

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

15 de Mayo de 2025

Índice

Índice.....	1
Resumen.....	2
Objetivos.....	3
Introducción.....	4
Odometría de un robot móvil diferencial.....	4
Cálculo del error en un robot móvil diferencial.....	5
Controlador PI en un robot móvil diferencial.....	6
Robustez de un controlador.....	7
Procesamiento de imágenes en una tarjeta embebida.....	7
Interconexión entre jetson y la cámara.....	8
Métodos de visión por computadora.....	8
Robustez en sistemas de procesamiento de imágenes.....	9
Solución del problema.....	11
Resultados.....	14
Conclusiones.....	17
Referencias.....	18

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.

Se presenta la entrega de un proyecto donde se integra el conocimiento adquirido a lo largo de los retos donde abordamos temas como lo es la odometría, control y navegación punto a punto, a todo lo antes mencionado le agregamos una capa de inteligencia basada en visión computacional artificial. donde el robot modelo “Puzzlebot” deberá modificar su comportamiento dependiendo del color detectado en un semáforo, los colores detectados son rojo, amarillo o verde. Para esto, se requerirá un sistema de detección visual donde se aplica la robustez con el fin de que por alguna ocasión o si llegara el caso en el puzzlebot pueda detectar el color de manera óptima y que ya tenga como un protocolo de seguridad por si no llegara a detectar los colores o simplemente algún dato mandado erróneamente, y por eso es fundamental incorporar un controlador en lazo cerrado ajustado de manera adecuada con el fin de aumentar la parte de exactitud del sistema. Este documento presenta todos los temas que se abordaron para su realización, así como una presentación de la metodología e investigación realizadas para la obtención y presentación de resultados de manera en la que se puede observar cómo se cumple con el objetivo principal y los objetivos generales de este proyecto.

Objetivos

En esta sección se presentan el objetivo general y los objetivos particulares del reto, los cuales están enfocados en la implementación de un control de lazo abierto para la trayectoria de un Puzzlebot.

Objetivo general: Desarrollar un sistema de control autónomo que permita al Puzzlebot tomar decisiones de navegación en base a la detección de colores en un semáforo virtual, basando el apoyo en ROS 2 para poder completar este objetivo.

Objetivos específicos:

- Diseñar un nodo de visión artificial capaz de detectar los colores rojo, amarillo y verde en un flujo de video (en tiempo real).
- Implementar un sistema de control en lazo cerrado que adapte la velocidad del robot según el color detectado.
- Integrar y validar el sistema completo en el entorno de simulación del Puzzlebot tanto de manera teórica como en tiempo real.
- Documentar el proceso de desarrollo y pruebas mediante un informe técnico detallado.

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.

La robótica móvil Puzzlebot no solo implica el movimiento de un robot en un entorno, sino también su capacidad para adaptarse y reaccionar ante condiciones variables. Una de las aplicaciones clásicas de la visión artificial en robótica es la detección de señales de tráfico, semáforos o indicadores visuales (señales). Este reto busca simular una situación realista donde el robot debe responder al color de un semáforo para decidir si detenerse, avanzar o reducir su velocidad. Se espera que los estudiantes implementen esta inteligencia visual y la integren en un sistema de control robusto, considerando aspectos como no linealidades, ruido y variabilidad de condiciones. Este tipo de integración representa un paso hacia el diseño de robots más inteligentes y autónomos.

Odometría de un robot móvil diferencial

En los robots móviles diferenciales, la odometría es un método fundamental para estimar la posición y orientación del robot a lo largo del tiempo, utilizando la información proporcionada por los encoders de las ruedas. Este proceso se basa en un modelo cinemático diferencial que describe cómo las velocidades individuales de las ruedas determinan el desplazamiento del robot en un plano (ManchesterRoboticsLtd, s. f.-h).

$$V = r \frac{\omega_R + \omega_L}{2} \quad (1)$$

$$\omega = r \frac{\omega_R - \omega_L}{l} \quad (2)$$

Donde:

- ω_R es la velocidad angular de la rueda derecha
- ω_L es la velocidad angular de la rueda izquierda
- V corresponde a la velocidad lineal del robot
- ω corresponde a la velocidad angular del robot
- l es la distancia entre las ruedas
- r es el radio de las ruedas

Estas velocidades se integran con el paso del tiempo para estimar la posición del robot (x , y) y su orientación θ . Para ello, se utiliza una forma discretizada del modelo cinemático implementada con el método de Euler (ManchesterRoboticsLtd, s. f.-h).

$$x_{k+1} = x_k + V \cdot \cos(\theta_k) \cdot dt \quad (3)$$

$$y_{k+1} = y_k + V \cdot \sin(\theta_k) \cdot dt \quad (4)$$

$$\theta_{k+1} = \theta_k + \omega \cdot dt \quad (5)$$

Este procedimiento, conocido como dead reckoning, es útil en entornos donde el robot no tiene acceso a sensores externos como GPS. Sin embargo, hay que considerar que su precisión se ve afectada por errores acumulativos provocados por el deslizamiento de las ruedas o la discretización del tiempo. Por ello, la odometría suele complementarse con técnicas de localización más robustas.

Cálculo del error en un robot móvil diferencial

El cálculo del error es esencial para el diseño de sistemas de control destinados a que guíen al robot hacia una posición deseada. Este proceso se conoce como estabilización de punto, y consiste en definir una pose objetivo $X_g = (x_g, y_g, \theta_g)$ que el robot debe alcanzar, comparándola con su pose actual estimada $X_r = (x_r, y_r, \theta_r)$ (ManchesterRoboticsLtd, s. f.-h).

A partir de esta comparación, se obtiene un vector error en coordenadas cartesianas y orientación:

$$e_x = x_g - x_r \quad (6)$$

$$e_y = y_g - y_r \quad (7)$$

$$e_\theta = \text{atan2}(e_y, e_x) - \theta_r \quad (8)$$

Este último término descrito en la ecuación (8), representa el ángulo entre la orientación actual del robot y la línea que conecta su posición con el objetivo. Dado que los ángulos pueden crecer indefinidamente, es común utilizar una función de envoltura para limitar los valores de e_θ dentro del rango $[-\pi, \pi]$, lo que garantiza un control estable (ManchesterRoboticsLtd, s. f.-h).

El error de distancia es una magnitud escalar que representa que tan lejos está el robot de su destino:

$$e_d = \sqrt{e_x^2 + e_y^2} \quad (9)$$

Con estos errores definidos, se puede aplicar una ley de control proporcional simple:

$$V = K_d \cdot e_d \quad (10)$$

$$\omega = K_\theta \cdot e_\theta \quad (11)$$

Donde K_d y K_θ son constantes que ajustan la respuesta del sistema. Esta forma de control es suficiente para lograr un comportamiento estable en la mayoría de los casos.

Si queremos mejorar el controlador, podemos añadir la componente integral al sistema, teniendo así un controlador PI:

$$V = K_d \cdot e_d(t) + K_{vi} \cdot \int_0^t e_d(\tau) d\tau \quad (12)$$

$$\omega = K_\theta \cdot e_\theta + K_{wi} \cdot \int_0^t e_\theta(\tau) d\tau \quad (13)$$

Donde K_{vi} es la ganancia integral para la velocidad lineal y K_{wi} es la ganancia integral para la velocidad angular. Los integrales se pueden implementar numéricamente como sumas acumulativas en tiempo discreto.

El cálculo de errores, es conjunto con la odometría, permite al robot moverse de forma autónoma hacia una meta definida en un entorno conocido.

Controlador PI en un robot móvil diferencial

Uno de los principales enfoques fue el uso de control en lazo cerrado, en el cual el robot utiliza su propia estimación de posición para corregir continuamente su movimiento y acercarse al punto final de su trayectoria. Para estimar dicha posición, se empleó la información proveniente de los encoders de las ruedas, mediante el método conocido como dead reckoning, que permite calcular el desplazamiento del robot integrando su velocidad lineal y angular a lo largo del tiempo. Aunque este método puede acumular errores, representa una base adecuada para la implementación de control autónomo.

El sistema de control implementado fue de tipo proporcional-integral (PI). Este tipo de controlador no solo considera el error instantáneo entre la posición actual y la deseada del robot, sino que también acumula dicho error a lo largo del tiempo para corregir desviaciones persistentes. Esto permite una respuesta más precisa que la obtenida con un controlador proporcional puro, especialmente al reducir el error en el estado estacionario. Las velocidades lineal y angular del robot se ajustan en función de estos componentes, dentro de límites previamente definidos para garantizar un comportamiento seguro y estable.

Para dotar al sistema de autonomía, se integró un nodo generador de trayectorias que permite al usuario ingresar múltiples puntos objetivo desde consola. Este nodo se encarga de enviar los objetivos al robot uno por uno, esperando una confirmación por parte del controlador tras alcanzar cada punto, antes de continuar con el siguiente. Esto asegura una navegación secuencial ordenada.

Todo el desarrollo se realizó utilizando ROS 2 y validación heurística, manteniendo una estructura modular para facilitar la implementación, la depuración y la evaluación por

separado de los distintos componentes. En este reporte se describen los pasos seguidos para construir la solución funcional, detallando tanto el análisis teórico como la implementación práctica del código, y se presentan los resultados mediante evidencia visual en formato de video e imagen.

Robustez de un controlador

En el contexto de sistemas de control, la robustez se refiere a la capacidad del controlador para mantener un desempeño aceptable frente a perturbaciones externas, incertidumbre en el modelo del sistema o variaciones en las condiciones de operación. Un controlador PI (Proporcional-Integral) es una estrategia clásica que, bien ajustada, puede aportar robustez frente a diversos factores que afectan el comportamiento de un sistema dinámico.

El componente proporcional (P) reacciona directamente al error entre la señal deseada y la medida, mientras que el componente integral (I) acumula dicho error en el tiempo y busca eliminar el error en estado estacionario. Esta característica hace que el controlador PI sea especialmente útil en presencia de pequeñas perturbaciones o errores persistentes, compensando desviaciones sostenidas que un controlador P puro no podría eliminar por completo.

Desde el punto de vista de robustez, el controlador PI:

- Tolera ruido y oscilaciones pequeñas debido a que no depende exclusivamente de un modelo exacto del sistema.
- Corrige errores acumulativos causados por factores como el deslizamiento de ruedas, errores de odometría o variaciones en la fricción.
- Mantiene estabilidad bajo ciertos márgenes de incertidumbre, siempre y cuando las ganancias estén adecuadamente calibradas y el sistema no sea excesivamente rápido ni no lineal.

Es importante señalar que, aunque el controlador PI no garantiza robustez ante cualquier tipo de incertidumbre, ofrece una buena compensación entre simplicidad y tolerancia a errores típicos de un sistema físico como el Puzzlebot.

Un sistema robusto con control PI debe diseñarse considerando la respuesta a perturbaciones y el margen de ganancia y fase del sistema. En aplicaciones prácticas como robots móviles, esto se traduce en mantener un seguimiento estable de la trayectoria incluso ante lecturas imprecisas de sensores o pequeñas inconsistencias en los actuadores.

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, su principal función fue identificar el color de un semáforo para modificar el comportamiento del robot de acuerdo con una máquina de estados.

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.

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.

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.

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 sección se desarrolla una solución del reto, que consiste en establecer una visión por computadora que analice los colores del semáforo y dichos colores tengan una implicación directa en el controlador del puzzlebot. Con el objetivo definido podremos establecer una estrategia para desarrollar los los nodos de la Figura 1 con los que se nos sea fácil poder desarrollar individualmente para al final poderlos unir y conseguir solucionar nuestro reto.

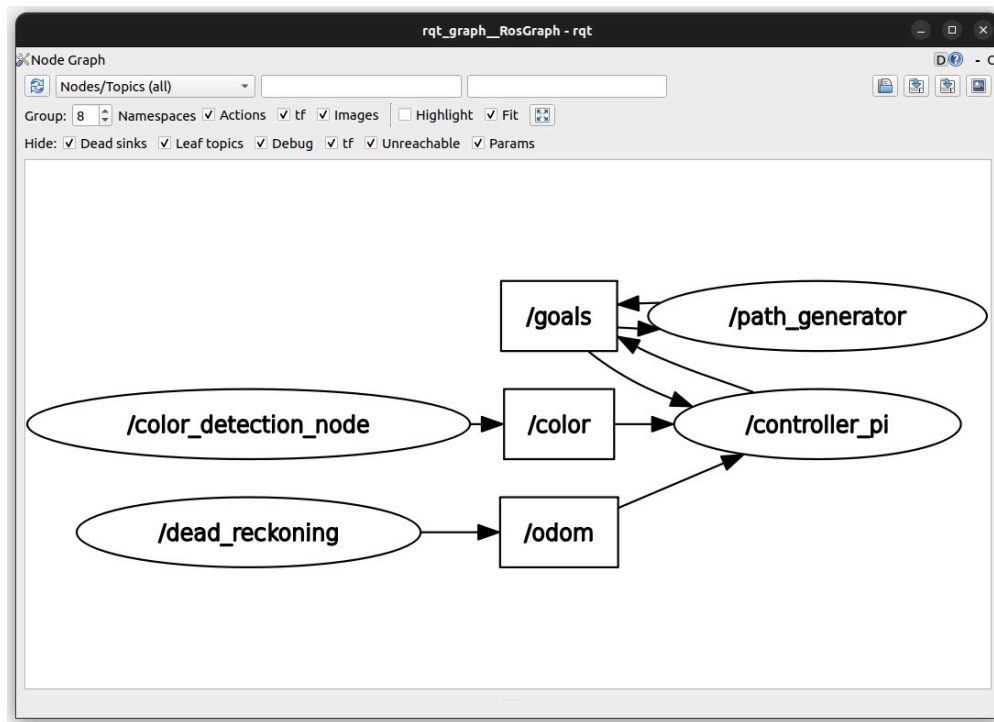


Figura 1: Diagrama de conexión de nodos

Se presentan los nodos obtenidos además de los elementos que tiene cada uno:

Nodo de odometría:

Siendo el encargado de establecer el posicionamiento de nuestro puzzlebot, hace el canal /odmo que servirá para poder ubicar el robot.

- Con las velocidades angulares de nuestras ruedas se desarrolla la Velocidad lineal y angular de nuestro robo con las fórmulas uno y dos previamente definida,
- Estima la posición (x,y) y orientación con el método reckoning, integrando las velocidades con el paso del tiempo, para obtener la posición de nuestro robot.
- Pública la odometría en el tópicos /odom y convierte los resultados a una representación con cuaterniones para la orientación.

Controlador de trayectorias (path generator.py):

Se encarga de establecer los puntos de trayectorias para nuestro robot publicando los valores en /goals y también recibiendo el mensaje desde el nodo del controlador.

- Permite al usuario ingresar múltiples puntos desde consola.
- Para cada uno el usuario ingresa X y Y.
- EL primer objetivo se envía inmediatamente tras recibir la entrada del usuario
- La bandera tiene que ser 1 y que la posición coincide, así se confirma el objetivo, espera un segundo y avanza al siguiente objetivo.

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 controller.py

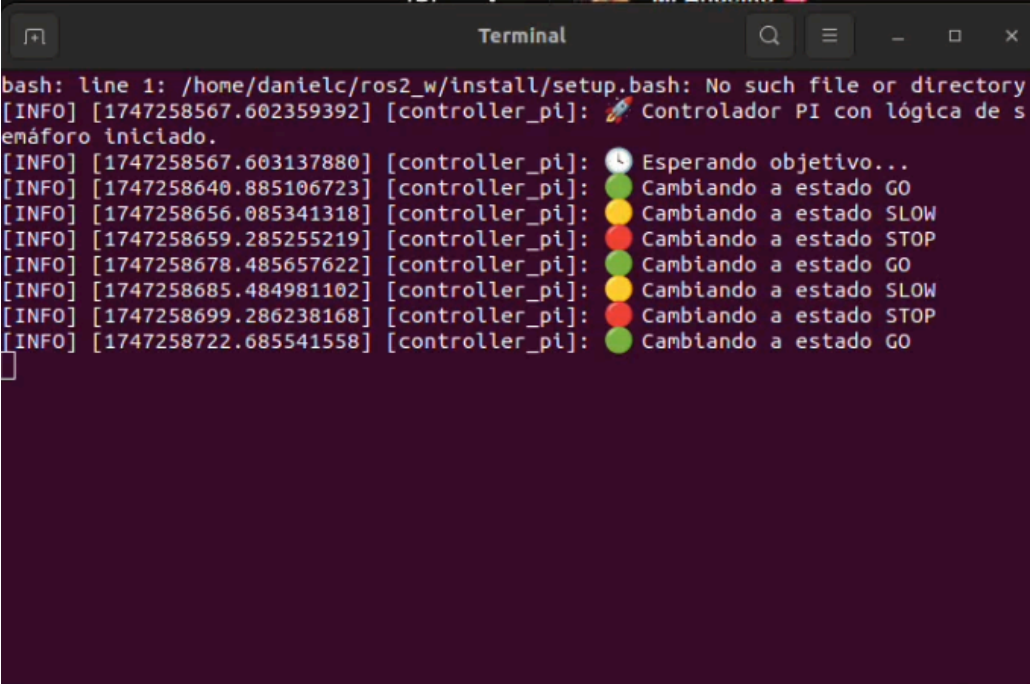
Controlador PI (controller_pi.py):

- Se define el nodo “controller_pi” que implementa un controlador PI, suscrito a /odom, /goals y /color, y publicando comandos de velocidad en /cmd_vel.
- Configuramos los valores de tolerancia para obtener nuestros objetivos deseados sin error, teniendo la tolerancia lineal y angular
- Definimos nuestras ganancias para el PI, kp_linear, ki_linear, kp_angular, ki_angular

- Se definen los límites mínimos y máximos para las velocidades, asegurando que su movimiento sea suave para evitar cambios bruscos que reinicien a nuestro puzzlebot.
- Hacemos una función llamada `goal_callback` que extrae los valores de nuestro mensaje `/goals`, que contiene las coordenadas del objetivo y marca los flags necesarios para iniciar el control hacia el nuevo objetivo.
- En la función `odom_callback` se obtiene la posición actual del robot desde `/odom` y calculamos el error de distancia y el error angular hacia el objetivo, también normalizando nuestro ángulo entre los valores de $-\pi$ y π .
- Evaluamos el estado del objetivo, para saber si el robot se encuentra en los parámetros definidos y se confirma de objetivo alcanzado regresando el mensaje de `/goals`
- EL control PI prioriza la corrección de orientación y se aplica el PI lineal, ajustando con el valor dependiendo de lo que reciba en el canal `/color`

Resultados

En este apartado, se muestran los resultados de los nodos propuestos en la solución del problema, tomando en consideración las diferentes terminales unidas a un launcher.sh que unifica las terminales siendo mucho más sencillo poder abrir las necesarias con un solo comando.

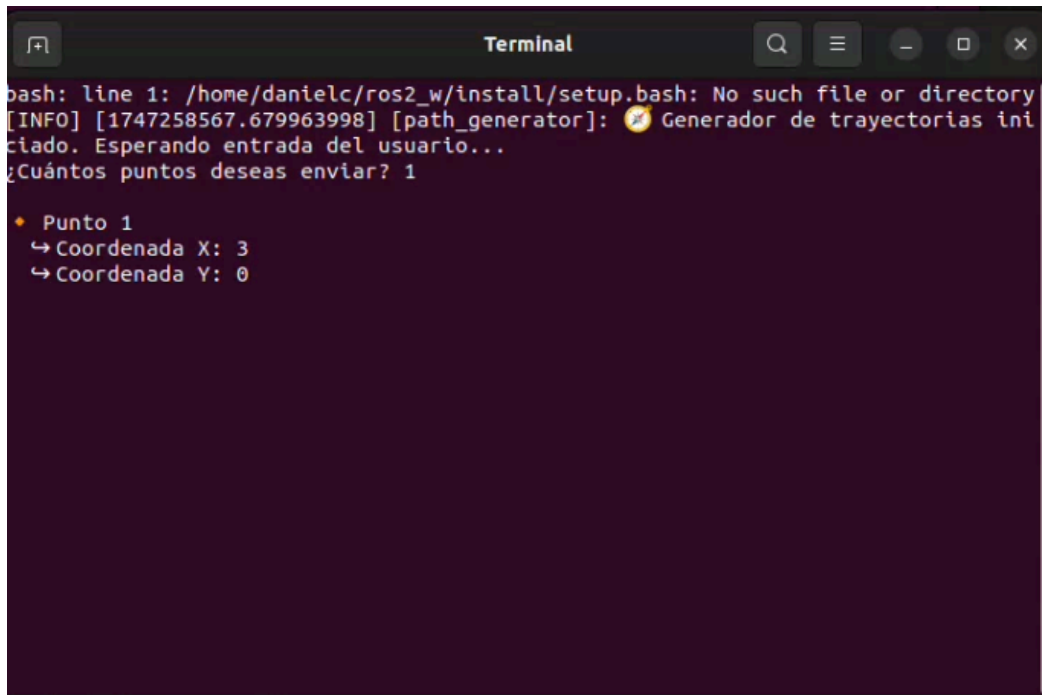


```
bash: line 1: /home/danielc/ros2_w/install/setup.bash: No such file or directory
[INFO] [1747258567.602359392] [controller_pi]: 🚀 Controlador PI con lógica de s
emáforo iniciado.
[INFO] [1747258567.603137880] [controller_pi]: 🕒 Esperando objetivo...
[INFO] [1747258640.885106723] [controller_pi]: 🟢 Cambiando a estado GO
[INFO] [1747258656.085341318] [controller_pi]: 🟡 Cambiando a estado SLOW
[INFO] [1747258659.285255219] [controller_pi]: 🔴 Cambiando a estado STOP
[INFO] [1747258678.485657622] [controller_pi]: 🟢 Cambiando a estado GO
[INFO] [1747258685.484981102] [controller_pi]: 🟡 Cambiando a estado SLOW
[INFO] [1747258699.286238168] [controller_pi]: 🔴 Cambiando a estado STOP
[INFO] [1747258722.685541558] [controller_pi]: 🟢 Cambiando a estado GO
```

Figura 2: Terminal controller_PI

Como presenta la Figura 2 nuestro nodo controller_PI, muestra los valores actuales de nuestra máquina de estados previamente definida en color_detection, haciendo énfasis que nuestro nodo al iniciar el valor actual de nuestra máquina el 1 siendo rojo (STOP), lo cual consiste en que nuestro robot espera la luz verde para dar inicio al movimiento.

Si notamos los diferentes cambios se cumple una robustez haciendo que evite algún falso en reconocer otro color, del verde es necesario pasar al amarillo para poder reconocer el color rojo, esto con la finalidad de seguir un patrón que sea mucho más fácil su detección.

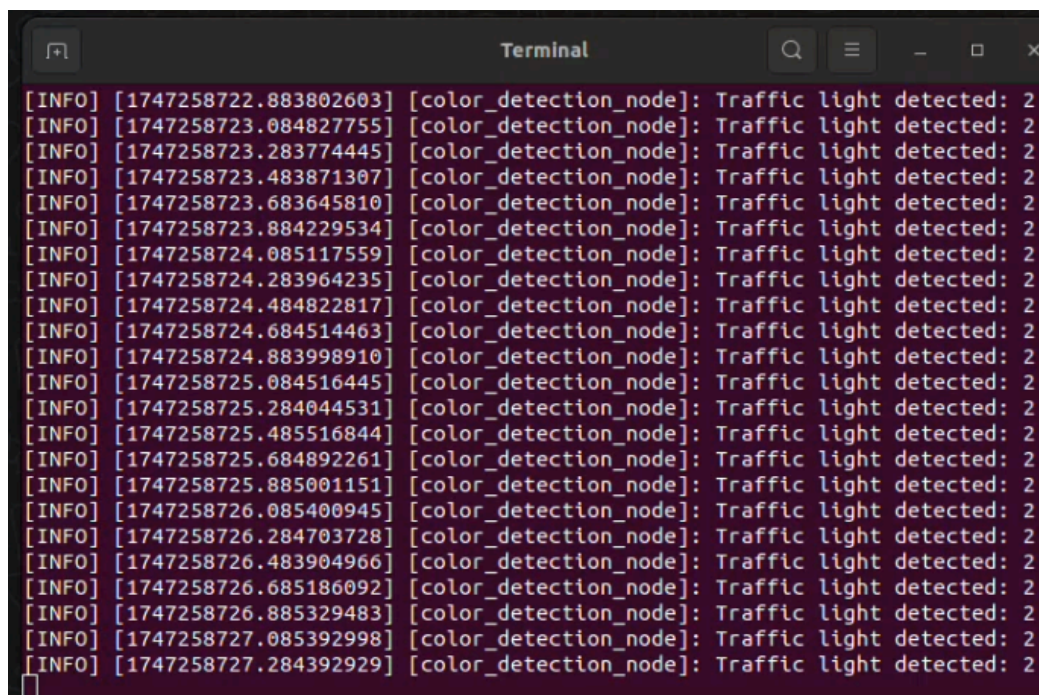


```
bash: line 1: /home/danielc/ros2_w/install/setup.bash: No such file or directory
[INFO] [1747258567.679963998] [path_generator]: 🚦 Generador de trayectorias ini
ciado. Esperando entrada del usuario...
¿Cuántos puntos deseas enviar? 1

• Punto 1
  ↳ Coordenada X: 3
  ↳ Coordenada Y: 0
```

Figura 3: Terminal Path_generator

Al iniciar nuestro Path_Generator como muestra la Figura 3, primero empieza en preguntar al usuario la cantidad de puntos que decida enviar, esto para establecer una ruta clara sobre el camino que va estar siguiendo, después establece el punto en el que podremos definir los valores en X y Y, dichas coordenadas están en metros, para el movimiento de nuestro robot.



```
[INFO] [1747258722.883802603] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258723.084827755] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258723.283774445] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258723.483871307] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258723.683645810] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258723.884229534] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258724.085117559] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258724.283964235] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258724.484822817] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258724.684514463] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258724.883998910] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258725.084516445] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258725.284044531] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258725.485516844] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258725.684892261] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258725.885001151] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258726.085400945] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258726.284703728] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258726.483904966] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258726.685186092] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258726.885329483] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258727.085392998] [color_detection_node]: Traffic light detected: 2
[INFO] [1747258727.284392929] [color_detection_node]: Traffic light detected: 2
```

Figura 4: Terminal Color_Detection

Nuestra Figura 4 muestra como el color_detection indica en que valor está nuestra máquina de estados, confirmando los colores que después se enviará al controller, esto es útil al poder establecer que es lo que necesitamos que se vea para pruebas individuales.

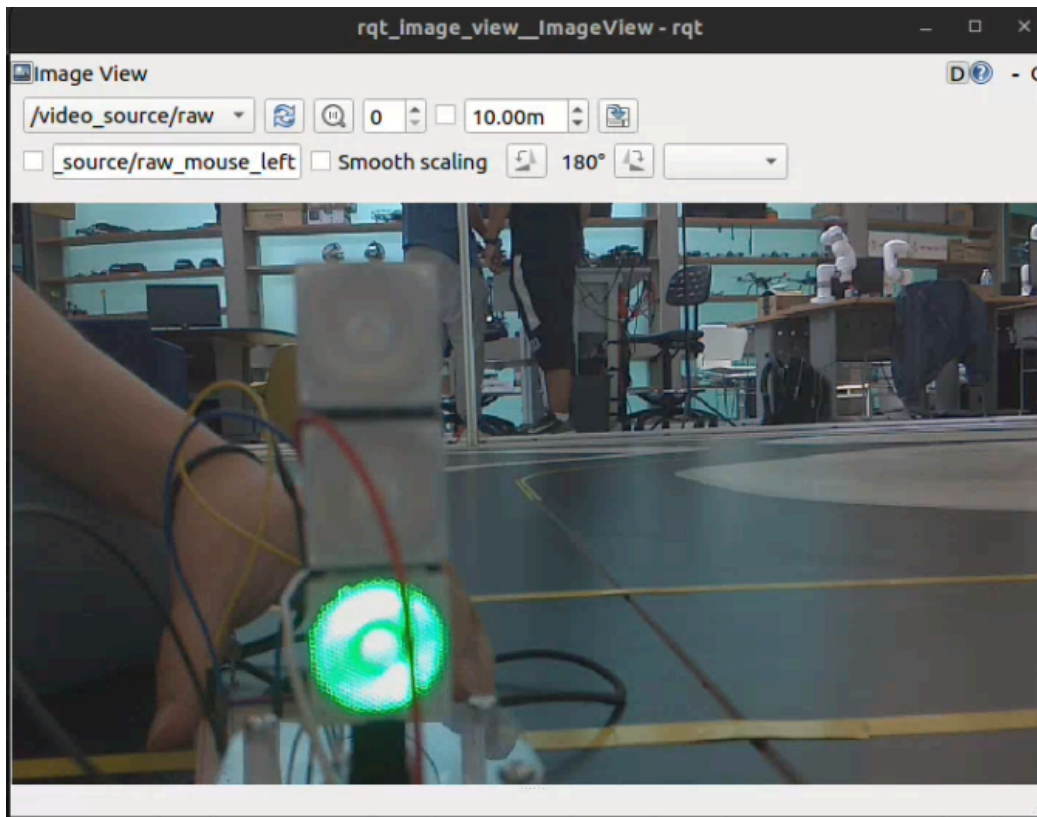


Figura 5: image_view canal /video_source/raw

Lo que nos muestra la Figura 5 es como desde la perspectiva de la cámara del puzzlebot se encuentra actualmente la luz de nuestro semáforo, esto siendo con un ligero retraso, pero lo suficiente para tener una visión sobre los colores que están actualmente

El video explicativo del reto (<https://youtu.be/bXYmZne3Mdw>), presenta una demostración completa del sistema desarrollado en este reto. En él, se observa al robot seguir una trayectoria definida por el usuario, quien introduce múltiples puntos objetivo mediante consola. A lo largo de la ejecución, se evidencia cómo el controlador PI guía al robot hacia cada punto, reaccionando correctamente ante los cambios de color del semáforo detectados por visión computacional. La grabación permite validar tanto la precisión del seguimiento de trayectoria como la robustez del sistema ante variaciones del entorno y errores de odometría, destacando el correcto funcionamiento del control en lazo cerrado.

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.

El desarrollo del presente reto integró exitosamente diversos componentes clave de la robótica móvil, combinando el control en lazo cerrado, la odometría y la visión por computadora en una solución funcional y autónoma para un robot tipo Puzzlebot. A través del uso de ROS 2 y una arquitectura modular, se logró implementar un sistema capaz de detectar los colores de un semáforo y actuar en consecuencia, simulando una situación realista de navegación urbana.

La odometría permitió estimar la posición del robot mediante información de los encoders, mientras que el controlador PI ofreció una estrategia robusta para guiar al robot hacia múltiples objetivos, compensando errores acumulativos y garantizando una navegación estable. Por su parte, el sistema de visión implementado sobre una Jetson Nano demostró ser eficiente y suficientemente robusto frente a variaciones del entorno, permitiendo identificar de manera confiable los colores rojo, amarillo y verde mediante segmentación en HSV y filtrado morfológico.

El enfoque de diseño empleó buenas prácticas como la calibración empírica, el uso de tópicos bien definidos y el procesamiento en paralelo, lo que facilitó la depuración y validación de cada módulo. En conjunto, la solución desarrollada representa un ejemplo claro de cómo combinar teoría, simulación y práctica en robótica para abordar problemas de percepción y control, acercando al estudiante a los desafíos reales en el diseño de robots inteligentes y autónomos.

Referencias

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

ManchesterRoboticsLtd. (s. f.-g).

*TE3002B_Intelligent_Robotics_Implementation_2025/Week3/Presentations/PDF/MC
R2_Closed_Loop_Control_v3.pdf at main ·
ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025.*

GitHub.

https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation_2025/blob/main/Week3/Presentations/PDF/MCR2_Closed_Loop_Control_v3.pdf

Freddy. (s. f.). *Implementacion-de-robotica-inteligente-2025/Implementación de Robótica*

*Inteligente (sesion 5).pptx at main ·
freddy-7/Implementacion-de-robotica-inteligente-2025.* GitHub.

[https://github.com/freddy-7/Implementacion-de-robotica-inteligente-2025/blob/main/Implementaci%C3%B3n%20de%20Rob%C3%B3tica%20Inteligente%20\(sesion%205\).pptx](https://github.com/freddy-7/Implementacion-de-robotica-inteligente-2025/blob/main/Implementaci%C3%B3n%20de%20Rob%C3%B3tica%20Inteligente%20(sesion%205).pptx)

Freddy. (s. f.-b). *Implementacion-de-robotica-inteligente-2025/Implementación de Robótica*

*Inteligente (sesion 8).pptx at main ·
freddy-7/Implementacion-de-robotica-inteligente-2025.* GitHub.

[https://github.com/freddy-7/Implementacion-de-robotica-inteligente-2025/blob/main/Implementaci%C3%B3n%20de%20Rob%C3%B3tica%20Inteligente%20\(sesion%208\).pptx](https://github.com/freddy-7/Implementacion-de-robotica-inteligente-2025/blob/main/Implementaci%C3%B3n%20de%20Rob%C3%B3tica%20Inteligente%20(sesion%208).pptx)

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/MCR2_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