# RETO SEMANAL 6. Manchester Robotics

Abraham Ortiz Castro
*Dpto. de Ingenierías Tec de Monterrey*
*Tecnológico de Monterrey*
Puebla, México
A01736196@tec.mx

Alan Iván Flores Juárez
*Dpto. de Ingenierías Tec de Monterrey*
*Tecnológico de Monterrey*
Puebla, México
a01736001@tec.mx

Jesús Alejandro Gómez Bautista
*Dpto. de Ingenierías Tec de Monterrey*
*Tecnológico de Monterrey*
Puebla, México
A01736171@tec.mx

Ulises Hernández Hernández
*Dpto. de Ingenierías Tec de Monterrey*
*Tecnológico de Monterrey*
Puebla, México
A01735823@tec.mx

## I. Resumen

Este documento continúa el desarrollo de algoritmos de visión computacional para la detección de elementos de nuestro entorno, en esta ocasión especializándonos en la detección y seguimiento de la línea. Este reto, similar a la detección de semáforos, cambiará su locomoción en función de la línea que detecte, cambiando su velocidad angular en las curvas y manteniendo una velocidad en mapas ruedas igual en las rectas.

El PuzzleBot deberá emplear la cámara y los algoritmos de visión computacional así como del diseño de un controlador que se adapte a las necesidades del entorno propuesto por una línea en una pista previamente diseñada con un inicio y fin de trayectoria, se espera un sistema robusto capas de hacer frente a elementos que puedan interferir en la detección apropiada de la línea.

## II. Objetivos.

- Mejorar la interacción entre PuzzleBot y las herramientas provistas por ROS para crear comandos sólidos aplicados a tareas específicas.
- Diseñar un controlador P, PI o PID para ajustar el movimiento del robot con base al error en la posición en el seguidor de línea.
- Implementar detección de líneas utilizando la cámara integrada al PuzzleBot.
- Cambiar la locomoción del robot en función de la línea de trayectoria.
- Diseñar un control robusto capaz de adaptarse a cambios detectados en la posición y visión del robot.

## III. Introducción.

### A. Control de lazo cerrado para un robot móvil

El control de lazo cerrado o de retroalimentación, es un tipo de control en el que se utiliza la salida del sistema para ajustar la entrada con el objetivo de mantener un sistema en un estado deseado.

Para un robot móvil diferencial, el control de lazo cerrado implica el uso de la retroalimentación para ajustar las velocidades de las ruedas del robot con el fin de mantenerlo en una trayectoria esperada o deseada. Este proceso implica lo siguiente:

- **Modelación del sistema:** Implica entender el cómo las velocidades de las ruedas del robot afectan el movimiento, a partir de las ecuaciones de cinemática que describen al robot.
- **Medición del estado:** Esto puede ser logrado a partir del uso de sensores como el encoder en las ruedas, los cuales nos permiten medir la rotación de las ruedas y con algunos datos más propios del robot diferencial, calcular la distancia recorrida del robot.
- **Cálculo del error:** Conociendo el estado actual del robot, se puede conocer el error de la posición y del ángulo al conocer la diferencia entre esos valores deseados y los valores reales o actuales que conocemos nos dará el error existente para poder lograrlo.
- **Ajuste de las velocidades de las ruedas:** A partir del error, ajustamos las velocidades de las ruedas del robot, para nuestros fines a través de un controlador P, lo cual permitirá ajustar la velocidad de las ruedas en función del error.

[1]

### B. PID aplicado a un robot móvil diferencial

Un controlador PID (Proporcional Integrador Derivativo) es una técnica de control empleada para sistemas de lazo cerrado. Enfocado en un robot móvil diferencial, el PID tendrá la tarea de ajustar las velocidades de las ruedas del robot en función de la medición del error. Cada una de las siglas de PID describe el tipo de control que ejercerán sobre el sistema, estas acciones son:

- **Control Proporcional (P):** Basándose en el error actual del control, si el error medible es grande, el controlador de tipo P intentará corregirlo rápidamente al aumentar la velocidad de las ruedas al producir una salida proporcional al error actual. La constante de proporcionalidad

determina cuánto cambio en la salida se produce por unidad de cambio del error.

- **Control Integral (I):** Este elemento del control se enfoca en los errores pasados, es decir, si el robot ha estado fuera de la trayectoria por un tiempo, el controlador de tipo I comenzará a acumular este error con el tiempo y lo solucionará al ajustar la velocidad de las ruedas, proporcionando así un control de acción lenta, pero contante, ayudando a eliminar el error residual que a menudo ocurre al usar el controlador de tipo P.
- **Control Derivativo (D):** Se basa en la tasa de cambio del error, en otras palabras, si el robot vuelve rápidamente a su trayectoria, el controlador de tipo D disminuirá la velocidad con el fin de evitar sobre ajustes. Este control proporciona una acción rápida y proporcional a la tasa de cambio del error, disminuyendo así la respuesta del sistema y evitando oscilaciones.

Si bien es cierto que el controlador PID puede ser muy efectivo dentro de un robot diferencial móvil, necesita de un ajuste cuidadoso de los parámetros P, I y D para un mejor rendimiento. Aunque los valores se pueden obtener mediante métodos heurísticos, podemos recurrir a técnicas más efectivas como el método Ziegler-Nichols.

Adicionalmente, a lo anterior, no debemos dejar de lado que el control PID asume un modelo lineal del sistema, por lo que ciertos comportamientos del robot móvil de tipo diferencial puede mostrar comportamientos no lineales como consecuencia del deslizamiento de las ruedas o la dinámica del robot. [2]

### *C. Cálculo del error*

Para generar un sistema robusto de navegación y control del robot, el cálculo del error en un robot móvil de tipo diferencial es fundamental, y se basa fundamentalmente en la diferencia entre la posición actual del robot y la posición deseada u objetivo.

El error en un robot diferencial puede abordarse en dos componentes primordiales.

- **Error de posición:**

$$e_p = P_o - P_a \tag{1}$$

  Es la diferencia entre la posición deseada y su posición actual, básicamente, es la diferencia entre las coordenadas de la posición objetivo y las actuales del robot.
- **Error de orientación:**

$$e_\theta = \theta_o - \theta_a \tag{2}$$

  Es la diferencia entre la orientación deseada y su orientación actual, básicamente, es la diferencia entre el ángulo objetivo y actual del robot.

[3]

### *D. Robustez de un controlador*

Se entiende a la robustez del controlador a la capacidad de mantener un rendimiento de forma adecuada pese a las variaciones o incertidumbres en el sistema donde se está ejerciendo el control, se puede entender por ende que un controlador robustez es aquel capaz de manejar variaciones en los parámetros del sistema con el fin de mantener los objetivos, su estabilidad y el rendimiento.

Particularmente en un sistema PID, la robustez hace referencia a la capacidad del controlador para manejar variaciones sin que su rendimiento se reduzca, es decir, pese a que el sistema experimente cambios en sus parámetros, un controlador robusto es capas de adaptarse al cambio mientras continúa dando un control apropiado o mejor dicho, un control que menos sensible a cambios bruscos del sistema. [4]

### *E. Procesamiento de imágenes en la Jetson Nano*

La Jetson Nano es una tarjeta embebida empleada ampliamente en el procesamiento de imágenes y en el desarrollo de visión computacional debido a sus capacidades de GPU y a su compatibilidad con bibliotecas diseñadas para el procesamiento de imágenes tales como OpenCV y CUDA. Aunado a lo anterior, este sistema embebido es capaz de ejecutar varias redes neuronales en paralelo diseñadas para algoritmos de clasificación de imágenes, detección de objetos, segmentación y el procesamiento de lenguaje. [5]

*1) OpenCV:* Open Source Computer Vision Library, mejor conocida como OpenCV es la biblioteca de código abierto más grande del mundo especializada en visión computacional, es por ello que comúnmente es empleada en una gran gama de aplicaciones. Además de realizar varias operaciones de procesamiento de imágenes, cuenta con una gran compatibilidad con muchos sistemas operativos (Android, iOS, Linux, Windows, etc.) y lenguajes de programación (Python, C++, Java, etc.) Algunas de las cosas que OpenCV permite al usuario son.

- Reconocimiento facial.
- Identificar objetos.
- Clasificar acciones humanas en videos.
- Seguir los movimientos de la cámara.
- Seguir objetos en movimiento.
- Extraer modelos 3D de objetos.
- Encontrar similitudes entre imágenes en una misma base de datos.
- Seguir el movimiento de los ojos.

[6]

*2) CUDA:* Compute Unified Device Architecture, o por sus siglas CUDA, es una plataforma de computación en paralelo desarrollada por NVIDIA, destinada a codificar algoritmos que se ejecutan en paralelo en la GPU, lo cual es especialmente útil a la hora de procesar imágenes en donde las operaciones pueden ejecutarse paralelamente a cada pixel. [7]

### *F. Interconexión entre la Jetson y la cámara*

La Jetson Nano de NVIDIA ofrece dos opciones para conectar cámaras: a través de la interfaz CSI (Interfaz de Sensor de Cámara) o mediante USB.

*1) Cámara CSI::* Para la conexión CSI, se debe realizar lo siguiente:

- Comprobar que la Jetson esté apagada y desconectada.
- Localizar el conector CSI en la placa.
- Levantar cuidadosamente la pestaña de plástico del conector.
- Deslizar lentamente el cable de la cámara.
- Vuelve a apretar la pestaña de plástico en su lugar cuidadosamente con el cable.

[8]

*2) Cámara USB::* Solo bastará con conectar a uno de los puertos USB de la Jetson una cámara mediante USB.

Una vez se realizó la conexión, puedes emplear la biblioteca OpenCV (previamente vista) para capturar y procesar imágenes. [8]

### *G. Detección de contornos o formas*

Detectar contornos o formas en una imagen es una tarea frecuente en el procesamiento de imágenes y la visión por computadora. OpenCV ofrece una variedad de funciones diseñadas para simplificar este proceso.

Aunque hay una gran variedad de maneras en las que se puede abordar esta tarea, generalmente se suele seguir una metodología concreta usando OpenCV, la cual es la siguiente:

- **Leer la imagen:** leemos la imagen que contiene las formas usando la función.
- **Convertir a escala de grises:** convertimos la imagen a escala de grises.
- **Aplicar umbral:** aplicamos un umbral a la imagen para obtener una imagen binaria.
- **Encontrar contornos:** encontrar los contornos externos de las formas en la imagen binaria.
- **Aproximar contornos a polígonos:** aproximar los contornos a polígonos con una precisión especificada.
- **Identificar formas:** usamos el número de vértices y la relación de aspecto de los polígonos para identificar las formas y etiquetarlas en la imagen. También podemos usar características propias como excentricidad o compacidad si se trata de detectar círculos o elipses.

[9]

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

La robustez en los sistemas de procesamiento de imágenes se define como la capacidad del sistema para manejar variaciones y perturbaciones en las imágenes de entrada y, a pesar de ello, generar resultados precisos y consistentes.

Por lo general, estas variaciones y perturbaciones suelen ser causadas por varios factores, siendo los más comunes los siguientes:

- **Ruido:** Un sistema robusto debe ser capaz de manejar el ruido y producir resultados precisos, pese a sufrir complicaciones con la calidad de la cámara o la compresión de la imagen.
- **Variaciones de iluminación:** Un sistema robusto debe tener la capacidad de afrontar estas variaciones y producir resultados consistentes.
- **Deformaciones y variaciones de escala:** Un sistema robusto es capaz de reconocer formas y objetos, pese a estos mostrar variaciones en sus formas como consecuencia de la perspectiva o distancia de la cámara.
- **Variaciones en la orientación:** Un sistema robusto debe tener la capacidad de reconocer objetos sin importar su orientación.

Así mismo, un sistema robusto deberá ser capaz de manejar diferentes resoluciones y/o tamaño de imágenes y procesarlas en tiempo real o cercano a ello. [11]

### *I. Seguidor de Línea con cámara*

Se entiende a un seguidor de línea con cámara es una forma avanzada de seguidor de línea con visión artificial, al capturar su entorno y procesarlo para identificar la línea de seguimiento.

*1) Consideraciones:*

- **Captura de imágenes:** El robot deberá contar con una cámara que capture el suelo mientras se desplaza, esta imagen se concentra en la línea y el fondo.
- **Procesamiento de imágenes:** Utiliza algoritmos de visión por computadora para detectar la línea. Algunos algoritmos empleados tradicionalmente implican la conversión de la imagen a escala de grises, la detección de bordes y la identificación de la línea en función de su color o contraste con el fondo.
- **Algoritmos de control:** Una vez que se detecta la línea, el robot ajusta su movimiento para seguir la trayectoria.
- **Corrección de la posición:** Si se desvían demasiado hacia un lado, el algoritmo de control ajusta la dirección para volver a la línea, todo esto en tiempo real.

[12]

### *J. Robustez de un Seguidor de Línea*

Algunos aspectos a considerar para la robustez del robot de seguidor de línea con cámara son:

- **Detección de línea:** Para un algoritmo sólido, se necesitan técnicas avanzadas de procesamiento de imágenes, como la umbralización adaptativa, filtrado de ruido y corrección de distorsiones.
- **Tolerancia a variaciones:** Para un algoritmo sólido es recomendable emplear técnicas de normalización y ajuste de parámetros para adaptarse a diferentes condiciones de pista.
- **Resistencia a obstáculos:** Para un algoritmo sólido, es necesario implementar estrategias de corrección de trayectoria y planificación de movimiento mediante código.
- **Adaptabilidad a diferentes superficies:** Para un algoritmo sólido es necesario implementar una calibración inicial y la capacidad de aprendizaje automático para mejorar la adaptabilidad ante tipos de línea no convencionales.
- **Estabilidad en velocidad y curvas:** Para un algoritmo sólido, un adecuado tunning del control PID permitirá controlar la velocidad y el ajuste de dirección en la navegación sin oscilaciones que afecten su movilidad.

[13]

### K. Algoritmos implementados para el seguimiento de líneas

Existen algoritmos previamente desarrollados especializados en la detección y seguimiento de líneas, algunos de ellos son los que se mencionan a continuación:

- **SIFT (Scale-Invariant Feature Transform):** Detecta y describe características invariantes a la escala en una imagen, especializado en el seguimiento en movimiento y en la caracterización de puntos clave en el objeto.
- **SURF (Speeded-Up Robust Features):** La base del algoritmo es similar a SIFT con la diferencia de ser más rápido, lo que le ha permitido estar presente en diferentes aplicaciones de seguimiento visual.
- **ORB (Oriented FAST and Rotated BRIEF):** Combina la detección rápida de características (FAST) con la descripción de características BRIEF. Es eficiente y adecuado para el seguimiento de objetos en tiempo real.
- **PID (Control Proporcional Integrador Derivativo):** Aunque propiamente no es un algoritmo especializado en este rubro, es necesario para controlar el movimiento, permitiendo el ajuste de la dirección y velocidad del robot.
- **CNN (Convolutional Neural Networks):** Una aplicación más avanzada y robusto implica el uso de Redes Neuronales Convolucionales, las cuales aprenden a detectar líneas y seguirlas, sin embargo, requieren de un conjunto grande de imágenes y de mucho tiempo de entrenamiento.

[14]

## IV. Solución del problema

### A. Lógica de solución

Para dar solución a este problema se planteó la siguiente solución, una lista resumida de los procesos se muestra a continuación.

1) Redimensión de la imagen.
2) Conversión de BGR a escala de grises.
3) Filtro promedio.
4) Binarización por método de Otsu.
5) Operación morfológica de cerrado.
6) Cálculo del centroide de la imagen.
7) Cálculo del centroide de la línea.
8) Cálculo del error.

Primero se optó por cambiar el tamaño de la cámara para reducir los recursos computacionales para el procesamiento de la misma, así como mejorar la respuesta de la jetson, de igual forma se optó por un recorte del eje horizontal eliminando un octavo de los extremos de la imagen, esto con el propósito de reducir las posibilidades de que el robot detecte las líneas externas en el trazo de la pista o que estas puedan afectar el cálculo del centroide, posteriormente se procedió a realizar una conversión de imagen BGR a GRAY para posteriormente realizar un filtro promedio, este con el objetivo de reducir el ruido de la imagen, dado que no se utiliza una cámara de gran calidad, posteriormente se realiza un proceso de binarización por el método de Otsu, ya que dentro las pruebas encontramos que arrojaba los mejores resultados.

Con esta imagen binarizada se procede a hacer una operación morfológica de closing, esto con tal de eliminar partes de la pista que puedan ser detectadas como parte de la línea central. Es con esta imagen procesada que se procede a calcular el error al medir la diferencia del centro de la imagen al centro de la línea. Para esto se cuentan los píxeles de la línea detectada, tomando en cuenta la posición del primer pixel, sumando todos estos y dividiendo por la cantidad de los mismos para así obtener el centroide de la línea. De igual manera se calcula el centro de la imagen y se realiza uma resta para calcular la diferencia o error que existe entre el robot y el centro de la línea.

### B. ROS

Para dar solución a este reto, se optó por la creación de dos nodos, uno de los cuales será encargado del control de la locomoción `controller`, el cual recibe el error y actúa a corde al mismo, y un nodo `error_line` el cual es el encargado de recibir la imagen de la cámara, preocesarla y calcular el error del centro del robot al centro de la línea, valor que será enviado al otro nodo, de igual manera se encarga de parar al llegar a la sección de las líneas punteadas.

### C. Nodo Controller

En este nodo iniciamos con la creación de un suscriber que recibe un tipo de dato Float32 del tópico `error_line`, así como la creación de un publicador, el cual es el encargado de una vez calculados los valores de las velocidades basándonos en el error recibido, los envía a través del tópico `cmd_vel`. Para esto se uilizó un control proporcional con valor de 0.15 para la velocidad angular. Un fragmento del código con la lógica implementada se muestra a continuación:

```
#Siendo self.frenado el dato recibido del tópico 'frenado'
if self.frenado == 1:
            self.velA = 0.0
            self.velL = 0.0

        if self.errorLineal >= -0.05 and self.errorLineal <= 0.05:
            self.velA = 0.0

        else:
            self.velA = self.errorLineal * self.kpTheta

        if self.velA > 0.2:
            self.velA = 0.2


        if self.errorLineal == 0.0:
            self.velA = 0.0
            self.velL = 0.0
        else:
            self.velL = 0.1
```

### D. Error Line

Se inicia por la creación de un suscriber que recibe la imagen de la cámara a través del tópico `'/video_source/raw'`, así como la creación de dos publicadores, el primero de los cuales se encarga de enviar el error calculado como un tipo de dato Float32 a través del tópico `error_line`, siendo el segundo el

encargado de detectar el final de la línea, enviando un valor booleano según sea el caso.

Dentro del mismo nodo se crea un `timer_callback` el cual es el encargado de llamar a la función `line_detection_callback` dentro de la cual se llama a la función `calculoError` que a su vez llama a las funciones correspondientes para redimensionar la imagen, realizar el procesado y el cálculo del error, para posteriormente ser enviado al nodo `controller`.

Los pasos a seguir para el procesamiento de imagen y cálculo del error se pueden observar en la sección [IV-A].

## V. Resultados

Los resultados obtenidos para este reto es el correcto funcionamiento del seguidor de línea, encontrando una metodología exitosa para el cálculo de una variable ('error') que nos permita controlar al robot con base en la información recibida a través de la cámara y su posición respecto a la línea. Logrando un control robusto en el que el robot mantiene una trayectoria adecuada al realizar el seguimiento de la ruta.

De igual manera se logró de manera con área de mejora que el robot se detuviera al detectar el final de la pista, una vez se ha terminado la línea de la trayectoria y se observan las líneas punteadas, aunque presenta un ligero un problema, ya que al avanzar y volver a detectar una línea este inicia una rotación por lo que el ángulo al que se detiene no es paralelo respecto a su trayectoria

A continuación se muestra el funcionamiento del robot seguidor de línea:

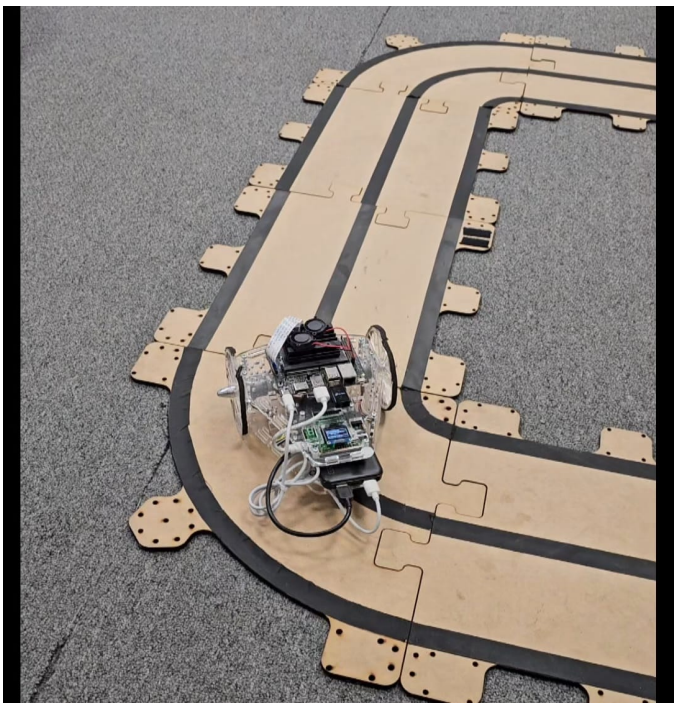

Fig. 1. Video funcionamiento

## VI. Conclusiones

El producto final, presentado esta semana como solución a la problemática de seguidor de línea continua sobre la misma línea de diseño de algoritmos mediante visión computacional para detectar elementos dentro del entorno de trabajo del PuzzleBot, en este caso de líneas indicadas en el suelo que definen la trayectoria a seguir por el robot. En términos generales, los resultados esperados cumplieron con lo esperado y más allá de ello, realizando el seguimiento de la trayectoria de manera estable y sin oscilaciones que afectarán los segmentos rectos tras tomar los curvos existentes. Así mismo, comprobamos un mejor funcionamiento al trabajar de manera local sobre la Jetson en lugar de una comunicación remota mediante la comunicación computador Jetson, eliminando los problemas que el retraso de procesamiento de imagen habíamos notado en prácticas pasadas, brindando una respuesta más rápida y sin tanto retraso, algo que también se vio agilizado debido al enfoque de nuestra solución que mediante el error entre la posición central de la cámara y la del centro de masa de la línea permite hacer una simple resta que en términos computacionales es mucho más ligera que divisiones o multiplicaciones que los otros algoritmos de solución proponían.

Aunado a lo anterior, los resultados obtenidos en conjunto con los previos y los contenidos vistos en el diseño de redes neuronales en módulos de la materia han empezado a encaminar ciertos elementos de la problemática final a una solución funcional y robusta que cumpla con lo esperado en el próximo reto final.

Aunque el sistema de detección de línea y control en torno a ella funciona de manera uniforme y consistente en los elementos rectos y de curvos de la pista, hemos detectado un elemento dentro de la detección y seguimiento de líneas que provoca ciertas anomalías en el desempeño del PuzzleBot, este elemento no es otro que los segmentos de línea no continuos (línea punteada) los cuales si bien se diseñó una forma de hacerles frente en código, aún no interactúa como se esperaría. Esto nos da una ventana de trabajo que creemos poder sobrellevar mediante la implementación de redes neuronales que detecten de mejor manera (en comparativa con el algoritmo de seguimiento de línea) estos sectores y que nos permitan atender de forma eficaz estos sectores, fuera de eso, el resto del algoritmo funciona de forma sólida con configuración de cámara enfocada en el suelo; sin embargo, trabajaremos para una localización más elevada que pueda detectar señales, semáforos y las líneas del recorrido para el siguiente reto.

## References


[1] Castellanos Pérez, M., Medina, A., Álvaro Hernández, S., Lester, S., Maza, A., León, I., y Revisores, O. (s.f.). Instituto Tecnológico de Tuxtla Gutiérrez Reporte Final Residencia Profesional: Desarrollo y Aplicación de Algoritmos de Cooperación en Robots Móviles. link.

[2] Cortés, U., Castañeda, A., Benítez, A., y Díaz, A. (2015). Control de Movimiento de un Robot Móvil Tipo Diferencial Robot link.

[3] Seguimiento de Trayectoria Mediante la Solución del Error Lateral en un Robot Tipo Diferencial. (s.f.). link.

[4] Mariana. (2024, Enero 4). Control robusto. Fisicotrónica. link.

[5] NVIDIA Jetson Nano. (s.f.). NVIDIA. link.

[6] OpenCV. (2024, Mayo 1). OpenCV - Open Computer Vision Library. link.
[7] NVIDIA CUDA toolkit - free tools and training. (s.f.). NVIDIA Developer. link.
[8] Instala OpenCV 4.5 en NVIDIA Jetson Nano — Configura una cámara para Jetson Nano. (2024, Marzo 4). link.
[9] Greyrat, R. (2022, July 5). ¿Cómo detectar formas en imágenes en Python usando OpenCV? – Barcelona Geeks. link.
[10] Administrador. (2020, Febrero 11). Detección de colores en OpenCV – Python (En 4 pasos). OMES. link.
[11] Procesamiento Digital de Imágenes. (Agosto 2006). link.
[12] ¿Qué es un robot seguidor de línea? Spiegato. (2021, Julio 12). link.
[13] A. Pinargote, B., Jean, D., y García, M. (s.f.). UNIVERSIDAD POLITÉCNICA SALESIANA SEDE GUAYAQUIL. Diseño e Implementación de un Robot Seguidor de Línea mediante Visión Artificial. link.
[14] Santos, D., Dallos, L., Gaona García, P. A. (2020). Algoritmos de rastreo de movimiento utilizando técnicas de inteligencia artificial y machine learning. Informacion Tecnologica. link.
[15] ManchesterRoboticsLtd. (s.f.). GitHub - ManchesterRoboticsLtd/TE3002B-2024: Intelligent Robotics Implementation. GitHub. link.