Documento Técnico de Nodos para el Reto Final

Nodo 1: camara_node.py

Nombre del Nodo: filter_node

Objetivo:
Preprocesar las imágenes obtenidas desde la cámara para generar dos salidas distintas: una imagen binaria enfocada a la detección de líneas negras, y una imagen comprimida en formato JPEG orientada a la detección de señales de tránsito con YOLOv8.

Funcionalidades Principales:

- Suscripción al tópico /video_source/raw para recibir la imagen de cámara.
- Aplicación de rotación de 180°, recorte de la región de interés (ROI), conversión a escala de grises y umbralización binaria inversa.
- Publicación de la imagen binaria filtrada en /filtered_image.
- Publicación periódica (15 Hz) de imagen comprimida JPEG redimensionada en /compressed_image.

Tópicos Utilizados:

- Entrada: /video_source/raw (sensor_msgs/msg/Image)
- Salidas:
    - /filtered_image (sensor_msgs/msg/Image)
    - /compressed_image (sensor_msgs/msg/CompressedImage)

Diseño Técnico:

- El procesamiento incluye operaciones morfológicas de apertura para limpieza de ruido.
- Se emplea cv_bridge para la conversión entre imágenes ROS y matrices OpenCV.
- La última imagen válida se mantiene en memoria para permitir la publicación periódica comprimida.

Nodo 2: line_follower.py

Nombre del Nodo: line_follower_node

Objetivo:
Detectar la posición de una línea negra sobre el suelo y calcular un error de seguimiento normalizado, utilizando una segmentación vertical de la imagen para extraer tres centroides.

Funcionalidades Principales:

- Recepción de imagen binaria desde /filtered_image.
- Detección del contorno más grande y segmentación vertical en tres regiones.
- Cálculo de centroides por región y estimación del error de posición con respecto al centro de la imagen.
- Implementación de un modo de recuperación si se pierde la detección de los centroides superiores.
- Publicación del error en /error y de la imagen con información visual en /debug_image.

Tópicos Utilizados:

- Entradas:
    - /filtered_image (sensor_msgs/msg/Image)
    - /reset_line (std_msgs/msg/Bool)
- Salidas:
    - /error (std_msgs/msg/Float32)
    - /debug_image (sensor_msgs/msg/Image)

Diseño Técnico:

- Se utiliza el centroide de la región media como referencia principal para calcular el error.
- En caso de pérdida de los centroides 2 y 3, se recurre al centroide 1 o al último error válido.
- El modo de recuperación es activado por un error especial (2.0) y se desactiva cuando el error se estabiliza dentro de un umbral.
- Las imágenes procesadas se graban localmente en un archivo de video para análisis posterior.

Nodo 3: detection.py

Nombre del Nodo: yolo_detector_node

Objetivo:
Realizar detección visual de señales de tránsito y semáforos utilizando un modelo de visión por computadora basado en YOLOv8, y publicar dicha información como códigos numéricos en los tópicos de control.

Funcionalidades Principales:

- Suscripción a imágenes comprimidas desde /compressed_image.
- Ejecución de inferencia con YOLOv8 y análisis de detecciones.
- Validación por área mínima y persistencia temporal para asegurar detecciones estables.
- Publicación de códigos correspondientes a semáforo en /color y a señales de tránsito en /sign.
- Grabación de video anotado con las detecciones.

Tópicos Utilizados:

- Entrada:
    - /compressed_image (sensor_msgs/msg/CompressedImage)
- Salidas:
    - /color (std_msgs/msg/Int32)
    - /sign (std_msgs/msg/Int32)

Diseño Técnico:

- Se establece un umbral de área mínima por clase para eliminar falsas detecciones.
- Se requiere una persistencia de al menos 0.5 segundos para confirmar una clase detectada.
- Las detecciones múltiples y los cambios de clase están regulados por una lógica de control para evitar publicaciones espurias.
- Se implementa un registro de detecciones confirmadas y su visualización sobre las imágenes.

Nodo 4: controllerf.py

Nombre del Nodo: controller_node

Objetivo:
Controlar el movimiento del robot diferencial a través de la información visual recibida desde los nodos de seguimiento de línea y detección, integrando una máquina de estados que regula la velocidad y el comportamiento según señales y semáforo.

Funcionalidades Principales:

- Control proporcional basado en el error de línea.
- Interpretación del estado del semáforo y ajuste de velocidad (STOP, SLOW, GO).
- Gestión de señales visuales: turn_left, turn_right, ahead, stop, giveway, work.
- Ejecución de giros con fases de avance inicial, rotación y avance final
- Implementación de un sistema de recuperación visual (modo RECOVER) que desactiva la navegación hasta recuperar visibilidad de la línea.

Tópicos Utilizados:

- Entradas:
    - /error (std_msgs/msg/Float32)
    - /color (std_msgs/msg/Int32)
    - /sign (std_msgs/msg/Int32)
- Salidas:
    - /cmd_vel (geometry_msgs/msg/Twist)
    - /reset_line (std_msgs/msg/Bool)

Diseño Técnico:

- Los giros dinámicos ajustan su duración según el último error registrado antes del inicio de la maniobra.
- Se da prioridad al semáforo sobre otras señales, impidiendo avances si está en rojo.
- El modo RECOVER actúa como medida de seguridad cuando la línea se pierde y se mantiene activo hasta que se corrige el error visual y se recibe una señal válida.
- Todas las velocidades y umbrales están parametrizados para permitir adaptabilidad a distintas pistas o condiciones.