Challenge Final- Manchester Robotics

Diego García Rueda, Jhonatan Yael Martínez Vargas , Jonathan Josafat Vázquez Suárez, Daniela Berenice Hernández de Vicente

[1] *Instituto Tecnológico de Estudios Superiores de Monterrey, Campus Puebla, Puebla, México*
**Autores correspondientes: A01735217@tec.mx,A01734193@tec.mx,A01734225@tec.mx,A01735346@tec.mx*

Publicado el 16/06/2023

En este reporte se pretende explicar el proceso de desarrollo de un robot seguidor de línea mediante procesamiento de imágenes, buscando crear un sistema que involucre tanto elementos de visión, toma de decisiones, control, cinemática y usos elementales de redes neuronales.
En el reporte se mencionan temáticas y procesos de desarrollo que fueron cruciales para el correcto funcionamiento del puzzlebot aplicando los conocimientos adquiridos para poder cumplir con los requerimientos solicitados.

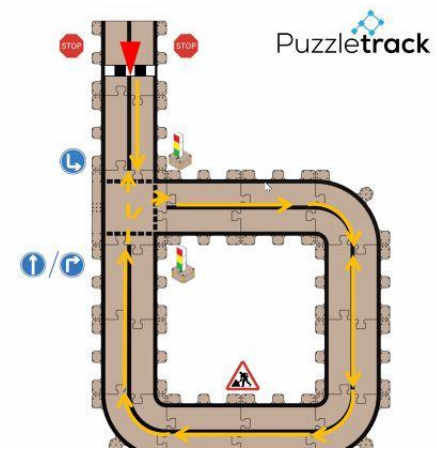


Imagen 1. Pista a recorrer

La imagen 1 hace referencia a la trayectoria final del reto presentado por MCR2, el cual se ha trabajado durante estas últimas semanas, donde el objetivo final es completar la trayectoria haciendo uso de procesamiento de imágenes y video, donde se busca detectar señales de tráfico y semáforos.

---

1. Introducción.

A. Introducción .

A lo largo de la historia la evolución de la tecnología ha sido inevitable, y con ella han surgido muchos retos relacionados con la misma.

Por lo que la robótica se ha vuelto un pilar destacado en cuestión de avances tecnológicos, de igual manera esta puede generar satisfacción al resolver necesidades donde se busca hacer más eficaz una tarea o un proceso determinado.

Como bien sabemos esta abarca bastantes disciplinas como lo es la programación, la mecánica, la mecatrónica, la electrónica, entre otras, siendo esta una ciencia que busca cierta innovación tecnológica.

Por lo que en la actualidad es un concepto que busca la facilidad de la sociedad y su mejora continua por medio de robots inteligentes y autónomos.

2. Objetivos.

A. Objetivos Generales.

A continuación se presentan los objetivos generales, donde se describe la función general con la que debe cumplir el challenge final. [Presentación Manchester]

- En este challenge se pretende que el estudiante revise y comprenda los conceptos introducidos en las sesiones anteriores.
- Este challenge tiene como objetivo enseñar el comportamiento de los sistemas de visión y procesamiento de imágenes en los robots móviles.
- Este challenge busca el manejo autónomo del puzzlebot con la trayectoria definida por Manchester Robotics.

B. Objetivos Específicos.

En este apartado se presentan los objetivos específicos del challenge final, donde se pretende explicar más a fondo lo que se espera realizar durante el mismo.

A continuación se establecen los requerimientos para la Pista final:

- La pista está conformada por diferentes piezas de MDF ensambladas entre sí en forma de “b”.
- Para este Challenge se utilizan tres diferentes piezas, la pieza recta, la pieza curva y la pieza de intersección.
- Cada una de las piezas debe estar compuesta por 3 líneas con ciertas medidas, como se puede observar en la imagen 2.

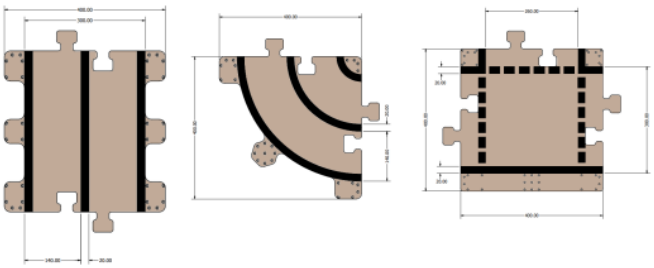


Imagen 2. Medidas específicas de cada pieza.

El comportamiento esperado durante el trayecto es el siguiente:

- Se espera que el estudiante complete la trayectoria definida por las señales de tráfico y los semáforos.
- El puzzlebot debe reconocer las señales de tráfico (Imagen 3) utilizadas durante la trayectoria final.

Imagen 3. Señales de tránsito implementadas

- El reconocimiento de las señales de tráfico debe ser mediante el uso de sistemas de visión.
- El *Puzzlebot* debe reconocer de manera autónoma los semáforos durante la trayectoria.
- El reconocimiento de los semáforos debe ser mediante el uso de sistemas de visión.
- El algoritmo de visión de seguimiento de línea y el controlador de circuito cerrado deben ser ***robustos***.
    - El estudiante debe definir qué significa robusto e implementar estrategias para lograr el éxito con el controlador.
- El algoritmo de visión, los algoritmos de seguimiento de línea y el controlador deben estar sintonizados apropiadamente.
- El controlador debe tomar en consideración los cambios del semáforo, cambios en las señales, perturbación, no linealidades y el ruido.
- Es recomendado, pero no requerido, que el estudiante use un archivo de configuración o un parámetro en el archivo de lanzamiento para establecer los objetivos de modo que puedan cambiarse fuera del código (no codificados).

A continuación se establecerá el comportamiento deseado con respecto a los semáforos:

- El comportamiento del algoritmo debe tomar en consideración el challenge de los semáforos, procediendo según sea el color de la luz.
    - Luz Roja: Detenerse hasta que aparezca una luz verde.
    - Luz Amarilla: Avance despacio hasta que aparezca una luz roja para detenerse completamente.
    - Luz Verde: Continúe por el camino.

De igual manera se presenta el comportamiento deseado con respecto a las señales de tráfico:

- Como se mencionó anteriormente el puzzlebot debe detectar y obedecer las señales de tráfico (véase Imagen 3) presentadas durante la trayectoria establecida.
    - **Ahead Only:** Continúe recto en la dirección indicada.
    - **Turn Right/Left Ahead:** Gire según sea el sentido de la flecha.
    - **Roundabout:** Circulación de rotonda. Reduzca la velocidad y de prioridad a los vehículos dentro de la rotonda, posteriormente ingrese en ella.
    - **Roadwork Ahead:** Reduzca su velocidad.
    - **Stop:** Deténgase por completo.
    - **Give way:** Ceda el paso al tráfico en las carreteras principales/ reduzca la velocidad al encontrar el cruce.

3. Unidades Funcionales

3.1 Puzzlebot.

"Un laboratorio en un robot", esta es una herramienta para el aprendizaje independiente y formación profesional, siendo una placa de desarrollo versátil, integrada y dedicada a la robotica.[1]

3.1.1 Características del Robot.

El *Puzzlebot* forma parte de la familia de robots diferenciales, los cuales requieren diferentes sensores y actuadores con la finalidad de leer la información del ambiente y con ello poder interactuar.

El cerebro del *Puzzlebot* es la placa de desarrollo de NVIDIA, una Jetson Nano, en este caso se utilizó un modelo con 2 GB de RAM, esta actúa como un nodo computacional robusto para componentes iniciales y una interfaz de comunicación a componentes externos (laptops, teléfonos, etc).

El diagrama de bloques de puzzlebot se compone de varios elementos. [véase imagen 4]

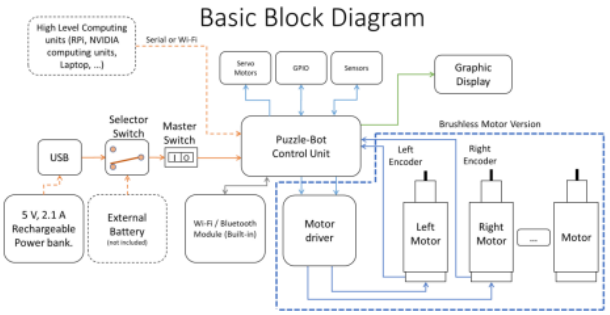


Diagrama 1. Diagrama de bloques del *Puzzlebot*

La configuración básica del robot se compone de dos motores sin escobillas (versión obsoleta) o motores con escobillas (según la versión), con dos encoders (uno por motor), una pantalla gráfica para mostrar información básica sobre el estado del robot, una batería recargable de iones de litio.

Aunado a esto cada motor tiene un controlador integrado para controlar la velocidad mediante señales PWM.

De igual manera contiene un microcontrolador el cual incluye comunicación WIFI y Bluetooth integrada. Por lo que se pueden conectar sensores y actuadores adicionales en los pines libres del microcontrolador (servomotores, sensores de reflectancia IR, sonares, etc.).

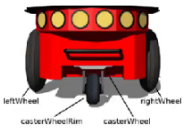
Es importante destacar que las unidades informáticas externas como (Raspberry pi, Nvidia Jetson, CPU) se pueden conectar a el robot usando WIFI o comunicación en serie.

Aunado a esto se utiliza una cámara **Raspberry Pi camera V2,** con la finalidad de obtención de video. [2]

3.2 Cinemática.

El modelo cinemático directo relaciona las velocidades del punto de interés **h(x,y)** con las velocidades de los actuadores considerando al vehículo como una masa puntual, es decir sin analizar fuerzas que ejercen sobre el mismo. [3]

Como se mencionó anteriormente este modelo proporciona únicamente velocidades, por lo que para obtener la posición y orientación del vehículo es necesario integrar las velocidades en un periodo de tiempo constante (**tiempo de muestreo**). [3]



$$h_x = x_1$$

$$h_y = y_1$$

$$\dot{h}_x = \mu \cos\varphi$$

$$\dot{h}_y = \mu \sin\varphi$$

$$\dot{\varphi} = \omega$$

$$\underbrace{\begin{bmatrix}\dot{h}_x\\ \dot{h}_y\\ \dot{\varphi}\end{bmatrix}}_{\dot{\mathbf{h}}} = \underbrace{\begin{bmatrix}cos\varphi & 0\\ \sin\varphi & 0\\ 0 & 1\end{bmatrix}}_{\mathbf{J}} \underbrace{\begin{bmatrix}\mu\\ \omega\end{bmatrix}}_{\dot{\mathbf{q}}}$$

Imagen 4. Análisis geométrico.

Como se mencionó anteriormente este modelo proporciona únicamente velocidades, por lo que para obtener la posición y orientación del vehículo es necesario integrar las velocidades en un periodo de tiempo constante (**tiempo de muestreo**). [3]

Por lo que en resumen la cinemática diferencial directa es la derivada con respecto al tiempo de la cinemática directa.

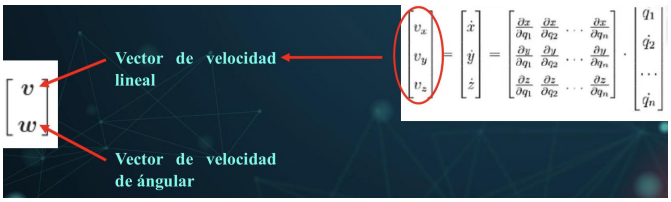


Imagen 5. Vectores de velocidad angular y lineal.

Por lo que en resumen la cinemática diferencial directa es la derivada con respecto al tiempo de la cinemática directa.

Un robot móvil con estas características normalmente cuenta con 3 grados de libertad respecto a una referencia, siendo posición en el plano (**x,y**) y **orientación (roll, pitch and yaw).**

Es importante destacar que todos los ajustes de orientación utilizados solo se realizan en **yaw**.

3.3 Control.

3.2.1 Control en lazo cerrado para un robot móvil.

Un robot móvil se presenta como un control de lazo cerrado, el cual puede realizar un cambio de posición de sus elementos, esto en función de la información captada de su entorno mediante sus sensores. [4]

Es importante reconocer que es casi imposible tunear el controlador para que este genere una trayectoria perfecta hacia la posición deseada en el robot. Ante esta situación también es importante reconocer que al no ser exactamente perfecto, se busca generar una tasa de error muy baja. [4]

Es importante destacar que este es un sistema que utiliza fundamentalmente la retroalimentación para mantener una variable controlada, al contar con la medición de dicha variable, esta se compara con el valor de referencia dado anteriormente, y cualquier diferencia entre estos dos valores es utilizada para ajustar el controlador empleado.

3.2.2 Control aplicado a un robot móvil diferencial y los temas derivados al cálculo del error.

Como anteriormente se mencionó la programación en ROS no ha sido lo más importante durante este proyecto, aunque sí es la parte que predomina en este proyecto, ya que la gran parte del proyecto es la creación de los nodos y la comunicación entre ellos, pero el sistema implementado no podría funcionar correctamente sin un controlador que satisfaga las necesidades del mismo.

Por lo que para poder implementar el controlador adecuado para el sistema, fue necesario comprender los tipos de controladores existentes y en qué momento es necesario utilizarlos, al igual que la función de cada una de las variables de los mismos, de igual manera es importante conocer que el diseño del sistema de control consistirá en elegir adecuadamente la localización de los ceros, polos y la ganancia K, por otro lado gracias al controlador podemos llegar a tener cierto tiempo muerto, por lo que es importante conocer el significado de dicho concepto. [5]

Ganancia de Proceso(K). Es la relación de cambio de salida al cambio de la variable de entrada. Esta define más específicamente la sensibilidad que tiene la variable de salida a un cambio dado en la variable de entrada. Dicho comportamiento se muestra matemáticamente en la ecuación 1. [4]

$$K = \frac{\Delta\ \text{Output}}{\Delta\text{input}}$$

Ecuación 1.

Esta solo puede describirse como un parámetro de estado estacionario y es independiente de las variables de diseño y operación. Cuenta con solo tres componentes, el signo, el valor y las unidades.

El primero indica cómo responderá la salida a la entrada del proceso. El segundo depende del proceso que se esté considerando, por lo que dependen de las variables que se mencionan.

Tiempo Muerto (t0). Sucede cuando se da el cambio en una variable de entrada y cuando inicia la variable de salida. Cabe mencionar que es muy importante, porque afecta a la controlabilidad de nuestro sistema de control, por lo que un cambio en el punto de ajuste no suele ser inmediato gracias a este parámetro. De igual manera, este tiempo muerto debe ser siempre considerado para los procesos de afinación y modelado. [5]

Control Proporcional. Es una forma de control de retroalimentación, siendo la forma más fácil de control continuo que se puede utilizar en un sistema de lazo cerrado. También minimiza la fluctuación en la variable de proceso, el lado negativo es que no siempre lleva al sistema al punto de ajuste que se desea. [5]

Por otro lado proporciona una respuesta más rápida que la mayoría de los otros controladores, lo que permite inicialmente que el controlador solo P llegue a responder unos segundos más rápido. Aunque el controlador P ofrece como ventaja un tiempo de respuesta más rápido, llega a producir una desviación del punto de ajuste, la cual se conoce como desplazamiento, y es algo que no se desea durante un proceso.

La existencia de ella implica que el sistema no podría mantenerse en el punto de ajuste deseado en estado estacionario. Este se puede minimizar combinando el control P con algún otro controlador (I o D)

Por lo que el controlador P correlaciona la salida del controlador con el error. Dicho comportamiento se muestra matemáticamente en la ecuación 2.

$$c(t) = K_c e(t) + b$$

Ecuación 2.

Control Integral. Es una segunda forma de control de retroalimentación. Usualmente se utiliza porque es capaz de eliminar cualquier desviación que pueda existir, de esta manera el sistema vuelve tanto al estado estacionario como a su configuración original. [5]

Por lo tanto, un error negativo hará que la señal al sistema disminuya, mientras que un error positivo hará que la señal aumente. Sin embargo este tipo de controladores suelen ser más lentos en su tiempo de respuesta que los controladores P solo porque dependen de más parámetros. Este tiempo de respuesta se puede aumentar si es combinado con algún otro controlador (P o D). De igual manera suelen usarse por separado cuando las variables medidas deben permanecer dentro de un rango muy estrecho y requieren un ajuste más fino.

Estos suelen afectar al sistema al responder a errores pasados acumulados, por lo que su filosofía es que las desviaciones se verán afectadas en proporción a la suma acumulada de su magnitud. Ahora bien, una gran ventaja de estos es que se eliminara el desplazamiento, mientras que sus grandes desventajas son que puede llegar a desestabilizar el controlador, y que hay una windup del integrador, lo cual aumenta el tiempo que tarda el controlador en realizar ciertos cambios.

El control correlaciona la salida del controlador con la integral del error, por lo que la integral del error se toma con respecto al tiempo. Es el error total asociado a lo largo de una cantidad de tiempo especificada. Dicho comportamiento se muestra matemáticamente en la ecuación 3.

$$c(t) = \frac{1}{T_i} \int e(t)dt + c\,(t_0)$$

Ecuación 3.

Control Derivativo. Es una forma de control de avance, ya que este anticipa las condiciones del proceso analizando el cambio en el error. Este funciona para minimizar el cambio de error, de esta forma mantiene el sistema en una configuración consistente. Uno de sus principales beneficios es que resisten el cambio en el sistema, siendo el más importante de estas oscilaciones. Ahora bien, la salida del control se calcula en base a la tasa de cambio del error con el tiempo, por lo que mientras más grande sea la tasa de cambio de error, más pronunciada será la respuesta del controlador. [5]

A diferencia de los otros controladores, el derivativo no guía al sistema a un estado estacionario, ahora bien, debido a esto se recomienda que deben estar combinados con los controladores P, I o PI para poder controlar de manera correcta el sistema.

El control D correlaciona la salida del controlador con la derivada del error. La derivada del error se toma con respecto al tiempo. Dicho comportamiento se muestra matemáticamente en la ecuación 5.

$$c(t) = T_d \frac{de}{dt}$$

Ecuación 5.

Control PI. El control PI es una forma de control de retroalimentación. Este proporciona un tiempo de respuesta más rápido que el control debido a la suma de la acción proporcional, aunque sigue siendo hasta un 50% más lento que el control P. Por lo que con el fin de aumentar el tiempo de respuesta, el control PI suele combinarse con el control D. De igual manera este impide que el sistema fluctúa, y es capaz de devolver el sistema a su punto de ajuste. [5]

Como se ha mencionado el control PI correlaciona la salida del controlador con el error y la integral del error. Dicho comportamiento se muestra matemáticamente en la ecuación 4.

$$c(t) = K_c \left(e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{de}{dt} \right) + C$$

Ecuación 4.

Control PD. El control PD es una combinación entre control de avance y retroalimentación, debido a que opera tanto en las condiciones actuales del proceso como en las condiciones del proceso predichas . En este controlador, su salida es una combinación lineal de la señal de error y su derivada. De igual manera contiene la amortiguación del control proporcional de la fluctuación y la predicción del error del control derivado. [5]

Como se ha mencionado el control PD correlaciona la salida del controlador con el error y la derivada del error. Dicho comportamiento se muestra matemáticamente en la ecuación 5.

$$c(t) = K_c \left(e(t) + T_d \frac{de}{dt} \right) + C$$

Ecuación 5.

Control PID. Este es el más utilizado, ya que combina las ventajas de cada tipo de control. Por lo tanto tenemos un tiempo de respuesta más rápido debido al control P, junto con el desplazamiento decrecido/cero de los controladores D e I. Es importante destacar que el desplazamiento anteriormente mencionado se elimina mediante el uso adicional del control I. [5]

La adición del controlador D aumenta en gran medida la respuesta del controlador cuando se usa en combinación con

los demás, ya que predice la respuesta del sistema midiendo el cambio en el error. Sin embargo, aunque el controlador PID parece ser el controlador más adecuado, también es el más caro. Por lo tanto, realmente no se utiliza a menos que el sistema necesite esa precisión y estabilidad. [5]

Por lo que el controlador PID correlaciona la salida del controlador con el error, siendo estos el integral del error y derivada del error. Dicho comportamiento se muestra matemáticamente en la ecuación 6.

$$c(t) = K_c \left(e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{de}{dt} \right) + C$$

Ecuación 6.

Es importante destacar que para nuestra implementación utilizamos un control PD adaptativo por lo que este es un conjunto de técnicas que permiten ajustar en tiempo real el valor de los parámetros de control, lo que permite contar con un buen seguimiento de las variables controladas aunque se desconozca los parámetros de la planta o estos cambien en el tiempo. [6]

3.5 Navegación.

Los sistemas de navegación que utilizan una única cámara como sensor para controlar vehículos autónomos se basan en la visión por computadora y algoritmos de procesamiento de imágenes. Estos sistemas permiten al vehículo "ver" su entorno y tomar decisiones en función de la información visual capturada por la cámara. A continuación, se presenta el proceso a seguir para poder lograr buenos resultados:

1. Adquisición de imágenes: La cámara captura imágenes del entorno en tiempo real.

2. Preprocesamiento de imágenes: Las imágenes capturadas se someten a un procesamiento previo para mejorar la calidad y eliminar el ruido. Esto puede incluir operaciones como la corrección del balance de blancos, la eliminación de distorsiones y la reducción de ruido.

3. Detección de objetos: Utilizando algoritmos de detección de objetos, se identifican y se localizan diferentes elementos en la imagen, en este caso señales de tránsito, líneas de carril y semáforos. Esto se logra mediante técnicas como detección de características, aprendizaje profundo (deep learning) y clasificación de objetos.

4. Seguimiento de objetos: Una vez que se detectan los objetos de interés, lo que se debe de hacer es un seguimiento de su movimiento en el tiempo. Esto permite al sistema comprender cómo se están moviendo los diferentes elementos en el entorno y predecir su trayectoria futura.

6. Toma de decisiones y planificación de rutas: Con la información recopilada de la detección y el seguimiento de objetos, el sistema de navegación puede tomar decisiones sobre la velocidad, dirección y acciones del vehículo.

7. Control del vehículo: Una vez que se ha tomado una decisión y se ha planificado una ruta, se envían comandos de control al vehículo para ajustar la aceleración, dirección y frenado. Estos comandos se basan en la información visual capturada por la cámara y procesada por el sistema de navegación.

Es importante tener en cuenta que los sistemas de navegación basados en una única cámara tienen limitaciones en comparación con los sistemas más complejos que utilizan múltiples sensores, como lidar y radar. La cámara puede tener dificultades para funcionar en condiciones de iluminación adversas, como la oscuridad o la luz solar intensa, y puede tener problemas para detectar objetos transparentes o con poca textura. Sin embargo, a través de técnicas de procesamiento de imágenes avanzadas y algoritmos de aprendizaje automático, los sistemas de navegación basados en cámaras han demostrado ser efectivos en muchos escenarios y continúan mejorando con el tiempo.

3.6 Visión Computacional.

Durante este reto se manejo el uso de tarjetas embebidas como lo es la tarjeta Jetson Nano, para realizar un procesamiento de imágenes, esto debido a la gran capacidad de procesamiento que tiene, permitiendo así realizar múltiples operaciones a la imagen de manera rápida y eficiente.

Para el procesamiento de imágenes fue necesario el uso de OpenCV, siendo una librería open-source de machine learning y visión artificial. Los algoritmos que la integran son más de 2500, los cuales incluyen machine learning y visión artificial. Estos permiten identificar objetos, clasificar imágenes con respecto a una base de datos, hacer tracking de movimiento de objetos, entre otras cosas. [5]

3.6.1 Interconexión entre tarjeta Jetson Nano y la cámara.

Siendo esta la conexión entre la tarjeta NVIDIA y la cámara que el *Puzzlebot* posee, se utiliza un archivo .launch proporcionado por MCR2 que permite inicializar la cámara y enviar las imágenes recibidas por un tópico llamado *video_source_raw* que posteriormente utilizando la librería *cv_bridge* permitirá utilizar librerías de OpenCV a la imagen recibida, esta imagen recibida será procesada y modificada por el nodo de visión, la estructura de este proyecto será explicada más adelante. Un ejemplo de imagen obtenida mediante este archivo launch puede observarse en la imagen 5.

Imagen 6. Ejemplo imagen obtenida por el *Puzzlebot*

3.6.2 Segmentación de la imagen mediante Regiones de Interés.

Es importante mencionar que el procesamiento de la imagen recibida durante este reto, no es aplicado a toda la imagen recibida por parte de la cámara del robot, es decir que se seleccionan específicamente ciertas regiones de interés (ROI, Region Of Interest).

Donde una región de interés (ROI) es una parte de la imagen que se desea filtrar o procesar de alguna manera. Es el área de una imagen que tiene cierta importancia o interés para un procesamiento en específico de esa zona, en el procesamiento

de una imagen puede haber más de una región de interés, esto es de suma importancia debido a que en este challenge se utilizaron 5 ROI. Una ROI se puede representar como máscara binaria. En la máscara, los píxeles que pertenecen a la ROI tienen el valor 1, y los que se sitúan fuera tienen el valor 0. La toolbox ofrece varias opciones para especificar una ROI y crear máscaras binarias.[6]

Se decidió implementar esto por el desarrollo que se explica en los siguientes puntos, simplificando el proceso, el tener 5 ROI nos permite "segmentar la pista" de manera que cuando se ubique el carril se pueda ubicar un centroide que represente el carril (sección …) y con esto crear un ajuste cuadrático que permita predecir una trayectoria que el robot debe de seguir.

Las dimensiones de la imagen recibida son de 1280 x 720 píxeles, las 5 ROI se encuentran en las siguientes coordenadas (asignadas en pixeles):

# ROI	Px Horizontales / Px Verticales
ROI - 1	360:432, 0:1279
ROI - 2	432:503, 0:1279
ROI -3	503:576, 0:1279
ROI - 4	576:648, 0:1279
ROI - 5	648:719, 0:1279

Tabla 1. Posición en píxeles de cada una de las Regiones de Interés.

Los valores anteriores muestran que todas las ROI recorren todo el ancho de la imagen y las 5 se encuentran una sobre otra a lo largo de la mitad inferior de la imagen iniciando en el pixel 360 (mitad de la imagen).

3.6.3 Centroides

El principio básico con el que la cámara del robot es capaz de identificar la trayectoria es mediante el uso de centroides. Un centroide es el punto que representa el centro geométrico de un objeto, este punto nos permite conocer información como la ubicación espacial del objeto coordenadas X,Y, la obtención de este punto se basa en la suma de las coordenadas de todos los píxeles del objeto dividida por el número total de píxeles. Algunas aplicaciones de los centroides son las siguientes:

1. Segmentación: El centroide puede ayudar a dividir una imagen en objetos individuales al encontrar el centro de cada objeto. Esto puede ser útil en aplicaciones como la detección y seguimiento de objetos.
2. Caracterización de objetos: El centroide proporciona información sobre la posición espacial de un objeto. Puede utilizarse como una característica para describir y comparar objetos en una imagen.
3. Control de movimiento: En aplicaciones de visión por computadora relacionadas con el seguimiento y control de movimiento, el centroide de un objeto puede ser utilizado para calcular desplazamientos y determinar la dirección del movimiento.

3.7 Aprendizaje.

3.7.1 Redes Convolucionales.

Una red neuronal convolucional (CNN), también conocida como Convolutional Neural Network en inglés, es un tipo de modelo de aprendizaje profundo utilizado ampliamente en el procesamiento de imágenes. Estas redes están diseñadas para imitar el funcionamiento del sistema visual humano y son especialmente eficientes en el reconocimiento visual. Básicamente las CNN utilizan filtros convolucionales para detectar características locales en diferentes partes de una imagen. Estos filtros se aplican a través de capas convolucionales, donde cada filtro se encarga de extraer características específicas, como bordes, texturas o formas, mediante operaciones de convolución. Estas operaciones generan mapas de características, que resaltan las regiones de la imagen donde se encuentran esas características.

A medida que los datos fluyen a través de la red, se agregan capas convolucionales adicionales para detectar características cada vez más complejas y abstractas. También se suelen utilizar capas de agrupación entre las capas convolucionales para reducir la dimensionalidad de los mapas de características y extraer características invariantes a pequeñas variaciones en la posición de los objetos en la imagen, finalmente, se incluyen capas totalmente conectadas para realizar la clasificación de las imágenes. Estas capas reciben los mapas de características y aprenden a asignar etiquetas o categorías a las imágenes. Durante el entrenamiento, la red ajusta los pesos de los filtros y las conexiones para minimizar el error en la clasificación de las imágenes de entrenamiento.

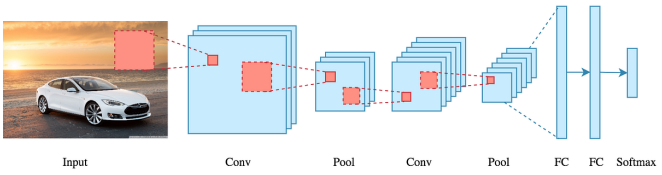


Imagen 7 Estructura básica de un red neuronal convolucional Aplicaciones De Las Redes Neuronales Convolucionales - kulturaupice

3.7.2 Detección de imágenes por CNN.

Una vez que hemos comprendido qué es una CNN, podemos explorar cómo este tipo de redes nos permiten detectar objetos en tiempo real. En nuestro caso, decidimos utilizar YOLO v5 debido a su velocidad y precisión en esta tarea. El funcionamiento de YOLO v5 se basa en dividir la imagen en una cuadrícula y predecir las regiones de interés y las clases de objetos en cada celda de la cuadrícula. A diferencia de otros enfoques, YOLO v5 realiza estas predicciones directamente, sin necesidad de utilizar regiones de interés propuestas.

La arquitectura de YOLO v5 consta de un backbone formado por capas convolucionales, que extraen características de la imagen a diferentes niveles de abstracción. Estas características se utilizan para generar predicciones de detección en múltiples escalas, lo que nos permite mejorar la detección de objetos de diferentes tamaños. Para lograr esto, el modelo utiliza una pirámide de características que fusiona la información de diferentes niveles de la red, capturando así detalles a diversas escalas. YOLO v5 utiliza una técnica llamada "atención espacial" la cual le permite aumentar la precisión de las predicciones, pues con esta técnica la red se enfoque en las regiones más relevantes de la imagen y presta mayor atención a los detalles importantes, mejorando así la detección de objetos pequeños.

La capacidad de YOLO v5 para realizar detecciones en tiempo real, sin comprometer demasiado la precisión, es una de sus

principales ventajas. Esto resulta especialmente útil en aplicaciones como conducción autónoma y detección de objetos en vídeos en tiempo real, donde es fundamental contar con respuestas rápidas. Por si fuera poco implementar y entrenar YOLO v5 es sencillo, ya que se proporciona un repositorio público con el código fuente y los pesos pre entrenados. Esto facilita que la arquitectura pueda ser probada y adaptada a las necesidades específicas de detección de objetos en tiempo real.

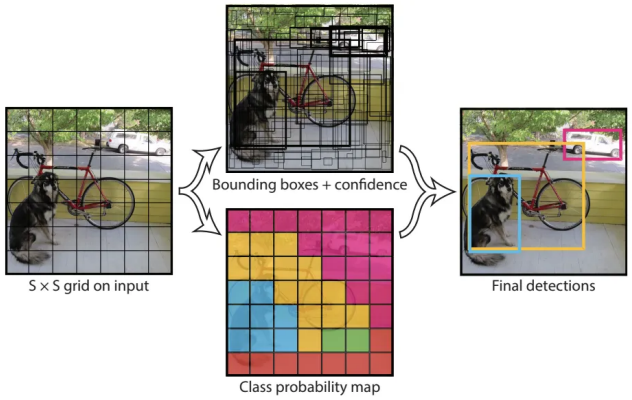


Imagen 8. Proceso de trabajo de una red YOLO v5 YOLO (You Only Look Once). A simple explanation of a real-time… | by Kevin Rexis Velasco | Towards Data Science

3.7.3 Robustez en sistemas de procesamiento de imágenes

El sistema de procesamiento de imágenes de este proyecto mostró ser bastante robusto, buscando el equilibrio entre un correcto procesamiento de las imágenes y un tiempo de procesamiento rápido para la obtención de las imágenes en tiempo real, el procesamiento se realiza para el seguimiento de la trayectoria y para la detección de semáforos, el procedimiento será descrito en la siguiente sección.

4.Interacción Inteligente.

Cada uno de los conceptos anteriormente mencionados se han utilizado para la resolución de manera práctica de los diferentes problemas que han surgido durante las 10 semanas del bloque.

4. 1 Arquitectura del sistema.

La estructura de este proyecto buscó ser lo más sencilla posible utilizando únicamente 3 nodos conectados mediante *ROS*

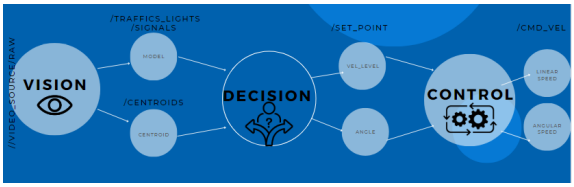


Diagrama 2. Arquitectura de la solución en ROS

4.2 Protocolo de comunicación interna.

En este siguiente apartado se busca explicar y desglosar el funcionamiento interno de cada uno de los módulos y elementos implementados para la solución de este reto.

4.2.1 Implementación de visión computacional.

Primero se explicará el proceso de la obtención de centroides realizando un procesamiento a la imagen recibida, obteniendo así la información del camino y la ruta a seguir, posteriormente se explicará el proceso en el cual se realiza la identificación de señales y de semáforos. Toda esta información será utilizada por el nodo de decisión para definir el comportamiento del robot.

4.2.1.1 Binarización de la imagen

Debido a que se busca identificar un carril de color negro se convierte la imagen a escala de grises, esto con motivo de que el color negro sea más sencillo de detectar unicamente aplicando un umbralizadoque permita generar una binarización detectando unicamente los carriles sin objetos inneccesariso que puedan generar ruido al sistema.

Imagen 9. Binarización inversa de la imagen detectando colores negros

4.2.1.2 Limpieza de la detección

En la imagen 6 se puede observar que se detectan ciertos elementos que no corresponden al carril, estas son las separaciones que existen entre cada pieza de la pista, para la limpieza de estos objetos se realiza un condicional en el que a partir del área de los objetos detectados es que se detectarán los objetos como carril. El valor del área que se usa como condicional es 450, este valor fue obtenido a base de prueba y error y concreto buenos resultados.

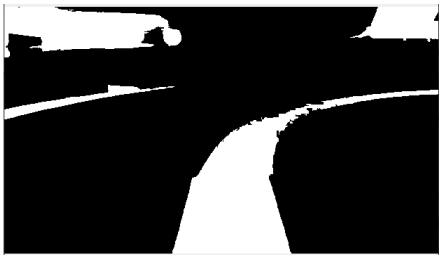


Imagen 10. Binarización inversa de la imagen tras limpieza.

4.1.2.3 Obtención de centroides

Tras haber realizado la limpieza de la binarización, como se puede apreciar en la imagen 7, se procede a obtener los centroides de cada uno de los cuerpos detectados en cada una de las ROI, de haberse realizado correctamente el proceso se marcará el camino de manera adecuada para posteriormente poder realizar el modelo de la ruta de manera adecuada.

A partir de los puntos de los centroides, dependiendo de la cantidad detectada puede ser de 0 hasta 5, se genera un modelo cuadrático de predicción

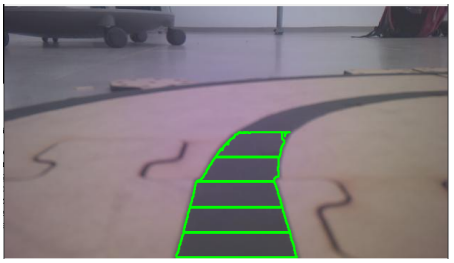


Imagen 11. Zonas de camino detectado a partir de las Regiones de Interés

4.1.2.3 Obtención de modelo cuadratico

Debido a que el objetivo principal de nuestro proyecto era diseñar el robot para que pudiera realizar el recorrido de la manera más rápida posible es que se implementó este sistema predictivo a manera de poder realizar la planeación de trayectoria considerando el tiempo de procesamiento y la velocidad del robot.

Utilizando los centroides detectados se establecen valores para A, B y C, esto siguiendo la fórmula de una ecuación cuadrática, ecuación 7.

$$y = ax^2 + bx + c$$

Ecuación 7.

Estos 3 valores son los que se envían al nodo de decisión para realizar el modelo de predicción necesario para definir las velocidades y otros aspectos del Puzzlebot.

4.2.2 Implementación de la Red Neuronal.

Para la detección de objetos después de explicar el tipo de modelo utilizado y cómo funciona. Lo primero que se hizo fue crear una base de datos utilizando alrededor de 120 fotografías de señales y semáforos, tomadas desde el robot en movimiento desde diferentes perspectivas. Esto se hizo con el objetivo de mejorar los resultados del entrenamiento del modelo.
Sin embargo, se determinó que 120 imágenes no eran suficientes, por lo que se implementó un código[10] para realizar el aumento de datos (data augmentation). Este código generó seis imágenes distintas por cada una de las imágenes originales, aplicando diferentes filtros como "shift", "zoom", "gaussian blur", "noise", "contrast" y "brightness". Así, se obtuvieron 48 variaciones de cada imagen original, lo que resultó en un conjunto de datos de 4320 imágenes diferentes.

Luego, se realizó manualmente el etiquetado de cada objeto en cada imagen utilizando la herramienta "labelimg". Este proceso resultó tedioso y largo, por lo que del conjunto de datos de 4320 imágenes, únicamente se utilizaron 1674 imágenes.

Posteriormente, la base de datos se exportó a la plataforma web "Roboflow". Esto permitió separar los datos en tres carpetas: "Training", "Validation" y "Test", con 1200, 325 y 188 imágenes respectivamente.

Con la base de datos preparada, se procedió a entrenar el modelo, para esto el primer paso fue clonar localmente el repositorio de YOLOv 5. Luego, se ingresó al repositorio clonado y se ejecutó un comando para iniciar el entrenamiento con los parámetros definidos. [véase imagen 9]

```
!python3 train.py
--img 640
--batch 6
 --epochs 30
 --data /home/yamajo/yolov5/Signal_detection-2/data.yaml
 --weights yolov5s.pt --cache
```

Imagen 12. Comando para entrenar el modelo de YOLO v5

Es importante mencionar que la ruta especificada como "data" se refiere al paquete descargado de "Roboflow", y debe ser movido dentro del repositorio local antes de ejecutar el comando, además de esto el número de épocas y baches no se usó demasiado grande por dos razones, la primera es que el modelo se iba a entrenar de manera local en una computadora y dos queríamos evitar un overfitting, ya que si bien 1674 imagenes son mas que 120, seguimos considerando que no son las suficientes para entrenar un modelo muy preciso.

Finalmente una vez entrando el modelo, lo que se hizo fue validar los resultados, por lo que se implementó otro script [11] el cual en un código normal de python carga los datos de los pesos obtenidos en la etapa de entrenamiento y los usa como base poder trabajar con la imagen en tiempo real , obteniendo así los siguientes resultados:

Imagen 13. Modelo CNN detectando señales

Imagen 14. Modelo CNN aplicado para detección de señales

4.2.3 Implementación del modelo de trayectoria.

Debido a la existencia de la pieza del cruce (imagen 2) es que se plantearon múltiples modelos de cómo es que el robot se comporta, para esto se definieron 3 estados, el estado 0 es cuando no se detecta ningun carril negro en la pista, ocurriendo unicamente en ciertas partes del cruce, el estado 1 es el resto de la pista, cuando

4.2.4 Implementación de decisión.

Como su nombre lo dice, este nodo es el encargado de tomar la decisión de qué camino debe seguir el robot. Para esto, empleamos dos cosas: en primer lugar, los tópicos de '/signals' y '/traffic_lights' se encargan de establecer el nivel de velocidad al que el robot debe moverse para alcanzar su próximo estado. Para ello, multiplicamos el valor actual de la velocidad por un número entre 0 y 100. El valor de este factor de escalamiento se determina según las siguientes reglas:

- Si se identifica una señal de "stop" o "red", el factor de escala es 0.
- Si se identifica una señal de "left", "right", "workers" o un semáforo en estado "yellow", el factor de escala es 50.
- Finalmente, si se identifica "forward" o "green", el factor de multiplicación es 100.

Cabe resaltar que esto solo modifica el nivel de velocidad al que debe moverse el robot. Para indicar la siguiente orientación que el robot debe tener en su próximo estado, se aplican nuevamente 3 casos:

- En caso de que el robot sea capaz de detectar una trayectoria recta, la siguiente orientación se determina mediante el modelo cuadrático propuesto para predecir y ajustarse a la trayectoria detectada por el robot.

- Si el robot detecta una curva, nuevamente se ajustan los coeficientes del modelo que predice este tipo de trayectorias.

- Finalmente, si el robot detecta que está en un cruce, se ajustan los coeficientes de la trayectoria para que sean similares en proporción a los que tenía el robot en alguna curva al momento de detectar una línea. Esto es válido solo porque se sabe que la pista tiene dimensiones fijas, por lo cual solo es necesario identificar correctamente en qué sentido debe dar la vuelta y ajustar los coeficientes del modelo.

El primer y el segundo estado es para el caso 1 en los que la cámara únicamente encuentra un centroide, es decir, se encuentra en la trayectoria normal, en este caso el robot funciona como un seguidor de línea, con los centroides detectados se obtienen modelos cuadráticos y en base a estos modelos cuadráticos es que se obtienen ángulos que son los que el robot tiene que ir siguiendo, se desarrollan 4 ángulos para hacer que las trayectorias sean lo más suaves posibles y en caso de curva, el robot intente igualar la trayectoria lo mejor posible en lugar de hacer únicamente una trayectoria recta.

```
self.ang[0] = np.round(np.arctan2(160, x_2 - x_1) * 180 / np.pi - 90, 2)
self.ang[1] = np.round(np.arctan2(120, x_3 - x_2) * 180 / np.pi - 90, 2)
self.ang[2] = np.round(np.arctan2(80, x_4 - x_3) * 180 / np.pi - 90, 2)
self.ang[3] = np.round(np.arctan2(40, x_5 - x_4) * 180 / np.pi - 90, 2)
```

Imagen 15. Cálculo de ángulos a partir de los modelos cuadráticos.

Específicamente del último caso, los coeficientes utilizados para las opciones existente del cruce para la elaboración del modelo cuadrático son los siguientes:

```
self.coeff = [-6.830424097526668e-19, 1.5206532033710582e-15, 639.9999999999992]
self.coeff_curv_l = [ 2.63871310e-03,  2.11350180e-01, -1.04474683e+03]
self.coeff_curv_r = [-3.08541632e-03,  3.96166189e-01,  2.12105016e+03]
```

Imagen 16. Coeficientes del modelo cuadrático para el cruce

Cada uno de estos coeficientes permite crear un modelo cuadrático para posteriormente poder obtener los ángulos de giro necesarios de acuerdo al cruce y la indicación de la señal siendo movimiento en recto, giro a la izquierda y giro a la derecha respectivamente.

El comportamiento en el cruce y el uso de los coeficientes ya definidos se usará cuando el robot detecte un espacio vacío (caso 0) posteriormente detecte la línea punteada (caso 2) y posteriormente detecte nuevamente un espacio vacio(caso 0).

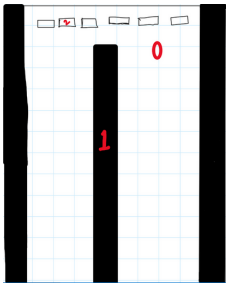


Imagen 17. Casos existentes en la pista.

Estos coeficientes para el cruce se definieron poniendo el robot en una pieza curva derecha e izquierda para obtener valores de modelos cuadráticos en los que ya sabemos que el robot gira correctamente debido a que las piezas siempre son las mismas, de esta manera en el cruce con estos coeficientes ya definidos se simulan la spiezas de la pista recta y las curvas hacia las 2 direcciones.

4.2.5 Implementación del controlador.

Para lograr los ajustes de velocidad lineal y angular necesarios para que el robot se ajuste a la velocidad y orientación deseada, se implementaron dos controladores de tipo PD en el nodo de control.

El primer controlador PD se encarga de ajustar la velocidad lineal del robot. El componente proporcional considera la diferencia entre la velocidad deseada y la velocidad actual del robot, y genera una señal de control proporcional a ese error. El componente derivativo tiene en cuenta la velocidad de cambio del error y ayuda a mejorar la respuesta del controlador, con esto lo que se espera es que a medida que el robot se acerca a la velocidad deseada, el controlador disminuye la señal de control para evitar oscilaciones excesivas.

El segundo controlador PD se encarga de ajustar la velocidad angular del robot, aquí el error se calcula como la diferencia entre la velocidad angular deseada y la velocidad angular actual del robot. El controlador ajusta la señal de control para reducir este error y lograr la orientación deseada.

Además, se ha implementado un enfoque de controlador adaptativo. Esto significa que los valores de las constantes Kp (ganancia proporcional) y Kd (ganancia derivativa) se ajustan automáticamente en relación a la velocidad del robot. Esto es importante porque la respuesta del control debe ser proporcional a la velocidad de movimiento del robot, y no solo al error entre su estado actual y el estado siguiente. Al adaptar las constantes Kp y Kd, el controlador puede proporcionar una respuesta más precisa y adecuada para diferentes velocidades.

Para garantizar la estabilidad del sistema, se han definido límites superiores e inferiores para los ajustes de velocidad lineal y angular. En el caso de la velocidad angular, se ha establecido un rango máximo de giro permitido entre dos estados sucesivos, que no debe superar las 70 unidades. Esto evita que el robot realice giros bruscos y se mantenga dentro de límites controlables. Para la velocidad lineal, se ha establecido un rango entre 0 y 120 unidades, lo que garantiza que el robot no se desplace a velocidades excesivas y pueda ser controlado de manera segura.

4.3 Protocolo de comunicación externa

El protocolo más utilizado para la interacción con el Puzzlebot fue SSH, este permite su uso, programación y ejecución de nodos de manera remota. Para la ejecución de nodos, acceso y control de ROS de manera remota se utilizó *ROS_MASTER* permitiendo ejecutar comandos principalmente para la modificación de los códigos que conforman los nodos de la estructura.

Uno de los paquetes de ROS más utilizados en este proyecto fue *teleop_keyboard*, un paquete de teleoperación que permitía el control de las velocidades lineales y angulares del robot utilizando teclas del teclado de manera remota, permitiendo mandar comandos de movimiento, esta herramienta fue bastante útil ya que permite realizar pruebas con el *Puzzlebot* para el cálculo de las velocidades angular y lineal máximas que soporta el robot sin reiniciarse, dato necesario para el control del sistema y evitar que este provoque un reinicio inesperado.

5. Conclusión.

El diseñar un sistema de seguimiento de línea utilizando únicamente la cámara como sensor principal, fue una tarea bastante compleja, sobre todo porque siempre se ha estado lidiando las diferentes exposiciones de luz a las que está expuesta el robot en diferentes horas del día, o diferentes marcas que pueden ser confundidas por el robot como parte de la misma línea a seguir. Por lo que en un inicio para solucionar solamente el seguidor de línea se implementó un algoritmo mediante los rangos de valores HSV, sin embargo para la detección en tiempo real entre más procesamiento se le haga a imagen original, más pesado y lento va a ser el nodo que realice esta acción, por lo que realmente al basarnos en un algoritmo en base a centroides, se realiza un trabajo más eficiente y eficaz. Además de que diseñar una estrategia que nos permitiera indicarle al robot cual es la dirección y los pequeños ajustes que este tiene que hacer a su ángulo de orientación mediante la cámara también resultó ser un trabajo complicado de implementar.

Aunado a esto consideramos que si se realizó de manera correcta una implementación adecuada de acuerdo con los requerimientos del reto, sin embargo evidentemente las dos áreas de oportunidad más grandes que encontramos durante su solución, fueron las siguientes.

1. **Implementación del algoritmo de la red neuronal dentro de la tarjeta Jetson nano:** Realmente el problema más relevante que enfrentamos fue la dificultad de incluir la red neuronal a la tarjeta Jetson Nano, por que realmente el funcionamiento de ambas partes era correcto, lo sucedido fue un conflicto entre librerías, versiones de python y/o versiones de las mismas librerías con ciertas versiones de python.
2. **Materiales/condiciones:** De igual manera consideramos que los materiales tanto del robot como de la pista, no eran los más óptimos, en la pista realmente esta contaba con ciertas manchas, que si bien hicieron que crearamos un algoritmo más robusto, de alguna manera si no hubieran existido, hubiera sido más sencillo. Aunado a esto los materiales del robot fue sin duda algo que causó mucho conflicto, porque las llantas de muchos se barrieron e hicieron que fuera más difícil el recorrido, porque estas perdían tracción, por otro lado de igual manera la cámara no era la mejor y en ocasiones detecta en una mala calidad las imágenes y hace algo más complicado su procesamiento.

Destacamos que de alguna manera si no hubieran existido estos inconvenientes todo hubiera sido más sencillo, sin embargo la idea igual era tener robustez en el mismo, por lo que consideramos un éxito la implementación.

6. Referencias.

1. ManchesterRoboticsLtd. (s. f.-a). TE3001B_Robotics_Foundation/Week 1/Presentations/MCR2_ROS_Introduction.pdf at main · ManchesterRoboticsLtd/TE3001B_Robotics_Foundation. GitHub. https://github.com/ManchesterRoboticsLtd/TE3001B_Robotics_Foundation/blob/main/Week%201/Presentations/MCR2_ROS_Introduction.pdf
2. Manchester Robotics [MCR2]. (s. f.-b). Puzzle-Bot (2.1).
3. Edisonsasig. (2023). Modelo cinemático y simulación de un robot móvil diferencial. Roboticoss. https://roboticoss.com/modelo-cinematico-y-simulacion-con-python-robot-movil-diferencial/
4. ManchesterRoboticsLtd. (s. f.-b). TE3002B_Intelligent_Robotics_Implementation/Week 8/MCR2_Final_Challenge.pdf at main · ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation. GitHub. https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation/blob/main/Week%208/MCR2_Final_Challenge.pdf
5. ManchesterRoboticsLtd. (s. f.-a). TE3001B_Robotics_Foundation/Week 1/Presentations/MCR2_ROS_Introduction.pdf at main · ManchesterRoboticsLtd/TE3001B_Robotics_Foundation. GitHub. https://github.com/ManchesterRoboticsLtd/TE3001B_Robotics_Foundation/blob/main/Week%201/Presentations/MCR2_ROS_Introduction.pdf
6. Embention. (2021, 20 septiembre). Control adaptativo y sus aplicaciones en la industria. Embention. https://www.embention.com/es/news/control-adaptativo-y-sus-aplicaciones/#:~:text=%C2%BFQu%C3%A9%20es%20el%20control%20adaptativo,o%20cambien%20en%20el%20tiempo.
7. Manchester Robotics [MCR2]. (s. f.). Closed Loop Control_v2. Github. Recuperado 25 de abril de 2023, de https://github.com/ManchesterRoboticsLtd/TE3002B_Intelligent_Robotics_Implementation/blob/main/Week%203/Challenges/MCR2_Mini_Challenge2.pdf
8. Libretexts. (2022, 2 noviembre). 9.2: Control de P, I, D, PI, PD y PID. LibreTexts Español. https://espanol.libretexts.org/Ingenieria/Ingenier%C3%ADa_Industrial_y_de_Sistemas/Libro:_Din%C3%A1mica_y_Controles_de_Procesos_Qu%C3%ADmicos_(Woolf)/09:_Control_proporcional-integral-derivado_(PID)/9.02:_Control_de_P,_I,_D,_PI,_PD_y_PID
9. OpenCV. (2023, 24 mayo). OpenCV - Open Computer Vision Library. https://opencv.org/
10. Procesamiento basado en ROI - MATLAB & Simulink - MathWorks América Latina. (s. f.). Procesamiento basado en ROI - MATLAB & Simulink - MathWorks América Latina
11. Color luz y color pigmento. (s. f.). https://www.fotonostra.com/grafico/colorluzpigmento.htm
12. ¿Qué son las redes neuronales convolucionales? | IBM. (s. f.). https://www.ibm.com/mx-es/topics/convolutional-neural-networks

7. Anexos.

1. Repositorio de GitHub: https://github.com/JM-Yamajo/Puzzlebot_Challenges
2. Link del video explicativo: https://www.youtube.com/watch?v=pgI7NoFbu68