

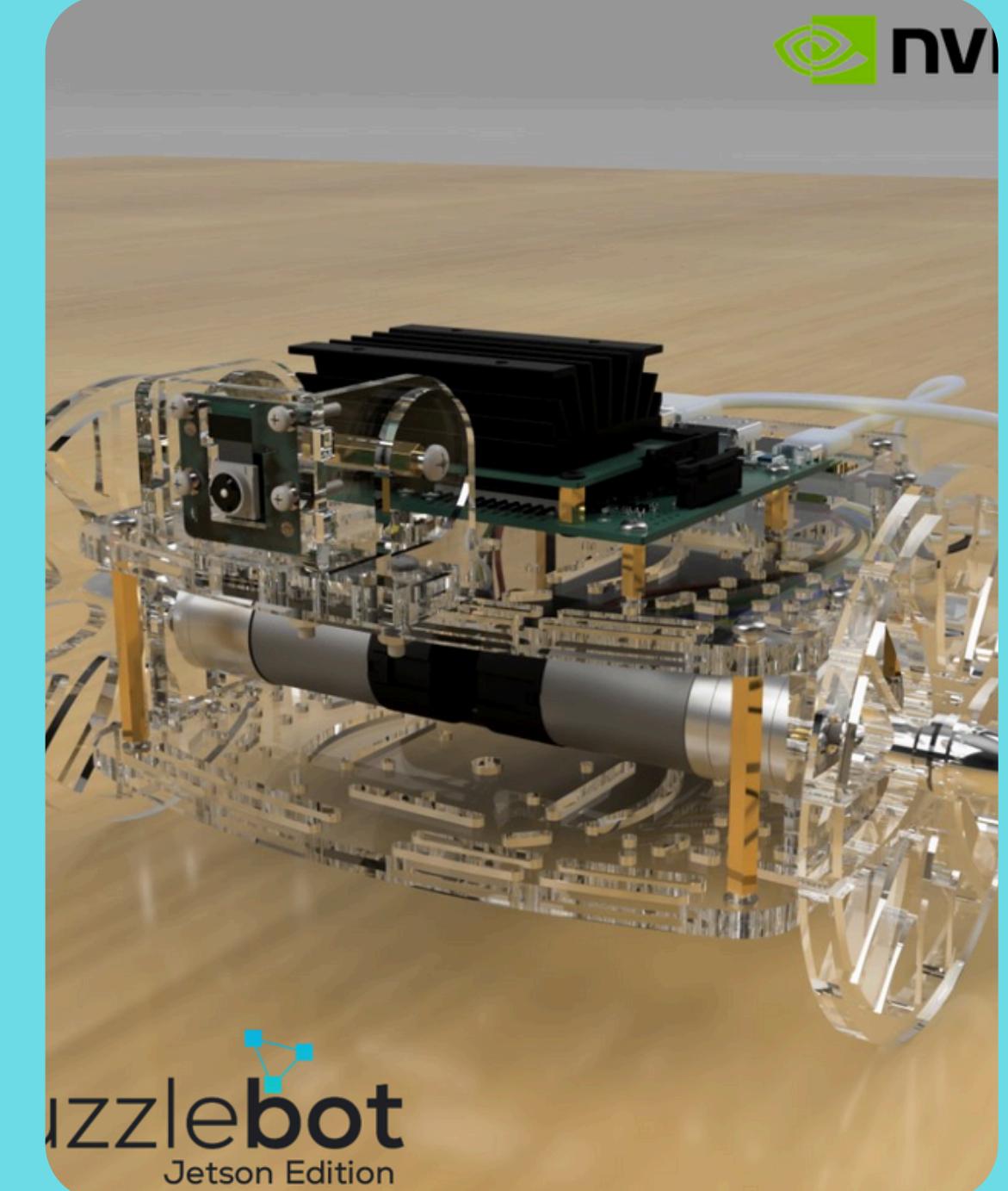
DESAFÍO DE CONDUCCIÓN AUTÓNOMA

Robonautas

Daniel Castillo López - A01737357

Emmanuel Lechuga Arreola - A01736241

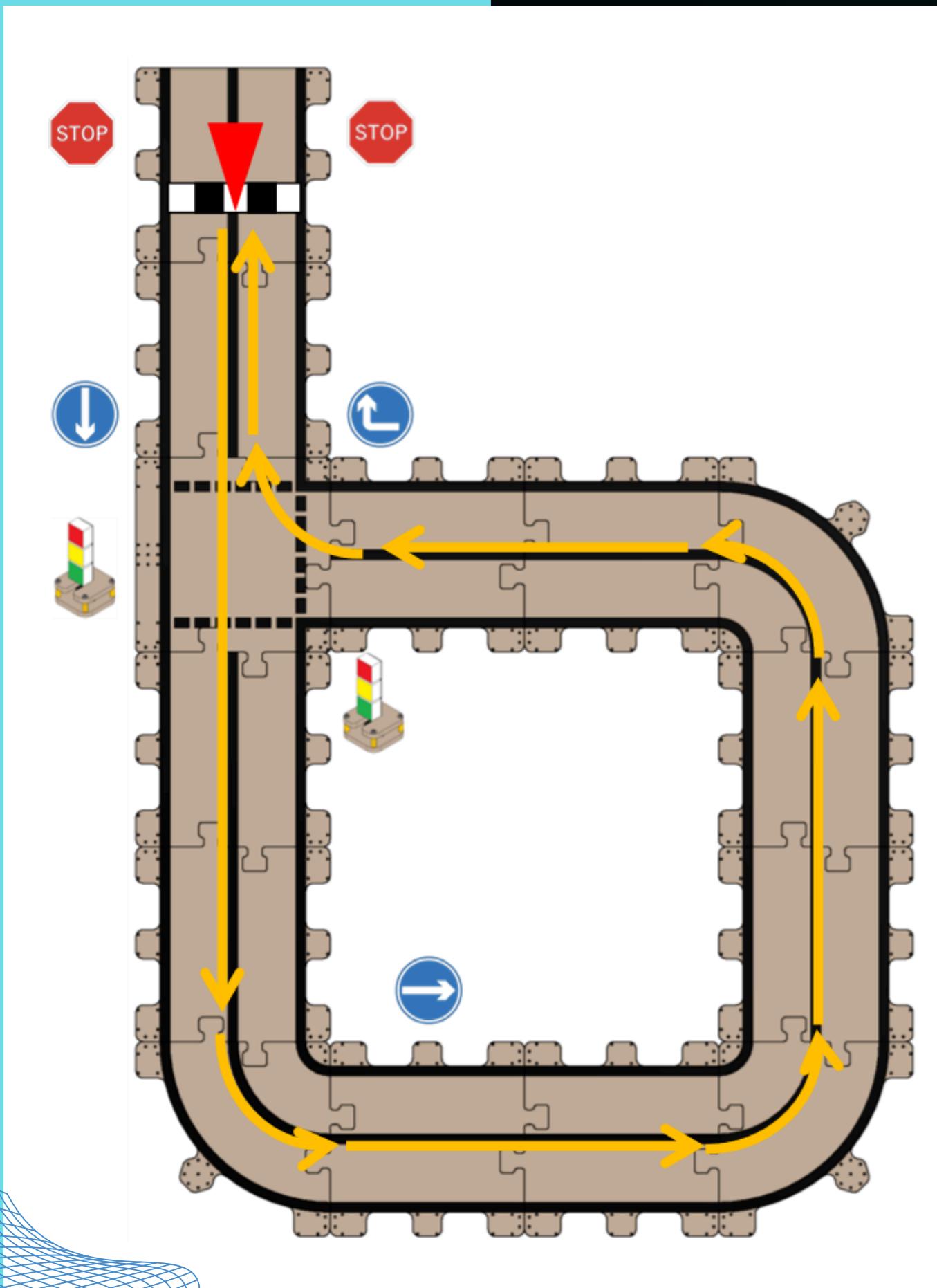
Paola Rojas Domínguez - A01737136




Puzzlebot
Jetson Edition

RETO PUZZLEBOT AUTÓNOMO

El reto final de MCR2 consiste en desarrollar un sistema de conducción autónoma para el robot móvil Puzzlebot, el cual debe recorrer una pista en forma de "b" siguiendo una línea negra central, reaccionando de forma inteligente a semáforos y señales de tránsito mediante visión por computadora. El robot debe ser capaz de detectar e interpretar diferentes señales así como los colores del semáforo, y actuar en consecuencia. Todo el recorrido debe realizarse de manera completamente autónoma, sin intervención humana, respetando las reglas del entorno y manteniéndose dentro de los límites del carril en todo momento.

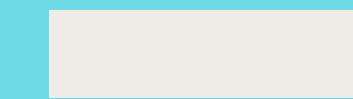


OBJETIVOS

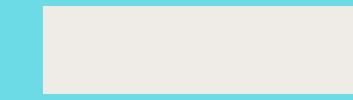
GENERAL

Desarrollar un sistema autónomo de navegación para el Puzzlebot que le permita recorrer una pista con forma de "b", siguiendo una trayectoria definida y reconociendo señales de tránsito y semáforos utilizando visión por computadora, con el fin de simular un entorno de conducción real y demostrar habilidades adquiridas en percepción, control y autonomía móvil.

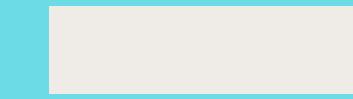
Particulares



Implementar un sistema de detección visual para identificar señales de tránsito mediante algoritmos de visión o aprendizaje automático.



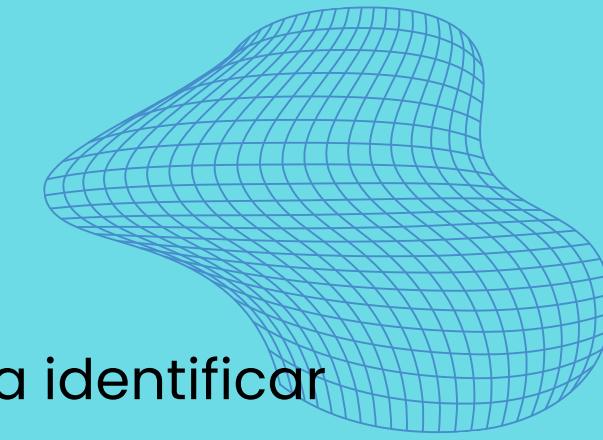
Diseñar un módulo de detección de semáforos que permita identificar los estados de luz y ajustar el comportamiento del robot en consecuencia.



Mantener la trayectoria del robot centrada sobre la línea negra de la pista utilizando algoritmos de seguimiento de línea robustos.



Garantizar la navegación autónoma sin intervención humana, permitiendo que el robot recorra el circuito completo, incluyendo curvas e intersecciones, cumpliendo con las reglas impuestas por las señales y semáforos.



METODOLOGÍA

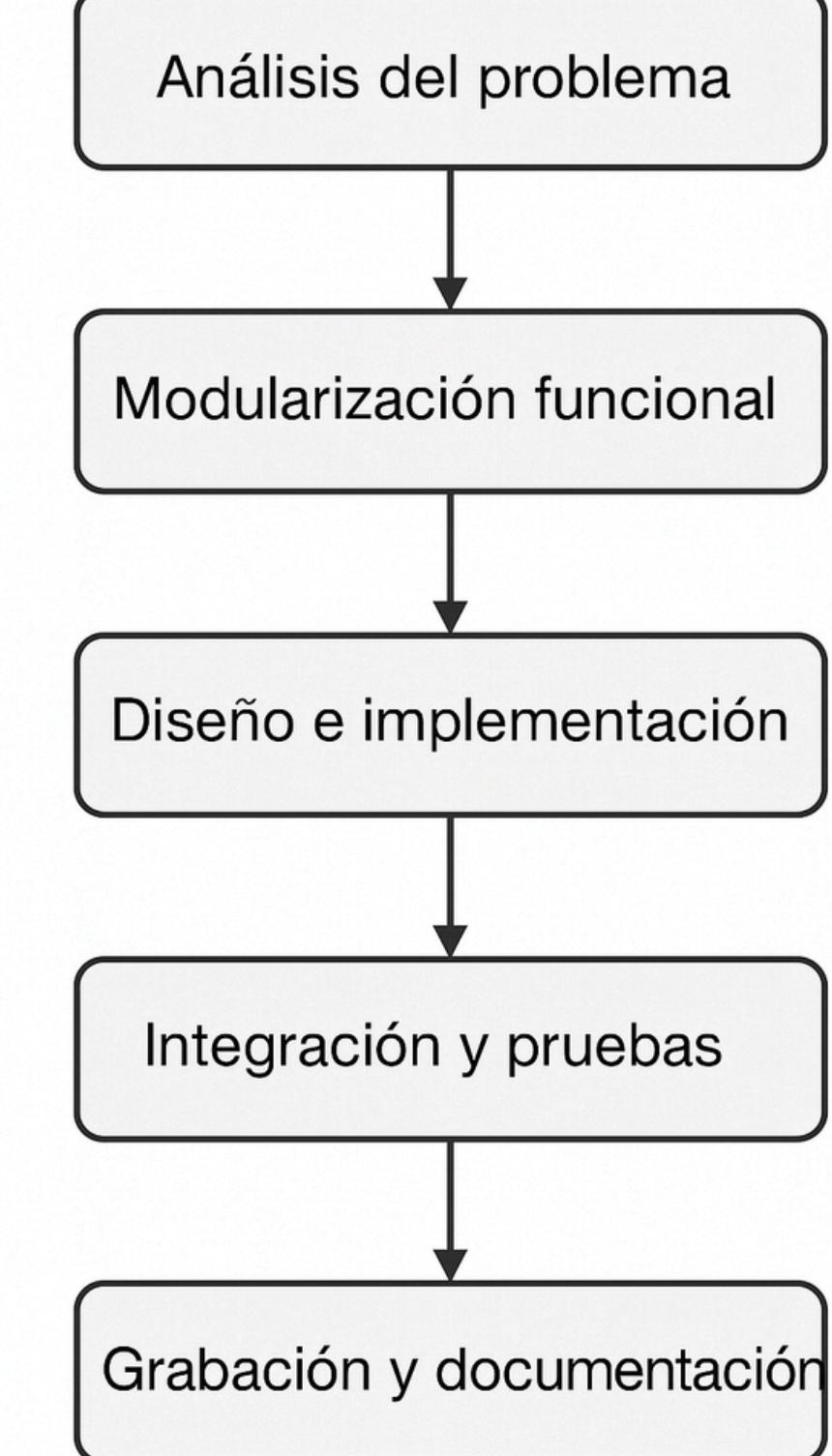
1. Análisis del problema

- Requerimiento: Navegación autónoma de un robot diferencial en una pista con línea negra, intersecciones y señales de tránsito.
- Restricciones: Interacción con semáforo, ejecución secuencial de maniobras, condiciones visuales variables.

2. Modularización funcional

Se definieron cuatro nodos independientes para desacoplar responsabilidades:

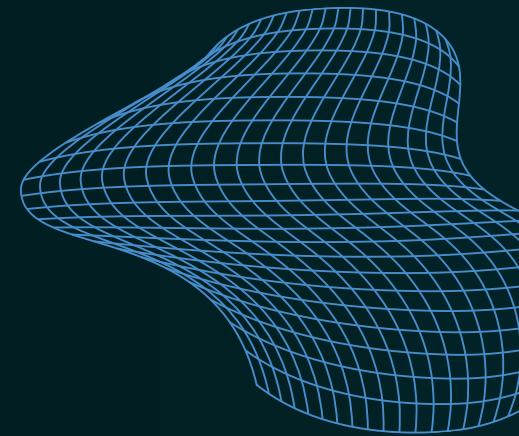
- Preprocesamiento de imagen (camara_node)
- Seguimiento de línea (line_follower)
- Detección de señales (detection)
- Control de movimiento (controllerf)



METODOLOGÍA

3. Diseño e implementación

- Procesamiento de imagen: ROI, binarización y segmentación.
- Visión por computadora: YOLOv8 entrenado y filtrado por área/persistencia.
- Control reactivo: Controlador proporcional P sobre el error de línea.
- Máquinas de estado: Semáforo y señales de tránsito definidas mediante lógicas independientes.
- Sincronización: Uso de tópicos ROS 2 para comunicación entre nodos.



4. Integración y pruebas

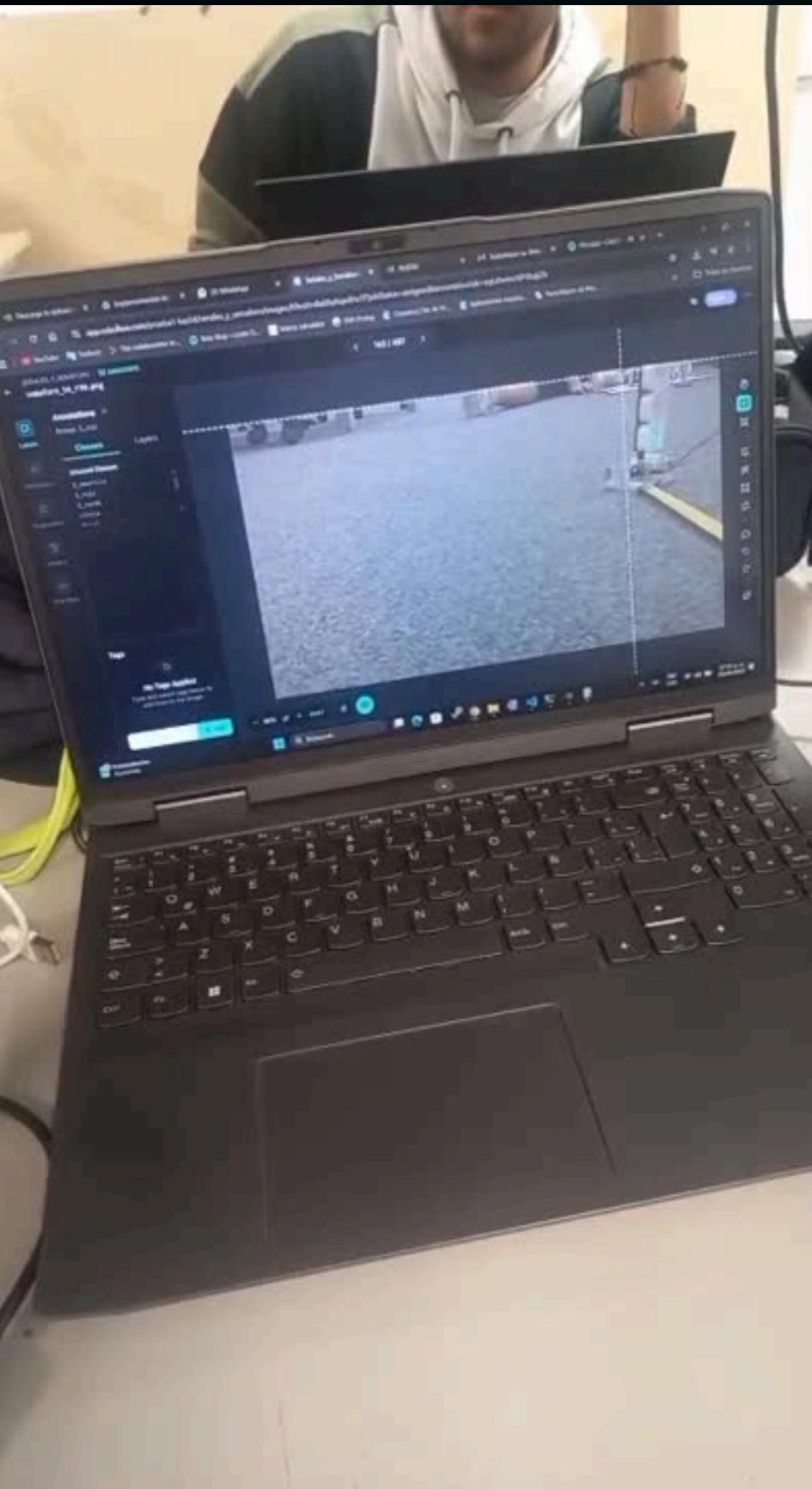
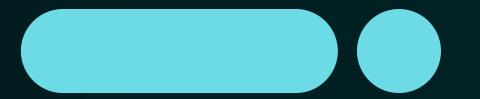
- Validación de cada nodo por separado.
- Simulación de situaciones reales: giros, pérdida de línea, interrupción por semáforo.
- Afinación de parámetros velocidades, tiempos de acción.

5. Grabación y documentación

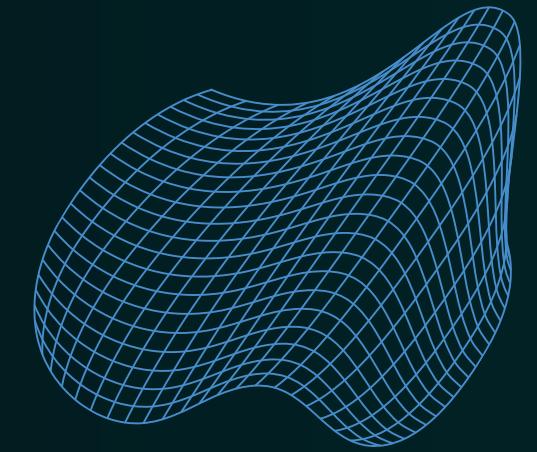
- Registro de comportamiento en video para análisis y evidencia.
- Documentación técnica por nodo y generación de gráficas de comportamiento.



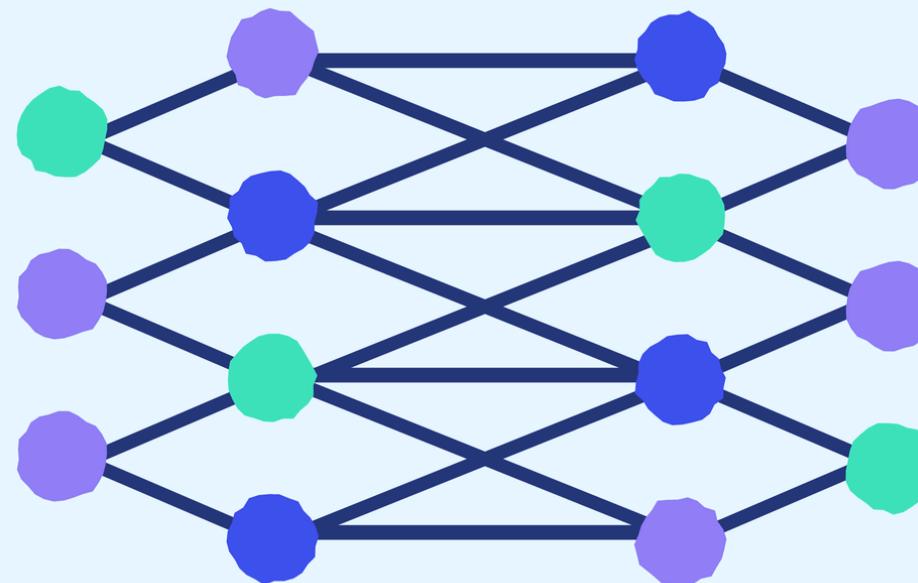
ROBOFLOW



- ✓ **Toma de fotografías**
- ✓ **Etiquetado de clases**
- ✓ **Creación del dataset**



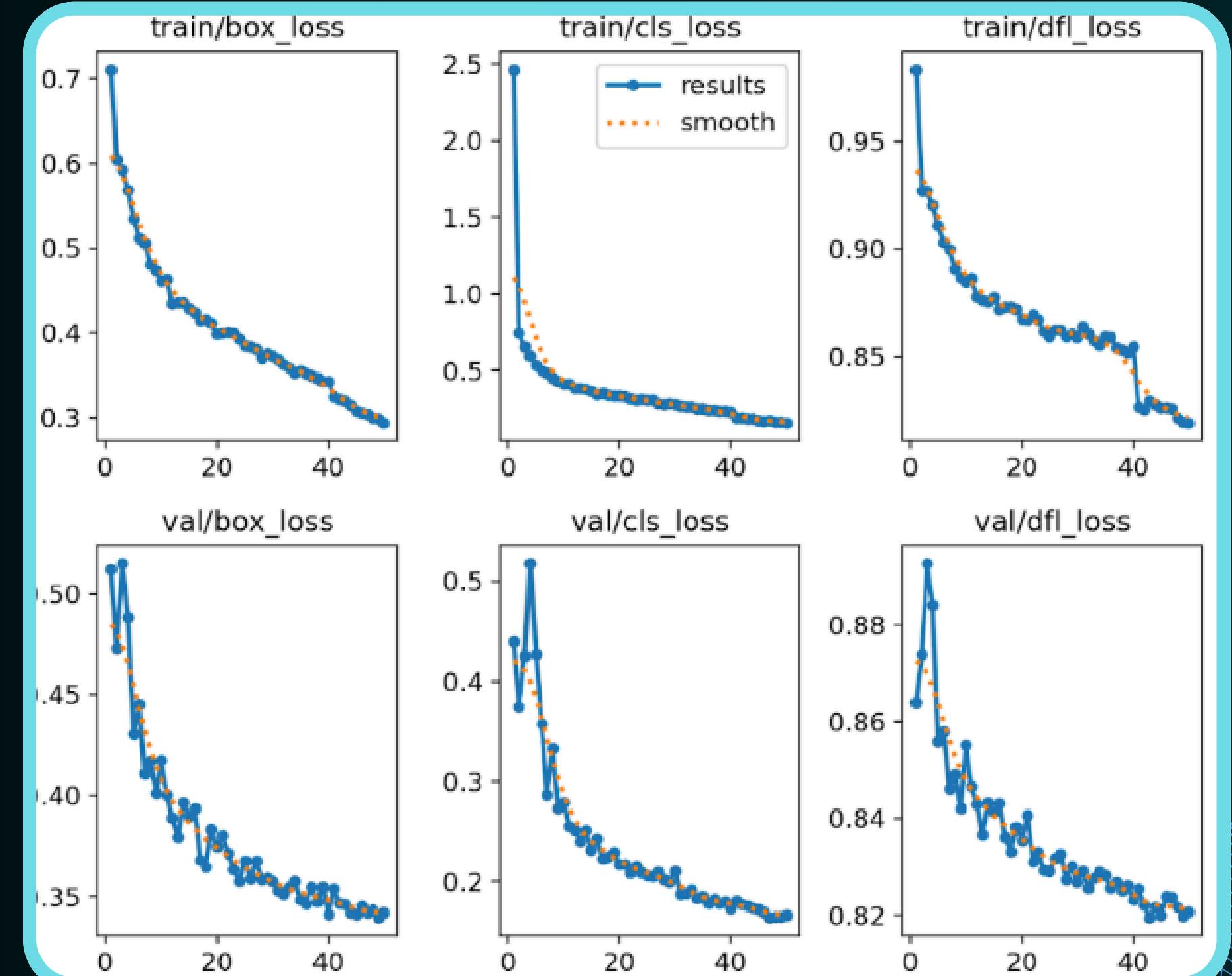
MODELO DE RED NEURONAL



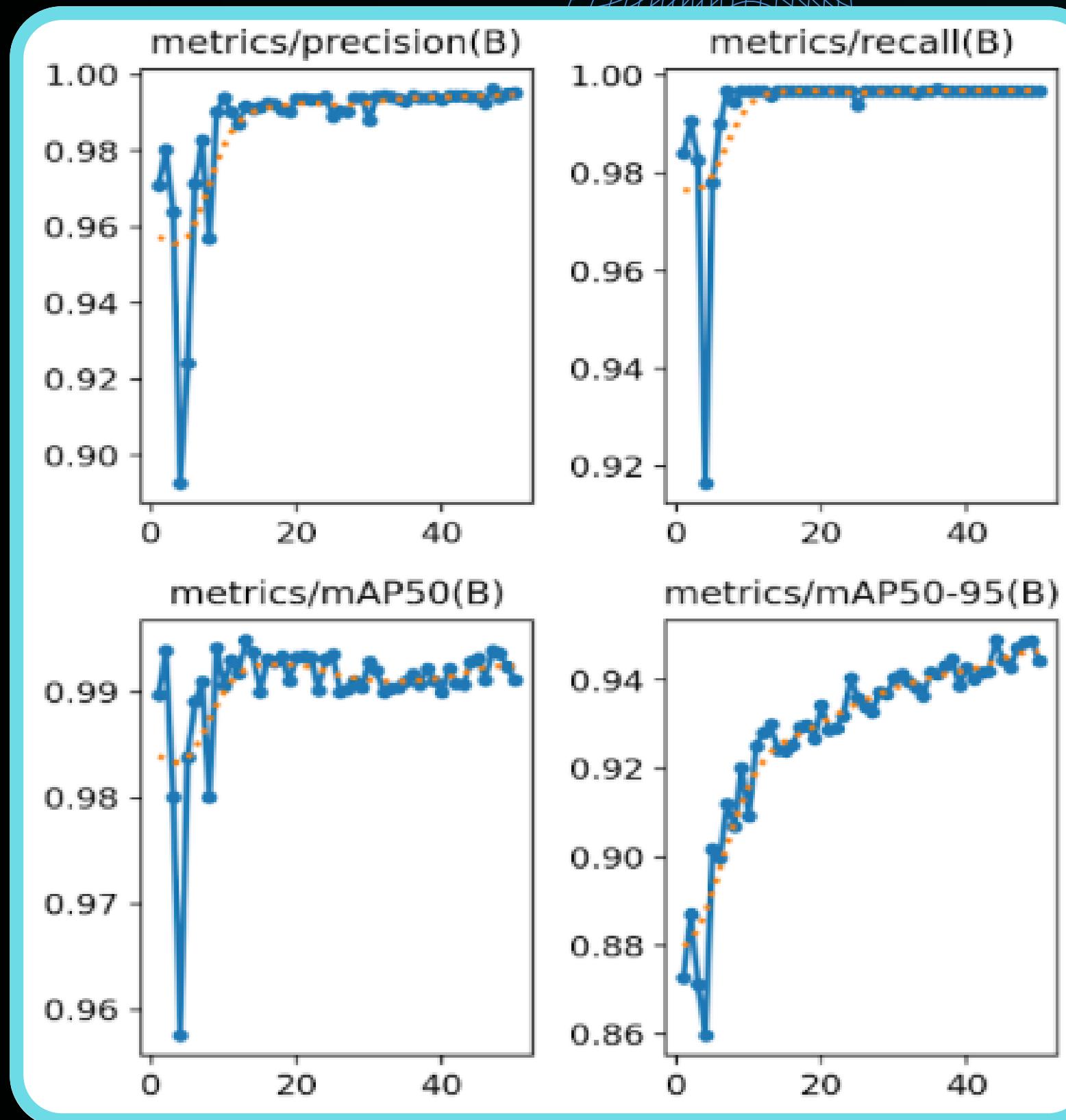
- Se usa la librería Ultralytics e importamos YOLO. También se importan funciones para mostrar resultados visuales como:
 - Loss
 - Métricas
 - Predicciones
 - Curvas Precision-Recall
 - Matriz de confusión

GRAFICOS DE PERDIDA

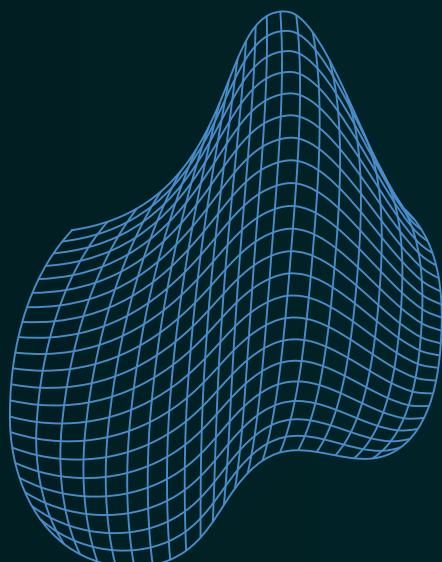
- **box loss:** Precisión de las cajas (bounding boxes).
- **cls loss:** Clasificación correcta de los objetos.
- **dfl loss:** Detección de presencia/ausencia de objeto.
- **Objetivo:**
Las curvas deben disminuir con el tiempo (menos error = mejor aprendizaje).



MÉTRICAS DEL MODELO



- Precision: Qué porcentaje de las detecciones fueron correctas.
- Recall: Qué porcentaje de los objetos reales fueron detectados.
- mAP50 y mAP50-95: Métricas combinadas de precisión y recall.



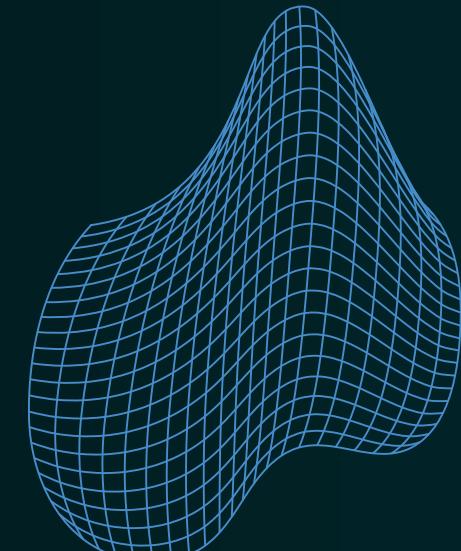
CURVAS PRECISION-RECALL

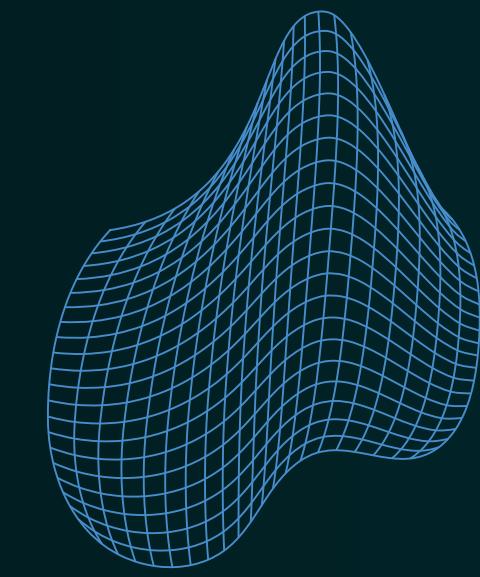
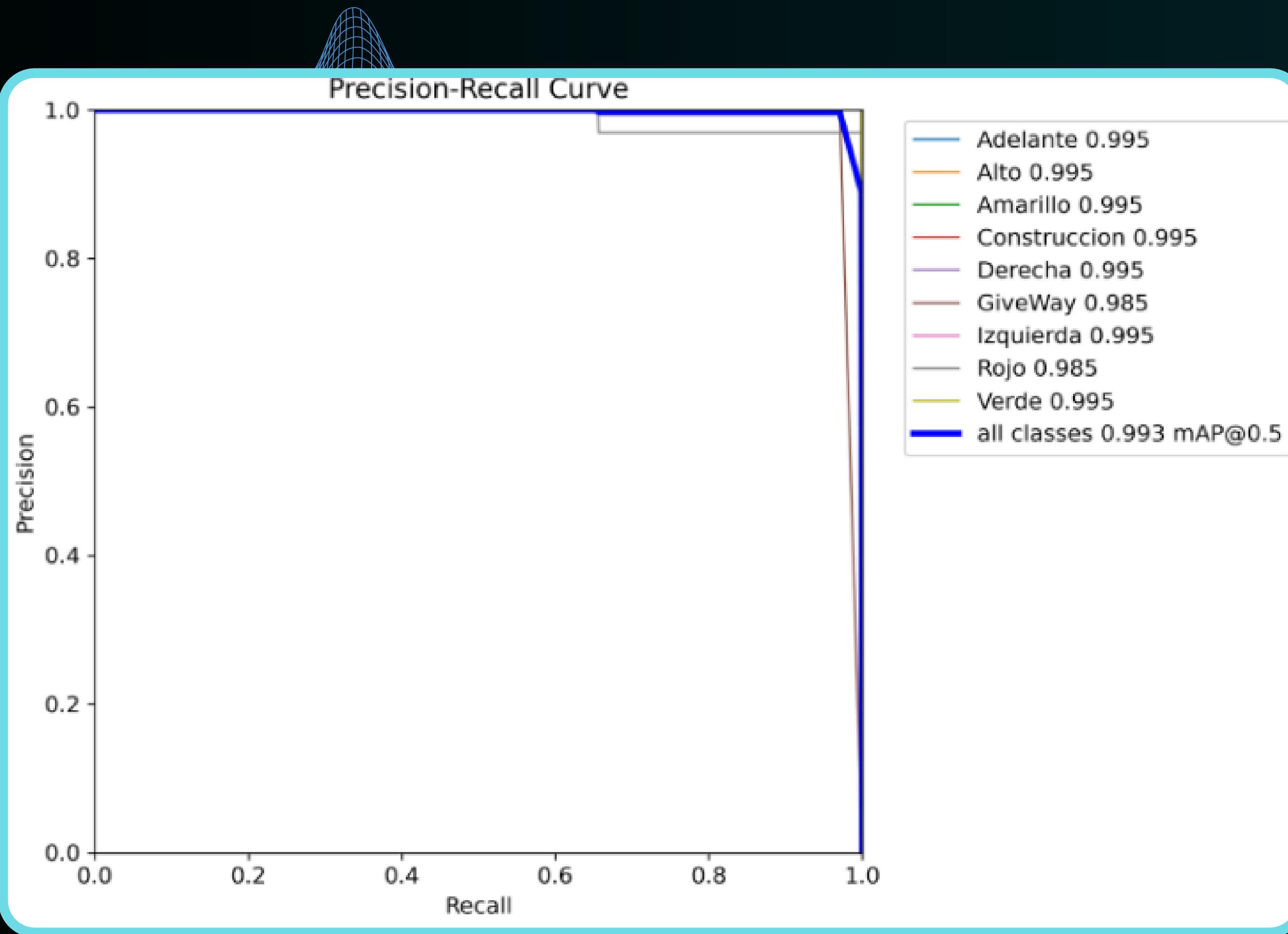
Precisión: % de aciertos entre las detecciones.

Recall: % de objetos reales detectados.

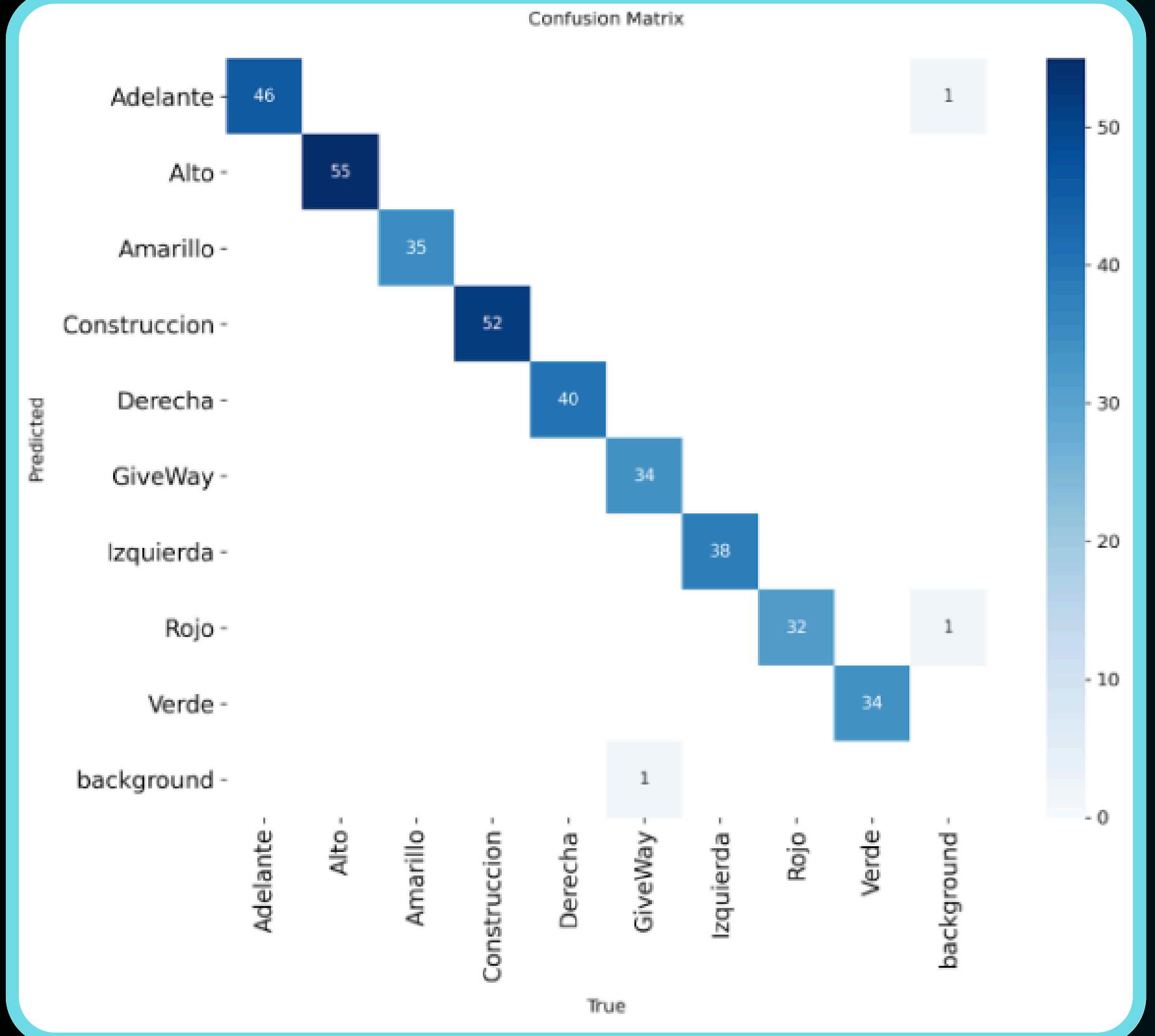
AUC (Área bajo la curva): Más cerca de la esquina superior derecha = mejor.

- Curvas bajas o a la izquierda indican bajo recall o precisión.
- Sirve para ver si el modelo detecta bien todas las clases o falla en alguna.





MATRIZ DE CONFUSIÓN



- Compara predicciones con etiquetas reales.
- **Diagonal (azul):** predicciones correctas.
- **Fuera de la diagonal:** errores o confusiones de clase.
- Permite analizar:
 - Qué tan bien clasifica cada clase.
 - Qué errores comete y cuán frecuentes son.

PREDICCIONES DEL BATCH DE VALIDACIÓN

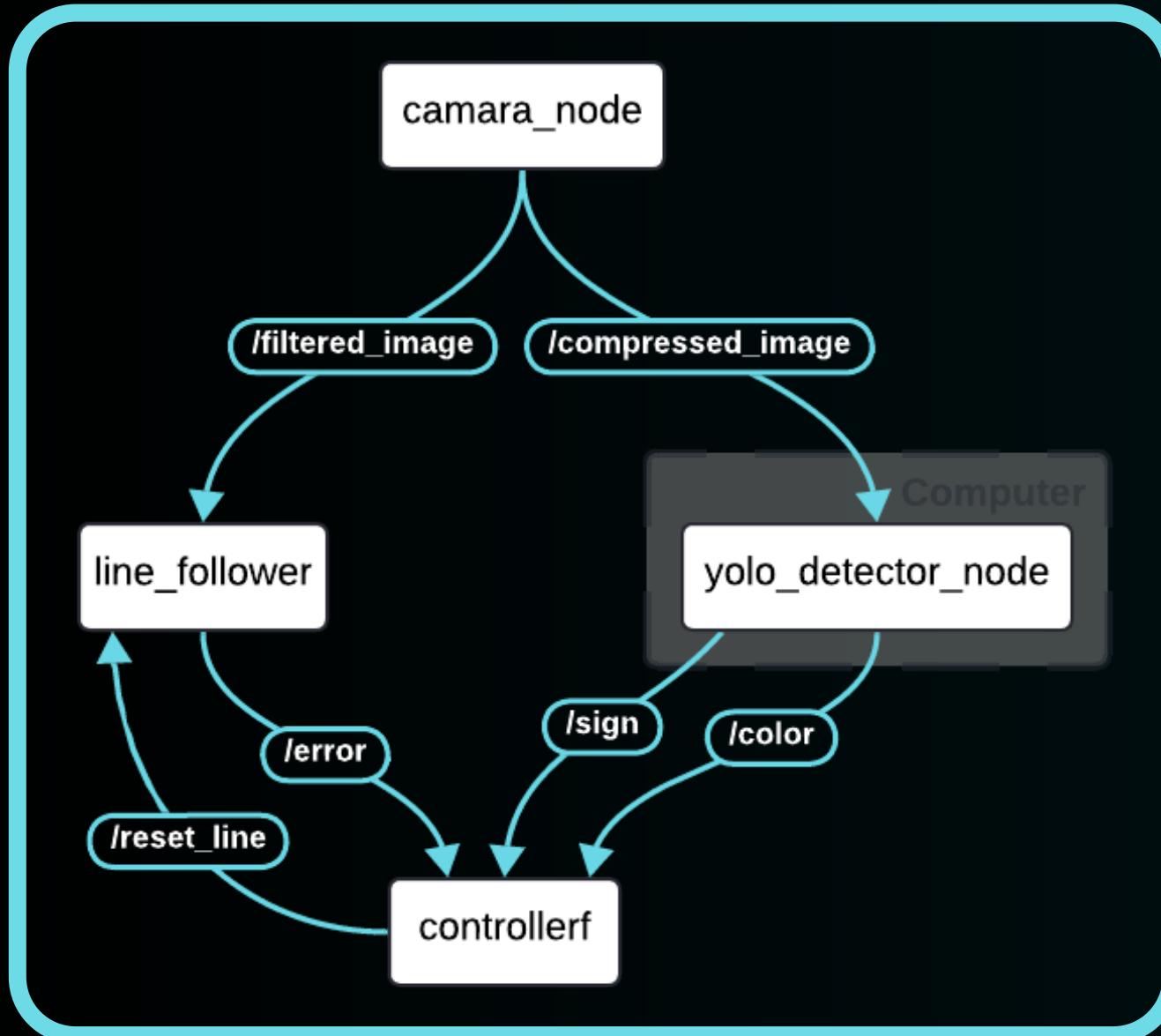
Se usan imágenes del conjunto de validación (no vistas durante entrenamiento).

Se visualiza:

- Objetos detectados.
- Clase asignada.
- Nivel de confianza.
- Localización con bounding boxes.



DIAGRAMA



INTERCONEXIÓN DEL SISTEMA

- **camera_node**

Captura, rota y filtra la imagen para el seguimiento de línea. Publica imagen binaria para procesamiento y JPEG comprimido para detección.

- **line_follower**

Detecta tres centroides sobre la línea negra. Publica un error proporcional robusto incluso en situaciones críticas.

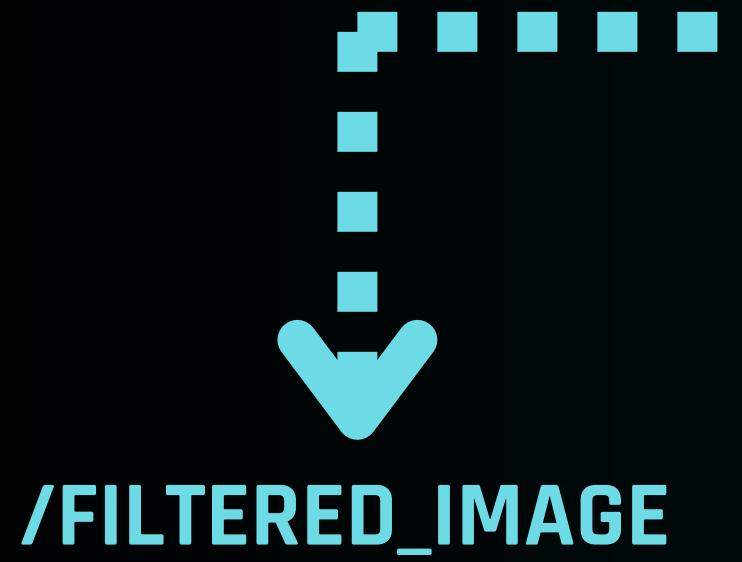
- **yolo_detector_node**

Usa YOLOv8 para detección de señales de tránsito y semáforos. Aplica umbrales de área, persistencia temporal y lógica de prioridad entre señales. Publica `/color` y `/sign` en paralelo

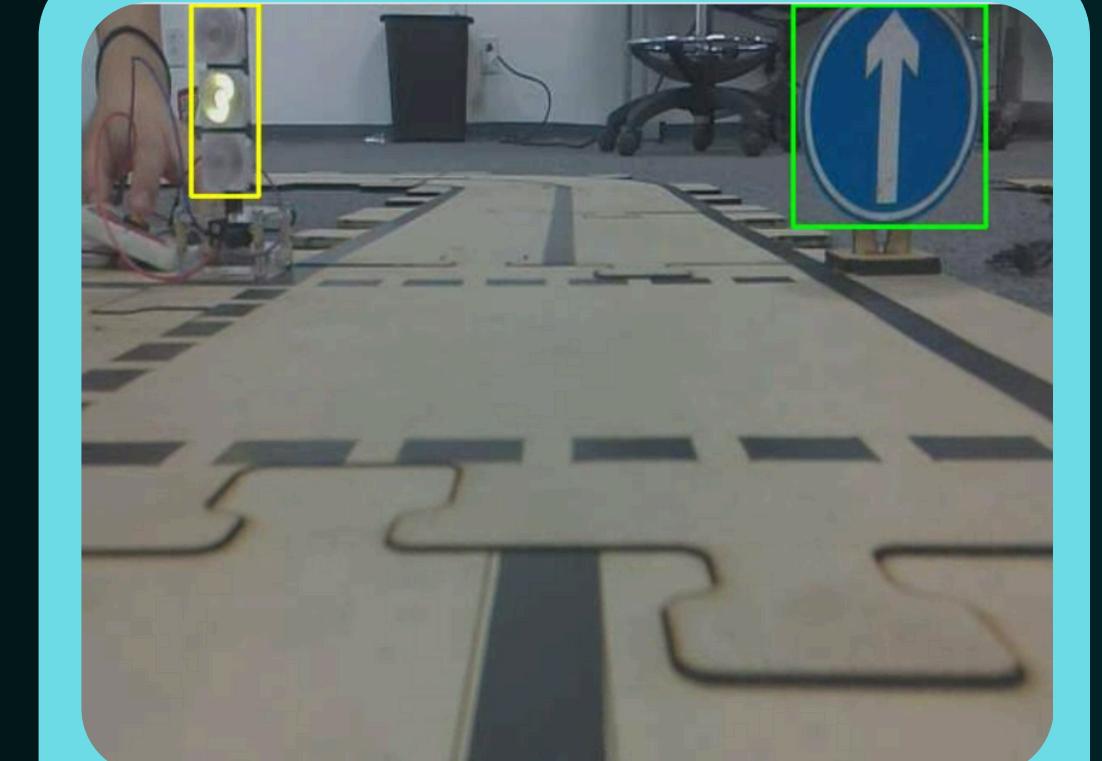
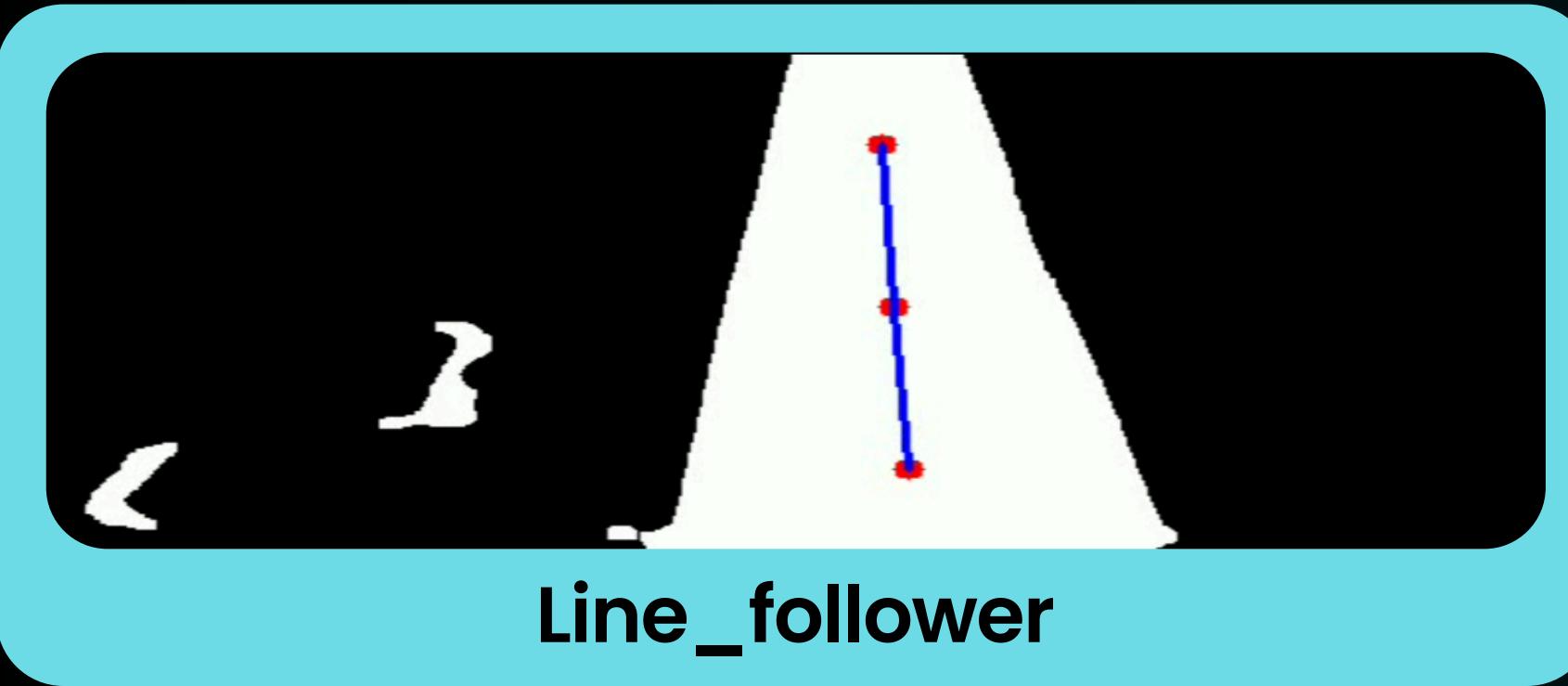
- **controllerf**

Controla la velocidad lineal y angular del robot. Interpreta el estado del semáforo para modular comportamiento. Implementa máquina de estados para `turn_left`, `turn_right`, `ahead`, `giveway`, `work` y `stop`.

CAMERA NODE



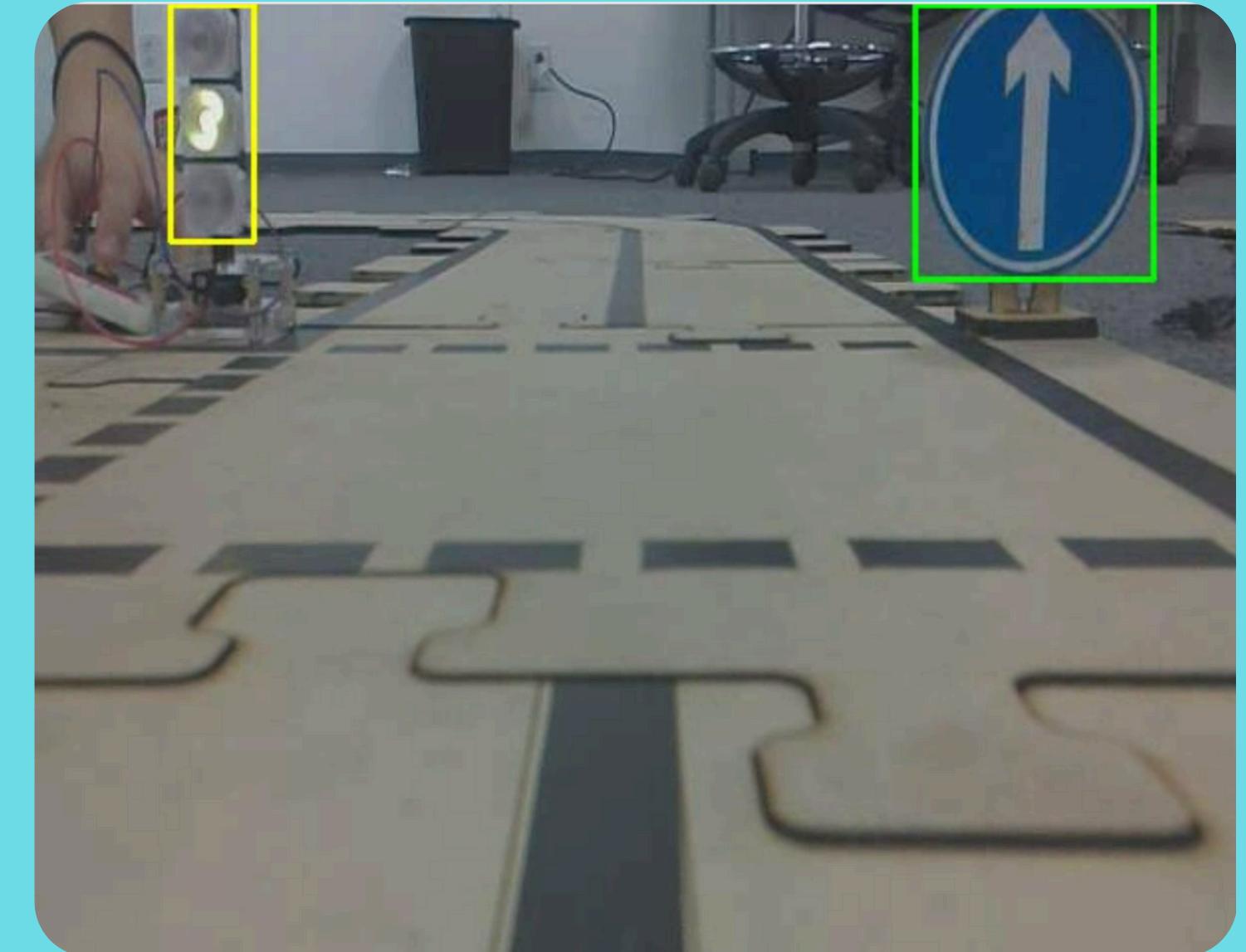
Se encarga recibir las imágenes del tópico `/video_source/raw` de la cámara CSI del robot, las procesa para extraer una región de interés y comprimir las imágenes para poder enviarlas periódicamente en formato JPEG de la imagen original para visualización y monitoreo



CAMERA NODE

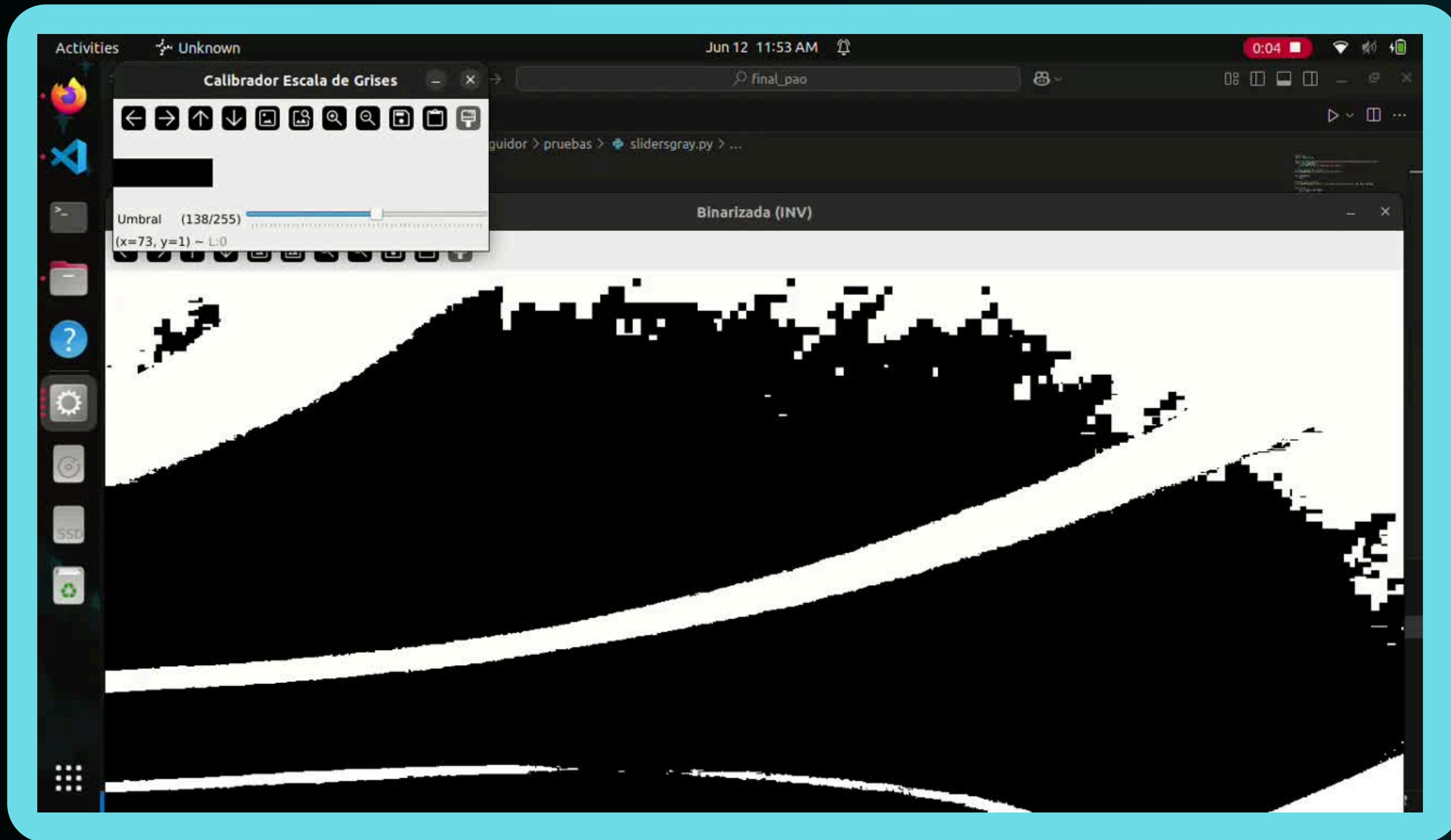
- Cada vez que llega una imagen, inicia con convertir la imagen ROS a un formato OpenCV BGR
- Corrige la orientación con rotación 180°
- Redimensiona la imagen a un tamaño 640x480
- Comprime la imagen a JPEG, usa la calidad en 50%, lo que permite un equilibrio entre el tamaño reducido para la transmisión y calidad visual

/COMPRESSED_IMAGE

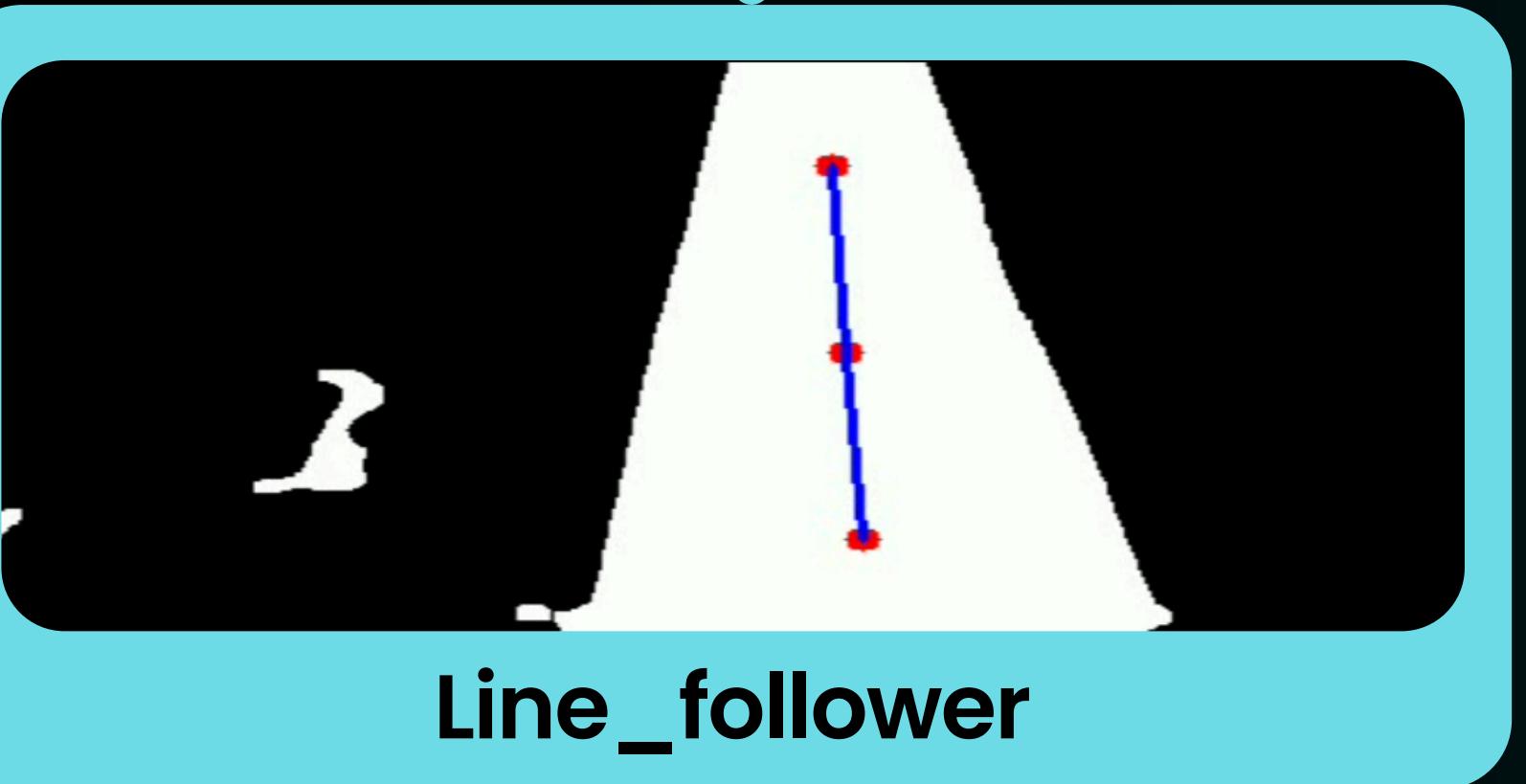
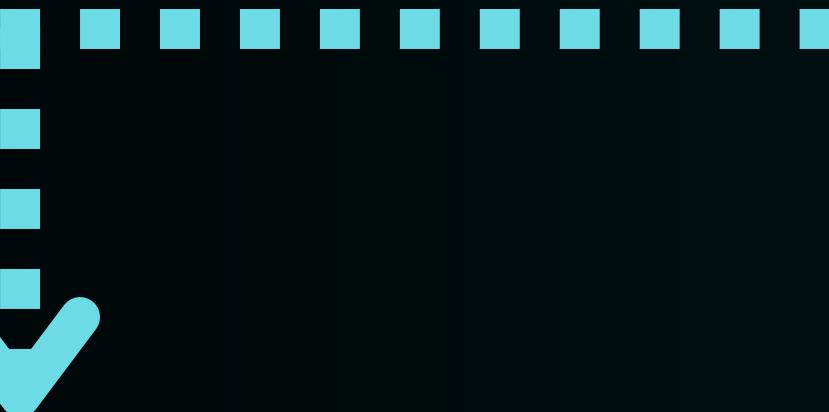


Detection

AJUSTE UMBRAL

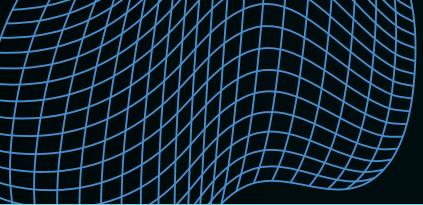


/FILTERED_IMAGE



CAMERA NODE

- Aplica a la imagen un tamaño estándar para el proceso de la parte inferior de [230:480, 200:1000]
- Convierte a escala en grises
- Aplica un umbral inverso donde una línea negra se verá ahora en blanco
- Establece un filtrado morfológico para eliminar ruido (Pequeños blancos aislados), busca patrones 3x3 píxeles y hace 2 iteraciones , eliminando el ruido sin afectar la línea principal
- Comprime la imagen a JPEG, usa la calidad en 50%, lo que permite un equilibrio entre el tamaño reducido para la transmisión y calidad visual



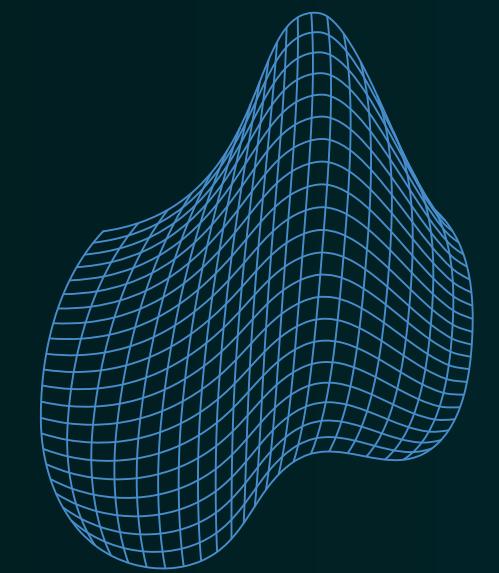
DETECTION.PY



✓ **Clase principal:**
YoloDetectorNode

- Suscripción a /compressed_image.
- Publicadores:
 - /color: Publica color de semáforo detectado.
 - /sign: Publica señal vial detectada.
- Carga un modelo YOLO entrenado (best.pt).
- Crea un video con las detecciones (bounding_box_vid.avi).

✓ **Función principal:** image_callback

- Recibe la imagen comprimida.
 - Decodifica la imagen con OpenCV.
 - Realiza inferencia con YOLO.
 - Filtra objetos según área mínima y persistencia temporal.
 - Anota la imagen (bounding boxes, etiquetas, colores).
 - Publica los códigos detectados.
 - Muestra logs con íconos y nombres de clases.
- 

DETECTION.PY

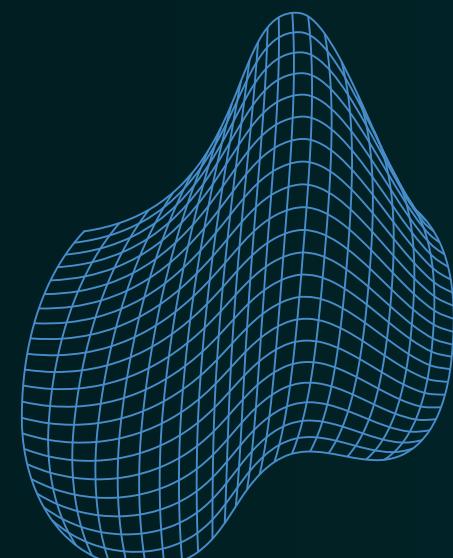
✓ Códigos Personalizados

```
self.custom_class_codes = {  
    "S_rojo": 1, "S_verde": 2, "S_amarillo": 3,  
    "right": 4, "left": 5, "front": 6,  
    "stop": 7, "giveway": 8, "work": 9  
}
```

- Se asocia cada clase detectada con un número.
- Cada señal tiene un Área específica.
- Sirve para simplificar la comunicación por tópicos.

✓ Persistencia y Filtrado

- Se guarda cuándo se detecta por primera vez cada clase.
- Se valida que la detección permanezca al menos 0.5 segundos antes de aceptarla (para evitar falsos).
- Esto reduce errores por detecciones espurias.





DETECTION.PY



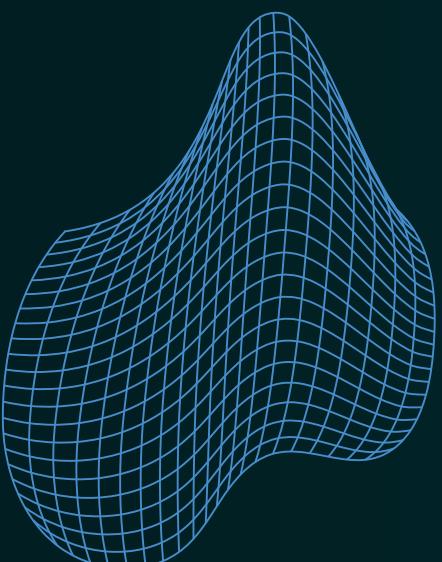
✓ Salida y Visualización.

El frame se anota visualmente:

- Rectángulo de color según clase.
- Etiqueta con nombre, confianza y área.
- Se guarda en video con cv2.VideoWriter.

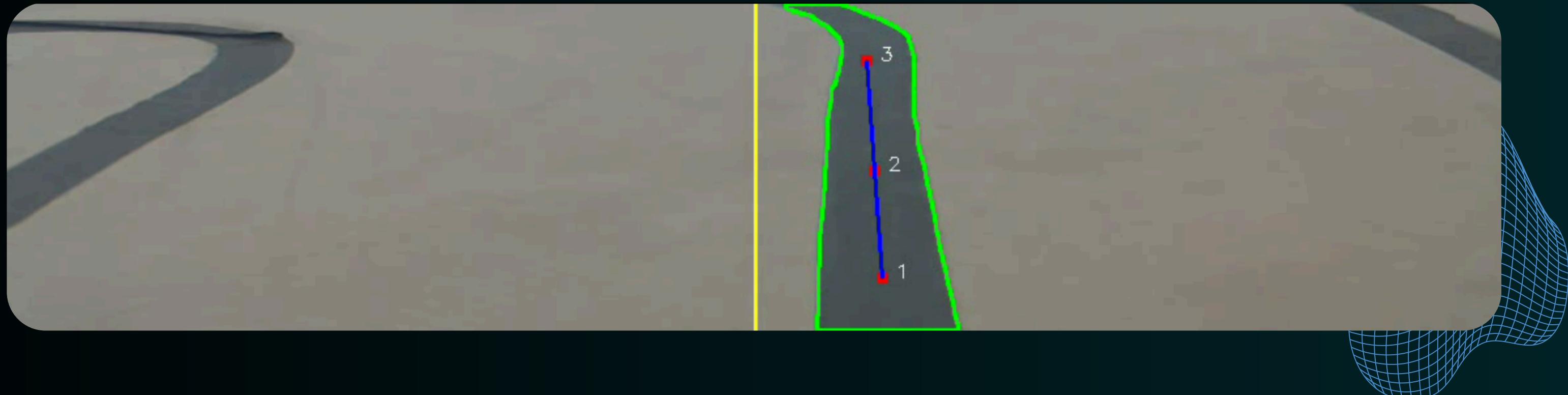
✓ Publicación de los resultados.

- Color del semáforo se publica en /color.
- Señal vial detectada se publica en /sign.
- Si no se detecta nada, se publica 0.



LINE FOLLOWER

- ✓ La imagen de ROS se convierte en formato OpenCv
- ✓ Busca con `cv2.findContours` los contornos blancos
- ✓ Detecta el área contorno más grande (500 px), para generar una máscara binaria con el contorno
- ✓ Tenemos división en Tres regiones dentro de la línea
- ✓ Con `cv2.moments` calculamos los centrodes y ponemos un círculo rojo



LINE FOLLOWER

CÁLCULO DEL ERROR

- ✓ El error representa cuánto se ha desviado el centro de la línea (detectado por visión) respecto al centro de la imagen de la cámara. Este valor se utiliza generalmente en un controlador proporcional (P) para corregir la dirección del robot.
 - ✓ Si los centroides existen se calcula el error comparando la posición de x contra el centro de la imagen
-
- error = $(x \text{ del centroide } 2 - \text{centro de la imagen}) / \text{centro imagen}$



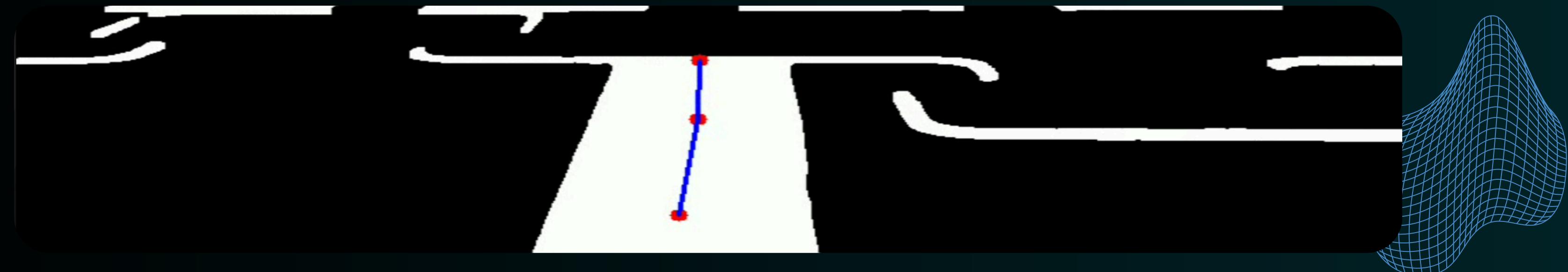
LINE FOLLOWER

MODO RECOVER

- ✓ SI el centroide 3 desaparece se evaluá si desapareció en el centro
- ✓ Se publica el error especial 2.0, para detener el movimiento del puzzlebot
- ✓ Envía el error usando el centroide 2, o el 1 si no existe el 2
Termina el modo el error sea $<\pm 0.05$

VERIFICACIÓN DE SALTOS

- ✓ Evaluá si el centroide 2 salta mucho comparado con el anterior, en un umbral de 90 px en "x" y "y"
- ✓ SI sí se ignora el centroide actual y se siguen usando las anteriores posiciones



CONTROLLER

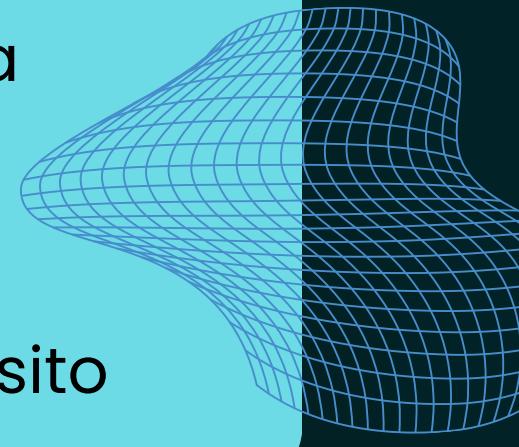
OBJETIVO

Controlar el movimiento del robot diferencial integrando:

- Seguimiento de línea (control proporcional).
- Interpretación del semáforo.
- Respuesta a señales de tránsito detectadas por visión.
- Gestión de recuperación visual en caso de pérdida de línea.

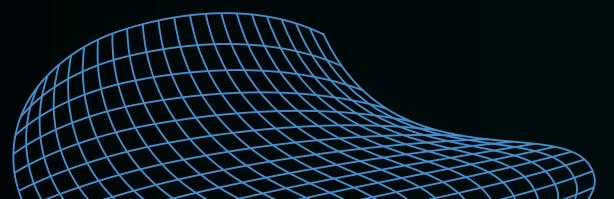
Entradas:

- **/error:** error de línea
- **/color:** estado del semáforo
- **/sign:** señal de tránsito



Salidas

- **/cmd_vel:** velocidad lineal y angular del robot
- **/reset_line:** reset del seguidor de línea



CONTROLADOR PROPORCIONAL

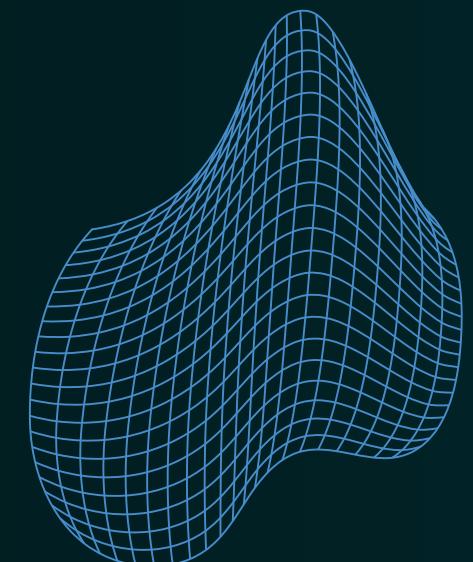
$$\omega = -K_p \cdot e$$

Donde:

- ω es la velocidad angular (rad/s).
- K_p es la ganancia proporcional (ajustada empíricamente a 0.5).
- El signo negativo asegura que el giro sea hacia la dirección opuesta al error.

Velocidad lineal:

- Normal: `base_speed = 0.09 m/s`



Máquina de estados

STOP

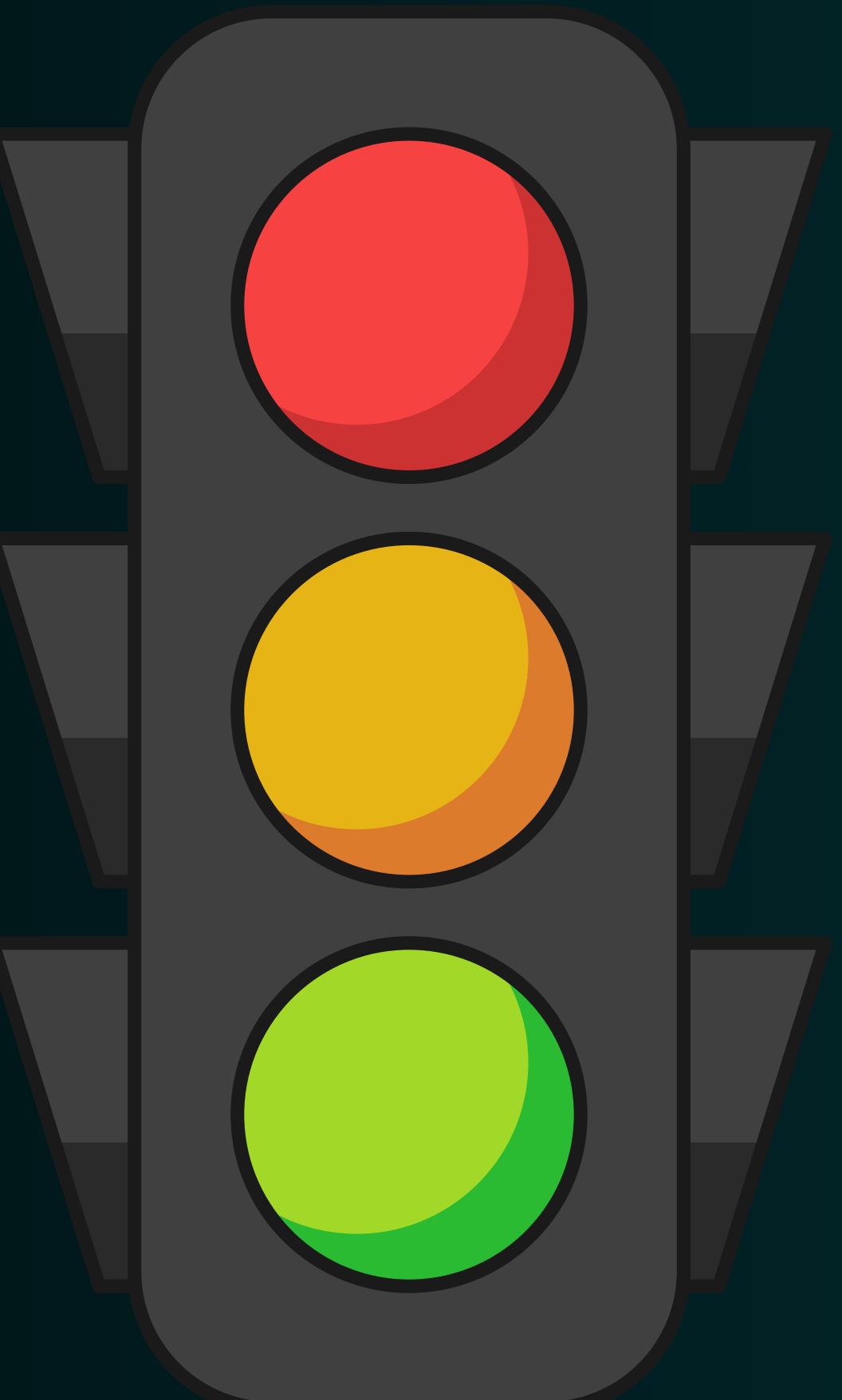
- Alto cuando vea luz roja
- Hasta que vea una luz verde.

SLOW

- Cuando vea una luz amarilla, conduzca lentamente
- Hasta que vea una luz roja para detenerse.

GO

- Al ver luz verde continuar con el recorrido



SEÑALES



Turn right

- Se prepara para girar a la derecha y ejecuta el giro.



Turn left

- Se prepara para girar a la izquierda y ejecuta el giro.



Ahead

- Va en dirección frontal (hacia delante).



Stop

- Se detiene hasta que la señal desaparezca.



Give way

- Disminuye la velocidad



Roadwork

- Disminuye la velocidad

FUNCIÓN DE RECUPERACIÓN VISUAL RECOVER

Activada cuando el nodo line_follower publica un error = 2.0

Dos fases:

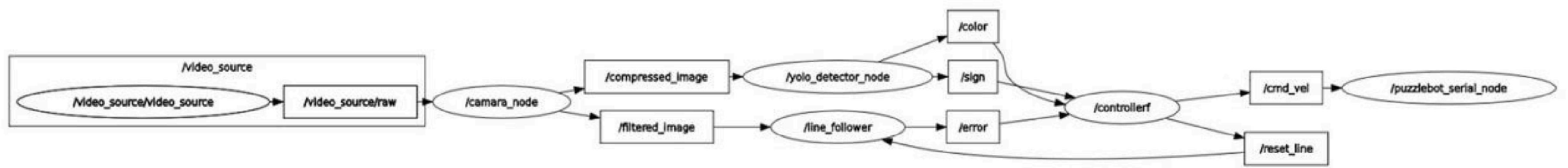
1. Corrección del error angular hasta que se alcance tolerancia $|e| < 0.05$ $|e| < 0.05$ $|e| < 0.05$
2. Esperar señal válida para reactivar el comportamiento normal

Durante RECOVER:

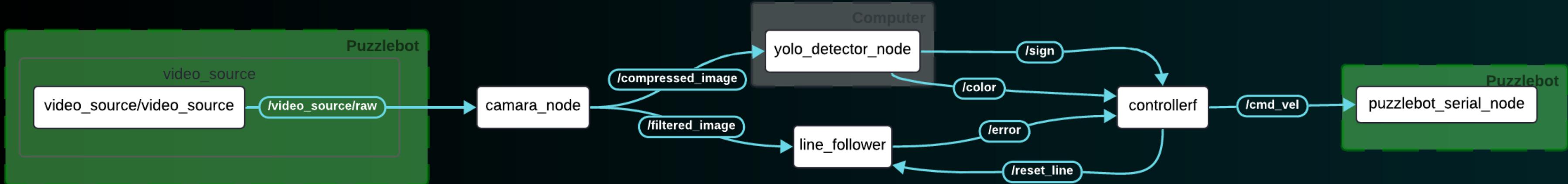
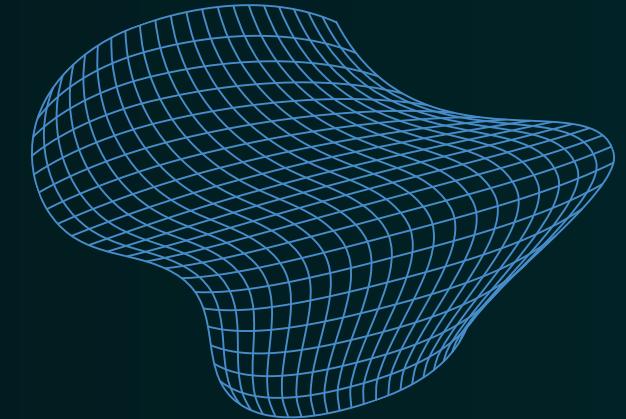
- Se pausa toda acción de señales.
- Se detiene avance hasta corregir visibilidad.
- Si el tiempo de espera excede 5 s, se sale automáticamente del modo.
- Mejora la robustez ante pérdida de línea o detección incompleta.

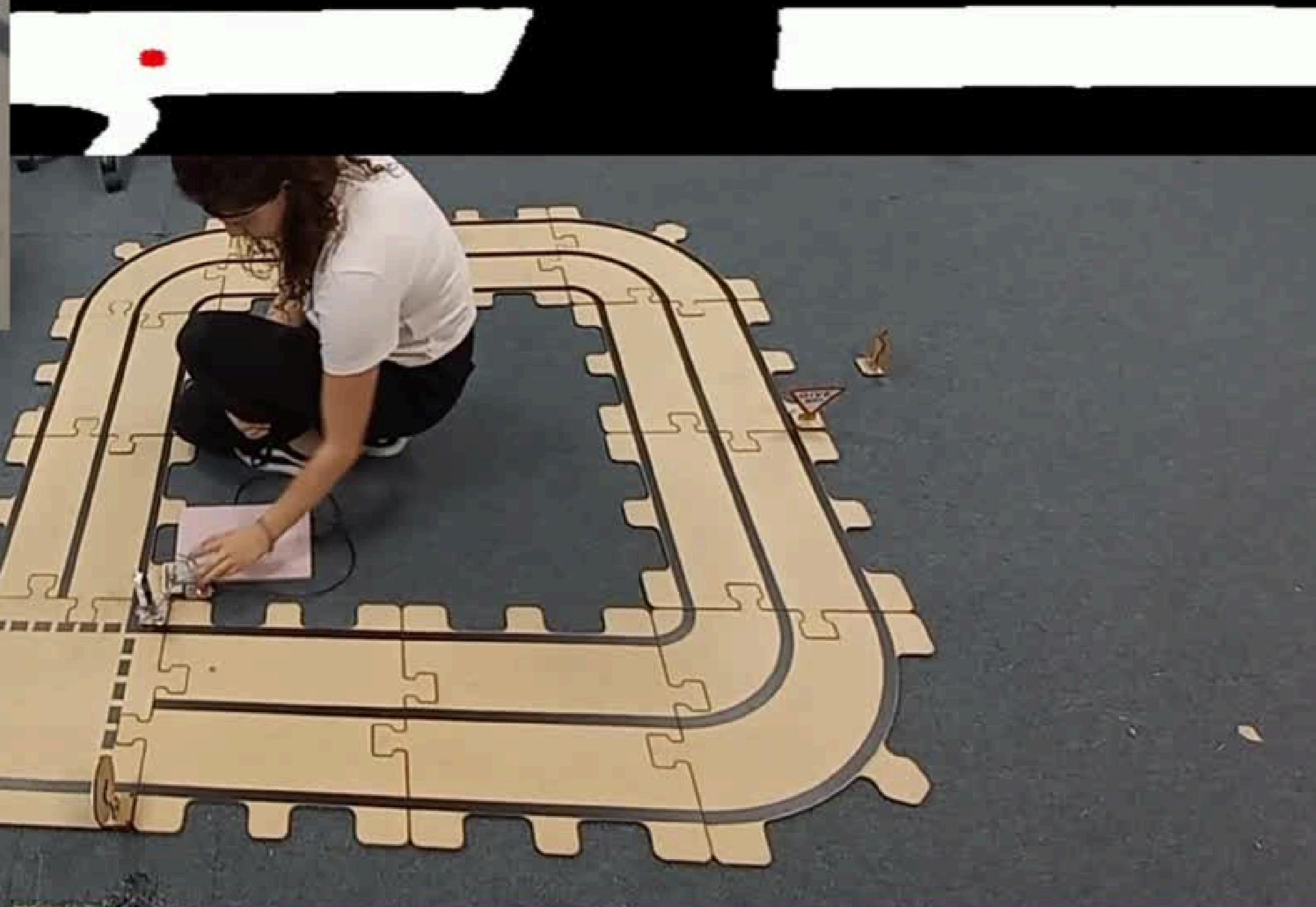
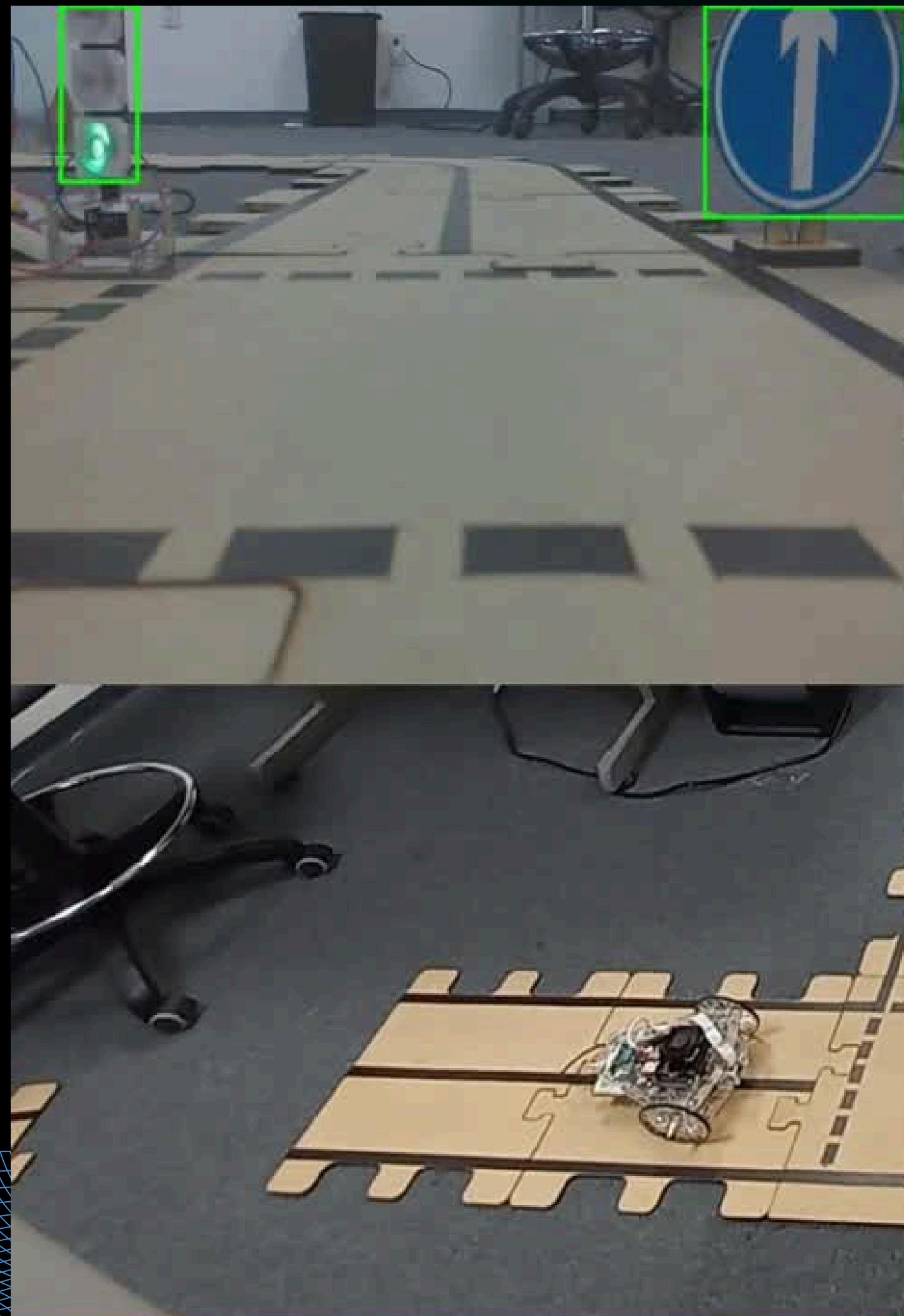
```
if self.recovering:  
    if self.recover_waiting_signal:  
        elapsed = (self.get_clock().now() - self.recover_start_time).nanoseconds * 1e-9  
        if elapsed > self.recover_timeout:  
            self.get_logger().warn("⌚ Tiempo agotado esperando señal en modo RECOVER")  
            self.recovering = False  
        else:  
            twist.linear.x = 0.0  
            twist.angular.z = 0.0  
            self.get_logger().info("🔴 Esperando señal tras RECOVER...")  
    self.cmd_vel_pub.publish(twist)  
return
```

RESULTADOS



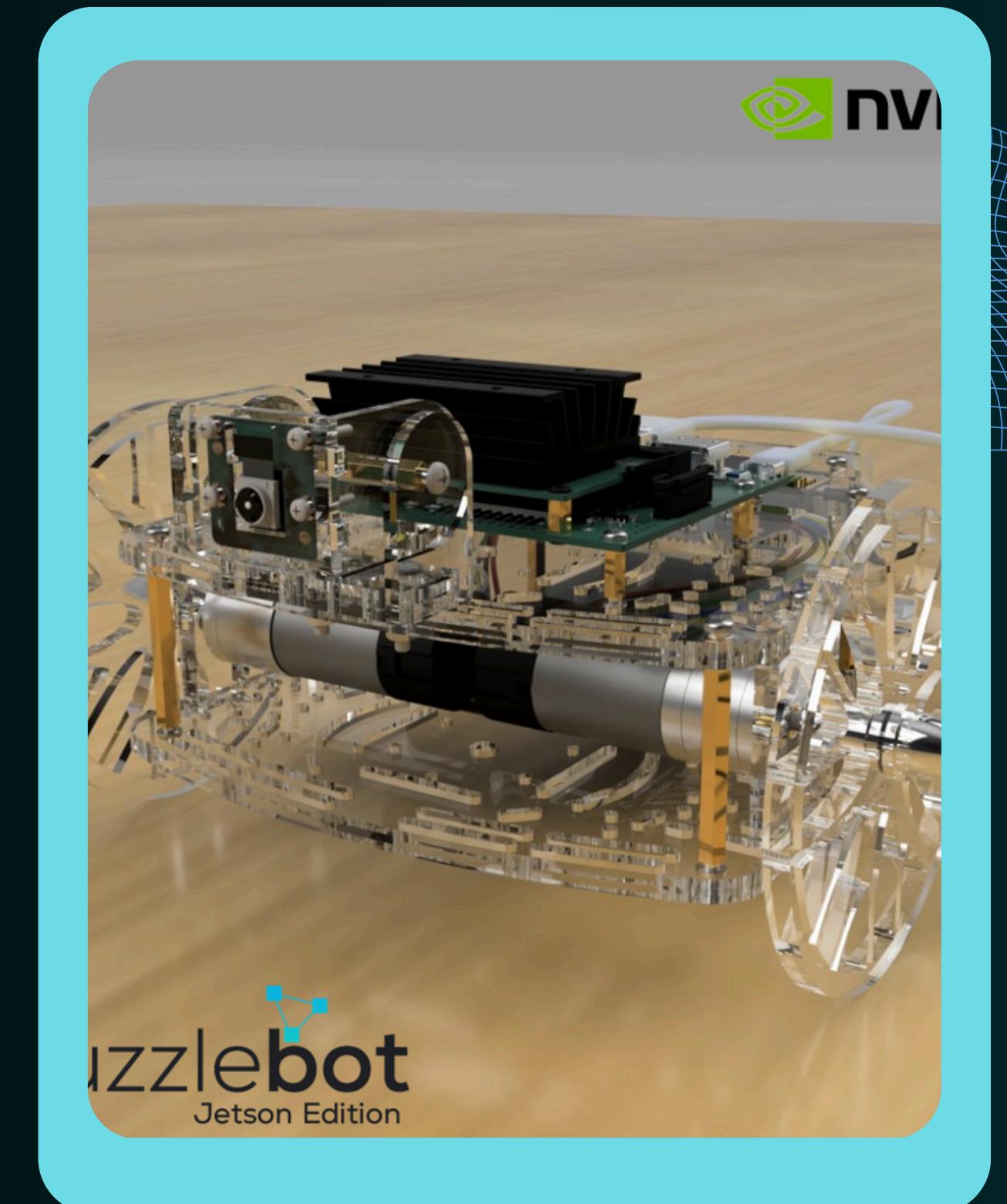
NODE GRAPH





CONCLUSIÓN

Se desarrolló un sistema autónomo de navegación para el Puzzlebot capaz de seguir una línea, interpretar señales de tránsito y responder al estado de un semáforo mediante visión por computadora. El robot ejecutó maniobras como giros, detenciones y cambios de velocidad de forma autónoma, cumpliendo los objetivos propuestos y consolidando habilidades en percepción, control y autonomía móvil.



GRACIAS



Github