Conversión del proyecto de ROS 1 a ROS 2

Análisis inicial del proyecto rbcar_steering_controller (enfoque hardware)

A alto nivel, este paquete ROS controla el giro (steering) de un vehículo (Rbcar) usando:

- Un variador/drive de motor (AMC DZCANTE-020L080) controlado por CANopen para mover el actuador de dirección en modo posición.
- Un encoder absoluto EPC (familia MA) conectado también por CANopen para medir el ángulo real de la dirección.
- Una tarjeta CAN PEAK (PCAN) como interfaz física con el bus CAN.

A continuación desgloso los bloques de hardware y cómo se integran:

1) Bus CAN y hardware de comunicaciones

- Interfaz: PEAK PCAN (usuarios típicamente PCAN-PCI, PCAN-USB, etc.). En el launch se usa el dispositivo “/dev/pcanpci1”; por defecto en código “/dev/pcan32”.
- Velocidad: 1 Mbit/s (CAN_BAUD_1M).
- Librerías: libpcan (PCan.h/PCan.cc) para abrir y configurar el canal, y enviar/recibir tramas CAN.
- Protocolo: CANopen “ligero” implementado manualmente, sin stack externo. Se envían NMT (Start, Stop, Reset Comm/Node), SDO (objetos de configuración/lectura) y PDOs (tramas de proceso).
- Mecanismo de supervisión: Node Guard/Heartbeat (cíclicamente se lanza un RTR a 0x700+NodeID y se comprueba el estado; si no hay respuesta dentro de NMT_TIMEOUT=2s se considera fallo de comunicación).

2) Drive de motor: AMC DZCANTE-020L080

- Función: accionar el actuador de dirección en modo posición (Position Mode).
- Nodo CAN: ID configurable (por defecto can_motor_id=1).

- Clase base **Dzcante020l080**.* encapsula objetos CAN (índices SDO/PDO) para este drive:
  - ControlWord (0x6040) con comandos: ENABLE_OP, SHUTDOWN, SWITCH_ON, QUICK_STOP, DISABLE_VOLT, RESET.
  - Modos de operación (0x6060) = Position/Velocity/Home.
  - RPDO y TPDO: configuración de COB-IDs y modos de transmisión (objetos 0x140x/0x180x y similares).
  - Lecturas: StatusWord (0x6041), Drive Status extendido (0x2002 subíndices 1..6), posición (0x6064), velocidad (0x60FF), corriente (0x6077), tensión bus DC (0x200F.01).
- Clase **MotorDrive**.* añade:
  - State machine interna (INIT/READY/EMERGENCY...) y comunicación (OPERATIONAL/FAULT...).
  - Hilos auxiliares:
    - ControlCommunicationThread: envía Node Guard y vigila tiempos para marcar FAULT si no hay respuesta.
    - ReadSDOMessagesThread: lectura cíclica de Drive Status, tensión del bus, entradas digitales, posición actual, etc., y cálculo de medias.
  - Configuración CAN:
    - Reset/Preoperational, setting de Node Guard (Guard Time 0x100C, Life Factor 0x100D), Event Action (0x2065.21), Recovery Time, TPDOs/RPDOs para posición/velocidad.
  - Gestión de estados de variador: dsDesiredStatus vs dsStatus. Cambia de estados enviando ControlWord con tiempos de guarda.
  - Señales de diagnóstico: descompone StatusWord y 0x2002.* en flags con máscaras y temporizadores para logging-control.
- Clase **MotorPosition**.* especializa a Position Mode:
  - Inicialización: secuencia ControlWord (Disable Voltage, Shutdown, Switch On, Enable Operation) y lectura de modo operación.
  - Homing: soportado (START_HOME / STOP_HOME), flags de homing completo y cambio a modo posición.
  - SetMotorPosition: envía objetivo vía RPDO21 (POSITION_PDO_MSG) en cuentas (counts).

3) Encoder absoluto EPC (familia MA) por CANopen

- Función: medir posición de la dirección en cuentas absolutas.
- Nodo CAN: por defecto can_encoder_id=0x7F.
- Clase **EpcMaCanOpen**.* maneja NMT/SDO/PDO básicos para estos encoders (IDs TPDO1 0x180+ID, TPDO2 0x280+ID).
- Clase **EpcEncoder**.*:
  - State machine similar (OPERATIONAL/FAULT...).
  - Hilo de comunicación para Node Guard.
  - Procesa TPDO1/TPDO2 con cuentas del encoder (encoder_counts).
- El nodo de alto nivel convierte estas cuentas a ángulo mediante una calibración lineal.

4) Nodo ROS de control de dirección

- Binario: rbcar_steering_controller.cpp.
- Flujos:
  - Parametrización vía ROS params y YAML (config/config.yaml): rangos de encoder, zero, resolución, calibración lineal (y=Ax+B), límites angulares y puntos extremos.
  - Inicializa PCan (a 1M), crea objetos MotorPosition (drive) y EpcEncoder (encoder) con sus IDs y desired_freq (50 Hz).
  - start(): Setup y Start de ambos (abre CAN, setea, pone OPERATIONAL, lanza hilos).
  - Loop spin(): a 50 Hz:
    - Envío SYNC (0x80).
    - Lectura de hasta 100 mensajes CAN y dispatch a Motor/Encoder para proceso (PDO/SDO/NMT/EMCY).
    - Publicación del estado: SteeringControllerStatus (estado motor+encoder, cuentas abs y motor, posición angular).
  - Suscripciones:
    - command_position_counts (std_msgs/Int32): manda counts al motor (posición absoluta en cuentas del motor).
    - command_angle (std_msgs/Float64): convierte ángulo deseado a cuentas absolutas y luego a cuentas de motor según relación abs_to_motor, y envía referencia de posición al drive.
  - Servicio:
    - set_motor_status (SetMotorStatus.srv): permite fijar estado del drive: OPERATION_ENABLED, QUICK_STOP, READY_TO_SWITCH_ON.
  - Diagnósticos:
    - diagnostic_updater reporta estado de comunicación y estado del drive, y supervisa frecuencia de comandos.

Qué tener en cuenta para replicar:

- **Calibración mecánica**: el YAML aporta relación entre encoder absoluto y cuentas del motor, y los extremos físicos. Esto es clave para que el comando en rad -> counts funcione correctamente.
- La variable abs_to_motor se calcula como motor_counts_range / abs_counts_range. Debes medir ambos rangos en tu mecánica y actualizar config.yaml.
- value_calibrate_a y value_calibrate_b definen la recta ángulo = -a*counts + b. Ajustarlos a tu encoder y cinemática.
- Límites de ángulo: extrem_angle_left/right y sus points asociados (extrem_points_left/right) fijan saturaciones de seguridad.

5) Resumen de la secuencia de arranque/operación

1. PCan abre el dispositivo y configura el bitrate.
2. MotorDrive:
   - Reset CAN Comm -> Preoperational
   - Configura PDOs/Node Guard/Eventos
   - Init drive (ControlWord sequence), lee modo operación
   - Start Communication (NMT Start), pasa a OPERATIONAL

- Lanza hilos de Node Guard y lecturas SDO periódicas

3. EpcEncoder:
   - Reset/Preoperational
   - (En este código no reconfigura PDOs del encoder)
   - Start Communication a OPERATIONAL
   - Lanza hilo de Node Guard
4. Bucle 50 Hz:
   - Envía SYNC
   - Lee paquetes CAN y los distribuye
   - Publica estado (counts + angle)
5. Comandos:
   - command_angle -> se traduce a counts del encoder y luego a counts del motor y se envía la posición al drive por RPDO21.

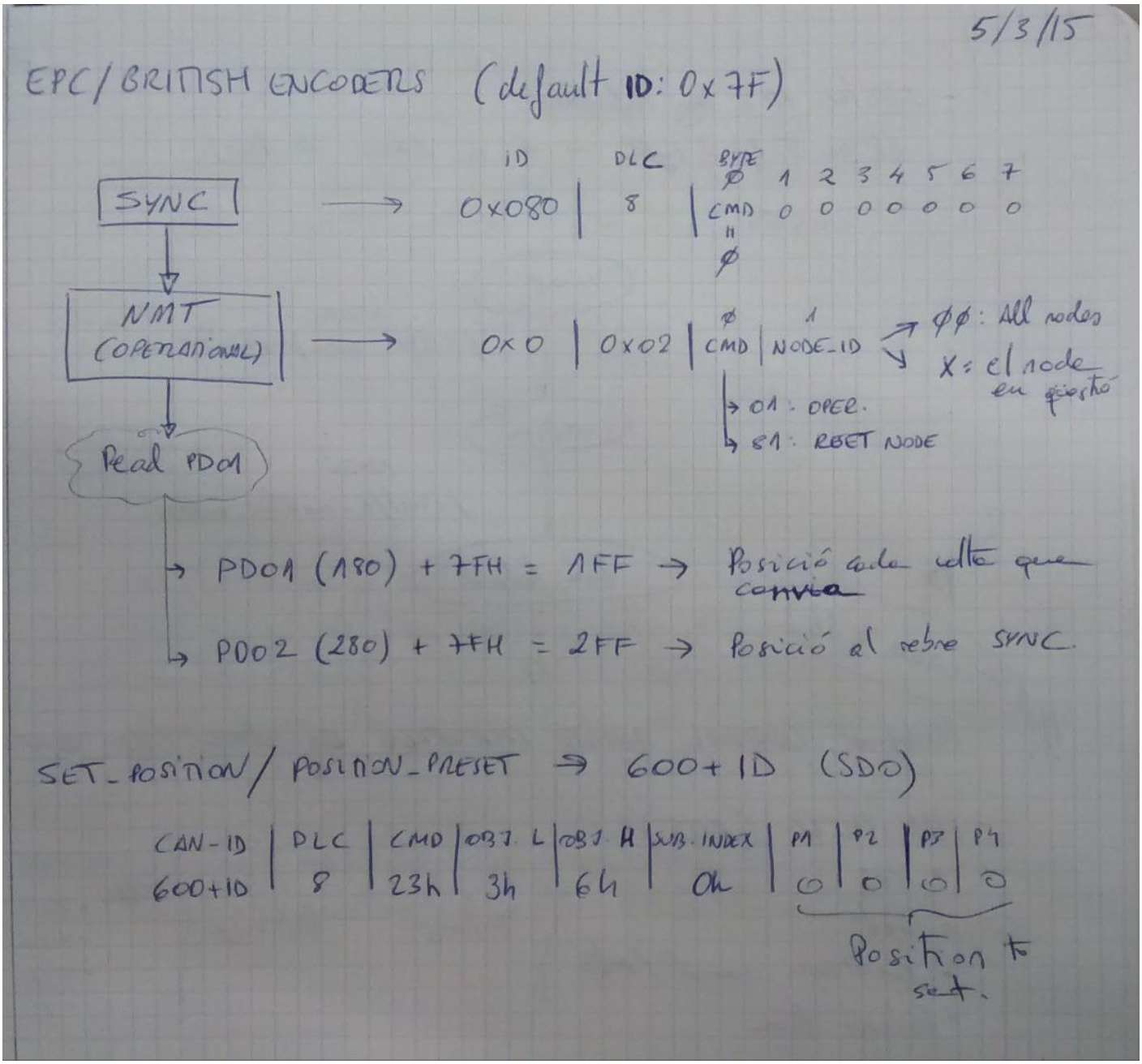
6) Señales CAN-clave y mapeo rápido

- NMT (ID 0x000): Start 0x01, Stop 0x02, Reset Comm 0x82, Reset Node 0x81, Preop 0x80 (DATA[0]=cmd, DATA[1]=node).
- Node Guard RTR a 0x700+NodeID, respuesta en 0x700+NodeID con estado (OPERATIONAL=0x05/0x85, PREOP=0x7F/0xFF, STOPPED=0x04/0x84).
- SYNC: 0x80 sin datos.
- Drive:
  - SDO server 0x600+ID (Write/Read a objetos 0x6040 ControlWord, 0x6060 Mode, 0x607A Position, 0x60FF Velocity, 0x2002 Drive Status, 0x200F.01 DC Bus, 0x6064 Position actual).
  - RPDO21 (0x280+ID) para consigna de posición (4 bytes little-endian).
  - TPDO1/21/22 para StatusWord/Position/Velocity según configuración.
- Encoder EPC:
  - TPDO1 (0x180+ID) / TPDO2 (0x280+ID): se usan los 2 bytes bajos como cuentas (el código así lo interpreta).
  - Node Guard/Heartbeat igual que el drive.

7) Qué adaptar en tu nuevo proyecto (hardware)

- Si cambias:
  - Interfaz CAN: sustituye PCan.* por tu driver y conserva la API Send/Read/Setup/Configure.
  - Drive: si el drive no es AMC DZCANTE-020L080, necesitarás:
    - Revisar/actualizar índices SDO, COB-IDs, TPDO/RPDO, escalados, y status words.
    - Ajustar secuencias de ControlWord y modos de operación.
  - Encoder absoluto: si otro fabricante/protocolo, cambiar mapeo de TPDOs y cómo se extraen cuentas.
  - Mecánica: recalibrar config.yaml para rangos, cero y linealización ángulo–cuentas.
- Mantener:

- Supervisión de comunicación (Node Guard/Heartbeat) o migrar a Heartbeat puro si tus nodos no usan Node Guard clásico.
- Seguridad: QUICK_STOP y manejo de fallos del drive.



Revision #1
Created 13 October 2025 06:23:21 by Rafael Carbonell Lázaro
Updated 13 October 2025 08:02:03 by Rafael Carbonell Lázaro