se hace una operación morfológica con un kernel 5x5 para eliminar ruido en las máscaras de color.
- Ahora haremos la detección de píxeles blancos de las máscaras para estimar la presencia de su color
- Con esto preparado establecemos una máquina de estados que basado en el conteo, hacemos el estado cero (sin detección), uno (STOP), dos (GO) y tres (SLOW), para evitar cambios repentinos.
- Publicamos el estado detectado en el canal /color con el valor de nuestra máquina de estados, para así ser enviado a “controllerf”

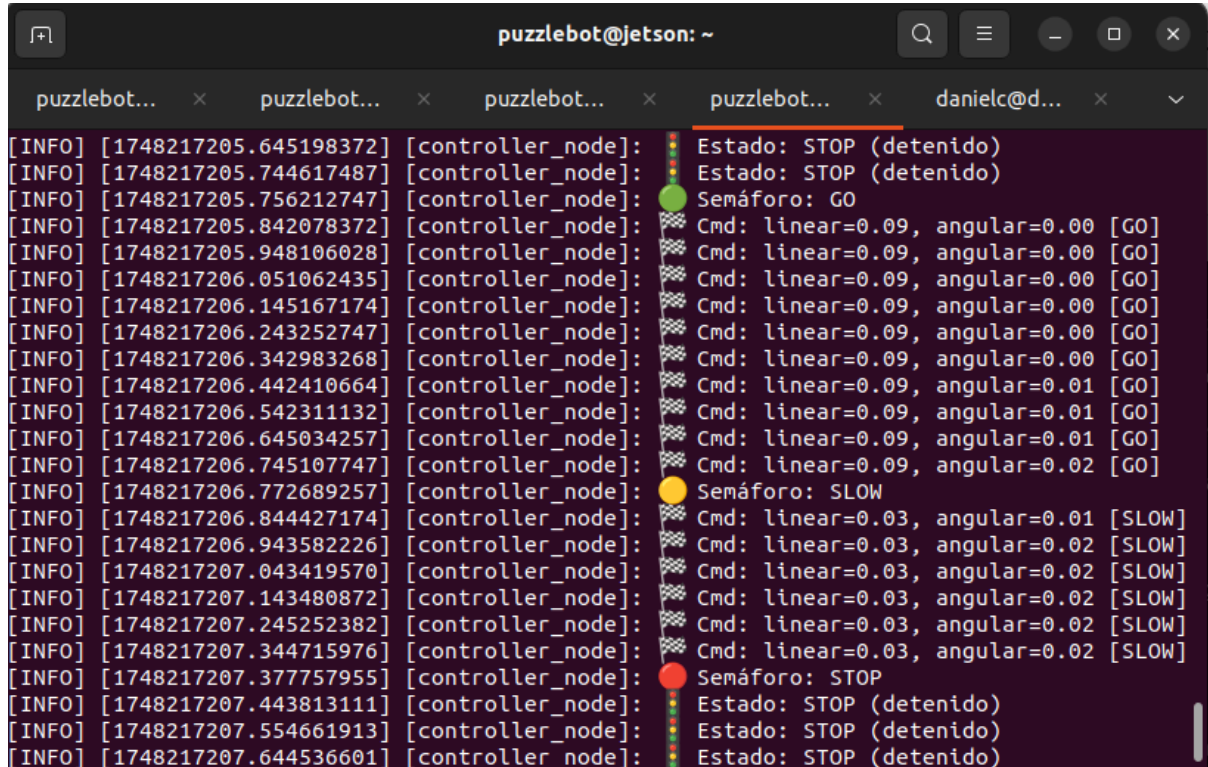
### **Controlador P (controllerf.py):**

Este nodo se encarga de implementar un control para un robot móvil que responde al estado de un semáforo y a un error de posición, generando los comandos de velocidad.

- Establecemos los parámetros de control “Kp” y “base\_speed”, con los estados de nuestro error y los valores del semáforo. Además para tener en STOP el iniciar de nuestro robot.
- Creamos suscripciones en /error, que representa el error de seguimiento de línea, /color, que representa el estado del semáforo. Junto a esto podemos hacer el mensaje tipo Twist al canal /cmd\_vel, para enviar el comando de velocidad del robot.
- Ejecutamos un timer\_callback cada 0.1 segundos para publicar comandos
- Utilizando la ganancia proporcional multiplicada por el error de la trayectoria, podremos corregir su curso en función a lo recibido.
- Agregamos una verificación periódica de 10 Hz para el estado actual del semáforo.
- Tenemos una limitación de velocidad angular que está entre -1 a 1 rad/s para evitar giros bruscos

## Resultados

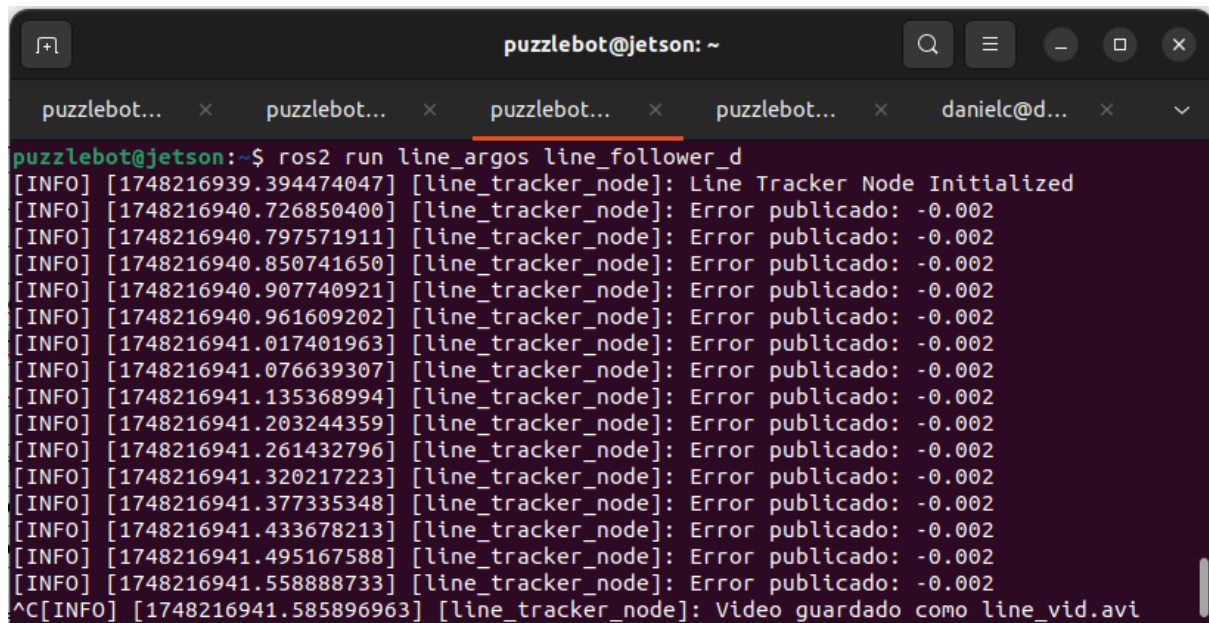
En este apartado, se muestran los resultados de los nodos propuestos en la solución del problema, tomando en consideración las diferentes terminales de cada nodo.



```
puzzlebot@jetson: ~  
[INFO] [1748217205.645198372] [controller_node]: Estado: STOP (detenido)  
[INFO] [1748217205.744617487] [controller_node]: Estado: STOP (detenido)  
[INFO] [1748217205.756212747] [controller_node]: Semáforo: GO  
[INFO] [1748217205.842078372] [controller_node]: Cmd: linear=0.09, angular=0.00 [GO]  
[INFO] [1748217205.948106028] [controller_node]: Cmd: linear=0.09, angular=0.00 [GO]  
[INFO] [1748217206.051062435] [controller_node]: Cmd: linear=0.09, angular=0.00 [GO]  
[INFO] [1748217206.145167174] [controller_node]: Cmd: linear=0.09, angular=0.00 [GO]  
[INFO] [1748217206.243252747] [controller_node]: Cmd: linear=0.09, angular=0.00 [GO]  
[INFO] [1748217206.342983268] [controller_node]: Cmd: linear=0.09, angular=0.00 [GO]  
[INFO] [1748217206.442410664] [controller_node]: Cmd: linear=0.09, angular=0.01 [GO]  
[INFO] [1748217206.542311132] [controller_node]: Cmd: linear=0.09, angular=0.01 [GO]  
[INFO] [1748217206.645034257] [controller_node]: Cmd: linear=0.09, angular=0.01 [GO]  
[INFO] [1748217206.745107747] [controller_node]: Cmd: linear=0.09, angular=0.02 [GO]  
[INFO] [1748217206.772689257] [controller_node]: Semáforo: SLOW  
[INFO] [1748217206.844427174] [controller_node]: Cmd: linear=0.03, angular=0.01 [SLOW]  
[INFO] [1748217206.943582226] [controller_node]: Cmd: linear=0.03, angular=0.02 [SLOW]  
[INFO] [1748217207.043419570] [controller_node]: Cmd: linear=0.03, angular=0.02 [SLOW]  
[INFO] [1748217207.143480872] [controller_node]: Cmd: linear=0.03, angular=0.02 [SLOW]  
[INFO] [1748217207.245252382] [controller_node]: Cmd: linear=0.03, angular=0.02 [SLOW]  
[INFO] [1748217207.344715976] [controller_node]: Cmd: linear=0.03, angular=0.02 [SLOW]  
[INFO] [1748217207.377757955] [controller_node]: Semáforo: STOP  
[INFO] [1748217207.443813111] [controller_node]: Estado: STOP (detenido)  
[INFO] [1748217207.554661913] [controller_node]: Estado: STOP (detenido)  
[INFO] [1748217207.644536601] [controller_node]: Estado: STOP (detenido)
```

Figura 2: Terminal controllerf

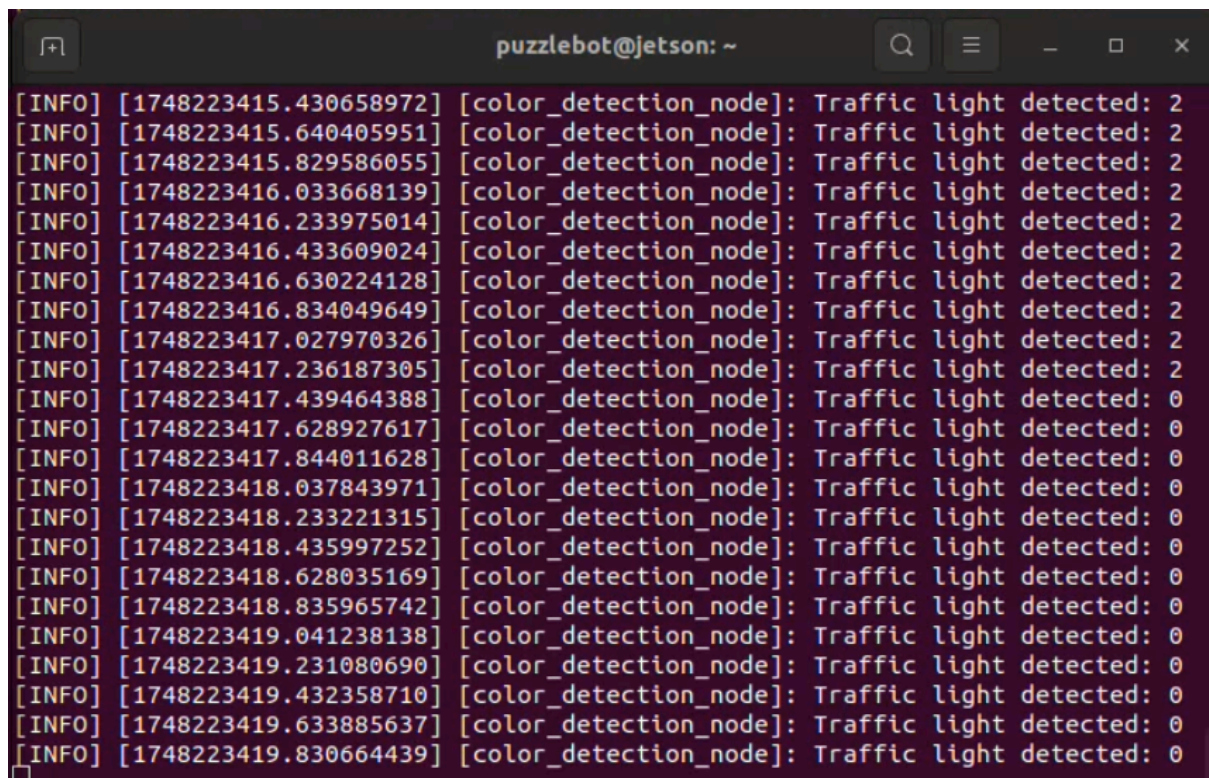
Como presenta la Figura 2 nuestro nodo “controllerf”, muestra los cambios de color detectados por el color\_detection, como vemos cumple el seguimiento de la máquina de estados, cuando el semáforo esté en verde muestra la velocidad lineal en 0.09 y viendo el movimiento angular actual, además después del cambio en el color amarillo se produce la reducción de la velocidad lineal.



```
puzzlebot@jetson: ~  
puzzlebot... x puzzlebot... x puzzlebot... x puzzlebot... x danielc@d... x  
puzzlebot@jetson:~$ ros2 run line_argos line_follower_d  
[INFO] [1748216939.394474047] [line_tracker_node]: Line Tracker Node Initialized  
[INFO] [1748216940.726850400] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216940.797571911] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216940.850741650] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216940.907740921] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216940.961609202] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.017401963] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.076639307] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.135368994] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.203244359] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.261432796] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.320217223] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.377335348] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.433678213] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.495167588] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.558888733] [line_tracker_node]: Error publicado: -0.002  
[INFO] [1748216941.585896963] [line_tracker_node]: Video guardado como line_vid.avi
```

Figura 3 : Terminal Line\_follower

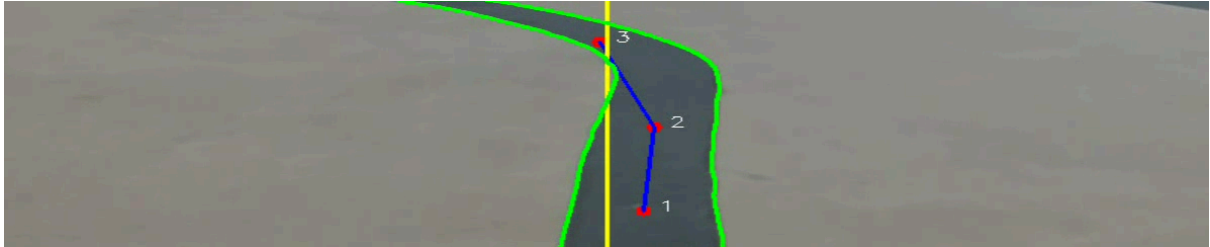
Cuando iniciamos nuestro line\_follower, la Figura 3 muestra cómo el error es establecido, si el error es definido en negativo, provocará que nuestro Puzzlebot se mueva a la derecha y de manera viceversa se verá si el error es en positivo, estos son los datos de salida de nuestro nodo y al finalizarlo muestra un mensaje confirmando que el video line\_vid.avi ha sido guardado.



```
puzzlebot@jetson: ~  
[INFO] [1748223415.430658972] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223415.640405951] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223415.829586055] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223416.033668139] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223416.233975014] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223416.433609024] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223416.630224128] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223416.834049649] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223417.027970326] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223417.236187305] [color_detection_node]: Traffic light detected: 2  
[INFO] [1748223417.439464388] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223417.628927617] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223417.844011628] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223418.037843971] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223418.233221315] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223418.435997252] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223418.628035169] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223418.835965742] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223419.041238138] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223419.231080690] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223419.432358710] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223419.633885637] [color_detection_node]: Traffic light detected: 0  
[INFO] [1748223419.830664439] [color_detection_node]: Traffic light detected: 0
```

Figura 4: Terminal Color Detection

El color\_detection, como podemos ver en la Figura 4, indica la luz del semáforo que esté detectando en este momento, dicho color será enviado al controller, para poder establecer cuál será el movimiento de nuestro robot, en el ejemplo evidenciamos como la luz detectada es de 2 significando que esta detectando el verde para después pasar a cero, que sería no detectar nada.



**Figura 5:** Salida video line\_follower

La Figura 5 es la perspectiva de nuestra cámara con los recortes necesarios, demuestra como la línea es seguida en base a los centroides que obtenemos y mostrando los cambios justo antes de entrar a una curva.

El video en youtube (<https://youtu.be/tFBr3m6d-IM>) muestra al robot siguiendo la pista, confirmando la funcionalidad del sistema de seguimiento de línea.

Finalmente, el video en youtube ([https://youtu.be/XEx\\_ZjFyzmc](https://youtu.be/XEx_ZjFyzmc)) muestra al robot siguiendo la línea de la pista y reaccionando a los colores del semáforo, además de un video que como el robot interpreta la línea negra, confirmando la funcionalidad de todo el sistema y su robustez.

Video final de evidencia: <https://youtu.be/boAW5BcXpDQ>

## Conclusiones

Por último, se presentan los logros alcanzados a lo largo del desarrollo del reto, analizando el grado de cumplimiento de los objetivos planteados. Se evalúa si estos fueron alcanzados en su totalidad, identificando las razones detrás de su éxito o, en caso contrario, los factores que pudieron haber limitado su cumplimiento.

En conclusión, este proyecto representa un paso importante en el desarrollo de un sistema de seguimiento de línea para un robot diferencial. Durante su implementación se enfrentaron diversas dificultades, principalmente en el reconocimiento preciso de la pista y de los objetos presentes en ella. A pesar de estos desafíos, se presentó una solución funcional, logrando cumplir con el objetivo general y acercándonos de manera significativa a la implementación del proyecto final.

La construcción de una pista representativa del entorno final permitió identificar limitaciones clave, tanto en el entorno como en el diseño del Puzzlebot. Esto nos motivó a buscar soluciones más óptimas, como el ajuste dinámico de velocidades, que facilita la navegación en curvas cerradas, así como la reducción de falsos positivos en la detección de elementos externos como líneas o figuras no deseadas. Este proceso evidenció la importancia de diseñar sistemas robustos y adaptables, capaces de operar en entornos complejos y cambiantes.

Finalmente, este trabajo no solo validó el funcionamiento básico del sistema, sino que también sentó las bases para futuras mejoras. Nos impulsa a continuar perfeccionando el control del robot e integrar sistemas de visión más precisos, optimizando la interacción entre percepción y acción. Con cada iteración, nos acercamos más a una solución confiable y lista para aplicarse en escenarios reales.



# Referencias

En este apartado se anexan los elementos consultados para el desarrollo del tema de investigación.

ManchesterRoboticsLtd. (s. f.-e).

*TE3002B\_Intelligent\_Robotics\_Implementation\_2025/Week1/Presentations/PDF/MC*

*R2\_Puzzlebot\_Jetson\_Ed\_ROS2.pdf* at main ·

*ManchesterRoboticsLtd/TE3002B\_Intelligent\_Robotics\_Implementation\_2025.*

GitHub.

[https://github.com/ManchesterRoboticsLtd/TE3002B\\_Intelligent\\_Robotics\\_Implementation\\_2025/blob/main/Week1/Presentations/PDF/MC\\_R2\\_Puzzlebot\\_Jetson\\_Ed\\_ROS2.pdf](https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week1/Presentations/PDF/MC_R2_Puzzlebot_Jetson_Ed_ROS2.pdf)

ManchesterRoboticsLtd. (s. f.-j).

*TE3002B\_Intelligent\_Robotics\_Implementation\_2025/Week4/Presentations/PDF* at

*main · ManchesterRoboticsLtd/TE3002B\_Intelligent\_Robotics\_Implementation\_2025.*

GitHub.

[https://github.com/ManchesterRoboticsLtd/TE3002B\\_Intelligent\\_Robotics\\_Implementation\\_2025/tree/main/Week4/Presentations/PDF](https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/tree/main/Week4/Presentations/PDF)

ManchesterRoboticsLtd. (s. f.-k).

*TE3002B\_Intelligent\_Robotics\_Implementation\_2025/Week6/Presentations/PDF/MC*

*R2\_shape\_detection\_watermark.pdf* at main ·

*ManchesterRoboticsLtd/TE3002B\_Intelligent\_Robotics\_Implementation\_2025.*

GitHub.

[https://github.com/ManchesterRoboticsLtd/TE3002B\\_Intelligent\\_Robotics\\_Implementation\\_2025/blob/main/Week6/Presentations/PDF/MC\\_R2\\_shape\\_detection\\_watermark.pdf](https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week6/Presentations/PDF/MC_R2_shape_detection_watermark.pdf)