Laboratory Project:
CAN in the Automotive Sector

Real-Time Systems

2020-21 Q2
May 6th, 2021

Victor Expósito Griñán
Group 10

1 - Introduction

Controller Area Network (CAN) is a communication protocol used in real-time systems, specially in the automotive world. Is the standard communication method that allows connexion and data exchange between the different devices of a vehicle (called Electronic Control Units (ECU)). Examples of ECUs are the engine control unit, the airbag system and the audio system of a car.

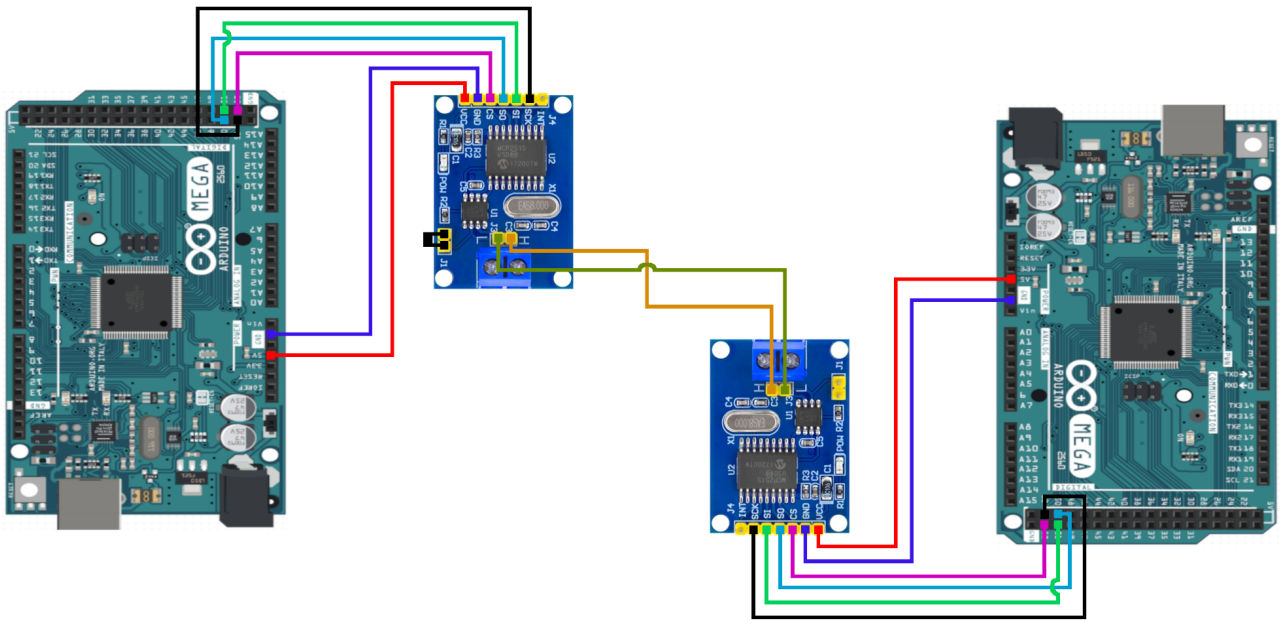
This project's objective is to introduce myself into the communication sector of automotive, learning more about how it works and how it is done, and also to try and communicate 2 ECUs using this protocol in order to implement specific functionalities.

2 - First Half - CAN Communication

The first part of this project consists in the simple communication of 2 Arduinos (which will represent 2 ECUs) using CAN. For this to be done, I need the following material:

- Arduino Mega 2560 x 2
- MCP2515 CAN Controller (Shield) x 2
- Jumper Resistor
- USB type B cable x 2
- Jumper cables

Many microcontrollers have specific peripherals for CAN, but unfortunately Arduino does not, so communication has to be made with controllers through SPI, like this:



Once the connections are made, I need to develop the codes, one for the transmitter ECU (which is going to be called *transmitterCode*) and another for the receiver ECU (which is going to be called *receiverCode*). In those scripts, a CAN library is needed, so I will use one provided by the Seeed Studio Hardware Platform's GitHub [1].

For the data sending, the function called *sendMsgBuf* will do the thing, needing an identifier, the length of the message and the data buffer. Identifiers are very important in CAN, as they serve a lot of purposes, like setting the message's destination or setting a priority between messages (the smaller the id, the greater the priority, which is useful when sending critical information).

For the data reception, the most important function is *readMsgBuf*, which reads the data buffer sent by another ECU and increments a counter that represents the message's length. Then, with a loop, it is very easy to go through the buffer and process the data.

After going through some problems, I finally accomplished my goal and sent data from one Arduino to the other successfully. (Codes can be found in the *tests* directory)

3 - Second Half - Parking Sensor System

The idea behind the second part of the project is to emulate a known parking system that plenty of cars have nowadays, an obstacle detection mechanism. When a car is set to parking mode, a set of sensors are activated to notify the driver of the position of objects using an acoustic signal and/or a graphic display with the purpose of avoiding potential collisions.

The recreation consists in the following: an Arduino with a proximity sensor will send information via CAN to another Arduino, which will set the brightness of an LED according to the value received, so the light is brighter if the distance is smaller.

Unfortunately for me, I did not dispose of the necessary time to implement this functionality. Nonetheless, I wanted to at least think about the project's structure, which is very important. In this kind of projects, the functionalities are divided by layers, and the major reason why this is done is to have a modular structure in which different people can work together and independently.

[1] *https://www.seeedstudio.com/*
https://github.com/Seeed-Studio/Seeed_Arduino_CAN

The layers are the following:

- APP (Application): contains the specific functionalities, like lighting or the wiper washer mechanism. In my project, this is the layer where the processing of the distance value should be made.
- BSW (Basic Software): contains the management code of what it is directly touching the hardware. In my project, this is the layer where the distance would be measured and where the LED would be set.
- COM (Communication): generally contains the CAN libraries (usually provided by Vector) and code of other communication protocols if needed.
- DGN (Diagnosis): contains CAN code but slightly different, as this layer is used to consult variables and states of the vehicle. This could be considered as a "car debug".
- OS (Operating System): usually contains an implementation of a cyclic scheduler, but it can be a little more complex.
- RTE (Run-Time Environment): this layer serves as a communication between layers, specifically variables. This method of communication is very useful, and it is known as "1 publisher - multiple subscribers", being this layer where things are published.

So, this is a brief explanation of how the project's system should work:

1. In BSW, the distance value is calculated via the proximity sensor and published in RTE.
2. APP picks the value from RTE, processes it, and then it is published in RTE again.
3. COM picks the value from RTE and sends it to the other ECU through CAN.
4. The receiver ECU picks the value from CAN and publishes it in RTE.
5. Finally, BSW picks the value and writes it in the LED.

All this process is managed by the OS.

4 - Final Thoughts and Conclusions

After this project, I can say that I have learnt about CAN and its applications and structure when working in automotive projects, which I appreciate because I have in mind to apply in this sector when graduating. Even though I have understood the structure, the fact that I could not implement the second part of the project leaves a bitter taste, but I might finish it in the future if I have the time. Despite everything, you never cease to learn, and I hope to discover even more things about the automotive sector.