Relatório de Projeto em Engenharia de Automação

Mestrado em Engenharia de Automação Industrial

Interface OBD para o AtlasCar2 e Monitorização do seu Estado

Autores:

Luís Cristóvão Nº: 80886

Professor:

Vitor Santos

Aveiro, Julho de 2018

Resumo

Este projecto consiste no desenvolvimento de uma interface OBD para monitorização do veículo **AtlasCar2**[1], que consiste no desenvolvimento e alteração de um carro, com o fim de lhe conferir características de condução autónoma.
Para o âmbito do presente projeto, é imprescindível a recolha de dados contidos na ECU (*Engine Control Unit*) do veículo, tais como:

- Posição do acelerador e do travão,
- Velocidade atual,
- Posição angular do volante,
- Autonomia,
- Entre outros.

Através da ficha OBD disponibilizada no veículo, é possível adicionar um dispositivo ao barramento **CAN** do mesmo, para obtenção de valores que auxiliem no processo de condução assistida por computador.
No presente documento será também descrito o desenvolvimento do dispositivo para descodificação dos valores anteriormente referidos.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

CAD *Computer Aided Design*

CAN *Controller Area Network*

CS *Chip Select*

Desc. Desconhecido

ECU *Engine Control Unit*

GPIO *General Purpose Input/Output*

I2C *Inter-Integrated Circuit*

kb/s Kilobit por segundo

km Kilometro

km/h Kilometro por hora

Mb/s Megabit por segundo

OBD *On-Board Diagnostic*

PCB *Printed Circuit Board*

ROS *Robot Operating System*

RPM Rotações por minuto

SPI *Serial Peripheral Interface*

Capítulo 1

Introdução

No âmbito do projecto AtlasCar2, surgiu a necessidade de obter valores registados na ECU (centralina) do veículo, sendo o mesmo um *Mitsubishi* modelo *i-Miev* produzido em 2015. Foi proposto o desenvolvimento de uma aplicação que, através de uma *interface* de baixo custo como, por exemplo, a exemplificada na Figura 1.1, que quando ligada à ficha OBD disponível no carro, possibilita a recolha de dados disponíveis na centralina do veículo.

Figura 1.1: Interface OBD comercial.

A aplicação deve apresentar e guardar valores lidos do veículo em tempo real. Os dados propostos a obter são:

- velocidade atual(m/s),
- autonomia (km),
- posição angular do volante (em graus),
- posições do acelerador e travão,
- ações do condutor (estado dos faróis, piscas, portas, entre outros),

- sensores integrados no carro (medições de temperatura do motor ou das baterias e monitorização de consumos).

O projeto foi dividido nos seguintes objetivos e tarefas:

- Instalação da ficha no veículo e teste de funcionalidade com uma aplicação *standard* (Android, IOS, etc.),
- Desenvolvimento de uma aplicação que leia os dados em bruto da ficha OBD,
- Implementação da descodificação do protocolo para o i-Miev,
- Desenvolvimento de uma aplicação para visualização dos parâmetros descodificados.

Capítulo 2

Apresentação de soluções para o problema

Neste capítulo é descrito o estudo feito para o desenvolvimento do projeto.
Inicialmente foi feito um estudo sobre as soluções de diagnóstico automóvel disponíveis no mercado. Foi verificado que a comunicação entre dispositivo e veículo é feita com protocolo *OBD*, através do barramento *CAN*.
Pode ser visto da seguinte forma, sendo o protocolo *OBD* a língua (Português) e o barramento *CAN* como o meio de comunicação (telefone)[7]. Com isto a centralina consegue saber o estado do veículo. Foi feita uma pesquisa sobre a informação disponível no barramento do veículo em questão, sendo esta apresentada no seguimento do documento.
Por fim será apresentada a solução implementada.

2.1 Soluções de mercado

As centralinas existentes no mercado são fabricadas por marcas que não se dedicam exclusivamente ao mercado automóvel. Assim, de modo a ser possível aplicar a mesma centralina em diferentes veículos, foram criadas normas que estabelecem uma solução generalizada para todos os fabricantes. Deste modo surgiu o protocolo *OBD*, para que a gestão de informação seja igual para todos os veículos, sendo possível também interligar dispositivos desenvolvidos unicamente pelo fabricante do carro.
Desde do ano 1996, todos os carros produzidos na Europa ou Estados Unidos são obrigados a ter na sua instalação elétrica uma ficha, como a representada na Figura 2.1.
Atualmente, para uma situação de diagnostico do veículo, são encontrados no mercado diferentes dispositivos de baixo custo. Os respectivos fabricantes disponibilizam aplicações,

Figura 2.1: Exemplo de porta *OBD* para um veículo.

para computador ou telemóvel(*iOS* e *Android*) para diagnóstico do veículo. A comunicação entre o dispositivo e a aplicação pode ser feita através de cabo ou sem fios(*Wifi* ou *Bluethooth*). Na Figura 2.2 estão representadas algumas das soluções disponíveis no mercado.

(a) (b)

Figura 2.2: Exemplos de soluções encontradas no mercado, com cabo (a) e *bluethooth* (b).

2.2 Comunicação CAN

O protocolo *CAN* foi desenvolvido pela *BOSCH*[8], sendo disponibilizado em meados dos anos 80. Atualmente é aplicado na indústria, em navios, veículos automóveis, entre outros. Na Figura 2.3 está representado um caso padrão de um barramento *CAN*.

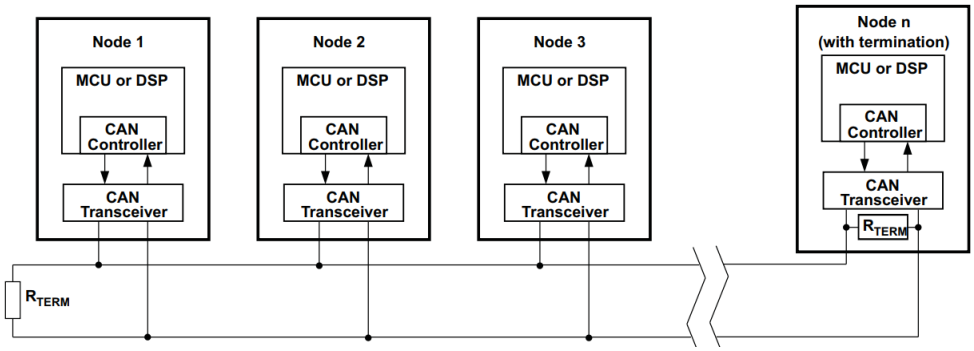


Figura 2.3: Barramento *CAN* padrão[2].

O *CAN* é um protocolo de comunicação série síncrono, de elevada fiabilidade, que permite

utilizar um controlo distribuído em tempo real. O barramento é constituído por nós, sendo que a comunicação entre eles é feita no conceito de multi-mestre. Isto é, todos podem ser mestres e escravos. A utilização deste sistema tem como principais vantagens [9]:

- Baixo custo de planeamento e instalação,
- Diagnóstico de erros,
- Redução da cablagem total da rede,
- Funcionamento em tempo real.

A norma ISO-11898:2003, estabelece o formato de mensagens *standard* para um barramento *CAN*, como representado na Tabela 2.1 . Este formato permite a utilização de 2048 identificadores. Posteriormente foi alterada de modo a ser possível utilizar um maior número de identificadores (aproximadamente 537 milhões), surgindo assim o formato alargado, apresentado na Tabela 2.2.

SOF	11-bit Identifier	RTR	IDE	r0	DLC	0...8 Bytes Data	CRC	ACK	EOF	IFS

Tabela 2.1: Constituição de uma mensagem *standard*.

SOF	11-bit Identifier	SRR	IDE	18-bit Identifier	RTR	r1	r0	DLC	0 ...8 Bytes Data	CRC	ACK	EOF	IFS

Tabela 2.2: Constituição de uma mensagem extensa.

Os parâmetros definidos nos dois tipos de mensagens possuem as seguintes funcionalidades:

- SOF - Um só bit que marca o inicio de mensagem, sendo usado para sincronismo dos nós.
- Identificador - Estabelece a prioridade da mensagem. O que tiver menor valor neste campo tem maior prioridade.
- RTR - Pedido de transmissão remota. Quando é feito um pedido informação por outro nó, este tem prioridade sobre outras que são transmitidas periodicamente. Os dados de resposta são recebidos por todos os nós e usados por qualquer um interessado.

- SRR - É um substituto do parâmetro (RTR), para o formato extenso.
- IDE - Bit de identificação do tipo de mensagem, ou seja, indica se vão ser recebidos mais bits de identificação.
- r0 e r1 - Bits de reserva.
- DLC - Composto por 4 bits, que indica o numero de *bytes* a serem transmitidos.
- Data - Dados a serem enviados.
- CRC - Verificação de redundância cíclica, que contem o numero de bits transmitidos para ajuda de detecção de erros na transmissão.
- ACK - Verificação feita por todos os nós, ou seja, se algum não receber correctamente a mensagem, é feito um novo envio da mesma. Este campo é composto por 2 bits, sendo um para reconhecimento e o segundo é usado como delimitador.
- EOF - Bit que indica o fim da mensagem.
- IFS - Contêm o tempo requerido pelo o controlador, ao mover correctamente a mensagem recebida para a área de *buffer*.

2.2.1 Descrisão do barramento *CAN* no veículo *I-MIEV*

De modo a ser possível desencriptar as mensagens disponíveis no carro, foi realizada uma pesquisa sobre quais os módulos existentes e a sua interligação, assim como quais são as mensagens disponíveis para leitura do barramento *CAN*. Através da Figura 2.4, é possível verificar quais os módulos existentes no barramento.

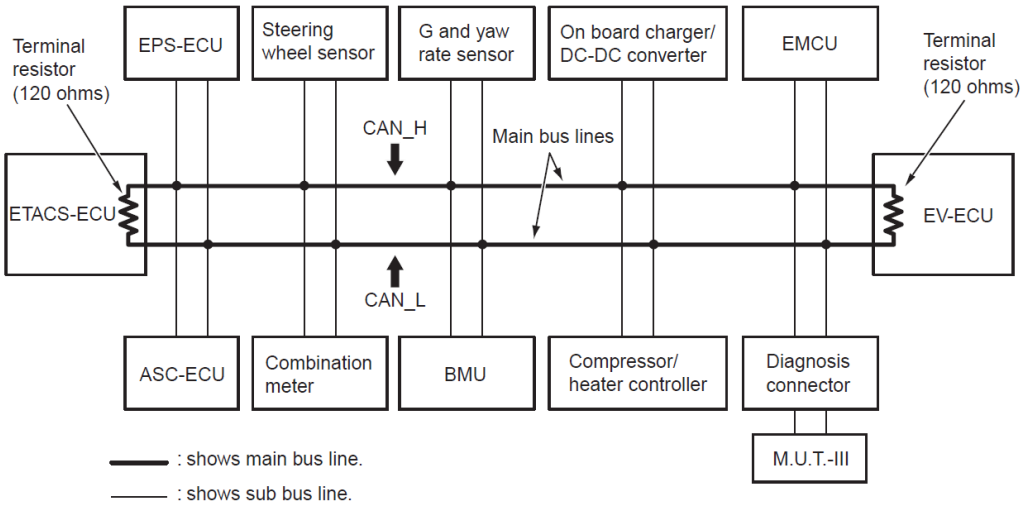


Figura 2.4: Estrutura do barramento *CAN* no *Mitsubishi iMiev*[3].

Através do diagrama apresentado anteriormente é possível concluir que no barramento estão inseridos os seguintes módulos:

- **ETACS-ECU** - Módulo para controlo de portas, espelhos, luzes, entre outros.
- **EPS-ECU** - Sistema de gestão da direção eletrica.
- **ASC-ECU** - Sistema ativo para controlo de estabilidade.
- ***Steering wheel sensor*** - Deteta o ângulo e torque do volante.
- ***Combination meter*** - Painel de instrumentos.
- ***G and yaw rate sensor*** - Sensor de aceleração lateral do veículo.
- **BMU** - Unidade de gestão de baterias.
- ***On board charger*** - Carregador de baterias incorporado para tomada AC 250 V 13 A.
- ***Compressor/heater controller*** - Controla o ar-condicionado e aquecimento do habitáculo.
- **EMCU** - Unidade de controlo do motor elétrico.
- **EV-ECU** - Centralina do veículo
- ***Diagnosis connector*** - Ficha OBD para efeito de diagnostico.

Como é de esperar nem todos os dispositivos existentes no veículo estão ligados directamente ao barramento *CAN*. Isto é devido ao limite máximo de nós num barramento, as leituras e os acionamentos são feitos pelo módulo correspondente. Cada módulo controla o fluxo de informação que é enviada e recebida pelo barramento, segundo a sua funcionalidade.

2.2.2 Mensagens do veículo *I-MIEV*

Nesta secção serão descritos os códigos descodificados. Foi feita uma pesquisa no manual de serviço do veículo, mas nele não são apresentadas nenhumas mensagens que circulam no barramento. Com isto a pesquisa foi efetuada em fóruns dedicados a carros elétricos. De seguida são apresentados os identificadores (valor hexadecimal) das mensagens e o período em que ocorrem no barramento *CAN* [4], sendo que estas cumprem o formato *standard* presente anteriormente na Tabela 2.1.

- 1000 *ms* - 01C.
- 200 *ms* - 568.
- 100 *ms* - 101, 286, 298, 29A, 2F2, 374, 375, 384, 385, 389, 38A, 3A4, 408, 412, 695, 696, 697, 6FA, 75A, 75B.
- 50 *ms* - 38D, 564, 565, 5A1, 6D0, 6D1, 6D2, 6D3, 6D4, 6D5, 6D6, 6DA
- 40 *ms* - 424, 6E1, 6E2, 6E3, 6E4
- 20 *ms* - 119, 149, 156, 200, 208, 210, 212, 215, 231, 300, 308, 325, 346, 418
- 10 *ms* - 236, 285, 288, 373

Na Tabela 2.3 estão apresentadas as informações contidas nas mensagens que foi possível descodificar bem como as unidades de medida associadas a cada uma. A forma como é descodificada a informação contida em cada mensagem será abordada no subcapítulo 3.2.

MsgID (Hex)	Descrição da mensagem recebida	Unidade de medida
0x101	Estado da chave na ignição	ON/OFF
0x208	Posição do pedal do travão	%
0x210	Posição do pedal do acelerador	%
0x231	Interruptor do pedal do travão	ON/OFF
0x236	Posição do volante	Graus
	Torque do volante	Desc.
0x286	*Temperatura do carregador on-board*	ºC
0x298	Temperatura do motor	ºC
	Rotações do motor	rpm
0x346	Autonomia restante	km
0x373	Valor de tensão aos terminais da bateria	Volt
	Valor da corrente da bateria	Amp.
0x374	Valor de carga restante	%
0x389	Valor de tensão AC	Volt
	Valor da corrente AC	Amp.
0x3A4	Estado dos botões do aquecimento e ar condicionado	Desc.
0x412	Velocidade atual do veículo	km/h
	Quilometragem total do carro	km
0x418	Posição da caixa de velocidades	Desc.
0x424	Estado dos faróis, piscas, portas, entre outros	ON/OFF
0x6E1 0x6E2	Valor da temperatura das 66 zonas da bateria	ºC
0x6E3 0x6E4	Valor de tensão das 88 células que constituem a bateria	Volt

Tabela 2.3: Mensagens do barramento, que é conhecida a informação contida[4][5][6].

2.3 Solução implementada

Inicialmente, foram feitas varias tentativas, para efetuar a comunicação com o carro. Começaram por ser abordadas soluções de mercado de modo a reduzir o recurso e desenvolvimento de *hardware* especifico para esta solução. Mas após várias tentativas com diferentes produtos existentes no mercado, foi verificado que não era possível comunicar com o carro devido ao facto de se tratar de um elétrico. Ou seja, as soluções de mercado de baixo custo, atualmente, não estão preparadas para identificar as diferentes mensagens recebidas por um barramento CAN de um carro elétrico, porque as mensagens de um carro de combustão interna ou híbridos são diferentes. Foi feita uma nova pesquisa sobre equipamentos que conseguissem ler este barramento, sendo que os produtos encontrados eram demasiado caros e tinham o risco de não funcionar. Com toda esta situação, optou-se por desenvolver um *sniffer* CAN, constituído por um *transceiver* CAN e uma unidade computacional de baixo custo.

Capítulo 3

Desenvolvimento

Neste capítulo é descrito o desenvolvimento do projeto, e que se divide em duas partes distintas, *hardware* e *software*.
É importante referir que foi necessário recorrer ao desenvolvimento de hardware porque as soluções de mercado disponibilizadas, não estão preparadas para carros eléctricos.
No decorrer da explicação do software desenvolvido, será abordado o funcionamento da *interface* bem como os métodos para descodificação das mensagens recebidas no barramento *CAN*.

3.1 Desenvolvimento de *hardware*

O hardware para este protejo foi desenvolvido tendo em conta que, no fim, este teria de integrar no projeto *ATLASCAR 2*. Assim, foi definido que a solução encontrada teria que ser modular, de baixo consumo e com dimensões reduzidas. O *hardware* consiste em três partes principais:

- *Raspberry Pi2 model B*,
- Módulo com comunicação *CAN* (MCP2515),
- PCB para integração dos diferentes módulos.

De início, foi escolhido um *Raspberry* para implementação do código desenvolvido. Esta plataforma foi escolhida tendo em conta a sua versatilidade dado que contêm as seguintes interfaces de elevada importância para este projeto:

- Porta Ethernet 10/100 Mbit/s,
- Porta HDMI,
- 4 Portas USB 2.0,
- Modulo SPI,
- Modulo I2C,

Para o projeto foi verificado que é importante o módulo incluir a porta *ethernet*, para futuramente este ser integrada numa rede ROS. Como atualmente é requerido que os valores obtidos sejam guardados num ficheiro de texto, a presença das portas USB também é importante para ser possível armazenar os dados obtidos através de uma *flash drive*. Também para a escolha desta plataforma foi tido em conta a possibilidade do uso de dispositivos com comunicação SPI/I2C. Na Figura 3.1 está representado um exemplar da plataforma escolhida.

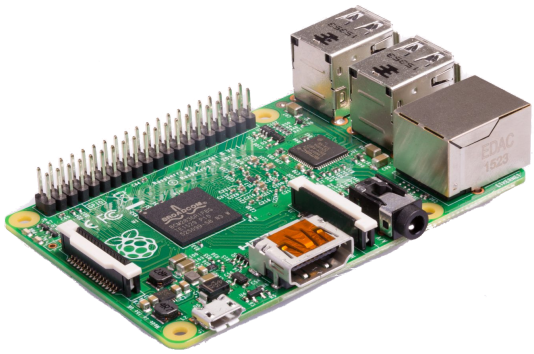


Figura 3.1: *Raspberry Pi2 model B.*

Para conseguir ler as mensagens do barramento *CAN* foi utilizado um módulo concebido para operar com *Arduino*[10] que é constituído por um controlador *CAN* autónomo com comunicação SPI, como o representado na Figura 3.2. Com o *datasheet*[11] do controlador, é possível observar que este tem as seguintes características:

- CAN com formato de mensagem V2.0B à velocidade de 1 Mb/s,
- Contêm *buffers* de receção, máscaras e filtros,
- Interface SPI de alta velocidade (10MHz),
- Opera no intervalo de tensão 2.7-5.5V
- É um dispositivo de baixo-consumo.

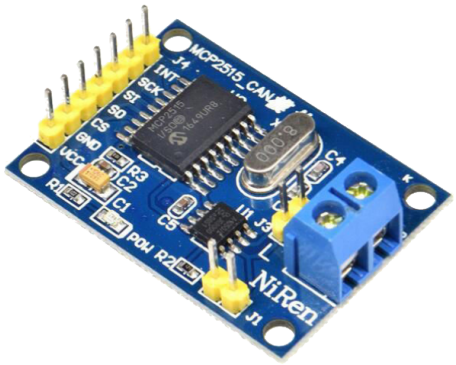


Figura 3.2: Módulo para barramento CAN MCP2515.

O módulo em questão tem um conector com dois terminais para ligar directamente no barramento CAN do carro. É importante referir que este pode ser alimentado com a tensão máxima 5.5V, e neste caso são lhe fornecidos 5V.

De modo a tornar o sistema desenvolvido mais compacto, foi desenvolvida um PCB dedicado. Este circuito foi feito de modo a interligar e fornecer uma alimentação fixa de 5V aos módulos referidos anteriormente. Foi feito um estudo sobre os consumos dos dispositivos utilizados, de modo a garantir a alimentação do sistema. É necessário garantir uma corrente, para o funcionamento do *Raspberry* (1.8A) e de 5mA para módulo CAN. Para isto foi escolhido um conversor DC-DC(AMSR-7805-NZ)[12] com as seguintes características:

- Tensão de entrada 7-18VDC,
- Tensão de saída fixa de 5VDC,
- Corrente de saída de 2A.

Tendo em conta que a ficha OBD disponibilizada no carro, tem o *pinout* presente na Figura 3.3, é verificado que através dela é possivel alimentar o sistema e ter acesso ao barramento *CAN*. Logo, foi integrado um conector no PCB desenvolvido de modo a garantir o funcionamento do sistema com um só cabo.

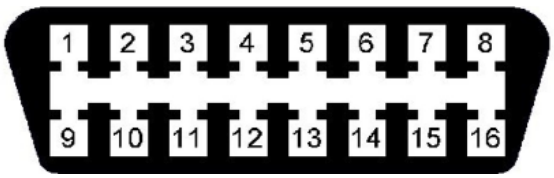


Figura 3.3: *Pinout* da porta OBD.

Com recurso ao *software EAGLE 7.5.0*, foi projetado o esquemático do PCB, bem como o seu desenho CAD. O esquemático final obtido está representado na Figura 3.4. Na Figura 3.5 está apresentado o aspecto final da PCB após o fabrico da mesma.

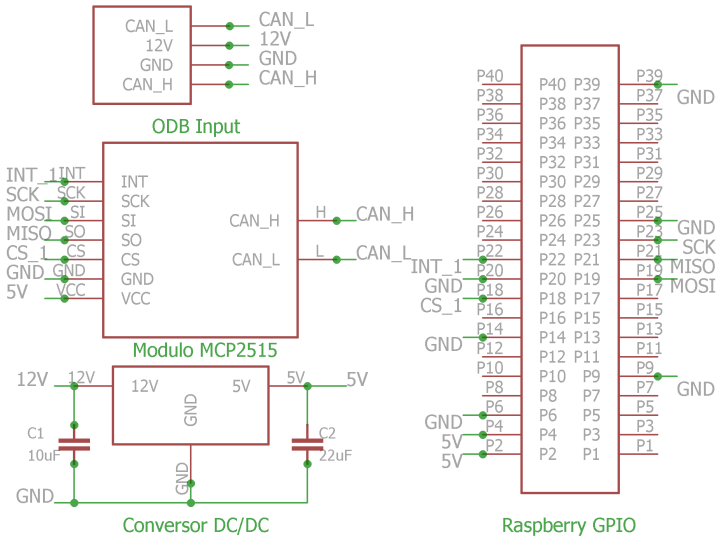
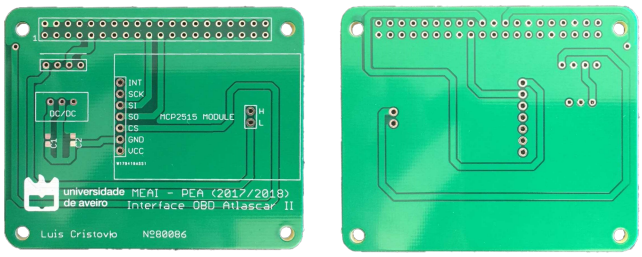


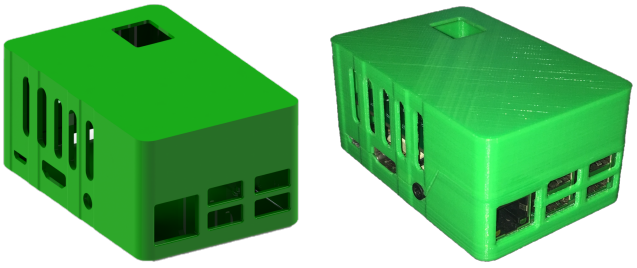
Figura 3.4: Esquemático da PCB desenvolvida.



(a) (b)

Figura 3.5: PCB produzida; (a) a camada superior ;(b) camada inferior.

Por fim, com recurso ao *software SolidWorks 2017* foi desenhada uma caixa para encapsular e proteger todo o sistema. A caixa final foi obtida através do processo de impressão 3D. Na Figura 3.6 está representado um modelo e o aspeto final da caixa.



(a) (b)

Figura 3.6: Caixa em desenho CAD(a) e obtida no final(b).

3.2 Desenvolvimento de *software*

Neste subcapítulo será descrito o *software* desenvolvido, de modo a explicar o funcionamento do programa implementado, bem como a descrição de como são desencriptadas as mensagens do barramento CAN.
O programa está desenvolvido em C++, a correr sobre uma plataforma baseada em *Linux*, sendo este *RASPBIAN STRETCH WITH DESKTOP* [13]. O programa desenvolvido é constituído por os seguintes ficheiros:

- **bcm2835.cpp** - Ficheiro com funções para funcionar com GPIO do *Raspberry*,
- **bcm2835.h** - Ficheiro de biblioteca com os registos do processador presente no *Raspberry*,
- **spi.cpp** - Ficheiro com as funções para o funcionamento com interface SPI,
- **my_spi.h** - Ficheiro com a declaração das funções do ficheiro "spi.cpp",
- **mcp_can.cpp** - Ficheiro com os métodos da classe MCP_CAN para funcionamento com módulo *CAN*,
- **mcp_can.h** - Ficheiro com a declaração da classe "MCP_CAN",
- **miev_obd.cpp** - Ficheiro com os métodos da classe "MIEV_CAN"para descodificação das mensagens recebidas do barramento *CAN*,
- **miev_obd.h** - Ficheiro com a declaração da classe MIEV_CAN,
- **main.cpp** - Contem a função *main* do sistema desenvolvido.

Através do diagrama de blocos geral apresentado na Figura 3.7 é possível observar de como são distribuídas as comunicações do sistema.

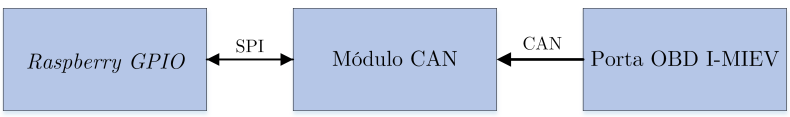


Figura 3.7: Diagrama de blocos das comunicações entre dispositivos.

Como demonstra o fluxograma da Figura 3.8, o programa, habilita os pinos GPIO do *Raspberry* para ser possível trabalhar com a interface SPI do mesmo, e de seguida é ativada a interface SPI com as seguintes características:

- Bit mais significativo primeiro (MSB),
- Configurado o modo de funcionamento SPI,
- Divisor de relógio, com o valor 32, para obter uma velocidade de 7.8125 MHz.

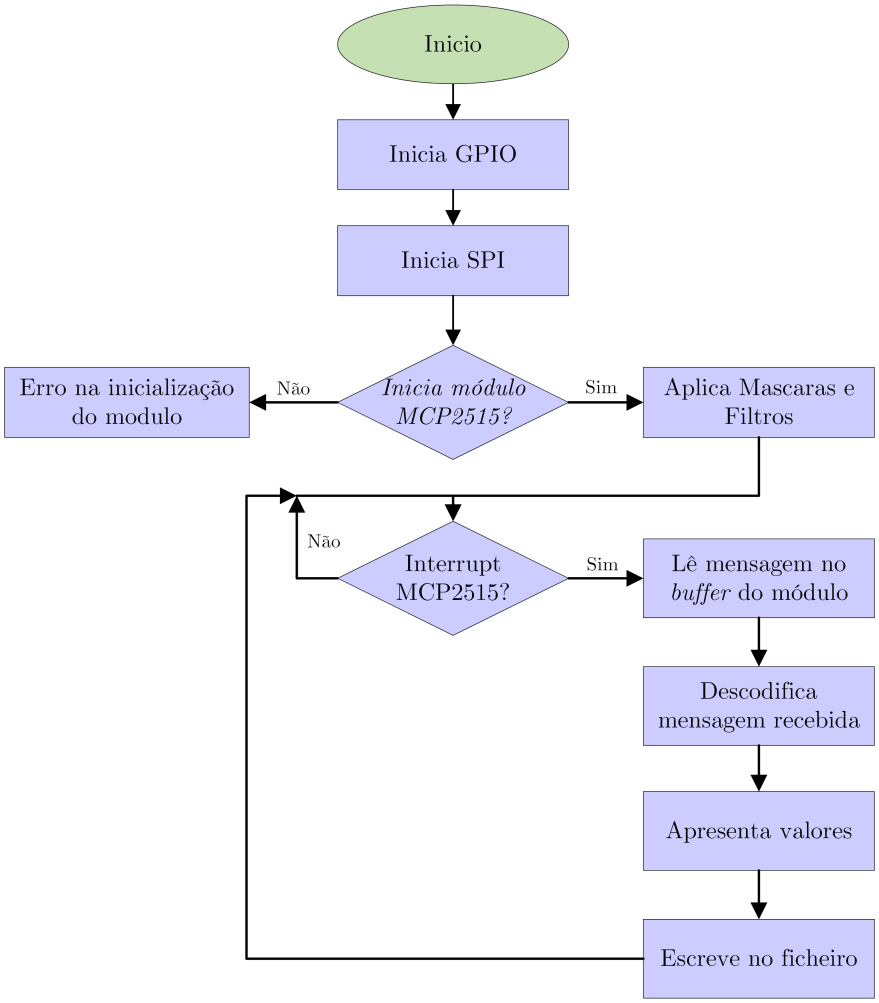


Figura 3.8: Fluxograma da função *main*.

É importante referir que a velocidade da comunicação SPI poderia ser maior, mas o módulo *CAN* trabalha com a velocidade máxima de 10 MHz de acordo com o fabricante. Para conseguir por fim comunicar com o barramento *CAN* do carro, é inicializado o módulo *CAN*. O módulo está configurado para o modo de leitura, para receber mensagens no formato *standard* a uma velocidade de 1000 kb/s. Para trabalhar com ele foram utilizadas bibliotecas desenvolvidas [14] para *Arduino*, sendo estas alteradas para o correto funcionamento na plataforma escolhida. As alterações efetuadas baseiam-se na implementação da função para alterar o estado do CS associado ao modulo *CAN* e da implementação das funções para transferência de dados através da interface SPI. Estas funções estão descritas no ficheiro **bcm2835.c**. Na Figura 3.9 é apresentado um exemplo de uma das alterações efetuadas ao ficheiro **mcp_can.cpp**.

```
47 /*********************************************************************************************************
48 ** Function name:           mcp2515_readRegister
49 ** Descriptions:            Read data register
50 *********************************************************************************************************/
51 uint8_t MCP_CAN::mcp2515_readRegister(const uint8_t address)
52 {
53     uint8_t ret;
54
55     bcm2835_gpio_write(RPI_GPIO_P1_18, LOW);
56     delay(10);
57     bcm2835_spi_transfer(MCP_READ);
58     bcm2835_spi_transfer(address);
59     ret = bcm2835_spi_transfer(0x00);
60     bcm2835_gpio_write(RPI_GPIO_P1_18, HIGH);
61     delayMicroseconds(10);
62     return ret;
63 }
64
```

Figura 3.9: Exemplo de um método alterado na classe MCP_CAN.

O módulo *CAN* quando tem mensagens no *buffer* para serem lidas, tem um pino de interrupção que é ativado, estando este associado ao pino 22 dos GPIO do *Raspberry*. É de referir que este não esta associada a uma interrupção no programa, mas sim é verificado no ciclo infinito do mesmo. Quando isto acontece é feita uma leitura das mensagens contidas no módulo CAN, para posteriormente serem descodificadas. A descodificação de cada mensagem é feita individualmente, ou seja, cada identificador é interpretado de forma diferente. É importante referir que a estrutura de dados da mensagem é dado por a Tabela 3.1 e que nem todos os identificadores utilizam os 8 *bytes*.

0...8 Bytes Data							
B0	B1	B2	B3	B4	B5	B6	B7

Tabela 3.1: Tabela com a identificação dos *bytes* de dados contidos numa mensagem.

De seguida, são mostradas as formas aplicadas para descodificação de cada mensagem, identificada com um número hexadecimal como referidas anteriormente na Tabela 2.3.

0x101

Informação sobre o estado da chave no carro através do valor contido unicamente em **B0**. Tem os seguintes estados:

0x00 - Chave desligada,

0x04 - Chave ligada.

0x208

Contem a informação em percentagem relativamente ao pedal do travão. Esta é obtida através da equação (3.1) em que são utilizados **B2** e **B3**.

$$\text{Travão} = \frac{(B2 \times 256 + B3) - 24576}{640} \times 100\ \% \tag{3.1}$$

0x210

Contêm a informação em percentagem relativamente ao pedal do acelerador. Esta é obtida

através da equação (3.2) em que é utilizado **B2**.

$$\text{Acelerador} = \frac{B2}{250} \times 100\ \% \tag{3.2}$$

0x231

Informação sobre o estado do travão se está pressionado ou não, contida em **B4**. Tem os seguintes estados:

0x00 - Pedal livre,

0x02 - Pedal pressionado.

0x236

Esta mensagem contêm duas informações. A informação da posição em graus e do torque aplicado ao volante são obtidas respectivamente com as equações (3.3) e (3.4).

$$\text{Posição do volante} = \frac{(B0 \times 256 + B1) - 4096}{2}\ \text{Graus} \tag{3.3}$$

$$\text{Momento do volante} = \frac{(B2 \times 256 + B3) - 4096}{2} \tag{3.4}$$

0x286

Informação da temperatura do carregador *on-board* obtida com a equação (3.5)

$$\text{Temp. Carregador } \textit{Onboard} = B3 - 40\ ^{\text{o}}\text{C} \tag{3.5}$$

0x298

Esta mensagem contêm duas informações. A informação da temperatura e rotações por minuto do motor são obtidas respectivamente com as equações (3.6) e (3.7).

$$\text{Temperatura do motor} = B3 - 40\ ^{\text{o}}\text{C} \tag{3.6}$$

$$\text{Rotações do motor} = (B6 \times 256 + B7) - 10000\ \text{RPM} \tag{3.7}$$

0x346

Informação sobre a autonomia restante em km, que obtém-se diretamente do valor de **B7**.

0x373

Esta mensagem apresenta o estado geral da bateria. O valor de tensão pode variar dentro do intervalo [343.2 , 389.7] V, e é obtido através da equação (3.8). Já o valor de corrente varia entre [-164.18 , 76.54] A obtido com equação (3.9), sendo que os valores negativos indicam que o carro esta a ser carregado.

$$\text{Tensão bateria} = \frac{B4 \times 256 + B5}{10}\ \text{V} \tag{3.8}$$

$$\text{Corrente da bateria} = \frac{(B2 \times 256) + (B3 - 128) \times 256}{100} \text{ A} \tag{3.9}$$

0x374

Representa a autonomia do carro em percentagem, resultado obtido da equação (3.10).

$$\text{Autonomia} = \frac{B1 - 10}{10} \text{ \%} \tag{3.10}$$

0x389

Contêm os valores de tenção e corrente AC do carregador *on-board*. O valor da tensão é obtido directamente através de **B1**. Já o valor de corrente é obtido com a equação (3.11).

$$\text{Corrente AC} = \frac{B6}{10} \text{ A} \tag{3.11}$$

0x3A4

Contêm todos os valores associados à climatização do habitáculo. Para estes serem obtidos é necessário examinar a mensagem recebida ao bit. Através de **B0** é obtida a informação da regulação de temperatura e o estado de três botões como indica a Tabela 3.2.

B0	b7	Estado do ar condicionado.
	b6	Aquecimento máximo ligado.
	b5	Recirculação de ar.
	b3...b0	Regulação de temperatura de ar entre [0,15], em que Frio>7>Quente.

Tabela 3.2: Descodificação do *byte* B0 da mensagem 0x3A4

Já através do valor **B1** é obtida a posição e a velocidade da ventilação como indica a Tabela 3.3.

B1	b7...b5	1 ou 2 - Direção da cara 3 ou 4 - Direção das pernas e cara 5 ou 6 - Direção das pernas 7 ou 8 - Direção do parabrisas e das pernas 9 - Direção do parabrisas
	b3...b0	Velocidade da ventoinha

Tabela 3.3: Descodificação do *byte* B1 da mensagem 0x3A4

0x412

Contêm os valores do número total de *km* e da velocidade atual em *km/h*. O valor do número total de km percorridos pelo o caro é obtido através da equação (3.12). O valor da velocidade atual é obtido directamente de **B1**.

$$\text{Total de km} = (B2 \times 65536) + (B3 \times 256) + B4 \tag{3.12}$$

0x418

Informação sobre a posição do manípulo das mudanças contida em **B0**. Tem os seguintes estados:

0x44 - Modo *drive* (**D**),
0x4E - Modo neutro (**N**).
0x50 - Modo de parqueamento (**P**),
0x44 - Modo de marcha atrás (**R**),

0x424

Nesta mensagem são obtidos diferentes estados do carro, como apresentado na Tabela 3.4.

B1	b6	Indicação luzes de travagem
	b5	Indicação luzes de estrada (máximos)
	b2	Indicação luzes de cruzamento (médios)
	b1	Indicação pisca esquerdo
	b0	Indicação pisca direito
B2	b7	Estado da conexão da ficha AC
	b0	Indicação de porta aberta

Tabela 3.4: Descodificação da mensagem 0x424.

0x6E1, 0x6E2, 0x6E3 e 0x6E4

Para compreender estas mensagens, é importante saber como é constituída a bateria do carro. Com a Figura 3.10 é possível identificar que a bateria é constituída por 88 células e tem 66 zonas para monitorização de temperatura. O conjunto destas quatro mensagens

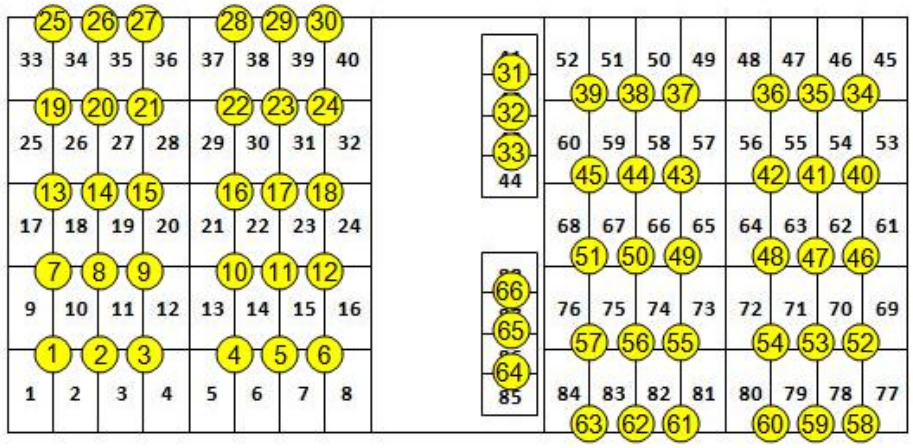


Figura 3.10: Constituição da bateria do carro.

permite obter o valor da tensão e temperatura de toda a bateria por secções, através das equações (3.13) e (3.14). É preciso ter em conta a Tabela 3.5, onde mostra como é mapeada a informação de toda a bateria por estas 4 mensagens.

$$\text{Tensão célula}(x) = \frac{B1_{\text{Célula}(x)} \times 255 + B0_{\text{Célula}(x)}}{100} \text{ V} \tag{3.13}$$

$$\text{Temperatura célula}(x) = B_{\text{Temp}(x)} - 50\ ^{\text{o}}\text{C} \tag{3.14}$$

Depois da mensagem recebida ser descodificada, é mostrada no terminal e adicionada a um ficheiro. De modo a, mais tarde, ser possível tratar os dados obtidos, por cada mensagem recebida é escrita uma nova linha no ficheiro. Cada linha do ficheiro, tem separado por vírgulas os seguintes parâmetros:

- Hora de registo da mensagem,
- Velocidade atual,
- Estado do pedal do travão,
- Posição do pedal do travão,
- Posição do volante,
- Momento do volante,
- Kilometros percorridos por o carro,
- Mudança engrenada,
- RPM do motor elétrico,
- Autonomia restante em km,
- Tensão da bateria,
- Corrente da bateria,
- Percentagem de carga da bateria.

Por fim a solução final está representada na Figura 3.11.

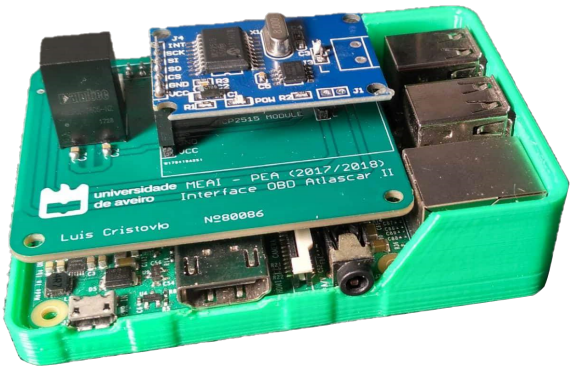


Figura 3.11: Solução final obtida.

Byte	B0	B1	B2	B3	B4 | B5	B6 | B7
0x6E1	1	0	Temp. 1	Temp. 2	Célula 1	Célula 2
	2	0	Temp. 3	Temp. 4	Célula 3	Célula 4
	3	0	Temp. 5	Temp. 6	Célula 5	Célula 6
	4	0	Temp. 7	Temp. 8	Célula 7	Célula 8
	5	0	Temp. 9	Temp. 10	Célula 9	Célula 10
	6	0	Temp.11	Temp. 12	Célula 11	Célula 12
	7	0	Temp. 13	Temp. 14	Célula 13	Célula 14
	8	0	Temp. 15	Temp. 16	Célula 15	Célula 16
	9	0	Temp.17	Temp. 18	Célula 17	Célula 18
	10	0	Temp. 19	Temp. 20	Célula 19	Célula 20
	11	0	Temp. 21	Temp. 22	Célula 21	Célula 22
	12	0	Temp. 23	Temp. 24	Célula 23	Célula 24
0x6E2	1	Temp. 25	Temp. 26	0	Célula 25	Célula 26
	2	Temp. 27	Temp. 28	0	Célula 27	Célula 28
	3	Temp. 29	Temp. 30	0	Célula 29	Célula 30
	4	Temp. 31	Temp. 32	0	Célula 31	Célula 32
	5	Temp. 33	Temp. 34	0	Célula 33	Célula 34
	6	Temp. 35	0	0	Célula 35	Célula 36
	7	Temp. 36	Temp. 37	0	Célula 37	Célula 38
	8	Temp. 38	Temp. 39	0	Célula 39	Célula 40
	9	Temp. 40	Temp. 41	0	Célula 41	Célula 42
	10	Temp. 42	Temp. 43	0	Célula 43	Célula 44
	11	Temp. 44	Temp. 45	0	Célula 45	Célula 46
	12	Temp. 46	0	0	Célula 47	Célula 48
0x6E3	1	Temp. 47	0	Temp. 48	Célula 49	Célula 50
	2	Temp. 49	0	Temp. 50	Célula 51	Célula 52
	3	Temp. 51	0	Temp. 52	Célula 53	Célula 54
	4	Temp. 53	0	Temp. 54	Célula 55	Célula 56
	5	Temp. 55	0	Temp. 56	Célula 57	Célula 58
	6	0	0	0	0	0
	7	Temp. 57	0	Temp. 58	Célula 59	Célula 60
	8	Temp. 59	0	Temp. 60	Célula 61	Célula 62
	9	Temp. 61	0	Temp. 62	Célula 63	Célula 64
	10	Temp. 63	0	Temp. 64	Célula 65	Célula 66
	11	Temp. 65	0	Temp. 66	Célula 67	Célula 68
	12	0	0	0	0	0
0x6E4	1	0	0	0	Célula 69	Célula 70
	2	0	0	0	Célula 71	Célula 72
	3	0	0	0	Célula 73	Célula 74
	4	0	0	0	Célula 75	Célula 76
	5	0	0	0	Célula 77	Célula 78
	6	0	0	0	0	0
	7	0	0	0	Célula 79	Célula 80
	8	0	0	0	Célula 81	Célula 82
	9	0	0	0	Célula 83	Célula 84
	10	0	0	0	Célula 85	Célula 86
	11	0	0	0	Célula 87	Célula 88
	12	0	0	0	0	0

Tabela 3.5: Mapeamento das células da bateria para monitorização

Capítulo 4

Testes e resultados

Este capítulo descreve os testes realizados com sistema desenvolvido e os resultados obtidos. Assim, o sistema desenvolvido foi instalado no veículo como mostra na Figura 4.1. Foi tido em consideração a instalação do sistema num local onde não perturbasse a condução do veículo.

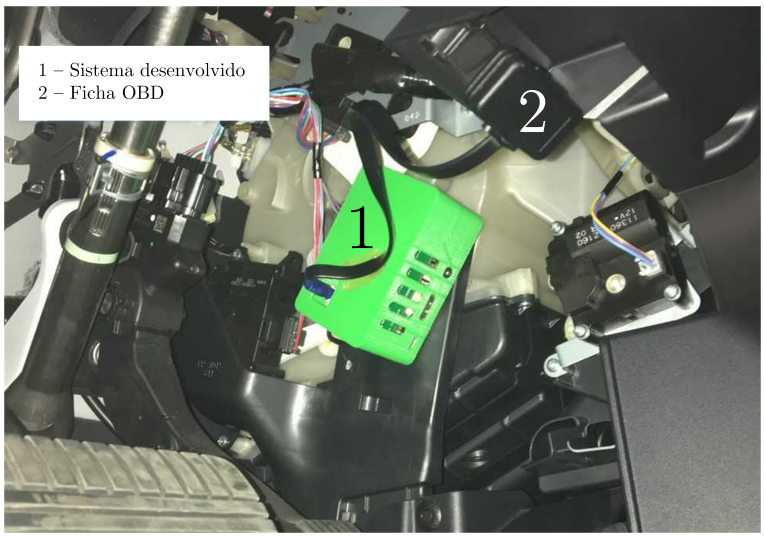


Figura 4.1: Instalação do sistema no carro.

Com a informação contida no subcapítulo 2.2.2, é possível calcular o número de mensagens que circulam por segundo no barramento CAN. Tendo em conta o período de cada identificador, foi obtido o valor de 1671 mensagens por segundo como apresentado na Tabela 4.1. Considerando o pior caso, que é quando "Data"é constituída por 8 *bytes*, é obtido o valor 180467 kb/s de dados que circulam no barramento.
Além do número de bits presentes no barramento ser inferior a 1000 kb/s, não foi possível ler todas as mensagens que circulam no barramento. Ou seja são recebidos todos os identificadores, mas não com a frequência esperada.

Período de ocorrência (ms)	Número de identificadores	Número de vezes por segundo	Total de mensagens por período
1000	1	1	1
200	1	5	5
100	20	10	200
50	12	20	240
40	5	25	125
20	14	50	700
10	4	100	400
Total de mensagens por segundo no barramento			1671

Tabela 4.1: Número de mensagens no barramento *CAN* por segundo

Após a ligação ao carro, e o código desenvolvido estar a funcionar, foi possível verificar quais a mensagens ou identificadores que estão presentes no barramento.
Também foi possível verificar se a desencriptação das mensagens estava correta ou errada.
De todos os identificadores conhecidos, só não foram verificadas as seguintes mensagens recebidas:

- 0x3A4
- 0x424
- 0x6E1...0x6E4

Todos os outros identificadores foram verificados e validados.

Capítulo 5

Conclusões

No decorrer deste projeto foram abordados vários conceitos tais como programação em C++, do funcionamento do protocolo CAN, desenvolvimento de circuitos elétricos e a respectiva PCB como por fim também de modulação 3D para fazer a caixa de proteção do sistema desenvolvido.
O estudo feito sobre o protocolo CAN ajudou bastante, de modo a ser sabido como iriam ser recebidas as mensagens, do carro em estudo, e como futuramente seriam desencriptadas.
Inicialmente estava pensado a utilização de uma solução de mercado para receber os dados do barramento CAN. Visto que nenhuma das disponibilizadas funcionou no carro, teve de ser desenvolvida uma forma que permitisse essa comunicação. No fim está escolha pode não ter sido a melhor para a aplicação desejada, devido a velocidade de recepção das mensagens.
O processo de desencriptação mostro-se um objetivo moroso de cumprir, pois o fabricante do carro não disponibiliza nenhuma informação relativamente as mensagens que circulam no barramento sendo esta pesquisa baseada em foruns de entusiastas na área. Também é preciso ter em conta que não foram descobertos todos os parâmetros que circulam no barramento CAN.
Em relação aos objetivos pretendidos inicialmente foram alcançados.
Como trabalhos futuros existem várias tarefas que podem enriquecer o projeto, sendo elas:

- Implementação do sistema numa rede ROS,

- Aumentar a frequência da recepção de mensagens CAN,

- Possibilidade de escrita no barramento,

- Continuação do estudo e da desencriptação do carro.

Por fim, a realização deste projeto mostro-se bastante desafiador e enriquecedor, não so pelos os conhecimentos abordados e adquiridos mas também pela a diversidade das tarefas realizadas.

Referências

[1] ATLASCAR. (Consultado em 21-04-2018). [Online]. Available: http://atlas.web.ua.pt/atlascar.html

[2] T. Instruments. (Consultado em 22-04-2018) Iso1050 isolated can transceiver. [Online]. Available: http://www.ti.com/lit/ds/symlink/iso1050.pdf

[3] S. M. i.-M. Mitsubishi. (2011)

[4] P. Laes. (Consultado em 26-04-2018) Mitsubishi i-miev obd-ii pid documentation. [Online]. Available: https://github.com/plaes/i-miev-obd2

[5] MiEV-CAN. (Consultado em 02-05-2018). [Online]. Available: http://myimiev.com/forum/viewtopic.php?f=25&t=727

[6] W. Fahl. (Consultado em 03-05-2018) Can4ever. [Online]. Available: http://can4eve.bitplan.com/index.php/Main_Page

[7] D. between OBD-II and CAN. (Consultado em 22-04-2018). [Online]. Available: https://mechanics.stackexchange.com/questions/25561/difference-between-obdii-and-can

[8] T. Instruments. (Consultado em 22-04-2018) Introduction to the controller area network (can). [Online]. Available: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf

[9] B. descrição protocolo de comunicações CAN. (Consultado em 21-04-2018). [Online]. Available: https://paginas.fe.up.pt/~ee99058/projecto/protocolo.html

[10] *Arduino*. [Online]. Available: http://www.arduino.cc

[11] MCP2515. *Microchip*. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/20001801H.pdf

[12] AMSR-78-NZ. *aimec*. [Online]. Available: http://www.aimtec.com/site/Aimtec/files/Datasheet/HighResolution/AMSR2-78-NZ.pdf?ft4=54-807

[13] R. D. Raspbian. (Consultado em 21-04-2018). [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/

[14] Seeed-Studio. Can bus shield. [Online]. Available: https://github.com/Seeed-Studio/CAN_BUS_Shield