RETO SEMANAL 4. 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 trabaja sobre los resultados de los anteriores desafíos referentes a la locomoción y control del PuzzleBot, particularmente centrado en la implementación de la detección de formas y colores con la finalidad de ajustar la velocidad de nuestro PuzzleBot en función del color del semáforo detectado.

El procesamiento de imagen en el contexto del PuzzleBot se enfoca en la detección de formas y colores para ajustar su velocidad según el semáforo detectado. Esto implica varias etapas: primero, la adquisición de la imagen del entorno; luego, el preprocesamiento para mejorar la calidad de la imagen. Posteriormente, se emplean algoritmos de detección de formas para identificar posibles semáforos, seguido de la segmentación de colores para determinar su estado. Con esta información, el PuzzleBot puede interpretar el semáforo y tomar decisiones apropiadas, como detenerse ante el rojo o continuar ante el verde. Este proceso requiere ajustes continuos para garantizar una detección precisa y una respuesta adecuada a las condiciones del entorno.

A continuación se presentan los resultados a los que llegamos al implementar nuestros algoritmos de detección de formas y colores y su comportamiento junto al controlador previamente desarrollado con ajustes mínimos en función de la detección obtenida.

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.
- Implementar detección de objetos utilizando la cámara integrada al PuzzleBot.
- Cambiar la locomoción del robot en función del color detectado por la cámara frontal del robot.
- 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. Detección de colores

Detectar colores en imágenes es una tarea habitual en el procesamiento de imágenes y la visión por computadora. OpenCV, una biblioteca popular para tales tareas, ofrece funciones específicas para simplificar este proceso. Específicamente, el espacio de color HSV (Hue, Saturation, Value / Matiz, Saturación, Brillo) resulta especialmente útil para la detección de colores.

El espacio de color HSV (Matiz, Saturación, Valor) es preferido sobre RGB para la detección de colores debido a su capacidad para separar los componentes del color de manera más intuitiva, su invariancia al brillo, su eficiencia computacional y su robustez frente a cambios de iluminación.

Al igual que con la detección de formas, la detección de colores sigue un esquema principal para lograr dicha tarea, este proceso es:

- **Leer la imagen:** leemos la imagen a analizar.
- **Convertir a escala de colores HSV:** convertimos la imagen a escala de colores HSV.
- **Aplicar umbral:** definimos los rangos de colores que queremos detectar en la escala HSV y aplicamos un umbral a la imagen para obtener una imagen binaria.
- **Encontrar contornos:** encontrar los contornos de las formas en la imagen binaria a resaltar.
- **Dibujar contornos y mostrar imagen:** dibujamos los contornos de las formas detectadas en la imagen original y mostramos la imagen resultante.

[10]

I. 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]

IV. Solución del problema

Para la solución de este reto se contruye sobre las bases previamente desarrolladas para las entregas anteriores, incluyendo la creación de tres nodos `odometry` el cual es el encargado de recibir los valores de la velocidad angular de cada una de las llantas y convertir dichos datos a desplazamiento en 'x', 'y' y theta, valores que posteriormente son enviados por un tópico del mismo nombre. El nodo `path_generator` el cual es el nodo encargado de enviar las coordenadas correspondientes para cada una de las figuras, desde un

triángulo hasta un hexágono, cuyas coordenadas se encuentran dentro de un círculo con diámetro 1, estas coordenadas son enviadas de igual forma por otro tópico con el mismo nombre. Por último el nodo `controller`, recibe los valores del tópico `odometria` y del `path_generator`, así como de nuestro nuevo tópico `color_detection`, el cual manda un entero dependiendo el color detectado del semáforo, tomando esta información, calculando la posición actual, la posición objetivo, el ángulo actual y el ángulo objetivo. Para calcular el error como se muestra en las ecuacuiones 1 y 2. Así como determina la velocidad lineal que debe tomar el robot, tomando en cuenta la luz del semáforo detectado.

A. Nodo Controller

Segmentos de código implementado para el nodo `Controller` se muestran a continuación:

```
#Se hacen las suscripciones pertinentes
        self.subscription_odometria = self.create_subscription(
            Vector,
            'odometria',
            self.signal_callback1,
            rclpy.qos.qos_profile_sensor_data )

# Callback para recibir la posición
actual del robot
    def signal_callback1(self, msg):
        if msg is not None:
            self.Posx = msg.x
            self.Posy = msg.y
            self.Postheta = msg.theta
```

En el código anterior se crea el suscriber que escucha al tópico 'odometria' con tipo de dato 'Vector', el cual contiene la posición actual del robot en los ejes 'x', 'y' y theta, para que posteriormente dentro de la función `signal_callback1` sean asignadas a otras variables locales.

```
#Se hacen las suscripciones pertinentes
        self.subscription_path = self.create_subscription(
            Path,
            'path_generator',
            self.signal_callback2,
            rclpy.qos.qos_profile_sensor_data )

def signal_callback2(self, msg):
    if msg is not None:
        self.trayectoria = [(0,0),(msg.x1, msg.y1),
        (msg.x2, msg.y2),  (msg.x3, msg.y3),
        (msg.x4, msg.y4), (msg.x5, msg.y5),
        (msg.x6, msg.y6)]
```

Dentro de esta sección de código se crea el suscriber que escucha al tópico `path_generator` que contiene las distintas coordenadas de cada una de las figuras para ser guardadas dentro de un vector desde el cual las accesaremos posteriormente.

```
#Se hacen las suscripciones pertinentes
        self.subscription_light = self.create_subscription(
            Int32,
            'color_detection',
            self.signal_callback_traffic,
            rclpy.qos.qos_profile_sensor_data )

def signal_callback_traffic(self, msg):
    if msg is not None:
        if msg.data == 0:
            self.color_traffic_light =
            self.color_traffic_light
        else:
            self.color_traffic_light = msg.data
```

Dentro de esta sección de código se crea el el suscriber que escucha al tópico `color_detection` el cual manda un entero, 1 para el color verde, 2 para el color amarillo y 3 para el color rojo del semáforo detectado por la cámara, el nodo `collor_detection` encargado de enviar estos datos se explicará más adelante.

Función Timer Callback de nuestro nodo controller:

```
#Timer para realizar los calculos correspondientes
self.timer = self.create_timer(0.1, self.timer_callback)

def timer_callback(self):
    [...]
    # Obtener las coordenadas del punto actual
    en la trayectoria
    target_x, target_y = self.trayectoria
    [self.indice_punto_actual+1]

    target_x_ant, target_y_ant =
    self.trayectoria[self.indice_punto_actual]

    # Calcular las coordenadas polares del punto objetivo
    self.distancia = math.sqrt((target_x - self.Posx)**2 +
    (target_y - self.Posy)**2)
    self.angulo_objetivo =
    numpy.arctan2(target_y-target_y_ant,
    target_x-target_x_ant)

    self.errorTheta = self.angulo_objetivo - self.Postheta
```

Dentro de nuestra función `timer_callback` se toman los valores obtenidos de nuestros suscribers y se realizan las operaciones para el cálculo de las coordenadas actuales del robot, las coordenadas objetivo, el valor actual de theta y su valor objetivo. Para que con las restas de los mismos se obtenga el error de tanto la posición, así como del ángulo. De igual manera se envían los valores del error a través de un tópico con el mismo nombre.

Se llevo a cabo la implementación del control, en nuestro caso decidiendo por un control 'Proporcional' ya que observamos daba resultados correctos para las trayectorias planteadas. Con valores para la variable proporcional de 0.25 para la velocidad angular. Siendo la implementación de un control 'PI' o 'PID' una de las áreas de oportunidad a mejorar en un futuro. Ejemplo de la implementación se muestra a continuación:

```
#Valor de k
    self.kpTheta = 0.25

#Control Angular
    self.PTheta = self.kpTheta*self.errorTheta

#Velocidad angular
self.velA = self.kpTheta*self.errorTheta
#Rango superior V.angular
if self.velA > 0.15:
    self.velA = 0.15
```

Dentro de la misma función `timer_callback` de nuestro nodo `controller` según los datos previamente recibidos del tópico `color_detection` se envían los valores de velocidad lineal correspondientes al color detectado, con valores de 0.2 para el color verde, 0.1 para el color amarillo y 0 para el color rojo. Un área de mejora en este apartado es la implementación de un control para la velocidad lineal, ya que de momento solo se envían constantes.

```
#Velocidad lineal según semáforo
if self.color_traffic_light == 1:
    self.velL = 0.2
elif self.color_traffic_light == 2:
    self.velL = 0.1
elif self.color_traffic_light == 3:
    self.velL = 0.0
```

```
if self.errorTheta > 0.05 or self.errorTheta < -0.05:
    self.velL = 0.0
```

De igual manera la función termina enviando los valores de velocidad lineal y angular calculados por el tópico `cmd_vel` como un tipo de mensaje Twist, como se muestra a continuación:

```
# Crear el mensaje Twist y publicarlo
twist_msg = Twist()
twist_msg.linear.x = self.velL
twist_msg.angular.z = self.velA
self.pub_cmd_vel.publish(twist_msg)
```

B. Nodo Color Detection

Para dar solución al reto de esta semana, el cual requiere de la detección del color de un semáforo a través de la cámara integrada dentro del Puzzlebot, se realizó la creación del nodo `color_detection` el cual requiere seguir el siguiente comportamiento según el color detectado:

- Rojo: Deténgase hasta que vea una luz verde
- Amarillo: Conduzca lentamente, hasta que vea una luz roja para detenerse.
- Verde: continúa con tu camino.

Dentro del código de este nuevo nodo se empezó por la declaración del suscriber, el cual se suscribe al tópico `/video_source/raw'` para recibir la imagen de la cámara y que a su vez llama a la función `image_callback`. De igual manera se creó un publisher encargado de publicar el color del semáforo detectado a través del tópico `color_detection` enviando un entero como valor.

```
#Suscriber camara
self.sub = self.create_subscription(Image, '/video_source/raw',
self.image_callback, rclpy.qos.qos_profile_sensor_data)

# Publica el color identificado en la imagen mediante un número
self.pub_color = self.create_publisher(Int32, 'color_detection', 10)
```

De igual manera se crean los vectores con los valores límite inferior y superior para cada uno de los colores del semáforo en formato de imagen HSV.

```
# Máscara para identificar colores rojos
self.redBajo = np.array([0, 20, 20])
self.redAlto = np.array([10, 255, 255])
self.redBajo=np.array([160, 60, 100])
self.redAlto=np.array([180, 255, 255])

# Máscara para identificar colores verdes
self.lower_green = np.array([70, 50, 50])
self.upper_green = np.array([100, 150, 150])

## Máscara para identificar colores amarillos
self.lower_yellow = np.array([25, 20, 20])
self.upper_yellow = np.array([35, 255, 255])
```

Es importante mencionar que estos valores se obtuvieron de manera heurística, con más de 100 pruebas diferentes, y que varían en gran medida según la cámara utilizada, así como de la iluminación del entorno y los colores del semáforo a utilizar. En nuestro caso estos valores se obtuvieron en el salón de clases donde se llevaron a cabo las pruebas con el robot y bajo un ambiente de iluminación relativamente controlado.

Es dentro del función `image_callback` llamda por nuestro suscriber que se realizan las operaciones y análisis de imagenes necesarias para la detección de los colores del semáforo, dentro de la misma se recibe la imagen del tópico `/video_source/raw'` para posteriormente ser convertida a una imagen OpenCV. Para reducir el procesamiento y con ello un intento de reducir la latencia en el procesamiento de los colores, se realiza un recorte horizontal de la imagen manteniendo la parte central de la misma, resultando en una imagen con un tercio de pixeles en su eje horiontal. Esta imagen recortada es posteriormente enviada a otra función `detectarSemaforo` que se encarga de detectar circulos presentes en la imagen, pasando parámetros como el radio mínimo y máximo de los circulos a detectar en la imagen.

```
# Rango de radios
min_radius = 50
max_radius = 300
semaforo = self.detectarSemaforo(middle_section, min_radius,
                                 max_radius)
```

Es dentro de esta función que se utiliza transformación de círculo de Hough para determinar la cantidad de circulos presentes y de detectar alguno retornar 'True', esta bandera se utiliza posteriormente para determinar si se llama a la función `detect_color` o no.

Es dentro de la función `detect_color` que se realiza el procesamiento de imagen necesario para la detección de los colores, los pasos seguidos para dicha detección se muestran a continuación:

1) Conversión de formato BGR a HSV.
2) Creación de un elemento estructurante de 15x15.
3) Operaciones de umbralización para el color correspondiente.
4) Uso de operadores Bitwise, aíslando las áreas del color requerido de la imagen según la máscara.
5) Reducción de ruido con filtro promedio de 9x9.
6) Uso de transformaciones morfológicas (Morfología de apertura).
7) Detección de bordes por método de Canny
8) Determinación del evento con mayor incidencia entre los colores.

Calculando el color con mayor incidencia en la imagen es como se determina que mensaje enviar a través del tópico `color_detection`, enviando un '1' en caso de detectar el semáforo verde, un '2' de detectar el semáforo amarillo y un '3' de detectar un semáforo de color rojo. De igual manera se implementa un condicional que únicamente envía estos valores de haber cambiado respecto al valor anterior, para ahorrar ancho de banda y no saturar el tópico.

V. RESULTADOS

Para finalizar obtenemos como resultado que el puzzlebot pueda seguir una trayectoria, en este caso, cuadrada y al mimso tiempo estar procesando la imagen de la camara para detectar si el semaforo que se le presente esta en color verde, amarillo o rojo, para poder avanzar, reducir la velocidad o detenerse respectivamente.

La trayectoria que recorre la hace por medio de coordenadas y obteniendo la posición del robot y al mismo tiempo el error para poder concluir el recorrido, esto lo resolvimos en un reto anterior.

En el video se puede observar como el puzzlebot realiza acciones dependiendo del color del semaforo, sin perder la

trayectoria cuadrada, de igual manera se puede observar que en ocasiones el robot tarda un poco en realizar las acciones, esto es debido a que por el procesamiento se retrasan los fotogramas, por lo que no realiza las acciones inmediatamente, tambien en el video se observa una terminal, la cual muestra al final de la linea el último color que detecto el robot, siendo 1 para verde, 2 para amarillo y 3 para rojo.

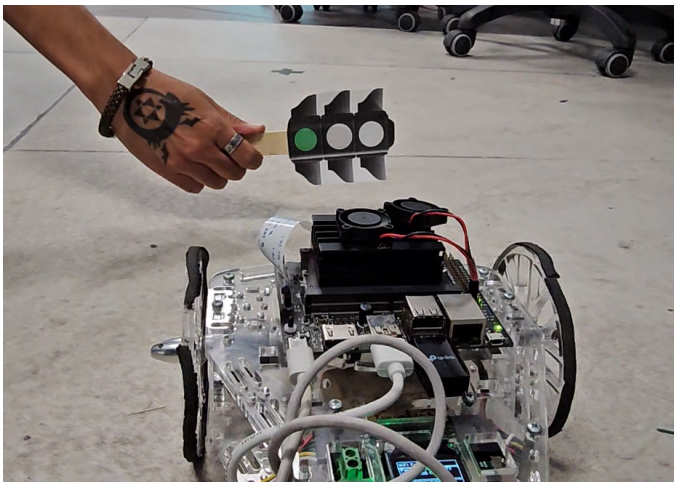

Fig. 1. Video funcionamiento

VI. Conclusiones

Gracias al trabajo previo en el módulo de procesamiento de imágenes y el uso de la librería OpenCV, pudimos implementar métodos especializados en la detección de formas y colores para el PuzzleBot. Este enfoque nos permitió lograr nuestro objetivo de ajustar la locomoción del PuzzleBot según los colores del semáforo, asegurando que se detenga ante el rojo, disminuya la velocidad en amarillo y mantenga un movimiento normal en verde. Aunque el sistema funciona de manera eficiente, reconocemos que aún enfrenta desafíos en condiciones de iluminación deficiente o cuando los colores detectados son similares a los clasificados.

Para mejorar en futuras entregas, podríamos explorar técnicas avanzadas de procesamiento de imágenes para una detección más robusta y implementar mecanismos de adaptación a cambios en la iluminación. Además, un refinamiento de los algoritmos de clasificación de colores podría ser beneficioso para mejorar la precisión del PuzzleBot en situaciones desafiantes. A pesar de estos obstáculos, estamos satisfechos con el resultado obtenido y vemos este proyecto como un testimonio de nuestro compromiso con la excelencia y la innovación en el campo de la robótica autónoma.

A través de un enfoque iterativo y un trabajo arduo, hemos logrado desarrollar un proyecto robusto que se desempeña adecuadamente en la mayoría de los entornos probados. A pesar de los desafíos como el calentamiento de la Jetson y las limitaciones en el código debido al tiempo de respuesta, hemos demostrado nuestra capacidad para resolver problemas y enfrentar obstáculos con determinación. Con cada desafío superado, hemos fortalecido nuestro conocimiento y habilidades, preparándonos para futuros proyectos con confianza y resiliencia.

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] ManchesterRoboticsLtd. (s.f.). GitHub - ManchesterRoboticsLtd/TE3002B-2024: Intelligent Robotics Implementation. GitHub. link.