



Escola Superior de Tecnologia

Mestrado em Engenharia Eletrónica e de Computadores

Laboratórios Integrados 2

Sistema de Identificação de condutor

Ricardo Pinto, 9336

José Maciel, 10288

Cláudia Ferreira, 443

Isaías Barbosa, 14424

Julho de 2017

Resumo

O presente relatório visa descrever o trabalho prático realizado no âmbito da disciplina Laboratórios Integrados 2, lecionada durante o 2º Semestre do 1º Ano do curso de Mestrado em Engenharia Eletrónica e de Computadores.

O trabalho realizado permite relacionar as disciplinas lecionadas durante este segundo semestre, e tem como objetivo a implementação de um sistema de identificação facial com uma interação de dispositivos em rede CAN.

No decorrer do trabalho prático foi necessário um estudo aprofundado sobre o que é a comunicação CAN desde os componentes necessários para a elaboração dos esquemas correspondentes à formação das tramas de envio. Também, foi necessário um estudo e implementação de uma aplicação capaz de efetuar a identificação facial, e consequentemente ativar possíveis saídas num dispositivo remoto consoante a pessoa reconhecida.

Concluímos que, com a combinação de um processamento de imagem capaz de identificar faces e uma comunicação com interações físicas (entradas e saídas) é possível comandar dispositivos sem ser necessário carregar em botões ou outro tipo de controlos físicos que seja necessário o “telecomando”, e assim deste modo automatizar um possível sistema.

Palavras Chave (Tema): Redes; condução; comunicação; Interação

Palavras Chave (Tecnologias): CAN; ATMEGA; OpenCV; Eagle.

Índice

1	<i>Introdução</i>	10
1.1	Enquadramento	10
1.2	Apresentação do trabalho prático	10
1.3	Planificação	10
1.4	Organização do relatório	12
2	<i>Enunciado</i>	13
2.1	Descrição e requisitos dos módulos do sistema	14
2.1.1	Módulo Master CAN (hardware)	15
2.1.2	Módulo de expansão I/O CAN (hardware)	15
2.1.3	Módulo de Comunicação PC <-> Módulo Master CAN	17
2.1.4	Módulo de Comunicação Módulo Master CAN <-> Módulo de expansão I/O CAN	17
2.1.5	Módulo de processamento de imagem: identificação e reconhecimento de faces	18
2.1.6	Módulo de interface com o utilizador	18
3	<i>Descrição técnica</i>	20
3.1	Módulo de expansão I/O CAN	20
3.1.1	Esquemático	20
3.1.2	Desenho da Board	22
3.1.3	Montagem PCB	23
3.2	Módulo Master CAN	24
3.2.1	Esquemático	24
3.2.2	Placa de Circuito impresso	34
3.3	Comunicação CAN	35
3.4	MCP25050	36
3.5	Módulo de Comunicação PC <-> Módulo Master CAN (Software)	39

3.6	Módulo de Comunicação Módulo Master CAN <-> Módulo de expansão I/O CAN	
		40
3.7	Software	41
3.7.1	Criação do Servidor Web com Express	42
3.8	Módulo de processamento de imagem	45
3.8.1	Identificação e reconhecimento de faces.....	45
3.9	Módulo de interface com o utilizador	49
4	Testes	52
5	Conclusões	59
6	Referencias Bibliográficas	60

Índice de Figuras

<i>Figura 1 - Reconhecimento do condutor</i>	13
<i>Figura 2 - Arquitetura do Sistema</i>	14
<i>Figura 3 - Módulo Master.....</i>	15
<i>Figura 4 - Módulo de expansão I/O</i>	16
<i>Figura 5 - Esquemático do modulo de expansão.....</i>	20
<i>Figura 6 - Esquema fonte comutada Módulo expansão.....</i>	20
<i>Figura 7 - Esquema transmissão CAN do módulo de expansão.</i>	21
<i>Figura 8 - Entradas a optoacopladores no módulo de expansão.</i>	22
<i>Figura 9 - Saída por Rele no modulo de expensão.....</i>	22
<i>Figura 10 - Desenho final da PCB modulo de expansão.</i>	23
<i>Figura 11 – Layout final da PCB Slave.</i>	23
<i>Figura 12 - Correção 5 Volts.</i>	24
<i>Figura 13 – Diagrama de Blocos do caderno de encargos do trabalho de LAB2 [1]</i>	24
<i>Figura 14 – Diagrama de blocos Módulo Master[2].....</i>	25
<i>Figura 15 - Power Module – Fonte de alimentação comutada, com entrada de 5.5 V a 36V</i>	25
<i>Figura 16 – Cálculo da tensão saída[2]</i>	26
<i>Figura 17 – Input Module. Modulo de entrada isolado por foto-acopladores</i>	27
<i>Figura 18 – Usb Module, com FT232rl.....</i>	28
<i>Figura 19 – Reset Module.....</i>	28
<i>Figura 20 – Cpu Module, com o microcontrolador Atmega328</i>	29
<i>Figura 21 – Signalizing Module.....</i>	30
<i>Figura 22 – CAN Module. Conjunto MCP2515 e MCP2551</i>	30
<i>Figura 23 – Out Module. Controlo da potência da placa master</i>	31
<i>Figura 24 – Can Out Module.....</i>	32
<i>Figura 25 – Master em PCB Botton Side.....</i>	34
<i>Figura 26 - Master em PCB Top Side</i>	34
<i>Figura 27 - Master com componentes soldados em funcionamento</i>	34

<i>Figura 28 - Teste de comunicação CAN (realizado em fritzing).....</i>	35
<i>Figura 29 – Teste de envio CAN.....</i>	35
<i>Figura 30 - Teste de receção CAN.....</i>	36
<i>Figura 31 – Esquema de programação MCP25050.....</i>	37
<i>Figura 32 – Interface inicial do MCP250xxProgrammer.exe</i>	37
<i>Figura 33 – Representação da configuração do GPIO no MCP25050</i>	38
<i>Figura 34 – CAN Registers MCP25050.....</i>	38
<i>Figura 35 – Programação do MCP25050</i>	39
<i>Figura 36 – Registros do MCP25050.....</i>	40
<i>Figura 37– Organização do projeto.....</i>	41
<i>Figura 38– Módulos a utilizar.....</i>	43
<i>Figura 39– Criação do ficheiro package.json.....</i>	43
<i>Figura 40– Criação do ficheiro package.json.....</i>	44
<i>Figura 41 - Diagrama de herança cv::face::FaceRecognizer[12].....</i>	47
<i>Figura 42 - Código do ficheiro train-user.js</i>	48
<i>Figura 43 - Ecrã principal.....</i>	49
<i>Figura 44 - Ecrã Criação de perfil inicial</i>	49
<i>Figura 45 - Ecrã de sucesso da tarefa.....</i>	49
<i>Figura 46 - Ecrã Adicionar Perfil</i>	49
<i>Figura 47 - Ecrã do utilizador</i>	50
<i>Figura 48 - Ecrã de perfis de utilizador</i>	50
<i>Figura 49 – Ecrã de remoção de perfil.....</i>	50
<i>Figura 50 – Excerto de código com aplicação do videoCanvas</i>	51
<i>Figura 51 – Output MCP25050.....</i>	52
<i>Figura 52 – Excerto de código de ligação ao Master.....</i>	53
<i>Figura 53 – Excerto de código de simulação de saídas</i>	53
<i>Figura 54 - Testes CAN.</i>	54
<i>Figura 55 - Trama CAN, parte dos dados.</i>	55
<i>Figura 56 - Combinações possíveis para o Slave CAN.....</i>	55

<i>Figura 57 - Envio consoante SerialPort.....</i>	56
<i>Figura 58 - Comparação da Trama e ativação de saídas no Slave.</i>	56
<i>Figura 59 – Representação física das saídas.</i>	57
<i>Figura 60 - Aplicação de ligação ao Slave por CAN.</i>	57
<i>Figura 61 - Criação da Porta Serie.</i>	58
<i>Figura 62 - Ligação à porta serie predefinida.....</i>	58
<i>Figura 63 - Formação da palavra para interpretação do Atmega.....</i>	58

1 Introdução

1.1 Enquadramento

Este trabalho prático surge no âmbito da disciplina de Laboratórios Integrados 2 do Curso Mestrado em Engenharia Eletrónica e de Computadores e compreende o desenvolvimento de um sistema integrado que engloba um projeto de *hardware* e *software*.

1.2 Apresentação do trabalho prático

O trabalho prático, permitiu a solidificação dos conhecimentos obtidos durante as aulas teóricas na disciplina em questão.

Ao longo da implementação do trabalho aplicamos conhecimentos sobre os softwares de visão por computador *OpenCV*, *Javascript*, *Eagle*, *Arduíno*.

O trabalho prático foi realizado em contexto de aula, mas a grande parte fora da sala de aula, estando sempre o docente disponível para respostas a possíveis dúvidas que pudessem surgir ao longo do desenvolvimento.

1.3 Planificação

Em termos de planificação, foi inicialmente definido o hardware e software a utilizar durante a execução do projeto. A seguir apresenta-se a listagem destas escolhas.

Escolhas Estratégicas

1. Sistema Operativo – Windows 10, OS X El Capitain
2. Ferramenta de gestão de projeto – Trello
3. Controlo de versões – Git (GitHub)
4. Hardware
 - a. realização de um protótipo em *breadboard*
 - b. software de desenho de PCB – EAGLE 7.6.0
 - i. ATMEGA 328 (AVR) – linguagem C
 - c. utilização de módulo conversor USB/RS232 FT232R
 - d. Componentes *through-hole*

- e. Fabrico da placa (pedido ao Prof. Moreira e empresa externa que respondeu em tempo útil).

5. Software

- a. Linguagem de programação – linguagem C / Node.js/Javascript
- b. IDE – *Sublime*
- c. Biblioteca de visão por computador – OpenCV 2.4 (Bindings oficiais para Node.js)
- d. Biblioteca de camada de interface – *HTML/CSS/Javascript*

1.4 Organização do relatório

O relatório encontra-se dividido em cinco capítulos principais:

1. Introdução;
2. Enunciado;
3. Descrição técnica;
4. Testes;
5. Conclusões;

No primeiro capítulo efetuou-se um enquadramento, apresentação e planificação do projeto.

Seguidamente, no segundo ponto foi colocada toda a informação relativa ao enunciado fornecido pelo docente.

No capítulo descrição técnica abordou-se todas as tarefas realizadas na implementação do sistema, tanto a nível de *hardware* como de *software*. A ilustração e esclarecimento sobre as arquiteturas utilizadas, a informação sobre a criação do código necessário para a execução do sistema e toda a análise sobre tudo o que foi implementado.

No capítulo seguinte temos a ilustração de alguns testes realizados de forma a comprovar o funcionamento do sistema.

Por fim, nas conclusões temos a análise sobre o projeto final desenvolvido identificando objetivos alcançados, problemas e dificuldades sentidas no seu desenvolvimento e propostas futuras.

2 Enunciado

A indústria automóvel procura constantemente novas formas de adaptar o funcionamento dos seus produtos às conveniências e conforto do cliente. O objetivo final deste projeto é o desenvolvimento e implementação de um sistema que permita que um automóvel reconheça a identidade do seu condutor, o que permitirá o ajuste automático de parâmetros do interior do automóvel (posição do banco, volume do rádio, etc.) às preferências do condutor que estiver a conduzi-lo em cada momento.

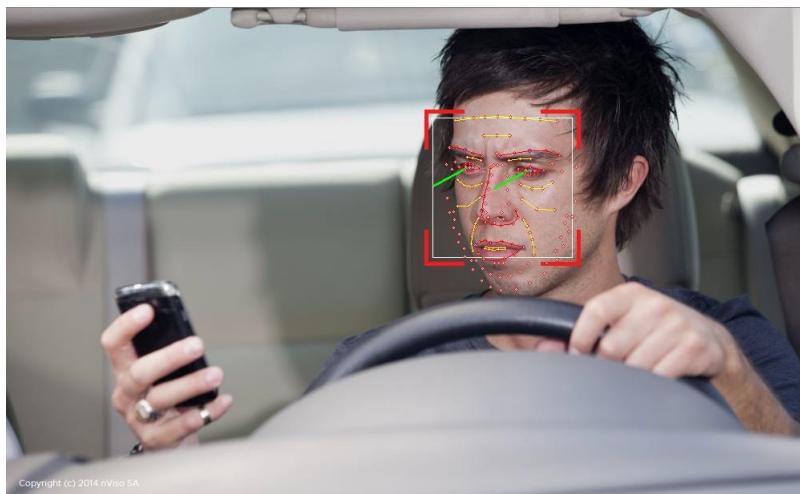


Figura 1 - Reconhecimento do condutor

A verificação da identidade do condutor será efetuada através de visão por computador, com recurso a imagens obtidas por uma câmara apontada à face deste, tipicamente montada junto ao retrovisor do automóvel. O módulo de processamento de imagem deverá comparar a face presente na imagem com faces de referência existentes numa base de dados com os condutores previamente introduzidos no sistema. O processamento de imagem será implementado em PC. A comunicação entre o PC e os restantes dispositivos do automóvel será efetuada através do barramento CAN, um standard de comunicação comum nos automóveis atuais.

Este trabalho comprehende, portanto, o desenvolvimento de um sistema integrado que engloba projeto de hardware e software. Neste projeto, a aplicação que correr no PC implementa funções de processamento e análise de imagem e comunica com um dispositivo através de USB, que por sua vez permite a comunicação com uma rede industrial CAN. O esquema global do sistema é o da Figura 2.

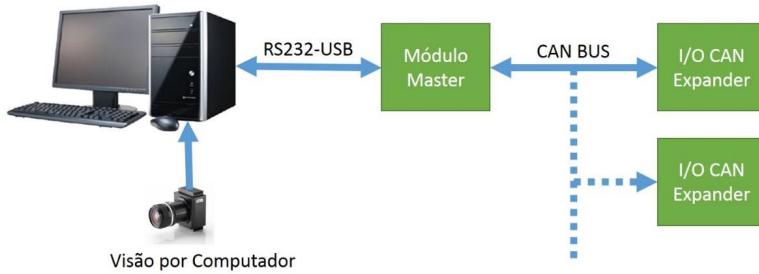


Figura 2 - Arquitetura do Sistema

2.1 Descrição e requisitos dos módulos do sistema

O sistema é constituído por dois blocos principais. O primeiro bloco, baseado em software, é responsável pela interface com o utilizador, gestão dos dados e configuração do sistema, processamento e análise de imagem e comunicação dos dados através de USB. O segundo bloco, baseado em sistemas embebidos, é responsável pela comunicação, através de uma rede industrial CAN, dos dados obtidos no módulo de software.

O bloco de software deverá ser constituído por:

- Módulo de interface gráfico com o utilizador
- Módulo de aquisição de imagem, obtida por uma câmara, deteção e reconhecimento de faces na imagem
- Módulo de comunicação de dados por protocolo série através de USB.

O bloco baseado em sistemas embebidos incluirá o desenvolvimento de dois tipos módulos de hardware:

- Módulo Master, que estará ligado ao PC por USB, responsável pela gestão e transmissão dos dados resultantes da aplicação de processamento e análise de imagem, através de uma rede de comunicação industrial baseada no protocolo CAN BUS.
- Módulo de expansão I/O, que estará ligado ao barramento CAN, traduzirá nas suas saídas os dados resultantes da análise de imagem, transmitidos pelo módulo master.

2.1.1 Módulo Master CAN (hardware)

Este módulo de hardware será utilizado para realizar a interface entre o PC a rede CAN.

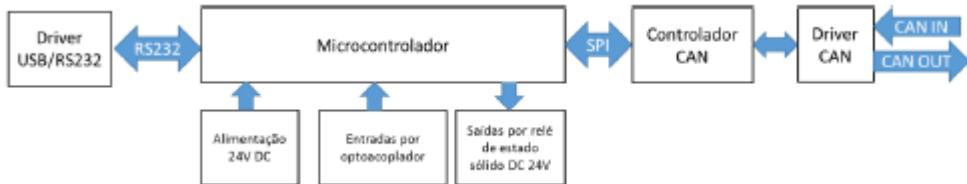


Figura 3 - Módulo Master

Para tal, no projeto deverão ser consideradas as seguintes características:

- Alimentação por fonte externa 12/24 V DC;
- Ligação ao PC por USB – deverá para isso incorporar um conversor USB/RS232 (FT232R ou similar);
- Transmissão de dados através do protocolo CAN – deverá para isso integrar um microcontrolador (Atmel 328), que comunicará com o PC através do protocolo RS232, e com um controlador Stand-Alone CAN (MCP2515) através da interface SPI.
- Regularização dos níveis de tensão para rede CAN através de um driver CAN (MCP2551SN);
- Disponibilização de um conjunto de 6 entradas (por opto-acoplador LTV-356T ou similar) e 2 saídas (por relé de estado sólido - CPC1002N ou similar), com indicação luminosa do estado.

2.1.2 Módulo de expansão I/O CAN (hardware)

Este módulo de hardware será utilizado como módulo de expansão da rede CAN.



Figura 4 - Módulo de expansão I/O

No projeto deverão ser consideradas as seguintes características:

- Alimentação por fonte externa 12/24 V DC;
- Implementação do protocolo CAN através de um bloco expansor de entradas e saídas MCP25020 ou MCP25050. A regularização dos níveis de tensão para rede CAN deverá ser estabelecida através de um driver CAN (MCP2551SN);
- Disponibilização de um conjunto de 2 entradas (por opto-acoplador LTV-356T ou similar) e 6 saídas (por relé de estado sólido - CPC1002N ou similar) com indicação luminosa do estado.

2.1.3 Módulo de Comunicação PC <-> Módulo Master CAN

É necessário definir um protocolo de comunicação, entre o PC e o módulo Master, que permita:

- Enviar, do PC para o Módulo Master, o resultado da identificação de faces, com identificação de quais os módulos de expansão I/O para onde deve ser enviado.
- Enviar, do PC para o Módulo Master, um pedido de identificação de todos os módulos de expansão I/O ligados à rede CAN.
- Enviar, do Módulo Master para o PC, um pedido de criação/alteração de registo de face na base de dados. Quando uma das entradas do módulo Master é ativada, a posição correspondente da base de dados deve passar a referir-se à face detetada nesse momento (caso alguma face esteja a ser detetada).

2.1.4 Módulo de Comunicação Módulo Master CAN <-> Módulo de expansão I/O CAN

É necessário definir um módulo de comunicação no módulo Master e a configuração do módulo de expansão, que permita:

- Configurar os registo do controlador Stand-Alone CAN para que este funcione no modo pretendido;
- Converter as mensagens recebidas do PC no protocolo CAN;
- Transmitir as mensagens através da rede CAN;

- Consultar o código de identificação de todos os módulos de expansão I/O ligados à rede CAN. Para programar os registos do módulo de expansão pode ser utilizado o software MCP250xxProgrammer.exe. Recomenda-se a utilização do programador PICKit 2.

2.1.5 Módulo de processamento de imagem: identificação e reconhecimento de faces

As principais tarefas associadas ao processamento de imagem são:

- Identificar o melhor setup para aquisição da imagem (câmara, iluminação, óptica, etc);
- Calibrar a distorção radial e tangencial provocada pela lente;
- Detetar a(s) face(s) na imagem, escolhendo a principal
- Segmentar a face
- Efetuar uma extração de características da face detetada
 - Definir quais as características relevantes a extrair
- Comparar características da face detetada com características de faces existentes na base de dados.

Os dados a enviar para a rede CAN são:

- Se foi identificada uma face
- A identidade da face identificada.

2.1.6 Módulo de interface com o utilizador

Deverá ser desenvolvida uma interface com o utilizador que permita uma fácil e intuitiva interação com o sistema. Esta deverá respeitar os seguintes requisitos:

- O interface deverá indicar o estado da ligação ao Módulo Master
- O interface deverá poder listar todos os módulos ligados à rede CAN
- A imagem da câmara deverá aparecer em tempo real no ecrã o PC
- A deteção de faces deverá ser assinalada na imagem

- A identificação de faces deverá ser assinada na imagem
- A não deteção de faces deverá ser assinada numa ou mais saídas digitais de um ou mais módulos de expansão I/O; a(s) saída(s) dos módulos a ativar deverão ser configuráveis
- A identificação de faces deverá ser assinada numa ou mais saídas digitais de um ou mais módulos de expansão I/O; a(s) saída(s) dos módulos a ativar deverão ser configuráveis
- A deteção de faces sem identificação deverá ser assinada numa ou mais saídas digitais de um ou mais módulos de expansão I/O; a(s) saída(s) dos módulos a ativar deverão ser configuráveis
- O interface deverá permitir a criação, edição e remoção de registo de faces na base de dados; para cada face deverá ser possível configurar quais as saídas de quais módulos de expansão I/O deverão ser ativados/desativados
- O interface deverá permitir a receção de pedidos de criação de novo registo de face na base de dados provenientes do módulo master, com a face identificada no momento.

3 Descrição técnica

3.1 Módulo de expansão I/O CAN

3.1.1 Esquemático

Seguidamente irá ser abordado as diferentes partes do esquemático do modulo de expansão e a explicação de cada parte resultante do esquema representado na Figura 5.

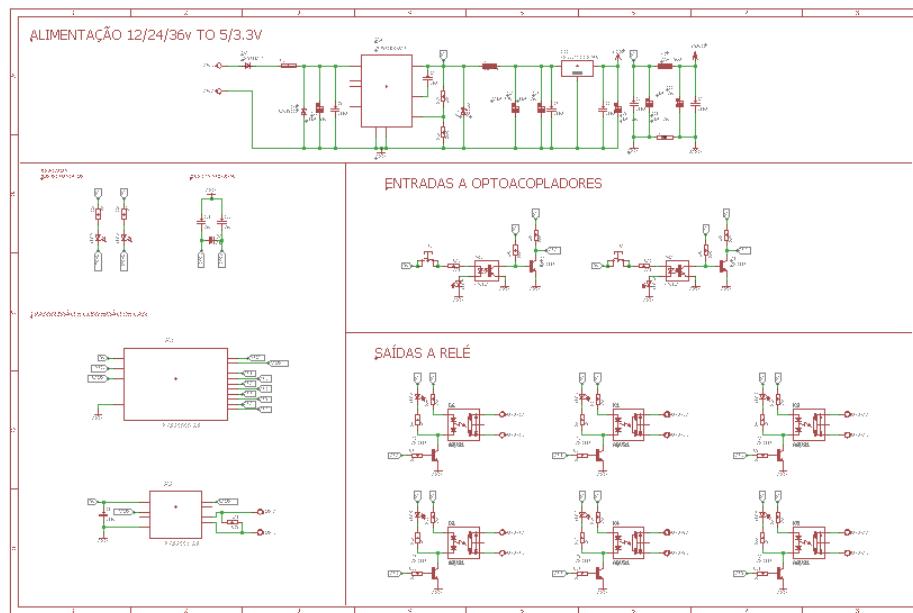


Figura 5 - Esquemático do modulo de expansão.

Como um dos pedidos do enunciado é a utilização de tensões de 12 a 24V, optou-se por uma fonte comutada, em que recebe a tensão que pode ir até 36V e garante sempre 5 Volts à entrada dos circuitos integrados como é retratado na Figura 6.

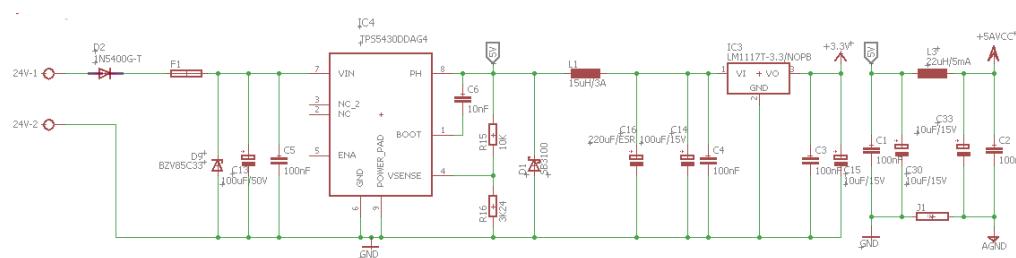


Figura 6 - Esquema fonte comutada Módulo expansão.

No que se refere à conexão CAN, o módulo tem o componente MCP2551 que passa a informação para os níveis de tensão CAN, seguidamente para interpretar a informação tem o MCP 25050, em que consoante a informação recebida ativa as suas saídas, e em função das suas entradas, envia a informação para o Master. Tal como é representado na Figura 7.

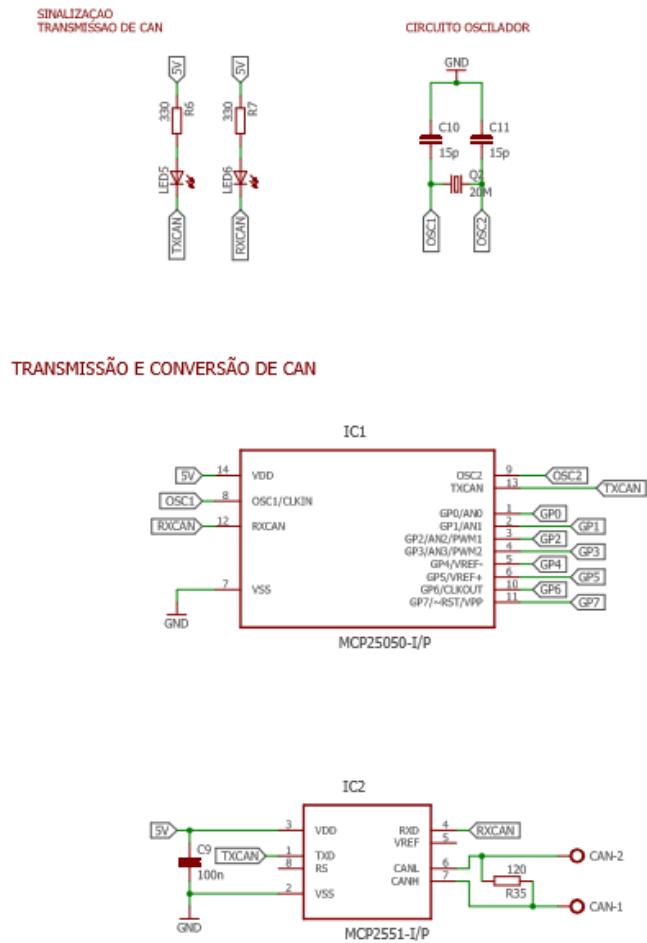


Figura 7 - Esquema transmissão CAN do módulo de expansão.

No que se refere à interação física com o utilizador, temos como *inputs* dois botões em que fazem a passagem de estado para o MCP25050 através de optoacopladores e passagem para transístor. Desta forma, garante-se a tensão certa nas entradas do MCP. O esquema foi baseado em exemplos encontrados durante a pesquisa inicial. A Figura 8 retrata uma as entradas, a outra tem a mesma configuração, ligando outro GPIO do MCP 25050.

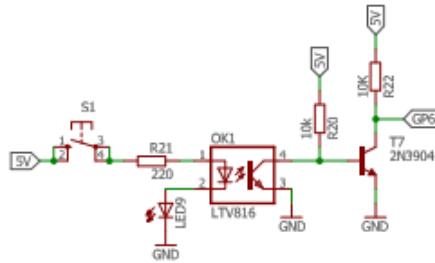


Figura 8 - Entradas a optoacopladores no módulo de expansão.

Por fim a nível do esquemático, termina-se com as saídas, neste projeto foram utilizados reles de estado sólido com a ativação através de transístor. Desta forma garante o isolamento elétrico entre o exterior e o MCP. Existem neste módulo seis saídas com a mesma configuração retratado na Figura 9.

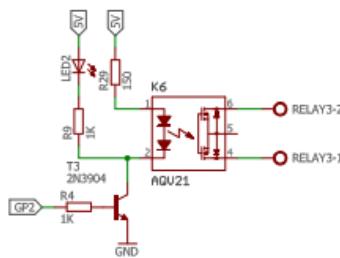


Figura 9 - Saída por Rele no modulo de expensão.

3.1.2 Desenho da Board

Após a criação dos esquemáticos realizou-se o desenho da PCB em Eagle. A Figura 10 retrata o resultado final.

A PCB está dividida em quatro partes, na parte superior direita temos a fonte de alimentação onde podemos receber uma tensão até a 36V, na parte central e inferior direita temos a comunicação CAN e o controlo de entradas e saídas. No que se refere à interação com o utilizador, as entradas estão na parte interior, e no lado direito tem todas as saídas, todas as interações com o utilizador estão sinalizadas luminosamente sobre os seus estados.

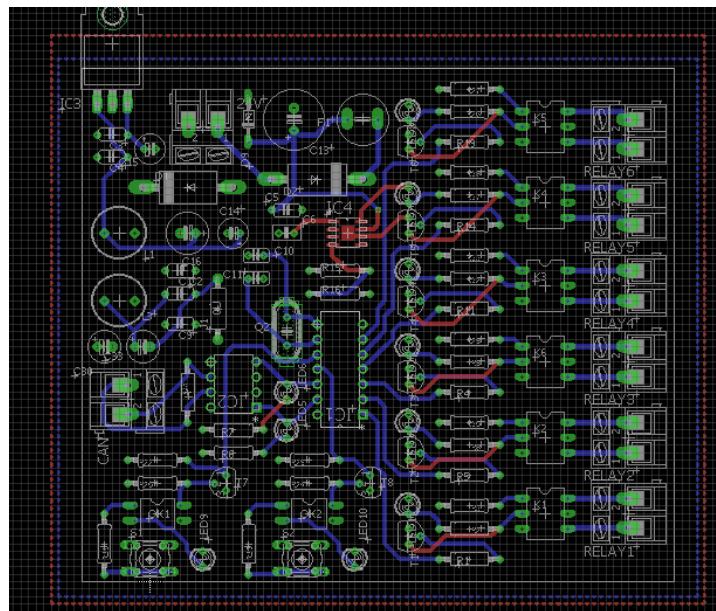


Figura 10 - Desenho final da PCB modulo de expansão.

3.1.3 Montagem PCB

Após o desenho da *Board* em *Eagle* e a soldagem, obteve-se o seguinte layout.

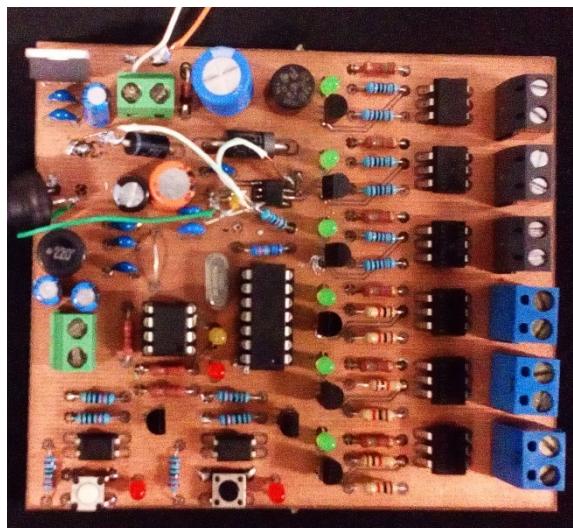


Figura 11 – Layout final da PCB Slave.

Como se pode verificar na Figura 11, existem componentes soldados de forma incorreta, isto deve-se ao facto de uma nova analise à fonte comutada em que a tensão de 5 volts só era obtida após a bobina de 15uH, e não antes como está no esquemático, representado na Figura 12, por isso foi necessário a decomposição dessa parte e a consequente correção das ligações.

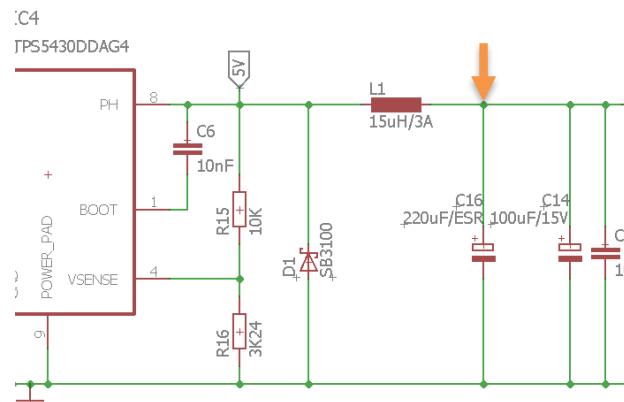


Figura 12 - Correção 5 Volts.

No que se refere aos outros componentes constituintes da placa, tudo foi testado e a funcionar em *breadboard*.

3.2 Módulo Master CAN

3.2.1 Esquemático

Seguindo o caderno de encargos apresentado para elaboração do trabalho de LAB 2[1], procedeu-se ao desenho de um sistema que suportará o pedido do docente da disciplina (Prof. José Brito). Na Figura 13, apresentamos o diagrama de blocos da proposta acima referida, onde sobressai a parte que a equipa de *hardware* Módulo Master fará o seu trabalho.

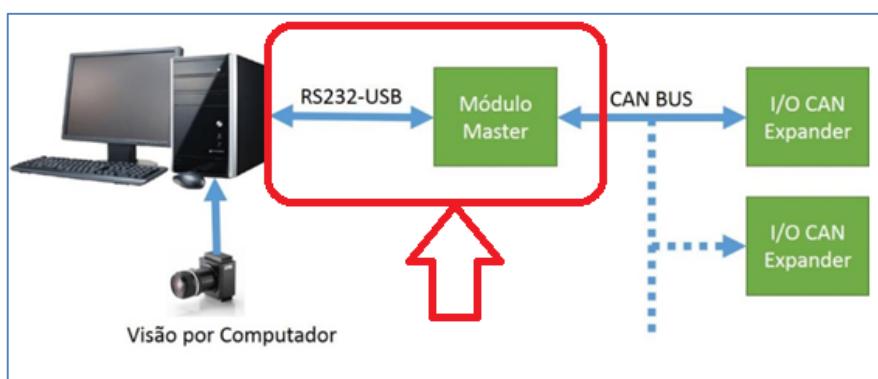


Figura 13 – Diagrama de Blocos do caderno de encargos do trabalho de LAB2 [1]

O Módulo Master, estará ligado ao PC pela porta USB, que será a “*responsável pela gestão e transmissão dos dados resultantes da aplicação de processamento e análise de imagem, através de uma rede de comunicação industrial baseada no protocolo CAN BUS*”. Para tal, foram consideradas as características já acima mencionadas para o Módulo Master.

Na Figura 14, o diagrama de blocos do Módulo Master, segundo a proposta do professor José Brito, à qual daremos corpo e conforme passaremos a explicar.

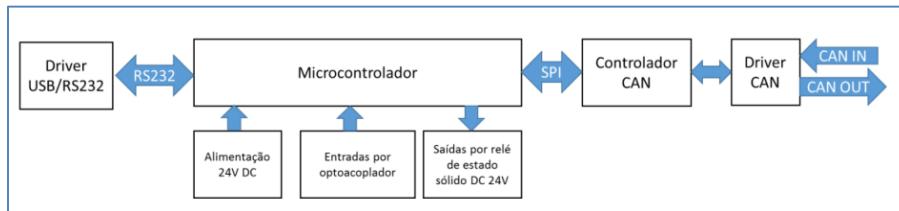


Figura 14 – Diagrama de blocos Módulo Master[2]

O esquemático do Módulo Master está dividido em partes, sendo que no **Power Module** apresentado na Figura 15, está incluída a alimentação, com fonte comutada, com aceitação de tensão de entrada de 5.5VCC a 36VCC, proteção de curto/circuito por fusível (1Amp) e inversão de polaridade pelo diodo BA340. Na entrada, a tensão de alimentação é filtrada pelo conjunto C4 e C12; seguidamente entra no IC TPS543d, que é um conversor por PWM, Canal -N, sem necessidade de transístor de saída, com cargas até 3 Amp [2].

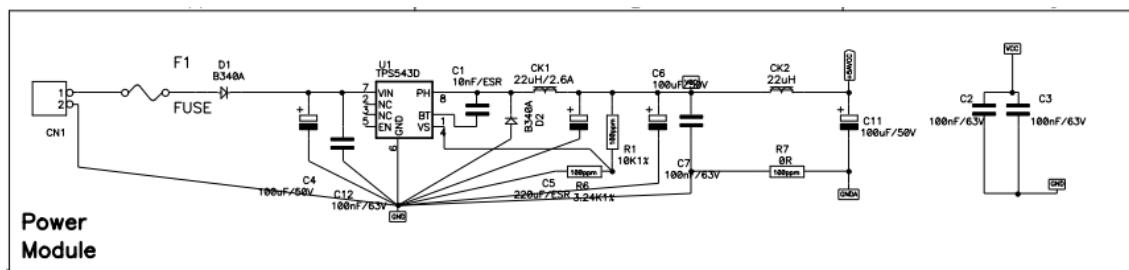


Figura 15 - Power Module – Fonte de alimentação comutada, com entrada de 5.5 V a 36V

Os componentes que fazem parte do resto deste módulo são os necessários para a operação normal deste integrado. Uma atenção especial ao conjunto bobine/díodo/condensador Ck1(22uH) e D2 (BA340a) essenciais na comutação, e condensador C5 (220uF) de baixas perdas (ESR); atenção também ao divisor de tensão, formado pelas resistências R1/R6 responsáveis pela tensão de saída (ao variar esta relação a tensão de saída é diferente) desde que obedeça à fórmula da Figura 16, sendo que para 5 Volts os valores encontrados são R1=10K e R2 (nossa R6) seja igual a 3.24K.

$$R_2 = \frac{R_1 \times 1.221}{V_{OUT} + 1.221}$$

Figura 16 – Cálculo da tensão saída[2]

Como é uma fonte comutada há que ter em conta o *Ripple*, e o máximo esperado, segundo o datasheet seria de 20mVAC, pelo que se faz a apostar num filtro composto pelo condensador C7, choque de 22uH e C11 para reduzir este valor. Uma vez que não são tratados nesta proposta sinais analógicos de grande sensibilidade, este ruído torna-se desprezível. No mesmo módulo, os condensadores C2 e C3, são os condensadores de desacoplamento dos ICs 74hc14.

O módulo **Input Module**, conforme se pode ver na Figura 17, tem a função de fazer as leituras de entrada, com isolamento por foto-acoplador, necessárias conforme especificação do caderno de encargos. Usa o foto-acoplador 4N25 [3], que tem no seu lado de entrada um divisor de tensão que estabelece como máximo 35V e como mínimo 12V, para o 4N25 trabalhar. O díodo 1n4001/7, serve de proteção ao led do 4N25, permitindo assim apenas 0.6V de tensão inversa, entre os pinos 1 e 2. Também é usado um shunt (soldado por defeito no PCB) para que se o cliente quiser a entrada totalmente isolada galvanicamente, o poder fazer abrindo essa ponte. Nas saídas é feito o “ataque” uma porta *schmitt trigger* do IC 74hc14 [4], que tem a função de definir os estados de transição de alto para baixo ou inverso.

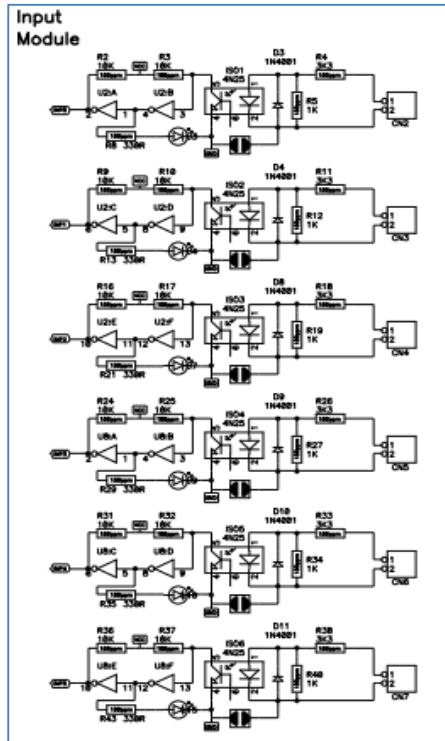


Figura 17 – Input Module. Modulo de entrada isolado por foto-acopladores

Também, tem incluído um circuito de visualização por led do estado de entrada de sinal. Na saída é utilizado também um *schmitt trigger* que está ligado às portas do microcontrolador, como entradas *falling edge* (alto para baixo).

Na Figura 18, está representado o módulo **USB Module**, que tem por missão fazer as comunicações USB/RS232 do PC para o Modulo Master, com o microcontrolador Atmega328. Também, faz a programação do mesmo microcontrolador quando o JMP1 (reset) está na posição pino 1 e 2, via DTR do FT232 [5], sendo que este circuito também é o responsável pela tradução dos protocolos USB/RS232. Os componentes que fazem parte de modulo, são os necessários conforme *datasheet* do fabricante. Uma vez que não é utilizada a tensão de alimentação da ficha Usb, foi colocado um diodo 1n4148, ligado a um led, sendo indicador de que a porta Usb está ligada e a trabalhar.

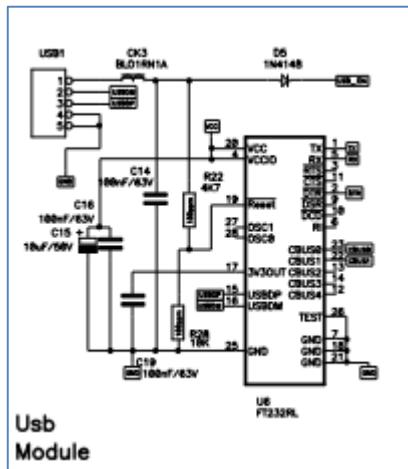


Figura 18 – Usb Module, com FT232rl

O módulo **Reset Module**, representado na Figura 19, tem por missão fazer o reset do microcontrolador (initialização). Parte de um estado baixo para alto (rising edge). Tem um ponto de acesso externo, via JMP3, necessário quando da necessidade de inicializar o sistema, fazendo uma ligação entre pinos de mesmo JMP3.

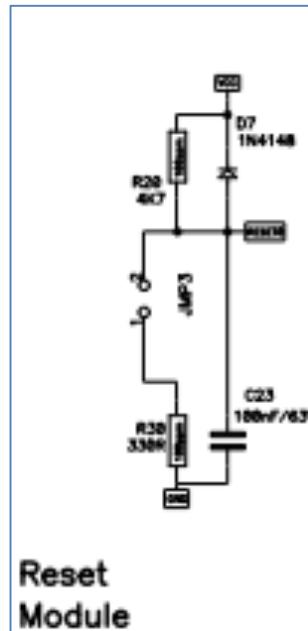


Figura 19 – Reset Module

O **CPU Module**, é a alma do sistema conforme se pode ver na Figura 20. Aqui são tratados todos os sinais e comandos para que o nosso módulo trabalhe segundo as

especificações. Este é um microcontrolador Atmega 328 [6], de 8 bits, flash e recebe via RS232 os comandos do PC. Também recebe as entradas do **Input Module**, e comanda via SPI o **Can Module**. Segundo a lógica de trabalho tem duas saídas PB0 e PB1 a estado alto, e aceita até 6 entradas a estado baixo. Recebe e envia segundo o protocolo RS232 (sinais digitais) nos pinos 2 e 3 (Tx e Rx), para o modulo Usb, para comunicar com o PC. Este trabalha a uma frequência de clock de 16Mhz. A parte analógica do microcontrolador não é utilizada, apesar de ser alimentada com tensão retificada e estabilizada, via fonte de alimentação (5VCC). Quando presente um *interrupt* no pino 4 (INT0), o sistema faz um desvio do seu ciclo normal de trabalho e faz as instruções presentes nas rotinas de CAN do microcontrolador, via SPI, tratando as tramas de envio/recepção do **Can Module**.

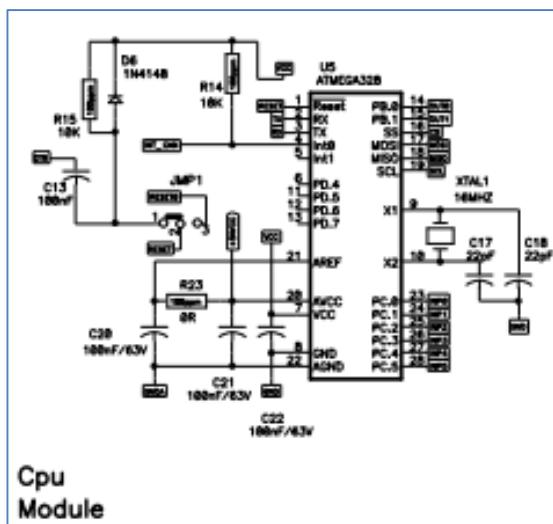


Figura 20 – Cpu Module, com o microcontrolador Atmega328

Na Figura 21, o módulo **Signaling Module** (sinalética), informa visualmente o utilizador de todas as operações que estão a ser executadas pelo Módulo Master, como:

- CBUS0 e CBUS1 – a piscar, indica que há dados a circular na RS232,
- RX0BF e RX1BF – a piscar, indica atividade no MCP2515,
- RX_CAN e TX_CAN – a piscar, indica atividade no Bus do Can,
- INT_CAN – quando o **Can Module**, envia ao microcontrolador um interrupt,
- Power – indica que o Módulo está a trabalhar,
- USB_ON – indica que a porta Usb foi ligada ao Módulo Master

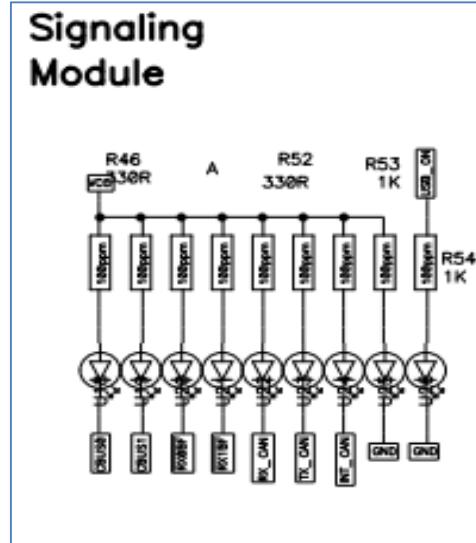


Figura 21 -- Signaling Module

O **CAN Module**, Figura 22, tem um controlador *Stand-Alone* CAN (MCP2515) [7] controlado através da interface SPI do Atmega328 e o regulador dos níveis de tensão para rede CAN através de um driver CAN (MCP2551SN) [8], que envia e recebe sinais do modulo para o barramento CAN. De referir a saída *Int* (*Interrupt*) pino 12 do CPU, para obrigar este a fazer um desvio do seu ciclo normal para as rotinas implementadas para o CAN, nos algoritmos que implementaremos.

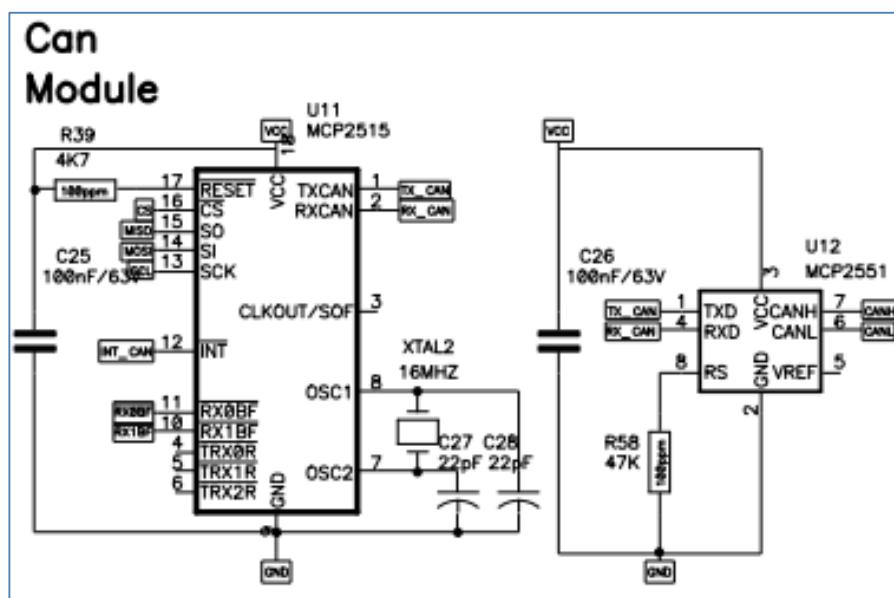


Figura 22 – CAN Module. Conjunto MCP2515 e MCP2551

Relativamente ao **OUT Module**, Quando no algoritmo é desencadeado um sinal de saída, este é transmitido aos pinos 14 e 15 do CPU (PB0 e PB1, respetivamente), com uma saída de um estado baixo para alto (rising edge), que irá ligar à base de um transístor em modo de emissor comum, por intermédio de uma resistência adaptadora de carga de $4.7\text{K}\Omega$. Este modo de trabalho necessita de poucos miliamperes de consumo nas portas do CPU, uma vez que o transístor pode ter um ganho entre 100 e 1000 vezes. Nestas condições, quando nas saídas de PB0/PB1 estiverem a 1 (5V), o transístor entra na zona de saturação, fazendo conduzir a corrente do emissor para o coletor, fechando o circuito, fazendo acender o led ligado ao seu coletor por intermédio de uma resistência redutora de tensão/carga. O mesmo acontece com relé de estado sólido LH1550[9], pois o mesmo está ligado em paralelo. A diferença das resistências tem a haver com o consumo dos respetivos leds ligados nestes circuitos, como o led de sinalização, como o led do relé de estado sólido, conforme se vê na Figura 23.

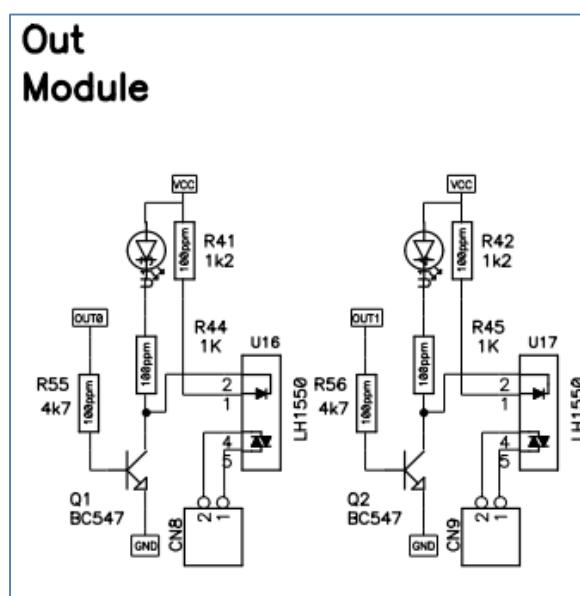


Figura 23 – Out Module. Controlo da potência da placa master

É por meio deste do **CAN OUT Module**, Figura 24, que a placa do Módulo Master se liga ao barramento CAN. Ter em atenção à posição dos condutores, pois um CANH

deve ligar a um CANH e um CANL a um CANL do barramento. Também, não menos importante, é a posição do *shunt* que ligará ou não à resistência de 120Ω do barramento. Num barramento CAN, tem que haver sempre resistências terminadoras, para assim estabilizar o barramento, e estas devem ser colocadas nos seus extremos. Apenas é necessário a utilização de duas de 120Ω .

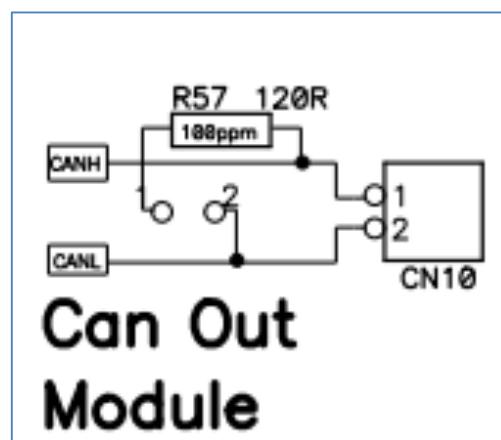
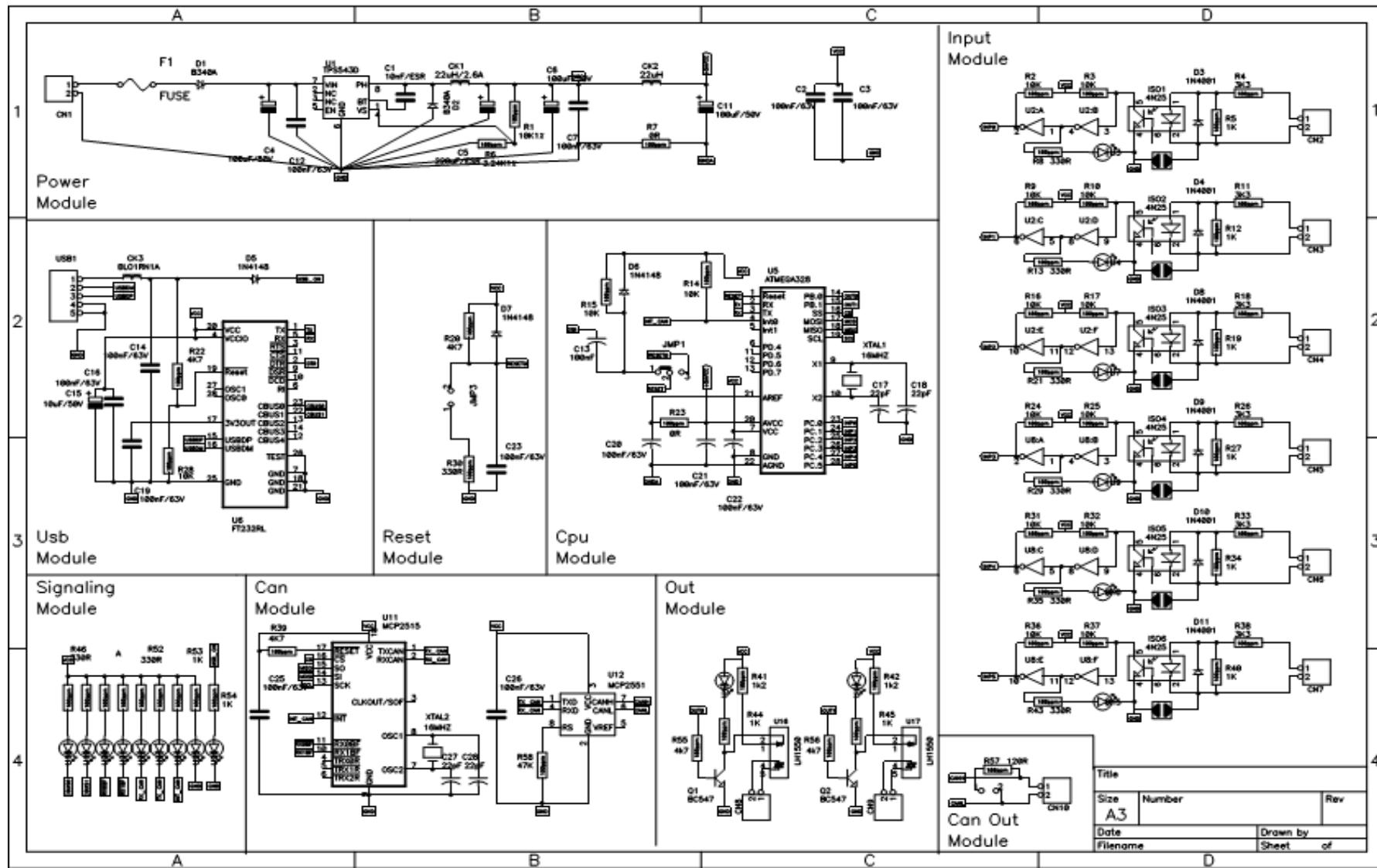


Figura 24 – Can Out Module

Na execução do pcb, utilizamos o software Eagle, versão de demonstração. O circuito impresso foi produzido, seguindo o esquema que se encontra a seguir.



3.2.2 Placa de Circuito impresso

Na página anterior, a versão completa do esquemático que deu origem ao PCB, representados nas Figura 25, Figura 26 e Figura 27.

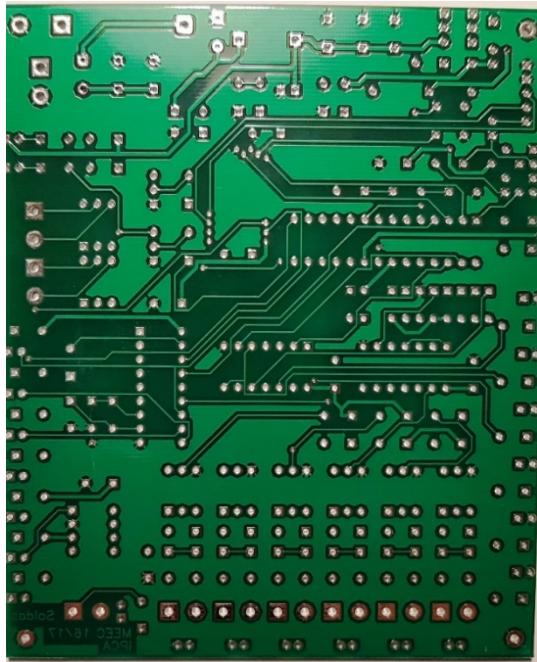


Figura 25 – Master em PCB Bottom Side

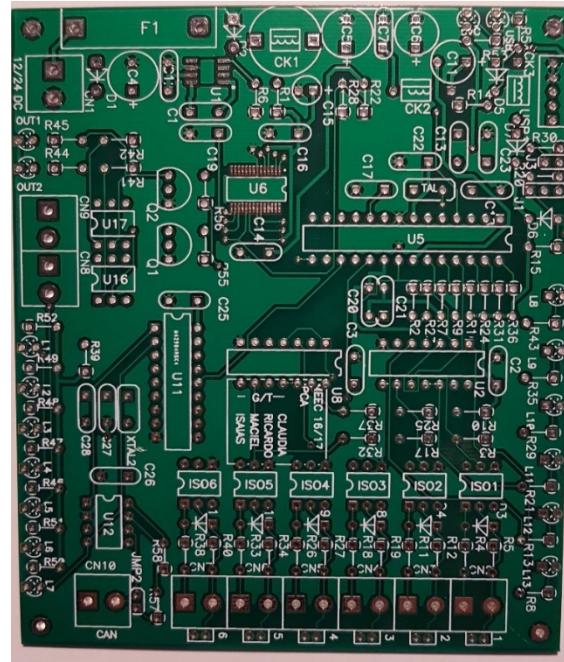


Figura 26 - Master em PCB Top Side



Figura 27 - Master com componentes
soldados em funcionamento

3.3 Comunicação CAN

Realizaram-se os testes de comunicação CAN, para certificarmos que a comunicação é possível. Para isso, colocamos a comunicar em barramento CAN dois Arduínos com o auxílio do MCP2515 e MCP2551, da forma como demonstra a Figura 28.

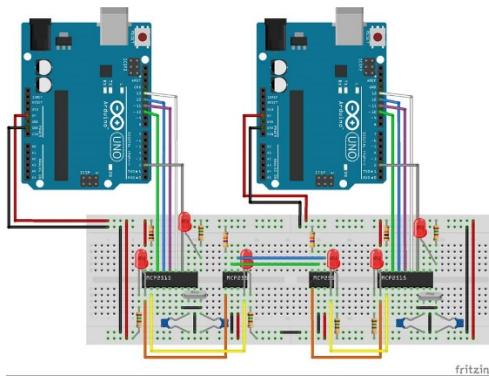


Figura 28 - Teste de comunicação CAN (realizado em fritzing)

No que se refe ao código utilizado, recorremos a uma biblioteca disponível de forma gratuita que tem o nome de *CAN_BUS_Shield-master* [10], que está preparada para iniciar e efetuar a comunicação entre dispositivos. Neste caso, efetuamos um envio de um Arduíno e colocar o segundo a receber.

```
#include <mcp_can.h>
#include <SPI.h>

// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 9;

MCP_CAN CAN(SPI_CS_PIN); // Set CS pin

void setup()
{
    Serial.begin(115200);

    while (CAN_OK != CAN.begin(CAN_500KBPS)) // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println("Init CAN BUS Shield again");
        delay(100);
    }
    Serial.println("CAN BUS Shield init ok!");
}

unsigned char stmp[8] = {1, 2, 3, 4, 5, 6, 7, 8};
void loop()
{
    // send data: id = 0x00, standrad frame, data len = 8, stmp: data buf
    stmp[7] = stmp[7]+1;
    if(stmp[7] == 100)
    {
        stmp[7] = 0;
        stmp[6] = stmp[6] + 1;

        if(stmp[6] == 100)
        {
            stmp[6] = 0;
            stmp[5] = stmp[5] + 1;
        }
    }

    CAN.sendMsgBuf(0x00, 0, 8, stmp);
    delay(100); // send data per 100ms
}
```

Figura 29 – Teste de envio CAN

```

#include <SPI.h>
#include "mcp_can.h"

// the cs pin of the version after v1.1 is default to D9
// v0.9b and v1.0 is default D10
const int SPI_CS_PIN = 9;

MCP_CAN CAN(SPI_CS_PIN); // Set CS pin

void setup()
{
    Serial.begin(115200);

    while (CAN_OK != CAN.begin(CAN_500KBPS)) // init can bus : baudrate = 500k
    {
        Serial.println("CAN BUS Shield init fail");
        Serial.println(" Init CAN BUS Shield again");
        delay(100);
    }
    Serial.println("CAN BUS Shield init ok!");
}

void loop()
{
    unsigned char len = 0;
    unsigned char buf[8];

    if(CAN_MSGAVAIL == CAN.checkReceive()) // check if data coming
    {
        CAN.readMsgBuf(slen, buf); // read data, len: data length, buf: data buf

        unsigned int canId = CAN.getCanId();

        Serial.println("-----");
        Serial.print("Get data from ID: ");
        Serial.println(canId, HEX);

        for(int i = 0; i<len; i++) // print the data
        {
            Serial.print(buf[i], HEX);
            Serial.print("\t");
        }
        Serial.println();
    }
}

```

Figura 30 - Teste de receção CAN

Confirmado ficou o envio e a receção, resultando no esquema final de ligação CAN até ao MCP25050 (não inclusive), faltando a confirmação da comunicação com o controlador de GPIO, o MCP25050.

3.4 MCP25050

Um dos pontos do trabalho era a manipulação do componente MCP25050, visto que é o elemento responsável pelo controlo do *Slave* no que se refere a entradas e saídas e as suas consequentes interrupções.

No que se refere à programação, utilizamos o *Pickit 2* para a transferência do ficheiro. HEX para o MCP25050. O esquema de ligações é demonstrado na Figura 31.

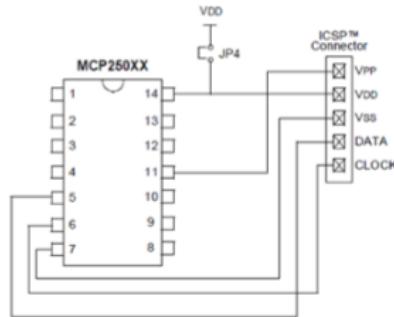


Figura 31 – Esquema de programação MCP25050

Para a criação do ficheiro HEX, utilizou-se o *MCP250xxProgrammer.exe*, em que se criou a configuração dos registos. Na interface inicial da aplicação, definiu-se a frequência de funcionamento, que neste caso é de 20MHz, esta parte é representada na Figura 32.

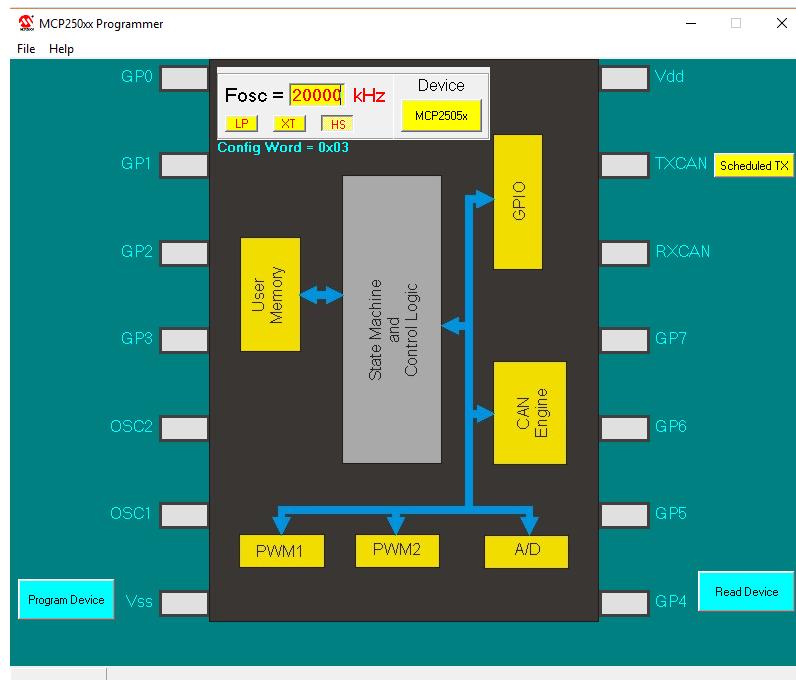


Figura 32 – Interface inicial do MCP250xxProgrammer.exe

De acordo com o estudo realizado e consequente esquemático idealizado, definiu-se as entradas e saídas no MCP25050. É utilizado como entradas os GPIOs de zero a cinco, e entradas os GPIOs seis e sete. A configuração é representada na Figura 33.

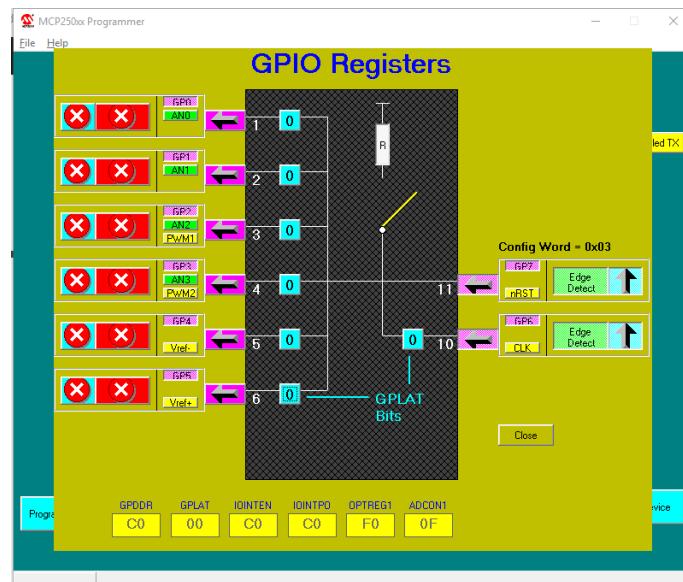


Figura 33 – Representação da configuração do GPIO no MCP25050

Seguidamente definiu-se a configuração dos registos, em que foi indicado o ID de leitura e escrita do MCP, foi escolhido o 0x680 para aplicar configurações e ler dos registos, e o 0x600 para o envio de mensagens que necessitem de resposta por parte do integrado.

Foi programado o TXID1 e TXID2 para 0x600 e 0x680 correspondente. Desta forma, quando é criada uma interrupção recebemos o valor de 0x680.

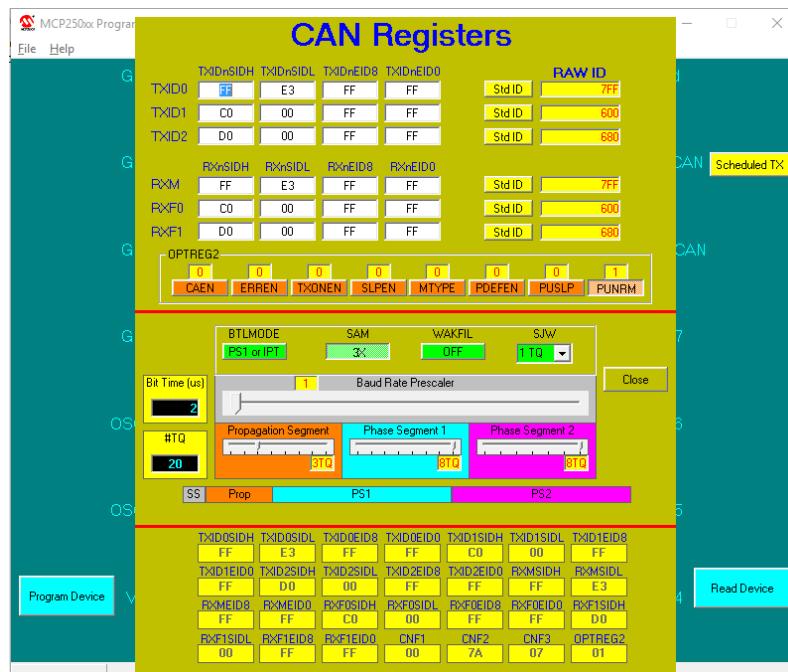


Figura 34 – CAN Registers MCP25050

Após a configuração, enviou-se o ficheiro HEX para o MCP25050, o programa para o envio foi o *PICkit(R) 2 Microcontroller Programmer*.

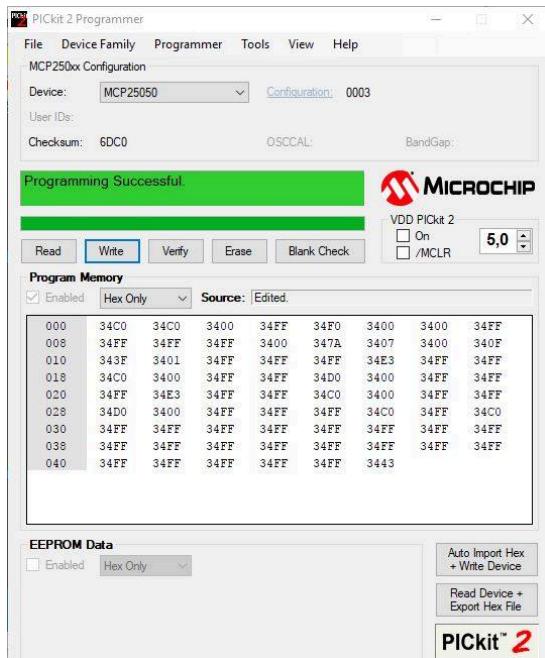


Figura 35 – Programação do MCP25050

A ligação física com o integrado utilizada está representado na Figura 31.

3.5 Módulo de Comunicação PC <-> Módulo Master CAN (Software)

De forma a desenvolver o código para a comunicação entre o PC e o módulo master foi necessário desenvolver um protocolo para envio de troca de mensagens pela série.

Esta comunicação foi realizada através da plataforma *Johnny-Five*, que foi criada para funcionar como uma “JavaScript Robotics & IoT Platform”. Basicamente, faz o mesmo que o IDE do Arduíno para comunicar através de *SerialPort*, mas com o JavaScript.

O Atmega328 é programado com uma biblioteca chamada “Firmata”, que faz o reconhecimento das funções usadas pela plataforma *Johnny-Five*, para realizarmos a ativação das portas de saída deste e interpretar os dados recebidos pela *SerialPort*.

Pretendemos, assim, utilizar um protocolo, no qual enviamos mensagens do PC para o módulo *Master*.

Este protocolo, até à data está em fase de testes e, portanto, ainda está a ser desenvolvido, não sendo assim apresentado no relatório.

3.6 Módulo de Comunicação Módulo Master CAN <-> Módulo de expansão I/O CAN

Na programação, pretendemos utilizar bibliotecas CAN que facilitam a leitura e envio de mensagens para o barramento CAN. Assim, para enviar as mensagens para a rede CAN, através do MCP2515, envia-se tramas de acordo com o protocolo CAN. Pretendemos utilizar os registos GPADDR (0x1F) e o GPLAT (0x02+0x1C de offset).

REGISTER 5-1: GPDDR - DATA DIRECTION REGISTER							
U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	DDR6	DDR5	DDR4	DDR3	DDR2	DDR1	DDR0
bit 7							bit 0
bit 7 Unimplemented: Read as '0'							
bit 6-0 DDR6:DDR0: Data Direction Register* bits							
1 = corresponding GPIO pin is configured as an input							
0 = corresponding GPIO pin is configured as an output							
* must be set if corresponding analog channel is enabled (see ADCON1)							
Legend:							
R = Readable bit		W = Writable bit		U = Unimplemented bit, read as '0'			
- n = Value at POR		'1' = Bit is set		'0' = Bit is cleared		x = Bit is unknown	

REGISTER 5-2: GPLAT - GPIO OUTPUT REGISTER							
U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	GP6	GP5	GP4	GP3	GP2	GP1	GP0
bit 7							bit 0
bit 7 Unimplemented: Read as '0'							
bit 6-0 GP6:GP0: GPIO Bits							
1 = corresponding GPIO pin output latch is a '1'							
0 = corresponding GPIO pin output latch is a '0'							
Legend:							
R = Readable bit		W = Writable bit		U = Unimplemented bit, read as '0'			
- n = Value at POR		'1' = Bit is set		'0' = Bit is cleared		x = Bit is unknown	

Figura 36 – Registos do MCP25050

Na programação do ATMEGA328 pretendemos utilizar uma trama que recebe como parâmetros (ID,Standard/Extended Frame, Tamanho da Data, Data, RTR). Assim, quando queremos comunicar com o expansor, simplesmente basta enviar o ID x680 (RTR=0) ou x600 (caso seja para pedir dados RTR =1), o tamanho da mensagem que queremos enviar, e a própria mensagem (buffer) que indica qual o registo a aceder e

o valor que queremos escrever utilizando máscaras. Desta forma, conseguimos ativar as saídas do expensor com facilidade.

3.7 Software

Em termos *Software* e de organização do nosso interface, apresenta-se a estrutura da Figura 37.

Na pasta “*components*” temos 4 ficheiros para criação e configuração do *layout.js*, *header.js*, *footer.js* e *training-overlay.js* da nossa interface. Estes ficheiros importam componentes do *react*, *react-redux*, *semantic-ui-react* e *next/link*.

Na pasta “*data*” temos dados/ficheiros associados à deteção das faces, às imagens geradas para cada utilizador e ao treino para reconhecimento das faces (*faces.yml*). Nesta pasta está também o ficheiro *users.json* que é a base de dados de utilizadores com as características individuais de cada utilizador.

A pasta “*pages*”, apresentam-se quatro ficheiros *_document.js*, *index.js*, *profile.js*, *select-profile.js* responsáveis pelas páginas/layouts da interface.

O ficheiro *server.js* é o servidor responsável pelo arranque da nossa aplicação utilizando o browser GoogleChrome em localhost:3000.

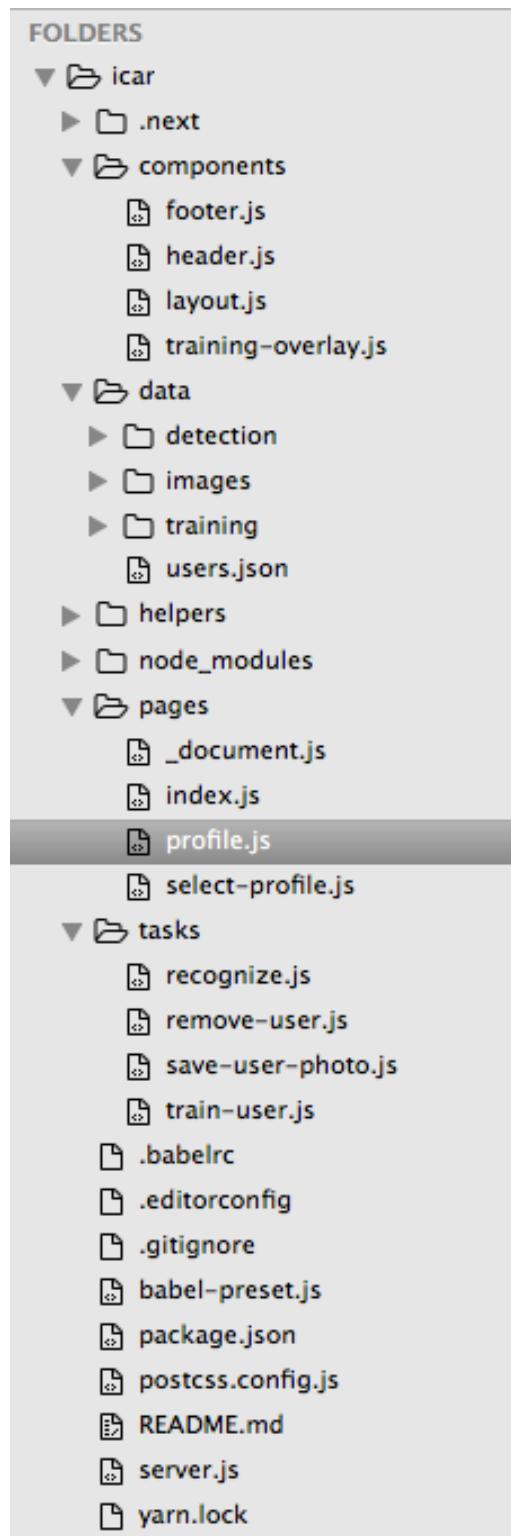


Figura 37 – Organização do projeto

3.7.1 Criação do Servidor Web com Express

O NodeJs é um interpretador de código *JavaScript* que funciona do lado do servidor. Esta plataforma permite aos programadores o desenvolvimento de aplicações em rede, em tempo real e de alta escalabilidade, de uma forma simples e rápida. O NodeJs é baseado no interpretador V8 da Google.

Antes do NodeJs, o *Javascript* era apenas utilizado no browser e para programarmos no lado do servidor teríamos que utilizar outra linguagem como o PHP.

Isto obrigava que um programador Web precisasse ter conhecimentos em HTML, CSS, *Javascript* e uma outra linguagem. O NodeJs veio mudar isso oferecendo-nos a possibilidade de escrever servidores Web utilizando o *javascript* e graças ao modelo não bloqueante obter uma boa performance. O NodeJs está disponível para várias plataformas. Como editor de texto usou-se o poderoso *Sublime Text*.

3.7.1.1 Estrutura do Projeto

A estrutura do projeto é bastante simples, como já observado na figura anterior. Criou-se uma pasta principal “iCar” e, dentro dessa pasta, colocou-se a nossa aplicação “server.js” e todo o conteúdo do nosso site, ficheiros HTML, CSS.

Procurou-se um *template* gratuito que fosse responsivo e com o aspeto pretendido para o nosso site. Um site com design responsivo é um site que se adapta a vários tamanhos de ecrã seja ele *Desktop*, *Tablet* ou *Smartphone*.

O *framework* mais popular que nos permite fazer isso chama-se *bootstrap* e foi criado pelo *Twitter* para facilitar o desenvolvimento de páginas web, dando-lhes um aspetto uniforme. São boas fontes de pesquisa para este tipo de *templates* os sites *startbootstrap* ou o *shapebootstrap*. Depois de encontrado o *template*, foram realizadas alterações conforme o pretendido e copiados todos os ficheiros e pastas para a pasta “iCar”. A seguir exemplifica-se o desenvolvimento do servidor no ficheiro “server.js”.

Em vez de reunir todos os componentes diretamente no node.js, optou-se por permitir adicionar funcionalidades ao node.js através de módulos. Logo no início da aplicação incluíram-se alguns módulos a utilizar como se observa na Figura 41.



```
server.js *  
const path = require('path')  
const next = require('next')  
const express = require('express')()  
const server = require('http').Server(express)  
const io = require('socket.io')(server)  
const five = require('johnny-five')
```

Figura 38– Módulos a utilizar

Na terceira linha do excerto de código exemplificado acima, temos a importação de um módulo com o nome *express*, que não instalamos no projeto. Quando instalamos o node.js temos disponível um gestor de pacotes chamado npm (*Node Package Manager*) e através dele, torna-se muito simples instalar as dependências do nosso projeto. Antes de instalar e definir o *express* como dependência do nosso projeto, criou-se um ficheiro chamado *package.json* na raiz do nosso projeto que terá algumas informações do nosso projeto como: Nome; Versão; Descrição; Repositório git remoto; Autores; Dependências e outras coisas.

Para criar o *package.json* utilizou-se o npm com a abertura do terminal e executando o seguinte comando (deve ser feito dentro da pasta “iCar”) como observado na Figura 39.



```
[MBP-de-Claudia:~ claudiaferreira$ cd /Users/claudiaferreira/GitHub/Projeto_lab_int_II/iCar  
[MBP-de-Claudia:iCar claudiaferreira$ npm init] ever servidores Web utilizando o javascript e graças ao modelo
```

Figura 39– Criação do ficheiro *package.json*

Após a execução deste comando o npm faz algumas perguntas sobre o projeto bastando pressionar a tecla ENTER.

Com o ficheiro *package.json* criado, é necessário instalar o *express* e defini-lo como dependência. Novamente no terminal é necessário executar:

MBP-de-Claudia:iCar claudiaferreira\$ npm install express –save

Através desta linha de comando executada no terminal fez-se o download do express para o nosso projeto e, por causa do parâmetro –save, foi também definido como dependência. Abrindo o package.json teremos uma nova chave chamada *dependencies* com o *express* como valor.

Após este processo, basta irmos até o terminal e executar o comando para a nossa plataforma NodeJs executar o server.js:

MBP-de-Claudia:iCar claudiaferreira\$ node server.js

Este comando tem que ser executado dentro da pasta iCar.

Depois de executado o comando acima receberemos a seguinte saída no terminal:

Ready on <http://localhost:3000>

Recorrendo ao browser *GoogleChrome*, e colocando <http://localhost:3000>, aparecerá um erro porque, o *express* usa rotas que não foram ainda definidas. Na Figura 40 observa-se a criação destas rotas com o método GET.

```
app.prepare().then(() => {
  express.get('/users', (req, res) => {
    const data = require(path.resolve(__dirname, 'data/users.json'))

    return res.json(data)
  })

  express.get('*', (req, res) => nextHandler(req, res))

  server.listen(3000, err => {
    if (err) throw err

    console.log('> Ready on http://localhost:3000')
  })
})
```

Figura 40– Criação do ficheiro package.json

Através da toda a descrição do processo, criou-se assim um servidor HTTP com o *ExpressJS*.

3.8 Módulo de processamento de imagem

3.8.1 Identificação e reconhecimento de faces

Para o módulo de processamento de imagem foram criadas várias “tarefas” em vários ficheiros, para a identificação e processamento das faces. Os ficheiros são: `recognize.js`; `save-user-photo.js`, `train-user.js` e `remove-user.js`. Em termos de base de dados, foi criado um ficheiro no formato JSON dado que a quantidade de dados a guardar é residual.

Relativamente à parte do processamento, foi necessário começar por capturar e guardar uma sequência de imagens do utilizador. O ficheiro `save-user-photo.js` inicia esse processo. O código implementado utiliza módulos de processamento de imagem do OpenCV (*Open Source Computer Vision Library*) e guarda numa pasta `../data/images/user1` (para o primeiro utilizador), as imagens correspondentes a esse utilizador. Essas imagens serão utilizadas posteriormente no treino para reconhecimento da face.

A deteção de faces em imagens digitais é um problema relevante e atual e como, uma biblioteca bastante robusta, o *OpenCV* tem um algoritmo para isso: trata-se do algoritmo de Viola-Jones para a deteção de objetos (também popularmente chamado de *Haar Cascade* ou apenas *Cascade*). Este classificador, utiliza essencialmente um modelo previamente treinado para detetar um tipo específico de objeto de interesse (no nosso caso, uma face humana). Para encontrar caras/faces na imagem foi utilizado o classificador `cv.FACE CASCADE`, existindo também classificadores para detetar olhos, sorriso, etc.

As imagens a guardar serão em tons de cinzento `cvtColor('CV_BGR2GRAY')` e é aplicada a função `equalizeHist()`.

A equalização do histograma é boa quando o histograma da imagem é confinado a uma região específica. Não funcionará bem em locais onde existam grandes variações de intensidade.

Após a captura das imagens é necessário proceder ao treino para reconhecimento da face, que é realizado recorrendo ao ficheiro *train-user.js*.

Neste ficheiro recorreu-se novamente ao *OpenCV*, uma vez que todos os modelos de reconhecimento de face no *OpenCV* são derivados da classe base *FaceRecognizer*, que fornece acesso a todos os algoritmos de reconhecimento de faces.

Além disso, a classe *FaceRecognizer* [11] oferece suporte para:

- ✓ Treino de um *FaceRecognizer* com um determinado conjunto de imagens (a nossa base de dados de rostos!).
- ✓ Previsão de uma imagem de amostra dada, ou seja, uma face.
- ✓ Carregamento e gravação do estado do modelo de/para um ficheiro XML ou YML.
- ✓ Configuração/Obtenção de informações de rótulos, que é armazenado como uma *string*. Estas informações são úteis para manter os nomes das pessoas reconhecidas.

O *OpenCV* fornece três métodos de reconhecimento de rostos:

- ✓ *Eigenfaces*
- ✓ *Fisherfaces*
- ✓ *Local Binary Patterns Histograms (LBPH)*

Os três métodos realizam o reconhecimento, comparando o rosto a ser reconhecido com um conjunto de treino de faces conhecidas. No conjunto de treino, fornecemos os rostos e identificamos a que pessoa correspondem. Quando o algoritmo é solicitado a reconhecer algum rosto desconhecido, usa o conjunto de treino para fazer o reconhecimento. Cada um dos três métodos acima mencionados usa uma forma de treino um pouco diferente.

Os métodos *Eigenfaces* e *Fisherfaces* encontram uma descrição matemática das características mais dominantes do conjunto de treino como um todo. O método LBPH analisa, separadamente e de forma independente, cada face no conjunto de treino.

Com o método LBPH, cada imagem é analisada de forma independente, enquanto o método de *Eigenfaces* analisa o conjunto de dados como um todo. O método LBPH é

um pouco mais simples, no sentido de que caraterizamos cada imagem no conjunto de dados localmente e quando uma nova imagem desconhecida é fornecida, realizamos a mesma análise e comparamos o resultado com cada uma das imagens no conjunto de dados.

Na Figura 41 apresenta-se o diagrama de herança de `cv::face::FaceRecognizer` [12] utilizado pelo *OpenCV*.

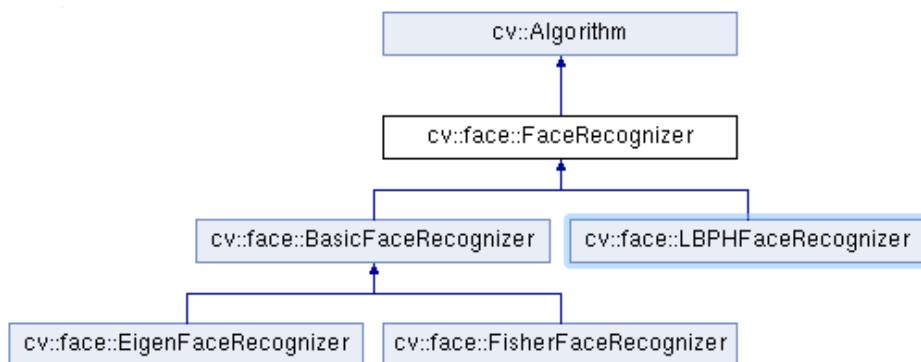
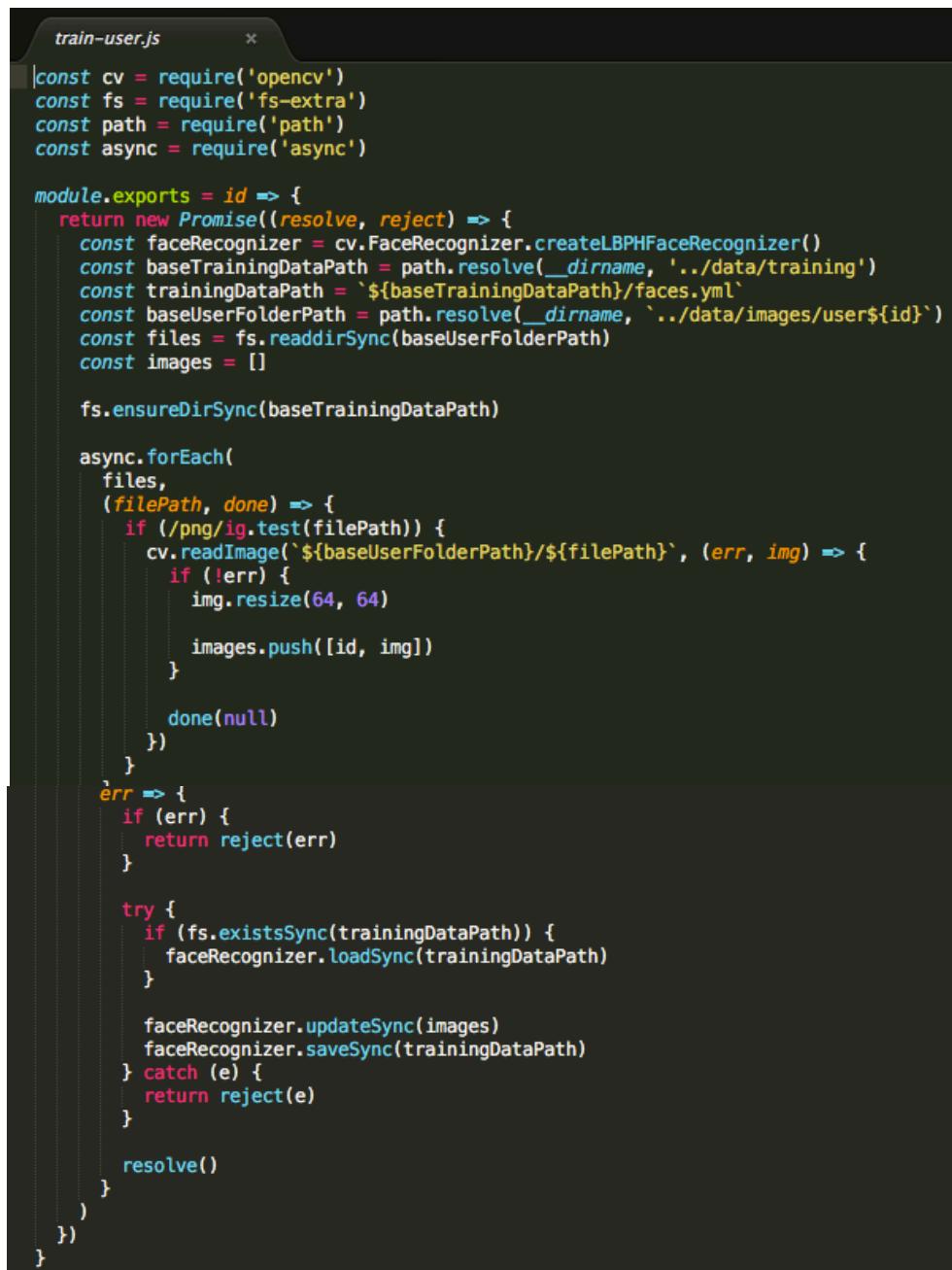


Figura 41 - Diagrama de herança cv::face::FaceRecognizer[12].

Observados os três métodos, optamos por usar o método LBPH por analisar cada imagem de forma independente e, provavelmente, funcionará melhor em diferentes ambientes e condições de luz, no entanto, dependerá também dos dados de treino e teste.

A Figura 42 apresenta como irá ser executado o treino para reconhecimento da face. Como já referido acima, utilizou-se o método `cv.FaceRecognizer.createLBPHFaceRecognizer()`.

No processo irá ler e redimensionar as imagens já colocadas na pasta User1 (para o primeiro utilizador) e cria um *array* de imagens. Após isto, irá realizar a sincronização e carregamento para “`trainingDataPath = `${baseTrainingDataPath}/faces.yml``”. O array de images é atualizado e os dados guardados em `faces.yml`.



```

train-user.js      *
const cv = require('opencv')
const fs = require('fs-extra')
const path = require('path')
const async = require('async')

module.exports = id => {
  return new Promise((resolve, reject) => {
    const faceRecognizer = cv.FaceRecognizer.createLBPHFaceRecognizer()
    const baseTrainingDataPath = path.resolve(__dirname, '../data/training')
    const trainingDataPath = `${baseTrainingDataPath}/faces.yml`
    const baseUserFolderPath = path.resolve(__dirname, '../data/images/user${id}`)
    const files = fs.readdirSync(baseUserFolderPath)
    const images = []

    fs.ensureDirSync(baseTrainingDataPath)

    async.forEach(
      files,
      (filePath, done) => {
        if (/png|ig/.test(filePath)) {
          cv.readImage(` ${baseUserFolderPath}/${filePath}`, (err, img) => {
            if (!err) {
              img.resize(64, 64)

              images.push([id, img])
            }

            done(null)
          })
        }
      },
      err => {
        if (err) {
          return reject(err)
        }

        try {
          if (fs.existsSync(trainingDataPath)) {
            faceRecognizer.loadSync(trainingDataPath)
          }

          faceRecognizer.updateSync(images)
          faceRecognizer.saveSync(trainingDataPath)
        } catch (e) {
          return reject(e)
        }

        resolve()
      }
    )
  })
}

```

Figura 42 - Código do ficheiro *train-user.js*

Após a fase de treino segue-se a fase do reconhecimento do utilizador/condutor do veículo. Parte deste processo está implementado no ficheiro *recognize.js*. Neste ficheiro estão implementadas funções, nomeadamente *detectFaces()* e *recognizeFace()* que auxiliarão no processo de deteção e reconhecimento de faces.

3.9 Módulo de interface com o utilizador

As figuras que se seguem, Figura 43 à Figura 49 demonstram o funcionamento da deteção de face e caracterização da pessoa. Na fase inicial do projeto procedeu-se à planificação e desenho dos mesmos, recorrendo à app *SketchApp* (<https://www.sketchapp.com/>) com elementos UI (*User Interface*) de uma libraria chamada *Semantic UI* (<https://semantic-ui.com/>) que será a framework a utilizar no desenvolvimento da aplicação, tendo componentes já feitos em termos de código e design.

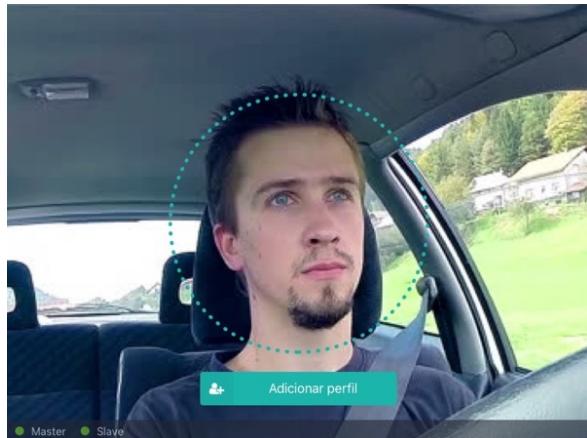


Figura 43 - Ecrã principal

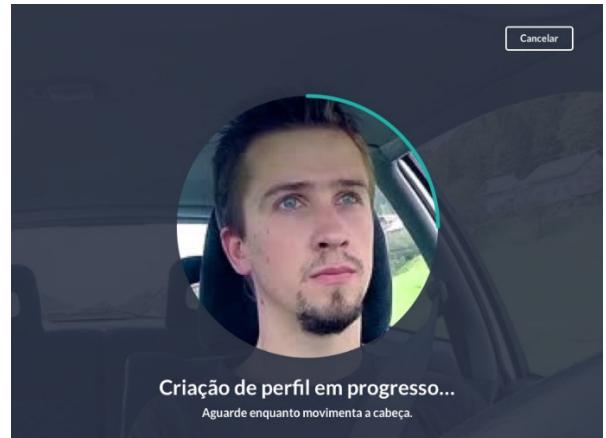


Figura 44 - Ecrã Criação de perfil inicial

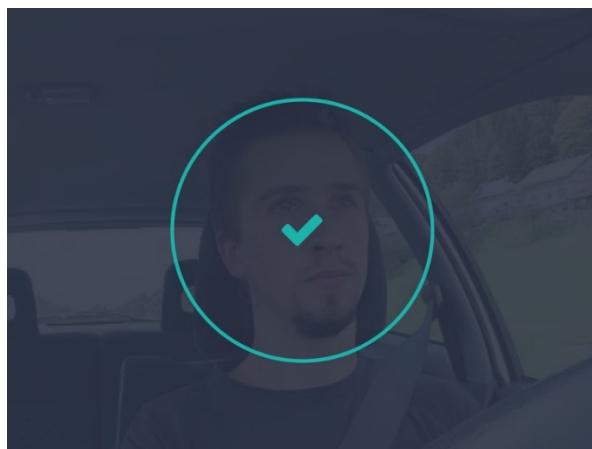


Figura 45 - Ecrã de sucesso da tarefa



Figura 46 - Ecrã Adicionar Perfil

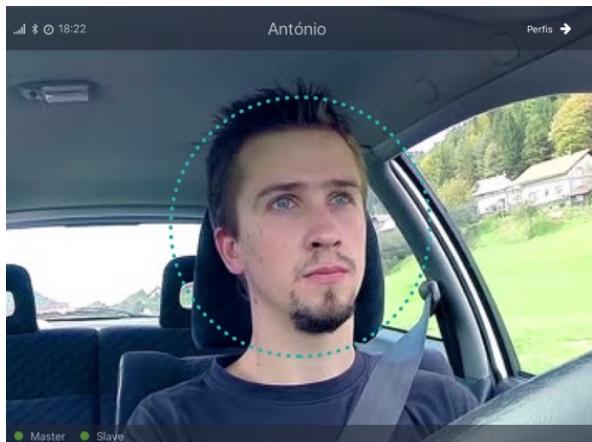


Figura 47 - Ecrã do utilizador

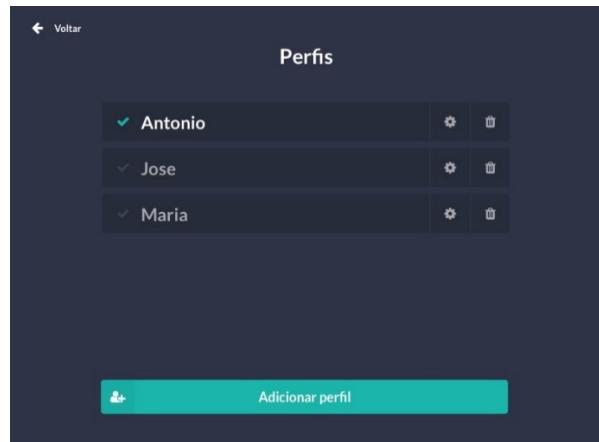


Figura 48 - Ecrã de perfis de utilizador



Figura 49 – Ecrã de remoção de perfil

Através das figuras representadas acima, podemos verificar que a interface indica o estado da ligação aos Módulos *Master* e *Slave*; lista todos os módulos ligados à rede CAN e a imagem aparece em tempo real no ecrã do PC. A deteção de face é assinalada na imagem através do círculo tracejado bem como a identificação da face, que aparece no *header*.

A interface permite ainda, a criação, edição e remoção de registos de faces/utilizadores na base de dados e para cada face é possível configurar quais as saídas de quais módulos de expansão I/O serão ativados/desativados. A interface permite a receção de pedidos de criação de novo registo de face/utilizador na base de dados provenientes do Módulo Master, com a face identificada no momento.

O ficheiro profile.js que terá como resultado a Figura 46, onde o utilizador colocará a sua identificação, ativará e configurará as saídas para o *Slave* (como exemplo colocamos Posição do banco, Posição dos espelhos e Posição do volante) consoante as suas preferências. Após estas configurações, a interface irá para a Figura 47, o ecrã principal que assinala em tempo real a face e coloca o nome do utilizador no cabeçalho da imagem. O utilizador tem disponível um botão para acesso aos perfis já existentes na base de dados. Ao clicar nesse botão será redirecionado para o ecrã da Figura 48. Neste ecrã poderá editar e atualizar os seus dados/características e remover um perfil não desejado. Na Figura 49, a remoção de utilizador do sistema, é realizada recorrendo ao ficheiro *select-profile.js*.

O ficheiro index.js, representa a página principal da interface. Utilizou-se *Canvas*, o que permitiu incorporar o conteúdo de vídeo capturado pela câmara web do nosso computador, dentro de uma página. A câmara captura o vídeo que é projetado no ecrã. Numa configuração mais fácil, pode ser fornecido um vídeo que é automaticamente ajustado às dimensões do ecrã da página e está pronto a ser usado. Um excerto de código implementado na nossa interface e referente a este processo está representado na Figura 50. Nesta fase, também obtemos o *stream* da câmara e mediante o que está a acontecer no vídeo, é iniciada a deteção de utilizador ou o treino de um novo utilizador.

```
this.videoContext = this.videoCanvas.getContext('2d')
this.overlayContext = this.overlayCanvas.getContext('2d')
this.overlayContext.strokeStyle = 'rgb(0, 184, 174)'
this.overlayContext.lineWidth = 4
this.overlayContext.lineCap = 'round'
this.overlayContext.setLineDash([2, 15])

window.socket.on('faceDetected', this.handleFaceDetected)
window.socket.on('userTrained', this.handleUserTrained)
window.socket.on('start-training', () => Router.push('/?training=1'))

if (this.video && navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {
  navigator.mediaDevices.getUserMedia({ video: true }).then(stream => {
    this.cameraStream = stream
    this.video.src = window.URL.createObjectURL(this.cameraStream)

    this.video.play().then(() => {
      if (!isTraining) {
        this.startDetecting()
      } else {
        this.trainNewUser()
      }
    })
  })
}
```

Figura 50 – Excerto de código com aplicação do videoCanvas

4 Testes

Na realização dos testes dos comandos CAN, foram realizadas várias experiências, tendo obtido sucesso na comunicação do PC com o Módulo Master, tal como demostramos na figura 51.

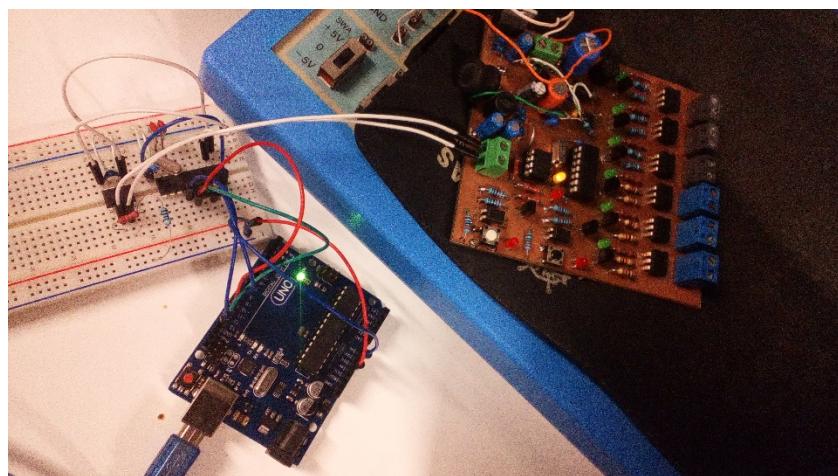


Figura 51 – Output MCP25050

Isto foi realizado, com um script com o IDE do Arduino que faz um SCAN aos IDs que estão presentes na rede CAN.

Para os testes de ligação PC e Master com o software desenvolvido, foi utilizada a framework *Johnny-Five*, que se trata de uma plataforma *IoT & JavaScript Robotics*, tal como referido anteriormente. Para a realização deste teste, procedeu-se à ligação de um teclado ao Master e utilizaram-se apenas 2 botões, que teriam como tarefas, o botão 1 iniciar a criação de um novo perfil/utilizador e o botão 2, simular o estado do *Slave* (ligado/desligado). O estado do Master (ligado/desligado) também foi igualmente testado. A Figura 52 apresenta um excerto de código criado para obter os resultados referidos.

```

board.on('ready', () => {
  const startTrainingButton = new five.Button({ pin: 14, invert: true })
  const slaveButton = new five.Button({ pin: 15, invert: true })

  io.on('connection', socket => {
    if (isConnected) {
      return
    }

    isConnected = true

    startTrainingButton.on('press', () => socket.emit('start-training'))

    board.on('close', () => socket.emit('connection-status', { type: 'master', status: false }))
    slaveButton.on('press', () => socket.emit('connection-status', { type: 'slave', status: false }))
    slaveButton.on('release', () => socket.emit('connection-status', { type: 'slave', status: true }))

    socket.on('recognizeFace', imageData => (
      faceRecognizer.recognizeFace(imageData)
        .then(data => {
          const { id } = data
          const user = DB.getUserById(id)

          socket.emit('faceDetected', user ? Object.assign({}, data, user) : { position: data.position })
        })
    ))

    socket.on('saveUserPhoto', (imageData, id, fileName) => saveUserPhoto(imageData, id, fileName))
    socket.on('removeUser', id => removeUser(id))
    socket.on('trainUser', id => trainUser(id).then(() => socket.emit('userTrained', id)))
    socket.on('saveUserSettings', (id, data) => DB.addOrUpdateUser(id, data))

    socket.on('update-settings', (settings) => {
      if (isUpdatingSettings) {
        return
      }
    })
  })
})

```

Figura 52 – Excerto de código de ligação ao Master

Como o módulo Master apresentava 2 saídas, foi também possível simular nestas saídas, as características do utilizador (posição do banco e posição dos espelhos). Esta simulação consistiu em fazer piscar os leds 8 e 9, das respetivas portas PB0 e PB1 do microcontrolador. Mediante o valor assumido na característica (0, 50, 100) o led piscaria o número de vezes por nós definido. A Figura 53 apresenta o código criado para o efeito.

```

const delay = 500
const seatLed = new five.Led(8)
const mirrorsLed = new five.Led(9)
const seatValue = { 0: 2, 50: 4, 100: 8 }[settings.seat.value];
const mirrorsValue = { 0: 2, 50: 4, 100: 8 }[settings.mirrors.value];

clearTimeout(seatLedBlinkTimout)
clearTimeout(mirrorsLedBlinkTimout)

seatLed.on().blink(delay / 2)
mirrorsLed.on().blink(delay / 2)

seatLedBlinkTimout = setTimeout(() => {
  seatLed.stop().off()

  if (delay * seatValue > delay * mirrorsValue) {
    isUpdatingSettings = false
  }
}, delay * seatValue + 50)

mirrorsLedBlinkTimout = setTimeout(() => {
  mirrorsLed.stop().off()

  if (delay * mirrorsValue > delay * seatValue) {
    isUpdatingSettings = false
  }
}, delay * mirrorsValue + 50)
}

```

Figura 53 – Excerto de código de simulação de saídas

No que se refere ao projeto, está a faltar unicamente a comunicação do *Master* com o *Slave*. Como a aplicação desenvolvida a nível de software não consegue enviar tramas do computador para o *Master* e consecutivamente enviar por CAN para o *Slave*, efetuamos a montagem de ligação à parte, de forma a implementar uma comunicação precisa.

Um dos pontos do enunciado proposto é a programação do MCP25050, como foi executado anteriormente e explicado neste presente relatório. Mas o funcionamento não se concretizou, os motivos poderão ser diversos, mas de acordo com os testes realizados, verificou-se estes como principais:

- O PicKit 2 fornecido pela escola poderá não estar a funcionar de forma correta e assim a transferência do ficheiro HEX não estar a ser enviado corretamente.
- As bibliotecas de comunicação com o MCP25050 foram atualizadas, e verificamos que a função de envio por CAN só aceita 4 argumentos, faltando um para indicar se pretendemos ativar o pino CS ou não.

Para colmatar este problema, e provar que os conceitos teóricos foram estudados foi proposto pelo professor colocar o Arduíno e um MCP2515 no lugar no MCP25050.

Dessa forma chegou-se ao exposto na Figura 54:

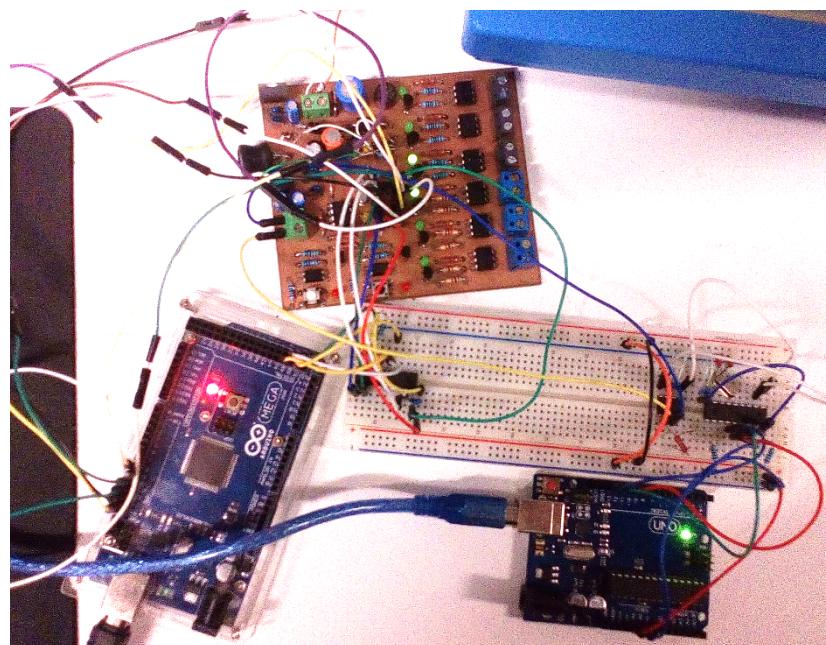


Figura 54 - Testes CAN.

Neste esquema, temos um Arduíno mega no lugar do MCP, e um Arduíno Uno a simular o *Master*, fazendo a comunicação CAN para o *Slave*.

No que se refere à programação, definimos a trama e uma forma intuitiva. O *Slave* passou a ter o ID 0x70, e o *Master* 0x112, ambos os ID's são *Standard*.

Já na parte dos dados, foi dado um tamanho de 8 *Bytes*, em que cada *byte* indica o estado de uma entrada/saída.

No início do projeto, sobre o conceito de máquina, definimos que cada saída do *Slave* irá corresponder ao estado (posições) dos espelhos e banco do veículo.

Os primeiros três dão a informação do estado do espelho, em que o primeiro *byte* corresponde a zero, o segundo a 50 e o terceiro a cem por cento. Os próximos três *bytes* indicam o mesmo mas para as posições do banco. Já por fim os últimos dois indicam o estado das entradas do *Slave*, tal como se apresenta na Figura 55.

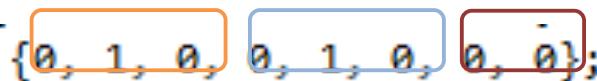


Figura 55 - Trama CAN, parte dos dados.

No que se refere à programação do *Master*, definiu-se as combinações possíveis. E de acordo com o que recebe da porta serie do PC, envia a trama correspondente. Desta forma é possível simular o envio de dados de um computador com o processamento de imagem para o CAN (Figura 56 e Figura 57).

```

unsigned char refresh[8] = {0, 0, 0, 0, 0, 0, 0, 0};
unsigned char esp0_banco100[8] = {1, 0, 0, 0, 0, 1, 0, 0};
unsigned char esp50_banco100[8] = {0, 1, 0, 0, 0, 1, 0, 0};
unsigned char esp100_banco100[8] = {0, 0, 1, 0, 0, 1, 0, 0};
unsigned char esp0_banco50[8] = {1, 0, 0, 0, 1, 0, 0, 0};
unsigned char esp50_banco50[8] = {0, 1, 0, 0, 1, 0, 0, 0};
unsigned char esp100_banco50[8] = {0, 0, 1, 0, 1, 0, 0, 0};
unsigned char esp0_banco0[8] = {1, 0, 0, 1, 0, 0, 0, 0};
unsigned char esp50_banco0[8] = {0, 1, 0, 1, 0, 0, 0, 0};
unsigned char esp100_banco0[8] = {0, 0, 1, 1, 0, 0, 0, 0};
```

Figura 56 - Combinações possíveis para o Slave CAN.

```

void loop()
{
    if(protocolo == "refresh")
    {
        CAN.sendMsgBuf(0x70,0, 8, refresh);
    }
    if(protocolo == "esp0_banco100")
    {
        CAN.sendMsgBuf(0x70,0, 8, esp0_banco100);
    }
    if(protocolo == "esp50_banco100")
    {
        CAN.sendMsgBuf(0x70,0, 8, esp50_banco100);
    }
    if(protocolo == "esp100_banco100")
    {
        CAN.sendMsgBuf(0x70,0, 8, esp100_banco100);
    }
    if(protocolo == "esp0_banco50")
    {
        CAN.sendMsgBuf(0x70,0, 8, esp0_banco50);
    }
    if(protocolo == "esp50_banco50")
    {
        CAN.sendMsgBuf(0x70,0, 8, esp50_banco50);
    }
    if(protocolo == "esp100_banco50")
    {
}

```

Figura 57 - Envio consoante SerialPort.

Já no *Slave*, processa a comparação de *byte* a *byte* e ativa a saída correspondente (Figura 58).

```

for(int i = 0; i<len; i++)    // print the data
{ Serial.print(buf[i]);
Serial.println();
}
    if(buf[0]==1)
    {digitalWrite(8, HIGH);}
    else digitalWrite(8, LOW);
    if(buf[1]==1)
    {digitalWrite(9, HIGH);}
    else digitalWrite(9, LOW);
    if(buf[2]==1)
    {digitalWrite(10, HIGH);}
    else digitalWrite(10, LOW);
    if(buf[3]==1)
    {digitalWrite(11, HIGH);}
    else digitalWrite(11, LOW);
    if(buf[4]==1)
    {digitalWrite(12, HIGH);}
    else digitalWrite(12, LOW);
    if(buf[5]==1)
    {digitalWrite(13, HIGH);}
    else digitalWrite(13, LOW);
}
}

```

Figura 58 - Comparação da Trama e ativação de saídas no Slave.

Fisicamente as saídas estão organizadas da seguinte forma:

- As primeiras três saídas correspondem às posições do espelho e as três saídas seguintes correspondem às posições do banco. Os valores correspondem a valores crescentes, em que o primeiro corresponde a zero, o segundo a

cinquenta e o terceiro a cem. Na Figura 59 é representado o valor da posição do espelho de cem e o banco corresponde a zero.

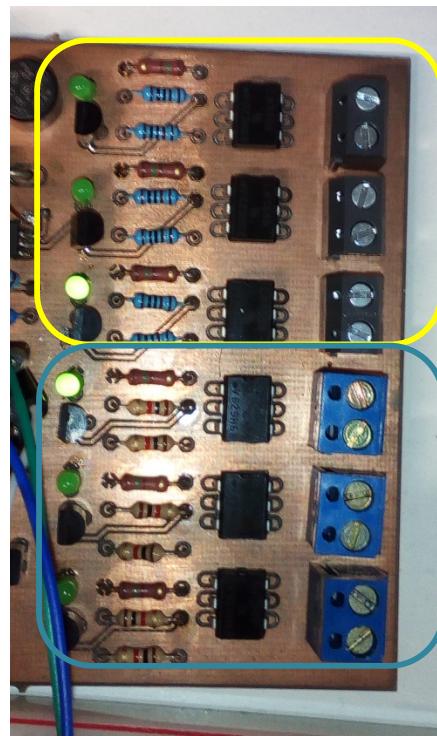


Figura 59 – Representação física das saídas.

Por fim, para comprovar que é possível comunicar entre o PC e o *Master*, foi criada uma aplicação capaz de definir o que se pretende ativar no *Slave*, como exemplificado na Figura 60.

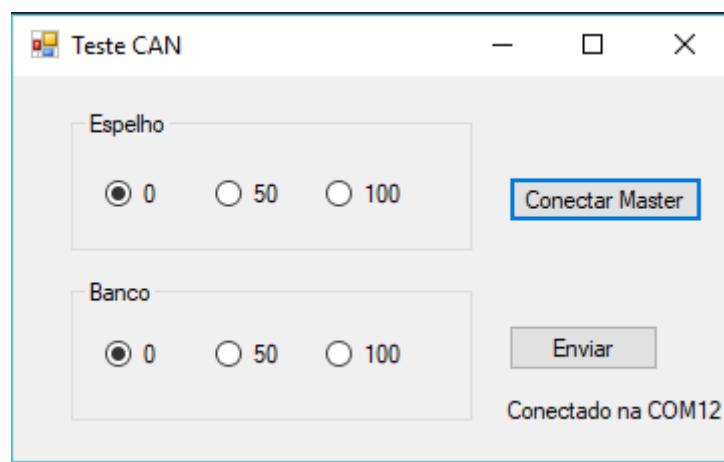


Figura 60 - Aplicação de ligação ao Slave por CAN.

Esta aplicação comunica por porta Serial com o Arduíno, envia a palavra com a junção dos valores dos espelhos e banco, para que no ATMEGA faça a comparação e envie a trama correspondente (Figura 61 e Figura 62).

```
SerialPort master = new SerialPort("COM12", 115200);
master.Open();
```

Figura 61 - Criação da Porta Serie.

```
1 reference
private void btConectar_Click(object sender, EventArgs e)
{
    master.Open();
    label3.Text = "Conectado na COM12";
}
```

Figura 62 - Ligação à porta serie predefinida

Consoante as opções selecionadas, forma a palavra de envio para que seja interpretada pelo Arduíno. Representa na interface para que o utilizador tenha a percepção do que envia (Figura 63).

```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    if(esp0.Checked){esp = "esp0";}
    if (esp50.Checked){esp = "esp50";}
    if (esp100.Checked){esp = "esp100";}

    if(banco0.Checked){banco = "banco0";}
    if (banco50.Checked){banco = "banco50";}
    if (banco100.Checked){banco = "banco100";}

    enviar = esp + "_" + banco;
    master.Write(enviar);
    label3.Text = enviar;
}
```

Figura 63 - Formação da palavra para interpretação do Atmega.

Assim desta forma confirmamos a ligação por serie, envio de dados por porta serie, interpretação pelo Arduíno, envio por CAN e interpretação certa do módulo de expansão, obtendo assim os objetivos pedidos para o trabalho, faltando somente nesta aplicação a identificação de faces e obtenção de características.

5 Conclusões

A elaboração do presente relatório teve a colaboração de todos os membros da equipa, que tentaram exprimir em palavras todo o árduo trabalho desenvolvido ao longo da realização deste projeto. Consideramos que o desenvolvimento deste projeto permitiu, entre outras coisas, fomentar o trabalho colaborativo com algumas reuniões de *brainstorm* e cooperação de todo a equipa para alcançar os objetivos propostos inicialmente.

O processo de planificação foi um processo extremamente crítico, pois envolveu muitas reflexões e variáveis a ter em conta, levando a algumas reformulações na parte de implementação do projeto. A complexidade da implementação foi por vezes subestimada e o vínculo a uma data de entrega do trabalho (um semestre) foram causadores de problemas adicionais que, infelizmente acabaram por afetar o projeto.

Uma das dificuldades sentidas ao longo do projeto relacionou-se com o *hardware* a desenvolver (compra de componentes no estrangeiro e pedidos de amostras aos fabricantes), e com o desenvolvimento do *software*, experimentando as comunicações via CAN previamente nas *veroboard* e *breadbord* utilizando Arduínos, encontrando dificuldades na programação do MCP25050. Realizou-se a ligação da interface ao módulo *Master*, testando e estabelecendo comunicações para teste de ligação ao módulo, com apresentação de indicador luminoso vermelho de falta de *Master*. Simulou-se com recurso a um botão a falta de ligação ao *Slave* e também através de um botão ligado ao *Master*, a criação de um novo utilizador.

No que se refere à comunicação CAN não ficou a funcionar como o pretendido, contudo toda a teoria necessária foi estudada, por isso mesmo após a entrega deste presente relatório, poderá haver melhorias, e consequente sucesso no que se refere ao funcionamento do módulo de expansão.

Apesar de todas as dificuldades, foi uma grande mais valia para apreender novos conhecimentos.

6 Referencias Bibliográficas

- [1] J. H. Brito, "Trabalho Prático Laboratorial. Identificação de condutor. Ipcá 2016/2017."
- [2] "TPS5430, TPS5431 TPS543x 3-A, Wide Input Range, Step-Down Converter."
- [3] Viscay.com, "4n25 - Optocoupler, Phototransistor Output, with Base Conne."
- [4] "SNx4HC14 Hex Schmitt-Trigger Inverters."
- [5] FTDI, "Ft232R Usb Uart Ic," *Technology*, pp. 1–40, 2008.
- [6] T. Atmel, H. Performance, L. Power, A. Avr, and M. Family, "ATmega328 / P," 2016.
- [7] Microchip, "Stand-Alone CAN Controller With SPITM Interface," pp. 1–84, 2005.
- [8] Microchip Technology Inc., "MCP2551 CAN Transceiver," *Communication*, pp. 1–24, 2010.
- [9] vishay.com, "LH1550AAB1,1 Form A High-Voltage Solid-State Relay."
- [10] S. Studio, "CAN-BUS Shield V1.2 - Seeed Wiki." [Online]. Available: http://wiki.seeed.cc/CAN-BUS_Shield_V1.2/. [Accessed: 25-Jul-2017].
- [11] OpenCV Dev Team, "FaceRecognizer — OpenCV 2.4.13.2 documentation." [Online]. Available: http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html. [Accessed: 16-Jun-2017]
- [12] "OpenCV: cv::face::FaceRecognizer Class Reference." [Online]. Available: http://docs.opencv.org/trunk/dd/d65/classcv_1_1face_1_1FaceRecognizer.html. [Accessed: 16-Jun-2017].