Taller 5: CAN y ROS

5.1 Objetivos

- Identificar el funcionamiento de CAN
- Integrar un bus can a través de Arduino y sensores CAN
- Gestionar la comunicación CAN con ROS

5.2 Marco conceptual

CAN **(Controller Area Network)** es un protocolo de comunicación en serie diseñado para permitir que microcontroladores y dispositivos se comuniquen entre sí dentro de una red, sin necesidad de una computadora central. Fue desarrollado por Bosch a mediados de los 80 para su uso en la industria automotriz, con el fin de reducir la cantidad de cables y simplificar la electrónica de los vehículos.Hoy en día, es un estándar internacional (ISO 11898) utilizado en una amplia gama de aplicaciones.
Como breve historia, en 1983, Bosch comienza el desarrollo del protocolo CAN. El objetivo principal era reemplazar los costosos y pesados arneses de cables en los automóviles por una red de comunicación multiplexada. En 1991, fue adoptado por la compañía Mercedes-Benz en el modelo S-Class, marcando su primera aplicación comercial. Desde 1993, se estandariza como ISO 11898, lo que facilita su adopción global en diversas industrias más allá de la automotriz.

5.2.1 Como funciona CAN

CAN se basa en identificadores de mensajes. Cada mensaje tiene un ID único que define su prioridad y contenido. Un ID con un valor más bajo tiene una prioridad más alta. Si varios nodos intentan transmitir al mismo tiempo, el que tiene el ID más bajo (mayor prioridad) "gana" el control del bus y su mensaje se transmite sin interrupción. Los nodos perdedores esperan su turno. La trama de datos tiene la siguiente estructura :

- **Start of Frame (SOF):** Inicia la transmisión.
- **Arbitration ID:** Identificador del mensaje (11 bits en CAN estándar, 29 bits en CAN extendido).
- **Data Length Code (DLC):** Longitud de los datos (0-8 bytes).
- **Data Field:** Los datos útiles del mensaje.
- **CRC:** Suma de verificación para la detección de errores.
- **Acknowledge (ACK):** Los receptores confirman que han recibido el mensaje sin errores.
- **End of Frame (EOF):** Marca el final de la trama.

Entre sus características clave están su robustez y alta fiabilidad, la eficiencia en la comunicación, permitiendo la transmisión de datos en tiempo real, la capacidad de multi-

maestro, es decir, que todos los nodos pueden empezar una transmisión, y la flexibilidad, ya que los nodos pueden ser añadidos o eliminados sin la necesidad de reprogramar toda la red. CAN tienen aplicaciones en multiples campos, entre los que destacan:

- **Automotriz:** Es el estándar de facto. Se utiliza para la comunicación entre el motor, los frenos (ABS), los airbags, el sistema de navegación y otros módulos electrónicos.
- **Automatización Industrial:** Control de robots, máquinas CNC y líneas de producción.
- **Aeroespacial:** Sistemas de control de vuelo, trenes de aterrizaje y sensores.
- **Médica:** Equipos de diagnóstico por imágenes, bombas de infusión y monitores de pacientes.
- **CRC:** Suma de verificación para la detección de errores.
- **Transporte público:** Se utiliza en trenes, autobuses y tranvías para controlar sistemas como puertas, iluminación y aire acondicionado.
- **Robótica:** Control de múltiples articulaciones y sensores en robots industriales.

5.3 Introducción

En esta práctica, el estudiante aplicará y reforzará los conocimientos adquiridos en el Taller 4, en el cual se implementó una red CAN utilizando dispositivos Arduino y el módulo `MCP2515`, que permite dotar a un Arduino de conectividad CAN mediante una interfaz SPI.

El módulo MCP2515 es una solución CAN completa que incluye el controlador CAN MCP2515 de Microchip y un transceptor CAN de alta velocidad TJA1050 de Philips. A través de las borneras o pines, se accede a las líneas CAN.H y CAN-L, que permiten comunicarse con otros nodos de la red CAN.

En la segunda parte del Taller 4, se emplearon dos Arduinos y dos módulos `MCP2515`, los cuales posibilitaron el envío de una temperatura simulada generada a través de un potenciómetro desde un Arduino hacia otro mediante un bus CAN, lo que permitió que el segundo Arduino interpretara dicha información y la mostrara en el monitor serial del IDE de Arduino.

Puede verificar un ejemplo del nodo transmisor y receptor en `https://github.com/Pbarbecho/CAN_Arduino.git`. La librería mcp_can.h permite la comunicación entre un microcontrolador Arduino y el controlador MCP2515, encargado de gestionar el protocolo CAN Bus mediante la interfaz SPI. Esta librería, desarrollada por Seeed Studio, facilita tanto el envío como la recepción de tramas CAN, simplificando la implementación de redes de comunicación entre dispositivos embebidos.

En este Taller 5, agregamos la comunicación del `USB-CAN converter` y la pc con Linux Ubuntu (Virtual Machine - VM) mediante el bus CAN, todo esto utilizando nodos de ROS para cada función.

5.4 Implementación de la Comunicación CAN mediante ROS

El Taller 5 consiste en realizar un proyecto pequeño que abarque ROS, CAN y utilice un Arduino como sensor. Usando el BUS-CAN, enviaremos información de temperatura a la PC-LINUX con Arduino.

Vamos a tener tres nodos en ROS (ver Fig. 5.1):

- **Nodo - Sensor (Arduino)**: El primer nodo es el sensor de temperatura, que utiliza el Arduino para leer el valor del potenciómetro y enviarlo al bus CAN; aquí no actúa

ROS.

- **Nodo - Transmisor (ROS)**: El segundo nodo (can-tx-node) recibe la trama CAN a través de la PC (Linux VM) utilizando el BUS CAN y el conversor USB-CAN, los convierte a grados centígrados y los envía al tópico `temperature`.
- **Nodo - Monitor (ROS)**: El tercer nodo `monitor-node` se suscribe a **la** temperatura e imprime en consola el valor adquirido del sensor conectado al Arduino.

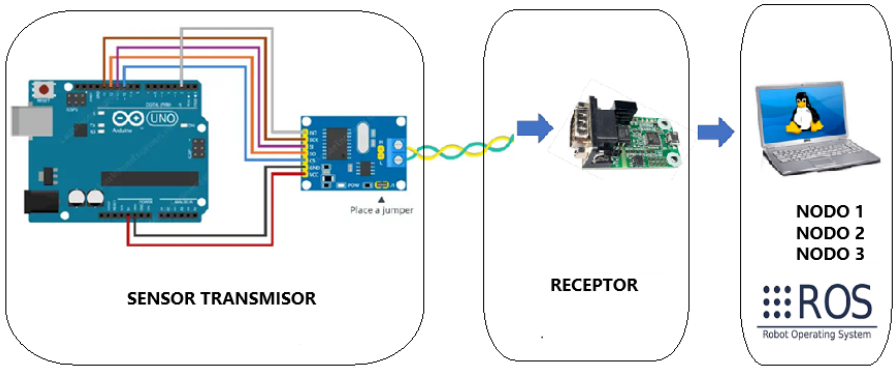

Figure 5.1: Elementos de la Red CAN y ROS

5.4.1 Desarrollo

1. **Habilitar interfaz `can` en la máquina `host`:** Primero, conectaremos el adaptador USB-CAN a un puerto usb de nuestra maquina Linux. Verificar que el adaptador fue detectado por nuestra maquina host. Utilice el comando `lsusb`, para verificar que se ha detectado el conversor, el `lsusb` debe mostrar algo como `OpenMoko...Schneider CAN adapter`. Verificamos que el adaptador USB-CAN se ha detectado como una interfaz de la máquina física host con el comando `ip link show`, el cual mostrara una interfaz que diga `can0:` (interfaz física).

   Note que el estado de la interfaz está en `DOWN`, por lo que debe cambiarse al estado `UP` y configurar la tasa de transmisión (bitrate) de la misma. El bitrate define la velocidad de transmisión de los mensajes CAN. Debe coincidir en todos los nodos de la red para que la comunicación funcione correctamente. Los valores comunes pueden ser 125 kbps, 250 kbps, 500 kbps y 1 Mbps.

   ```
   sudo ip link set can0 up type can bitrate 500000
   ip link show can0
   ```

   En caso de que no se detecte el adaptador USB-CAN, realice una `actualización` del sistema operativo (host) o, si utiliza una máquina virtual Linux, verifique que el USB esté conectado a la VM.
2. **Habilitar la interfaz can en el contenedor:** Una vez se ha detectado y se a activado la interfaz can en el host. Ahora se debe ejecutar el contenedor con soporte a la interfaz can del host. Tomar en cuenta que deben cargar las interfaces `ttyACM0` para el Arduino y la interfaz `net=host` para las interfaces del host, entre ellas la interfaz

can (convertidor USB-CAN). Además, otorgar permisos de administración sobre las interfaces de red del contenedor. Por ejemplo, puede ejecutar:

```
docker run -it --name=sensor_can --net=host --device=/dev/ttyACM0 --
    cap-add=NET_ADMIN ros:jazzy
```

3. **Habilite el Nodo-sensor (Arduino):** Puede usar el código del nodo-transmisor del Taller 4. Por ejemplo, vea el código del nodo TX en `https://github.com/Pbarbecho/CAN_Arduino.git`.
4. **Nodo - Transmisor (ROS):** Este nodo se encarga de leer la información del BUS CAN (e.j., interfaz can0) y le asigna un tópico `temperature` para que esté disponible para cualquier otro nodo de la red.

   Para interactuar con la interfaz `can`, puede usar librerías CAN, para lo cuál ha de instalar:

```
apt update && apt install -y python3-can
```

   Creamos el nodo transmisor en ROS. Note que debe configurar el ID correspondiente al ID configurado en el nodo - sensor del Arduino. Utilice la teoría de la trama de CAN para explicar que son los msg arbitration ID y por qué se toman 2 bytes como datos en el código:

```
import rclpy
from rclpy.node import Node
import can
from std_msgs.msg import Float32

class CANListener(Node):
    def __init__(self):
        super().__init__('can_listener')
        self.publisher_ = self.create_publisher(Float32, 'temperature
            ', 10)
        self.bus = can.interface.Bus(channel='can0', bustype='
            socketcan')
        self.get_logger().info('Listening on CAN interface can0...')
        self.timer = self.create_timer(0.1, self.read_can)

    def read_can(self):
        msg = self.bus.recv(timeout=0.01)
        if msg and msg.arbitration_id == 0x100 and len(msg.data) >= 2:
            t10 = (msg.data[0] << 8) | msg.data[1]
            if t10 & 0x8000:  # signed
                t10 -= 65536
            tempC = t10 / 10.0
            self.get_logger().info(f'Received Temp: {tempC:.1f}  C ')
            self.publisher_.publish(Float32(data=tempC))

def main():
    rclpy.init()
    node = CANListener()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

```
if __name__ == '__main__':
    main()
```

5. **Nodo_monitor:** Desarrolle el nodo que muestre la información en grados centígrados leída del bus CAN. Para esto, este nodo - monitor debe suscribirse al tópico `'temperature'` del nodo - transmisor.

Con este proyecto, nos hemos familiarizado con ROS y CAN. Hemos utilizado Arduino para simular sensores y también hemos configurado el conversor USB-CAN de modo que podamos utilizar nuestra PC con Linux como un nodo de la red CAN.

5.5 Actividad Reto

Reporte en el informe

Implemente una red CAN que tenga 2 nodos sensores de temperatura, usando dos Arduinos y una PC (Linux con contenedor ROS) que lea la información de la red CAN y la publique en un tópico de ROS para que cualquier otro nodo muestre la información de las dos temperaturas de manera gráfica.