Práctica 1. Robótica hardware

Autores: Hugo Sánchez e Iván Prieto

En esta práctica se han desarrollado una serie de nodos implementados con ROS2 los cuales llevan a cabo la conducción autónoma de un vehículo simulado en el entorno Webots y la detección de señales de este.

Vamos a definir uno a uno todos los nodos, su comportamiento y sus comunicaciones con los demás nodos:

- **Nodo 1: my_robot_driver.py:**

  Este nodo actúa como puente entre ROS2 y Webots ya que se comunica con los demás nodos y es el único que usa la API del robot simulado.

  Su propósito es traducir los comandos de ROS2 en llamadas concretas a la API del coche, este nodo esta subscrito al tópico **"/cmd_vel"** por el que recibe la velocidad y el ángulo de giro del coche y llama a la API del robot para mandarle las ordenes de movimiento al coche.

```python
def step(self):
    rclpy.spin_once(self.__node, timeout_sec=0)
    v = self.__target_twist.linear.x
    desired_angle = self.__target_twist.angular.z

    steering_angle = max(-MAX_STEERING_ANGLE, min(desired_angle, MAX_STEERING_ANGLE))

    self.__robot.setCruisingSpeed(v)
    self.__robot.setSteeringAngle(steering_angle)
```

  La función step se va a ejecutar cada 10 ms (variable definida en el .wbt del mundo de Webots), lo que hace esta función es recoger la velocidad y el ángulo deseados, después se asegura que el ángulo este dentro de los límites de giro permitidos del coche, por último, manda la velocidad con **"setCruisingSpeed()"** y el ángulo con **"setSteeringAngle()" .**

- **Nodo 2: lane_controller.py:**
  Este nodo recibe la información **lane_detector** y **sign_detector**, lo procesa, decide un comportamiento y lo manda a **my_robot_driver**.

  Este nodo esta subscrito a **"/lane_error"** que recibe el error en pixeles del carril y a **"/speed_command"** recibiendo la velocidad objetivo del coche. Además, publica en **"/cmd_vel"** la velocidad y el giro.

  La función encargada de controlar el giro del coche es error_callback:

```python
def error_callback(self, msg):
    error = float(msg.data)
    prop = self.Kp * error

    if self.current_speed_target != 0.0:
        self.integral += error
        if self.integral > self.integral_max:
            self.integral = self.integral_max
        elif self.integral < -self.integral_max:
            self.integral = -self.integral_max

    int_term = self.Ki * self.integral

    derivative = self.Kd * (error - self.last_error)
    self.last_error = error

    self.current_angular_target = -(prop + int_term + derivative)

    self.publish_command()
```

  Esta función recibe el error del carril, es decir, cuantos pixeles nos hemos alejado del centro del carril, y mediante un **controlador PID** publica el ángulo de giro deseado.

  Para la obtención de los parámetros de las constantes se ha ido probando valores hasta conseguir una configuración con la cual el coche sigue la línea y no se desvía. Estos son los parámetros:

```python
# PID
self.Kp = 0.004
self.Ki = 0.0
self.Kd = 0.004
```

- **Nodo 3: lane_detector.py:**

  El nodo lane_detector recibe a través del tópico /car/road_camera/image_color la imagen de la cámara que apunta al suelo, detecta la línea que separa los carriles y manda el error de centrado a través del tópico /lane_error.

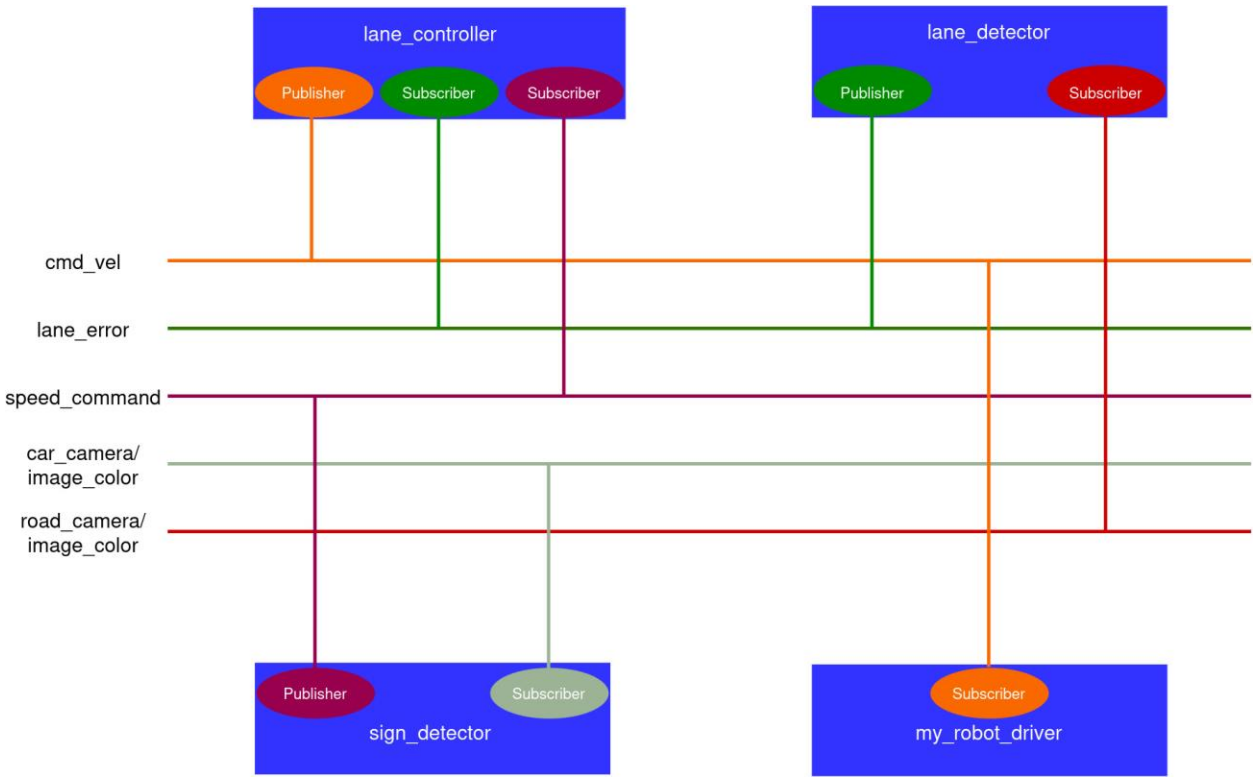
  Para detectar las lineas, se hace la media del brillo de la imagen de cada columna para, posteriormente, obtener el valor máximo de todas las columnas. Tras esto, se comprueba si es asfalto y se sigue recto sin desviarse, o tiene que hacer alguna corrección.

  El mayor reto del lane_detector, fue conseguir que el nodo consiguiese diferenciar entre el paso de cebra y la línea de separación al tener un color similar. La solución que se implemento fue medir el brillo de ambos colores y, gracias a un umbral de confianza, se ignora el paso de cebra al ser más brillante la línea de separación de carriles que este.

```
#Umbral de confianza
self.LINE_THRESHOLD = 170.0

# Media
column_means = np.mean(img, axis=(0, 2))

max_brightness = np.max(column_means)

if max_brightness < self.LINE_THRESHOLD:
    error = 0
    #Línea no detectada
else:
    center_index = int(np.argmax(column_means))
    error = int(center_index - 256)
   #Línea detectada

 #Centro detectado
self.error_pub.publish(Int32(data=-error))
```

- **Nodo 4: sign_detector.py**

  Este nodo se encarga de detectar las señales mediante opencv, primero se leen las imágenes y luego se comprueba si la camara captura alguna de las imagnes con la función “image_callback()”. Cuando se detecta una imagen se publica la acción que tiene que hacer el coche (reducir velocidad, pararse, aumentar la velocidad) en el tópico “speed_comand”



cmd_vel: tópico encargado del control del vehículo en el que se intercambian mensajes de tipo twist (es un paquete de varios datos). El nodo lane_controller actúa como publicador de la velocidad y, my_robot_driver que se comporta como suscriptor.

lane_error: tópico en el que se intercambian mensajes de tipo Int32. El nodo lane_detector actúa como publicador y, lane_controller, como suscriptor.

speed_command: tópico en el que se intercambian mensajes de tipo Float32. Sign_detector actúa como publicador y, lane controller como suscriptor.

car_camera/image_color: tópico en el que se intercambian mensajes de tipo Image. sign_detector actúa como suscriber.

road_camera/image_color: tópico en el que se intercambian mensajes de tipo Image. lane_detector actúa como suscriber.