Reto semanal 5. Manchester Robotics

Implementación de Robótica Inteligente

Integrantes:

A017314050 | Eleazar Olivas Gaspar

A01735696 | Azul Nahomi Machorro Arreola

A01732584 | Angel Estrada Centeno

A01735692 | Arick Morelos del Campo

Profesores:

Rigoberto Cerino Jimenez

Alfredo Garcia Suarez

Juan Manuel Ahuactzin Larios

Cesar Torres Huitzil

Mayo 2024

Resumen:

Este reto nos permitirá interactuar con el Puzzlebot, añadiendo una capa de seguimiento de líneas a la etapa de decisión y al algoritmo de navegación punto a punto desarrollados previamente, para seguir una línea utilizando la cámara del Puzzlebot a través de ROS. A lo largo del proyecto, abordaremos varios objetivos específicos para garantizar el éxito y la efectividad de nuestra solución.

Primero, utilizaremos la cámara del Puzzlebot junto con la infraestructura de ROS para detectar y seguir líneas en tiempo real. Este proceso implica modificar y mejorar el algoritmo existente, integrando una nueva funcionalidad de seguimiento de líneas que permita al robot ajustar su trayectoria automáticamente al seguir una línea marcada en el suelo. La implementación requerirá una comprensión profunda de los conceptos de visión por computadora y su aplicación en un entorno robótico.

En segundo lugar, estudiaremos cómo el sistema de visión artificial del Puzzlebot responde a diferentes condiciones de iluminación y variaciones en el entorno. Evaluaremos el rendimiento del algoritmo de seguimiento de líneas bajo diversas circunstancias, como cambios en la luz ambiental, sombras y diferentes colores de líneas. Este análisis nos ayudará a identificar puntos débiles y áreas de mejora en el sistema, asegurando que sea robusto y fiable.

Después diseñaremos y programaremos un algoritmo robusto utilizando técnicas avanzadas de procesamiento de imágenes. Este algoritmo deberá ser capaz de operar en tiempo real, gestionando eficientemente los ruidos en la imagen y adaptándose a diferentes tipos de líneas y condiciones del entorno. Nos enfocaremos en técnicas como la segmentación de imágenes, el filtrado y la detección de bordes para lograr una detección precisa y rápida de las líneas.

Además, trabajaremos en la optimización del código y del procesamiento de imágenes para asegurar que el sistema funcione de manera fluida y con una latencia mínima. Esto incluirá la mejora de la eficiencia del algoritmo, el uso eficiente de los recursos computacionales del Puzzlebot y la minimización del tiempo de procesamiento para mantener un rendimiento en tiempo real.

La validación del sistema incluirá la medición de métricas de desempeño como la precisión en la detección de líneas, la robustez frente a perturbaciones y la velocidad de procesamiento del algoritmo.

Objetivos:

- Utilizar la cámara del Puzzlebot y la infraestructura de ROS para detectar y seguir líneas en tiempo real, modificando y mejorando el algoritmo existente para integrar esta funcionalidad.

- Estudiar cómo el sistema de visión artificial del Puzzlebot responde a diferentes condiciones de iluminación y variaciones en el entorno, evaluando el rendimiento del algoritmo de seguimiento de líneas e identificando áreas de mejora.

- Diseñar y programar un algoritmo robusto utilizando técnicas avanzadas de procesamiento de imágenes, que opere en tiempo real, gestione eficientemente los ruidos en la imagen y se adapte a diferentes tipos de líneas y condiciones del entorno.

- Trabajar en la optimización del código y del procesamiento de imágenes para asegurar que el sistema funcione de manera fluida y con una latencia mínima, mejorando la eficiencia del algoritmo y el uso de los recursos computacionales del Puzzlebot.

- Validar el sistema mediante la medición de métricas de desempeño como la precisión en la detección de líneas, la robustez frente a perturbaciones y la velocidad de procesamiento del algoritmo.

Introducción:

Para abordar la solución de este reto tenemos que tener en cuenta algunos conceptos teóricos importantes que explican el funcionamiento, a continuación los conceptos a conocer:

- **Procesamiento de Imágenes en una Tarjeta Embebida:**

El procesamiento de imágenes en una tarjeta embebida, como las NVIDIA Jetson, permite realizar tareas de visión por computadora en tiempo real utilizando dispositivos compactos y especializados. Estas tarjetas integran CPU, GPU y memoria en un solo chip, facilitando la captura, preprocesamiento, análisis y toma de decisiones basadas en imágenes. El proceso comienza con la captura de imágenes mediante una cámara, seguida del preprocesamiento para mejorar la calidad de las imágenes a través de la conversión de color, reducción de ruido y normalización. Posteriormente, se aplican algoritmos de visión por computadora para extraer características relevantes como bordes, contornos y formas, lo que permite tomar decisiones y ejecutar acciones, como mover un robot siguiendo una línea.

En la captura de imágenes el tipo de cámara es lo que tiene mayor peso para el procesamiento que se realizará después, al procesar la imagen quitamos ruido y transformamos la imagen a un espacio de color que nos ayude mejor a la deteccion de las lineas que utilizaremos para el seguimiento, una vez procesada la imagen pasamos al analisis de la misma, en el que extreeremos las caracteristicas que nos interesan como los bordes, contornos y centroide. Este flujo de procesamiento permite que dispositivos embebidos como las tarjetas NVIDIA Jetson realicen tareas complejas de visión por computadora en tiempo real, haciendo posible su uso en aplicaciones industriales, vehículos autónomos, drones, y muchas otras áreas que requieren procesamiento eficiente y rápido de imágenes.

- **Interconexión entre Jetson y la Cámara:**

La interconexión entre la NVIDIA Jetson y la cámara se realiza generalmente a través de interfaces como USB, CSI (Camera Serial Interface) o Ethernet. La conexión física implica conectar la cámara a la Jetson mediante un cable compatible, mientras que la configuración de software requiere instalar y configurar los controladores y bibliotecas necesarias para que la Jetson reconozca y se comunique con la cámara. Una vez configurada, se pueden capturar imágenes o vídeos en tiempo real utilizando software como OpenCV para su procesamiento directo en la Jetson.

- **Detección de Contornos o Formas en una Imagen:**

La detección de contornos o formas en imágenes es esencial para aplicaciones de visión por computadora, permitiendo identificar bordes de objetos y reconocer formas clave en una escena

visual. Este proceso comienza con el preprocesamiento de la imagen, donde se aplican filtros como el de Canny para resaltar bordes incluso en presencia de ruido. Luego, la umbralización convierte la imagen en binaria, simplificando el análisis. Finalmente, algoritmos como cv2.findContours de OpenCV se utilizan para identificar y delinear los contornos presentes, proporcionando información estructural crucial para tareas como detección de objetos, reconocimiento de patrones y segmentación de imágenes. La detección de contornos es una parte importante en el desarrollo de sistemas de visión por computadora, capacitando a los dispositivos para comprender y tomar decisiones basadas en el contenido visual capturado por cámaras y otros sensores de imágenes digitales.

- **Detección de Líneas y Contornos en una Imagen:**

La detección de líneas y contornos juega un papel fundamental en aplicaciones como el seguimiento de líneas en robots autónomos. Este proceso se inicia con el preprocesamiento de la imagen, donde se aplican técnicas como el suavizado y la detección de bordes para resaltar características relevantes. El suavizado ayuda a reducir el ruido en la imagen, mientras que la detección de bordes identifica los cambios abruptos en la intensidad que pueden indicar la presencia de líneas o bordes de objetos. Posteriormente, se emplea la Transformada de Hough para detectar líneas rectas en la imagen. Esta técnica transforma los puntos de la imagen en parámetros de una línea en un espacio de votación, lo que permite identificar líneas incluso en presencia de ruido y discontinuidades en los bordes. Una vez identificadas las líneas, se pueden utilizar algoritmos de contornos para detectar y refinar aún más los contornos relevantes en la imagen.

La combinación de estos procesos proporciona una base sólida para el seguimiento de líneas en robots autónomos y otras aplicaciones similares. Al detectar y seguir las líneas en su entorno, los robots pueden navegar de manera efectiva por terrenos estructurados, como pistas o senderos, tomando decisiones en tiempo real basadas en la información visual capturada por sus cámaras. Este enfoque no solo es crucial para la locomoción precisa de los robots, sino que también tiene aplicaciones en la industria, la logística y la robótica móvil en general.

- **Algoritmos Implementados para el Seguimiento de Líneas y Trayectorias:**

Los algoritmos implementados para el seguimiento de líneas y trayectorias permiten a los robots seguir líneas o trayectorias predeterminadas en un entorno. Estos algoritmos funcionan detectando la línea a seguir mediante métodos de visión por computadora, calculando la trayectoria del robot en función de la posición de la línea detectada y ajustando la velocidad y dirección del robot para mantenerse en la línea, utilizando controladores PID u otros métodos de control.

- **Robustez en Sistemas de Procesamiento de Imágenes:**

La robustez en sistemas de procesamiento de imágenes es un factor crítico en aplicaciones donde la precisión y la confiabilidad son fundamentales. Este concepto se refiere a la capacidad del sistema para mantener un rendimiento consistente frente a variaciones y perturbaciones en el entorno. Para lograr este objetivo, se implementan estrategias como la redundancia mediante múltiples métodos de detección, lo que permite al sistema compensar posibles fallos de un solo método y tomar decisiones más informadas. Además de la redundancia, la adaptabilidad es esencial para mejorar la robustez del sistema. Esto implica ajustar dinámicamente los parámetros del sistema para adaptarse a cambios en las condiciones del entorno, como variaciones en la iluminación o la presencia de obstáculos. Los

algoritmos de procesamiento de imágenes pueden ser diseñados para monitorear continuamente estas condiciones y ajustar automáticamente sus configuraciones, garantizando un rendimiento óptimo en tiempo real.

Otra estrategia fundamental es la aplicación de técnicas avanzadas de filtrado para reducir el ruido y mejorar la calidad de las imágenes. El ruido puede afectar negativamente la precisión de los resultados del procesamiento de imágenes, por lo que eliminar o reducir su impacto es crucial. Mediante el uso de filtros adaptativos y técnicas de suavizado avanzadas, el sistema puede mejorar la fiabilidad de los datos obtenidos, asegurando una operación robusta incluso en condiciones desafiantes.

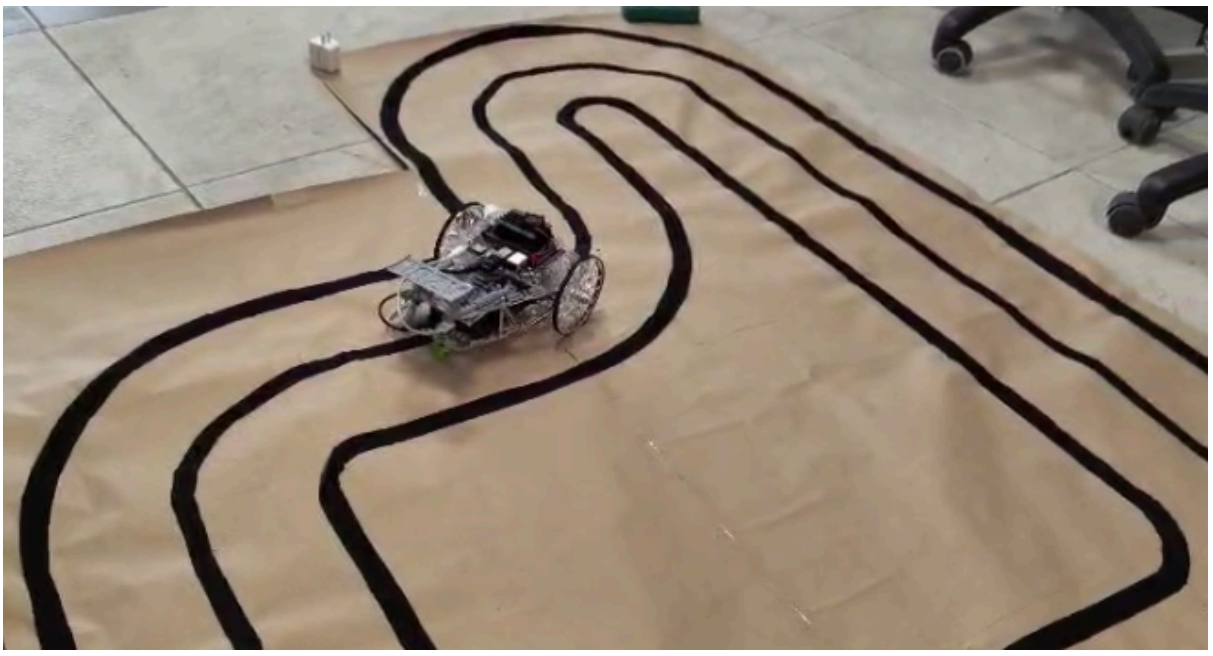
- **Centroide:**

El centroide representa el centro geométrico de un objeto o forma en una imagen, y su cálculo es crucial en muchas aplicaciones de procesamiento de imágenes y visión por computadora. Primero, para calcular el centroide, es necesario detectar y segmentar el objeto de interés en la imagen. Este proceso puede implicar el uso de algoritmos de detección de bordes, segmentación por umbralización u otros métodos de segmentación avanzados dependiendo de la naturaleza del objeto.

Una vez que el objeto ha sido identificado y segmentado, se procede a calcular el centroide utilizando las coordenadas de los píxeles que componen el objeto. Este cálculo implica determinar el punto medio promedio en términos de coordenadas x e y de todos los píxeles dentro del objeto. El resultado es un par de coordenadas que representan la posición del centroide en la imagen. El centroide tiene una amplia gama de aplicaciones, desde la robótica hasta el análisis de imágenes médicas. Por ejemplo, en robótica, el centroide puede utilizarse para dirigir un robot hacia el centro de un objeto detectado, como una línea o una forma, facilitando tareas de seguimiento y control.

Solución del problema:

- Pista a utilizar:



Código empleado:

```python
# Definición de la clase
class LineFollower(Node):
    # Constructor
    def __init__(self):
        # Definición del tópico
        super().__init__('Azul_LineFollower_node')

        # Variables para la camara
        self.valid_img = False
        self.bridge = CvBridge()
        self.sub = self.create_subscription(Image, '/video_source/raw', self.camera_callback, rclpy.qos.qos_profile_sensor_da

        #Creación del tópico publicador: cmd_vel
        self.publisher2 = self.create_publisher(Twist, 'cmd_vel', 10)

        # Timer period
        timer_period = 0.05
        self.timer = self.create_timer(timer_period, self.timer_callback)

        #Print info to confirm node was made.
        self.get_logger().info('Line Follower node successfully initialized!!!')

```

En esta parte del código se define el nodo que hará el seguimiento de la línea. Primero, se crea el suscriptor a la cámara del bot, posteriormente se hace el publicador a cmd_vel para mandar las velocidades lineales y angulares del robot.

```python
    def resize_image(self, img):
        # Crop the image
        height, width = img.shape[:2]
        start_y = int(height * 0.55)  # Adjusted to crop an additional 15% from the top
        end_y = height  # Keep the bottom part of the image
        start_x = 2 * width // 8
        end_x = 6 * width // 8

        cropped_img = img[start_y:end_y, start_x:end_x]
        return cropped_img

```

Esta función se encarga de recortar la imagen a sólo el pedazo de interés que es donde se sitúa la línea central de la pista.

```python
    def timer_callback(self):
        try:
            if self.valid_img:
                # Inicializamos a 0.0 las variables que no utilizaremos del Twist y mandaremos a cmd_vel
                cmd_vel = Twist()
                cmd_vel.linear.y = 0.0
                cmd_vel.linear.x = 0.0
                cmd_vel.linear.z = 0.0
                cmd_vel.angular.x = 0.0
                cmd_vel.angular.y = 0.0

                frame = cv2.cvtColor(self.img, cv2.COLOR_BGR2HSV)

                # Rotate the frame 180 degrees
                frame = cv2.rotate(frame, cv2.ROTATE_180)

                # Resize and crop the frame
                resized_frame = self.resize_image(frame)

                filtro_median = cv2.medianBlur(resized_frame, 3)

                # Convert the extracted portion to grayscale
                gray_image = cv2.cvtColor(filtro_median, cv2.COLOR_BGR2GRAY)

                # Binarize the grayscale image using Otsu's method
                _, binary_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

```

Se inicializan las componentes del mensaje tipo Twist que se enviará a cmd_vel. También se lee el primer cuadro de la imagen y se procesa para quitar ruido, suavizar y binarizar.

```
        # Apply morphological operations
        SE_d = np.ones((10, 10), np.uint8)
        morf_d = cv2.dilate(binary_image, SE_d, iterations=5)
        SE_e = np.ones((10, 10), np.uint8)
        morf_e = cv2.erode(morf_d, SE_e, iterations=1)
        imagenProcesada = cv2.bitwise_not(morf_e)

        #Conteo de pixeles
        centro_img_x = int(morf_e.shape[1]/2)
        centro_img_y = 10
        dimy = int(morf_e.shape[0]/2)

        cont = 0
        bandera =centroide_primer_punto_x = 0
        centroide_primer_punto_y = 0
        for i in range (morf_e.shape[1]):
            if morf_e[dimy][i] == 0:
                cont += 1
            if (bandera == 0 and morf_e[dimy][i] == 0):
                centroide_primer_punto_x = i
                centroide_primer_punto_y = dimy
                bandera = 1
        error = ((centroide_primer_punto_x + cont/2)-centro_img_x )/(morf_e.shape[1])
```

Se aplica la morfología para tener la imagen procesada y con ella detectar el centroide de la línea.

```
        # Determine the direction to turn
        if error != 0:
            if error >= -0.05 and error <= 0.05:
                direction = 'F'
                cmd_vel.angular.z = 0.0
                cmd_vel.linear.x = 0.05
            elif error <= -0.05:
                direction = 'L'
                cmd_vel.angular.z = (0.15 * error)
                cmd_vel.linear.x = 0.05
                if cmd_vel.angular.z < 0.05:
                    cmd_vel.angular.z = 0.05
            elif error >= 0.05:
                direction = 'R'
                cmd_vel.angular.z = (0.15 * error)
                cmd_vel.linear.x = 0.05
                if cmd_vel.angular.z > -0.05:
                    cmd_vel.angular.z = -0.05
            else:
                direction = 'F'
                cmd_vel.angular.z = 0.0
                cmd_vel.linear.x = 0.05

        # Create a String message and publish the direction
        self.get_logger().info(f"Deviation X: {error}, Direction: {direction}")

        self.publisher2.publish(cmd_vel)
```

Dependiendo del error que tenga el centroide de la línea con el centro de la imagen se define el movimiento que tendrá el robot.

Resultados:

Repositorio GitHub:
https://github.com/AzulMachorro/Retos_Manchester_Robotics_IRI-Week-6

Video del procesamiento del nodo:

https://youtube.com/shorts/Q6jfELbq6JY

En este video se muestran dos ventanas, la ventana en la parte superior del video es la visualización del rqt_image_view el cual muestra la imagen resultante de todo el

procesamiento que se aplica a la imagen original de entrada de la webcam del robot para la detección de la línea y su centroide.

En la terminal que se muestra en la parte inferior del video se ve la ejecución del nodo line_f que es el seguidor de línea, este imprime mediante la terminal las coordenadas del centroide de la línea, la coordenada X del centro de la imágen, la desviación en X del centroide de la línea con respecto al centro de la imagen, y la dirección de avance para el robot.

La imagen procesada que nos manda el robot no es muy fluída por la alta latencia que se presenta en la comunicación de los tópicos para visualizar la imagen de entrada procesada y la baja tasa de fotogramas por segundo que la propia webcam del robot tiene; aún así se logra apreciar cómo a medida que el punto rojo, que es el centroide de la línea, se desfasa del centro de la imagen el mensaje de dirección del robot cambia, diciendo si debe ir de frente “F”, izquierda “L”, o derecha “R”.

Video demostrativo del robot:

https://youtu.be/_lPCl_Olzgk

Este es un video del recorrido que hace el robot sobre la pista que creamos. La velocidad lineal se ajustó a ser muy baja (0.05 rad/sec) para poder procesar las imágenes y no perder detalles importantes de la pista como sucedería con una velocidad mayor. En este video se puede apreciar que el robot hace un buen recorrido sobre la pista pues se mantiene relativamente encima de la línea negra central. En algunos trayectos el robot puede presentar ligeras oscilaciones debido al control proporcional aplicado para girar de una manera más rápida: la velocidad angular si puede llegar a ser mucho mayor que la lineal dependiendo del error medido.

Conclusiones:

En este proyecto, se mejoraron los códigos usados en retos anteriores al integrar una capa de seguimiento de líneas en el sistema de navegación. se logró mediante la modificación y mejora de los códigos, permitiendo al robot seguir automáticamente una línea marcada en la pista, con esto profundizamos conceptos de visión por computadora. El diseño y la implementación de un algoritmo robusto fueron fundamentales ya que se utilizaron técnicas avanzadas de procesamiento de imágenes para garantizar un mejor rendimiento.

A partir de los videos demostrativos podemos comentar que el robot muestra una capacidad efectiva para mantenerse relativamente encima de la línea negra central, lo que indica que el algoritmo de seguimiento de líneas es bastante eficiente, la decisión de reducir la velocidad lineal del robot a 0.05 rad/sec, fue buena y garantiza una adecuada captura de la dirección de la pista sin perder información.

Aunque el robot hace un buen recorrido en general, se observan ligeras oscilaciones en algunos tramos. Estas oscilaciones pueden ser atribuidas al control proporcional aplicado para girar de manera más rápida. Este comportamiento sugiere la necesidad de ajustes adicionales en el control para

mejorar la estabilidad del robot en ciertos casos. En general, el video muestra un progreso prometedor en el desarrollo del sistema de seguimiento de líneas, con resultados satisfactorios en términos de mantenimiento de trayectoria y ajuste de velocidad. Sin embargo, también indica áreas específicas que podrían requerir optimización adicional para lograr un rendimiento aún mejor.

Por último se debe agregar que se tomó en cuenta la luz, la hora del día y la posición de la cámara en el entorno, estos aspectos externos afectan a la correcta funcionalidad del programa sin embargo se hizo lo posible para generar el mejor resultado.

Bibliografía o referencias:

- *CONTROL ROBUSTO DE UN SISTEMA MECÁNICO SIMPLE MEDIANTE UNA HERRAMIENTA GRÁFICA. revistas.unal.edu.co. (s.f).* https://revistas.unal.edu.co/index.php/dyna/article/view/15852/36191

- *Pérez, J. (2007). Análisis de sistemas robóticos. Revista de Ingeniería Robótica, 7(2), 55-65.*https://www.redalyc.org/pdf/784/78470208.pdf

- *NVIDIA Corporation. (s.f.). "Jetson Nano Developer Kit User Guide". Documentación oficial de NVIDIA.* https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit

- *Universidad de Toronto, Departamento de Ciencias de la Computación. (s.f.). "Detección de Contornos". Página web del proyecto SRAF (Sistema de Reconocimiento Automático de Formas).* https://www.cs.toronto.edu/~dmac/images/ProjectFiles/sraf/srafdoc/deteccion.html#:~:text=La%20detecci%C3%B3n%20de%20contorno%20es,una%20aplicable%20en%20distintas%20circunstancias.

- *Ángel Palomares Carrascosa. 2012. Universidad Politécnica de Madrid. Memoria de investigación.* https://oa.upm.es/19654/1/INVE_MEM_2012_130127.pdf