1 Introdução

O barramento CAN permite a comunicação entre outros dispositivos importantes, como o Raspberry Pi (usado para traqueamento) e o Windows (usado para simulação do robô, desenvolvida em Unreal Engine). Porém, alguns dispositivos usados no robô não suportam CAN, como alguns controladores (Spark e VictorSP) e a maior parte dos sensores (NavX ou limit switches, por exemplo). Por isso, desenvolvemos o nosso próprio protocolo de mensagens CAN por meio da programação em Java da RoboRIO.

2 Objetivo

O objetivo deste documento é apresentar um embasamento teórico por trás do desenvolvimento do protocolo CAN da equipe e explicá-lo para uso em dispositivos externos e manutenção preventiva.

3 Estrutura dos dados

Cada dado recebe as informações de ID, posição do byte (0 a 7) e bit. Os bits podem informar o valor absoluto (para sensores com valores de 0 a 255), estado ternário (em caso de controladores que entram em movimento retardado ou acelerado) ou estado booleano (em caso de sensores booleanos, como as limit switches).

Os dados são separados em tipos. Os principais tipos de dados são dados provindos de sensores e controladores. Eles possuem uma base de ID compartilhada: por exemplo, todo sensor começa com a ID “1F 64” e todo controlador começa com a ID “2F 64”. A estrutura de IDs compartilhadas pode ser encontrada no início de cada seção.

Alguns sensores serão virtualizados para simulação e para isso, precisam receber inputs. Há uma programação separada para uso nos simuladores do time e ela pode ser encontrada no GitHub. De acordo com as normas de controle e configuração, a programação para simulação estará dentro do repositório do robô simulado (BS-03 ou BS-04), sob a branch “simulation”.

4 Uso dos dados CAN

4.1 Raspberry Pi

O Raspberry Pi (em abreviação, RPi) é usado para enviar informações de traqueamento (ângulo e distância do robô em relação ao alvo de visão). Em 2019, essas informações eram enviadas por meio da Shuffleboard (conexão Ethernet com o rádio). No entanto, o FMS (Field Management System, ou Sistema de Gestão de Campo) usado pela FIRST durante as partidas recusava a conexão do RPi com a RoboRIO.

Figura 1 – Raspberry Pi 3 Modelo B, usado para traqueamento

O FMS não possui o direito de recusar uma conexão por meio da CAN, pois a CAN funciona por rede local. Por isso, o uso do barramento CAN é essencial para garantir a confiabilidade dos dados trocados entre a RoboRIO e o RPi.

O RPi não possui funcionalidade CAN por padrão. Por isso, é usada a placa MCP2515, que implementa a funcionalidade de interface CAN para qualquer dispositivo que tenha comunicação SPI.

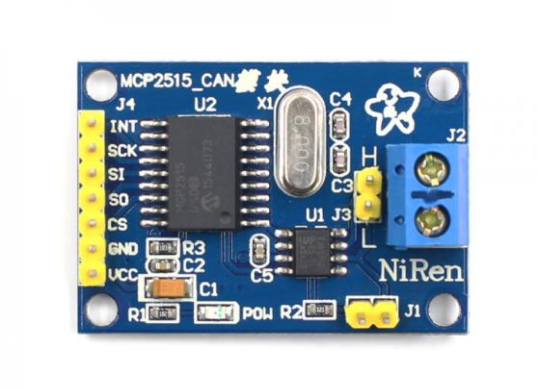

Figura 2 – Placa MCP2515

Ligações: não entendi até agora como funciona a ligação do MCP2515 com o RPi, portanto gostaria de pedir ajuda pro Danko pra escrever isso aqui

4.2 Arduino

O Arduino é usado para armazenar as informações enviadas pelo robô durante a partida. Com isso, é possível usar as informações enviadas para criar gráficos de aceleração, velocidade média, movimentação do robô na partida, dentre diversos outros dados que podem ser modelados a partir dos dados disponibilizados: voltagem da bateria, corrente puxada por cada motor, força aplicada em cada motor, rotações do giroscópio, pulsos do encoder, dentre vários outra possibilidades.

O papel do Arduino é de recebimento: ele recebe todos os dados enviados pelo robô, em todas as IDs possíveis. Para isso, basta usar um controlador CAN MCP2515 e conectar ele com a RoboRIO. Após receber os dados, ele os armazena em um arquivo de formato “.txt” em um cartão de memória

avulso. Ele também é capaz de se comunicar com a RoboRIO por Ethernet por meio da shield W5100 e armazenar os dados em cartão SD externo.

Figura 3 – Shield Arduino W5100

Há um sistema separado que lê esse arquivo ".txt" e transforma isso em gráficos sobre o robô, como aceleração, velocidade média, posicionamento na quadra em função do tempo, quantidade de pontos marcados, dentre outros dados importantes. Ele roda em Windows, é programado em C# e é executado com base no arquivo gerado em cada partida.

4.2 Simulador

O simulador é um sistema desenvolvido na Unreal Engine 4 e que lê os dados enviados por protocolo CAN para registrar todos os dados importantes enviados pelo robô (tração, escalada, dentre outros subsistemas) e transformar isso em movimentos dentro da quadra de jogo. Ele é capaz de testar o autônomo e qualquer funcionalidade usada no robô pode ser desenvolvida dentro dele.

Para comunicar a RoboRIO com o simulador, é necessária a comunicação da RoboRIO com o computador. Essa comunicação é feita por meio de um dispositivo que transforme a comunicação CAN em Serial, como o PCAN-USB. Para ler os IDs e enviar os dados para a Unreal Engine, é necessário desenvolver uma forma de receber os dados seriais pela UE4. A UE4 não recebe Serial por padrão. Se outra engine for usada para desenvolver o simulador, não é necessário desenvolver o sistema Serial.

5 Sensores

Device ID: 30

ID no código em Java: 1F 64

5.1 Sensores Digitais

ID no código em Java: 1F 64 04

5.1.2 Limit Switches de Garra e Torre

ID no código em Java: 1F 64 04 FF

5.1.2.3 Limit Torre - Superior

ID no código em Java: 1F 64 04 FF

Byte: 0

Bit: (byte) 1

Tipo: Output

5.1.2.4 Limit Torre - Inferior

ID no código em Java: 1F 64 04 FF

Byte: 0

Bit: (byte) 2

Tipo: Output

5.1.2.4 Limit Torre - Ambos estão ativados

ID no código em Java: 1F 64 04 FF

Byte: 0

Bit: (byte) 3

Tipo: Output

5.1.2.5 Limit Garra - Ambas

ID no código em Java: 1F 64 04 FF

Byte: 1

Bit: (byte) 1

Tipo: Output

5.1.2.6 Limit Garra - Superior

ID no código em Java: 1F 64 04 FF

Byte: 1

Bit: (byte) 2

Tipo: Output

5.1.2.7 Limit Garra - Inferior

ID no código em Java: 1F 64 04 FF

Byte: 1

Bit: (byte) 3

Tipo: Output

5.2 Sensores Analógicos

ID no código em Java: 2F 64 04 DD

5.2.1 Encoders de Tração

ID no código em Java: 2F 64 04 DD

5.2.1.1 Valor absoluto do encoder

ID no código em Java: 2F 64 04 DD

Byte: 0

Bit: Valor absoluto do encoder, de 0 a 255, em centímetros

Tipo: variável (input para a programação dos simuladores, output para a programação do robô)

5.2.1.2 Direção do encoder

ID no código em Java: 2F 64 04 DD

Byte: 1

Bit: 1 para positivo e qualquer outro valor para negativo

Tipo: variável (input para a programação dos simuladores, output para a programação do robô)

5.2.2 NavX Virtual

ID no código em Java: 2F 64 04 DD

5.2.2.1 Valor absoluto da NavX

ID no código em Java: 2F 64 04 DD

Byte: 2

Bit: Valor absoluto do encoder, de 0 a 180

Tipo: variável (input para a programação dos simuladores, output para a programação do robô)

5.2.2.2 Direção da NavX

ID no código em Java: 2F 64 04 DD

Byte: 3

Bit: 1 para positivo e qualquer outro valor para negativo

Tipo: variável (input para a programação dos simuladores, output para a programação do robô)

6 Controladores

Device ID: 31

ID no código em Java: 2F 64

6.1 Controladores de Subsistemas

ID no código em Java: 2F 64 04

6.1.1 Controladores da Torre

ID no código em Java: 2F 64 04 FF

6.1.1.1 Velocidade do Elevador

ID no código em Java: 2F 64 04 FF

Byte: 0

Bit: Velocidade, de 0 a 100% (decimal)

Tipo: Output

6.1.2 Controladores da Garra

ID no código em Java: 2F 64 04 AA

6.1.2.1 Controlador da Cargo - Ativado

ID no código em Java: 2F 64 04 AA

Byte: 0

Bit: (decimal) 1

Tipo: Output

6.1.2.2 Controlador da Cargo - Desativado

ID no código em Java: 2F 64 04 AA

Byte: 0

Bit: (decimal) 2

Tipo: Output

6.1.2.1 Controlador do Modo da Garra - Subindo

ID no código em Java: 2F 64 04 AA

Byte: 1

Bit: (decimal) 1

Tipo: Output

6.1.2.1 Controlador do Modo da Garra - Parado

ID no código em Java: 2F 64 04 AA

Byte: 1

Bit: (decimal) 2

Tipo: Output

6.1.2.1 Controlador do Modo da Garra - Descendo

ID no código em Java: 2F 64 04 AA

Byte: 1

Bit: (decimal) 3

Tipo: Output

6.1.3 Controladores da Tração

ID no código em Java: 2F 64 04 BB

6.1.3.1 Potência da Speed Controller Group - Esquerda

ID no código em Java: 2F 64 04 BB

Byte: 0

Bit: Potência, de 0 a 100 (em decimal)

Tipo: Output

6.1.3.2 Potência da Speed Controller Group - Direita

ID no código em Java: 2F 64 04 BB

Byte: 1

Bit: Potência, de 0 a 100 (em decimal)

Tipo: Output

6.1.3.3 Speed Controller Group - Esquerda - Frente

ID no código em Java: 2F 64 04 BB

Byte: 2

Bit: (decimal) 1

Tipo: Output

6.1.3.4 Speed Controller Group - Esquerda - Parado

ID no código em Java: 2F 64 04 BB

Byte: 2

Bit: (decimal) 2

Tipo: Output

6.1.3.5 Speed Controller Group - Esquerda - Trás

ID no código em Java: 2F 64 04 BB

Byte: 2

Bit: (decimal) 3

Tipo: Output

6.1.3.3 Speed Controller Group - Direita - Frente

ID no código em Java: 2F 64 04 BB

Byte: 3

Bit: (decimal) 1

Tipo: Output

6.1.3.4 Speed Controller Group - Direita - Parado

ID no código em Java: 2F 64 04 BB

Byte: 3

Bit: (decimal) 2

Tipo: Output

6.1.3.5 Speed Controller Group - Direita - Trás

ID no código em Java: 2F 64 04 BB

Byte: 3

Bit: (decimal) 3

Tipo: Output