IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Escola Superior de Tecnologia

MESTRADO EM ENGENHARIA ELETRÓNICA E DE COMPUTADORES

Laboratórios Integrados II

Identificação facial de um condutor

Brian Martins Nº 9980

Cristiano Carvalho Nº 9575

Nuno Gomes Nº 9689

Julho de 2017

Resumo

O presente trabalho prático visa a implementação de um sistema de reconhecimento facial que integra dois blocos principais: um baseado em PC que é responsável pelo processamento e análise de imagem e o outro é baseado em sistemas embebidos, que gere a comunicação através do protocolo de comunicação CAN dos dados analisados pelo módulo baseado em PC.

O sistema embebido foi composto pela componente de Hardware e Software. A componente de Hardware foi implementada utilizando o EAGLE onde foram desenvolvidas duas placas: uma placa para o Master e outra para o Slave (módulo de expansão com GPIO's). Para a criação das PCB foram seguidas as regras de desenho de PCB's anteriormente já aprendidas. Apesar de terem sido desenvolvidas estas acabaram por não ser produzidas pelo facto de o circuito não estar operacional em breadboard. Assim, utilizou-se a montagem em Breadboard onde foi simulado o comportamento do sistema pela utilização de LEDs e Botões.

A parte de Software que tratou do reconhecimento facial foi efetuada em Visual Studio, utilizando o EmguCV, que também serviu de base para a criação da interface do bloco baseado em PC.

Palavras – chave: Hardware, Software, Eagle, Reconhecimento Facial, CAN

Índice

Índice de Figuras

Lista de acrónimos

SPI - Serial Peripheral Interface

CAN – Controller Area Network

GPIO – General Peripherical Input/Output

PCB – Printed Circuit Board

LED – Light Emitting Diode

Ohm – Unidade de medição de resistências

GND – Ground

PC – Personal Computer

USB – Universal Serial Bus

V – Volts

CANH – Controller Area Network High pin

CANL – Controller Area Network Low pin

PWM – Pulse Modulation Width

DC – Direct Current

I/O – Input / Output

pF – p (pico) & F (Faraday), unidades de medição de condensadores

SMD – Surface Mount Device

A/D – Analog / Digital

1. Introdução

O presente trabalho prático foi realizado no âmbito da unidade curricular de Laboratórios Integrados II, lecionada durante o 2º semestre do 1º ano do Mestrado em Engenharia Eletrónica e de Computadores.

A indústria automóvel procura constantemente novas formas de adaptar o funcionamento dos seus produtos às conveniências e conforto do cliente. O objectivo final deste projecto 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.

A verificação e identificação do condutor será efetuada através de visão por computador, ao desenvolver esta aplicação terá de ser capaz de comparar a fase presente na imagem com faces de referência existentes na base de dados dos condutores possíveis.

Após a aplicação que correr em PC implementar as funções de processamento e análise de imagem este comunica com um dispositivo externo, baseados em sistemas embebidos, através de USB e por sua vês este sistema embebido comunicará com os restantes dispositivos de expansão através de barramento CAN, sendo este a comunicação *standard* na indústria automóvel, como mostra a figura 1.

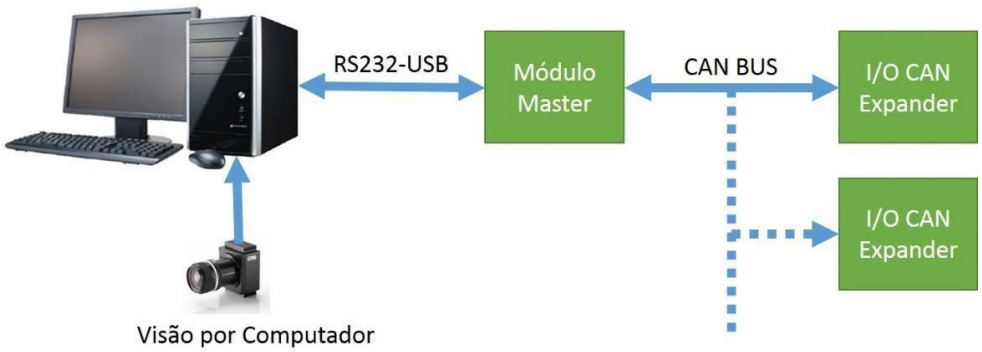

Figura 1 - Arquitetura sistema a montar

Resumidamente, o barramento CAN foi desenvolvido por Robert Bosh em 1986 para simplificar os sistemas de cablagem na indústria automóvel. A frame standard (figura 2) que utilizamos é bastante simples e permite ter muitos módulos conectados ao mesmo barramento através da variação dos 11 bits do Identifier. O RTR indica se a frame é recessivo caso RTR=1 ou dominante caso RTR=0. Por fim, temos o tamanho da mensagem em bytes e a própria mensagem que queremos enviar. Com base nestas frames conseguimos fazer a comunicação na rede CAN para ler dados ou escrever dados enviando mensagens através do módulo Master.

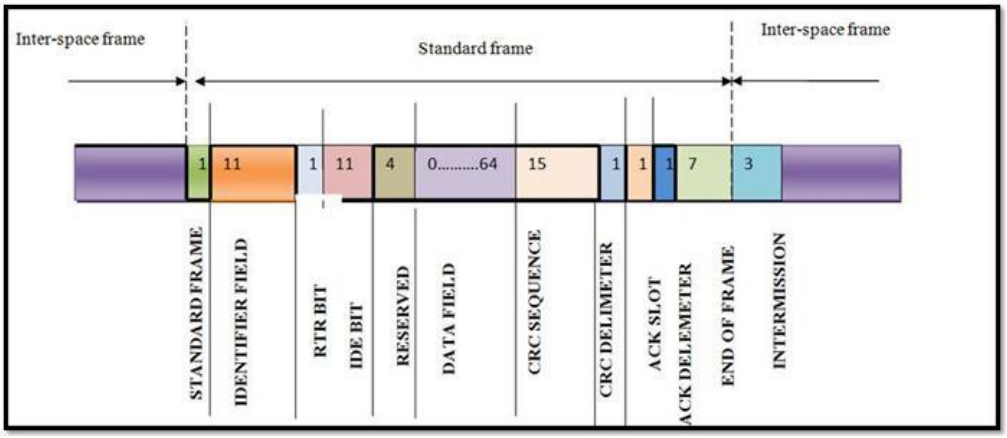

Figura 2 - CAN Standard Frame

2. Requisitos dos módulos do sistema

Como vimos anteriormente, este projecto encontra-se dividio em dois blocos principais.

2.1. Software

O primeiro bloco, baseado em software, é responsável pelo interface com o utilizador, gestão dos dados e configuração do sistema, processamento e análise de imagem e comunicação dos dados através de USB.

Para cumprir estes requisitos de comunicação foi necessário definir um protocolo de comunicação, entre o PC e o módulo Master, que permita enviar do PC para o Master o resultado da identificação de faces, com identificação de quais os módulos de expansão I/O que devem ser ativados/desativados. Deverá ser ainda possível realizar um pedido de identificação de todos os módulos de expansão I/O ligados à rede CAN e ainda permitir a criação/alteração de registo de faces guardadas na base de dados.

Relativamente à interface com o utilizador desenvolveu-se uma aplicação que permitisse uma fácil e intuitiva interação com o sistema. Como requisitos esta terá de respeitar os seguintes: indicar o estado da ligação ao Módulo Master; listar todos os módulos ligados à rede CAN; a imagem da câmara deverá aparecer em tempo real no ecrã do PC; a detecção e identificação de faces deverá ser assinalada na imagem; as saídas dos módulos a activar 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 activar deverão ser configuráveis; permitir a criação, edição e remoção de registos 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 activados/desactivados; e por fim permitir a recepção de pedidos de criação de novo registo de face na base de dados provenientes do módulo master, com a face identificada no momento.

2.2. Hardware

O segundo bloco, hardware, consiste no desenvolvimento de um bloco baseado em sistemas embebidos que incluirá o desenvolvimento de dois tipos de módulos.

Um Módulo Master (figura 3), 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.

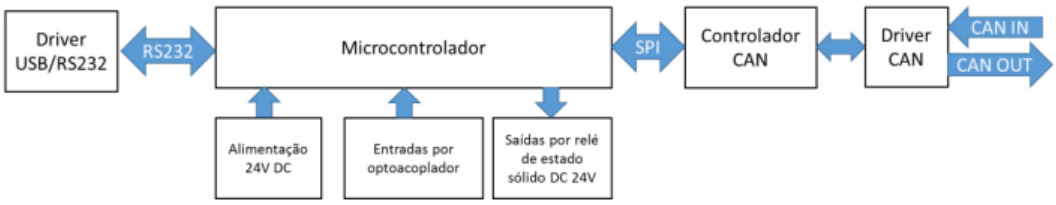

Figura 3 - Módulo Master

No desenvolvimento deste módulo master teremos de considerar as seguintes necessidades:

- Alimentação por fonte externa 12/24 V DC (o mais comum na indústria);
- Ligação ao PC por USB;
- 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 (MCP2551);
- Disponibilização de um conjunto de 6 entradas (por opto-acoplador) e 2 saídas (por relé de estado sólido), com indicação luminosa do estado atual.

Um Módulo de expansão I/O (figura 4), que estará ligado ao barramento CAN e que traduzirá nas suas saídas os dados resultantes do processamento de imagem, transmitidos pelo módulo master.

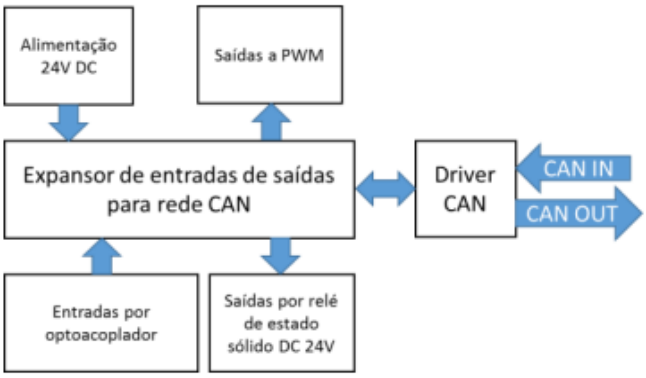

Figura 4 - Módulo de expansão

À semelhança do módulo master, também este módulo de expansão deverá respeitar os seguintes requisitos:

- Alimentação por fonte externa 12/24 V DC;
- Implementação do protocolo CAN através de um bloco expansor de entradas e saídas 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) e 6 saídas (por relé de estado sólido) com indicação luminosa do estado atual.

2.3. Controlo do estado do projeto

Tendo por base estes requisitos do sistema, quer a nível de software quer a nível de hardware, passou-se ao desenvolvimento do mesmo. Todo o trabalho realizado encontra-se relatado no capítulo seguinte.

Durante o desenvolvimento do projecto, todos os documentos, desenhos, esquemáticos, código fonte, etc., foram geridos através de um sistema de gestão de versões, do tipo Git. O servidor na cloud utilizado foi o *GitHub*[1]. A figura 5 apresenta o logótipo deste servidor e a figura 6 demonstra um excerto de commits efetuados ao longo dos últimos meses.

Figura 5 - Logótipo da plataforma Github

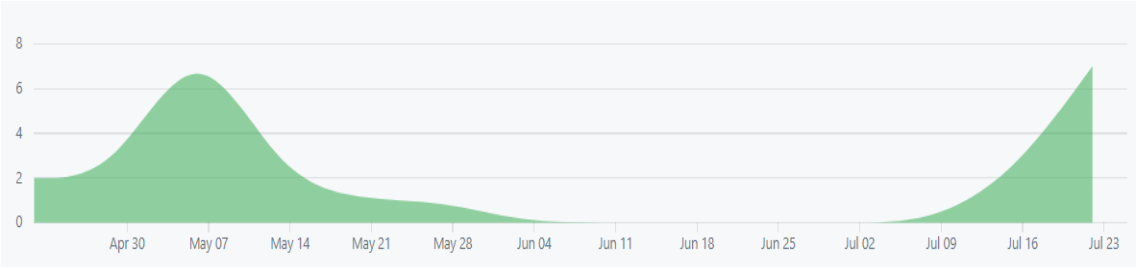

Figura 6 - Grafico com a quantidade de commits ao longo dos meses de 2017

Para além disso, para um melhor planeamento das tarefas a realizar por cada membro do grupo foi utilizado o *Trello*[2], tendo sido criados duas boards de trabalho, uma para a parte de Software e outra para a parte de Hardware. A figura 7 ilustra o logótipo da ferramenta *Trello* e a figura 8 ilustra um exemplo da lista de tarefas TODO da equipa de software.

Figura 7 – Logótipo da ferramenta Trello

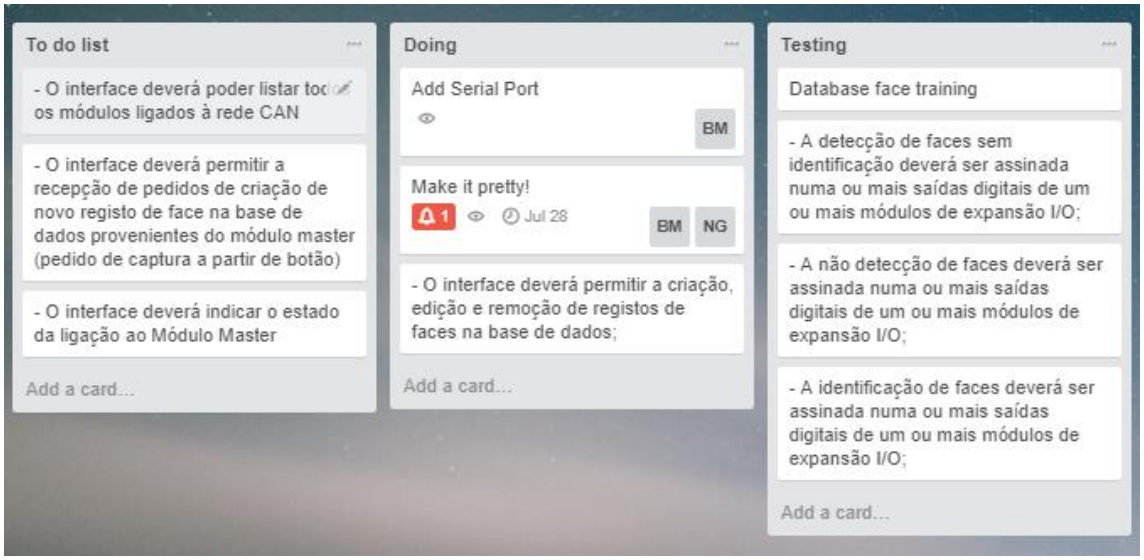

Figura 8 - Exemplo da list TODO da equipa de software

3. Desenvolvimento

3.1. Hardware

3.1.1. Módulo Master CAN

Previamente ao desenvolvimento da PCB do módulo Master em Eagle CAD[3], a funcionalidade do circuito foi testada de forma a entender o funcionamento de todos os elementos e a realizar ensaios de validação com o propósito de eliminar quaisquer erros após a produção da mesma. O esquemático deste módulo foi assim montado em breadboard, tendo-se utilizado um Arduino Uno para fazer a comunicação entre o PC e o módulo, uma vez que este utiliza um microcontrolador ATMega328[4], já com todos os elementos necessários à sua funcionalidade e incorpora também uma FTDI a qual permite fazer a comunicação série entre o PC e o microcontrolador via USB.
Embora os testes tenham sido realizados com um Arduino Uno, no desenvolvimento da PCB final foi utilizado um ATMega328[4] em Bootloader, com um cristal de 16MHz e dois condensadores cerâmicos de 22pF em paralelo.

A comunicação entre o ATMega328 e o controlador stand-alone CAN MCP2515 é feita através de SPI através dos pinos 16->16(Chip-Select), 17->14(Master-Out Slave-In), 18->15 (Master-In Slave-Out) e 19->13(Clock). No circuito do MCP2515 foi utilizado o mesmo esquemático do oscilador utilizado para o ATMega238.

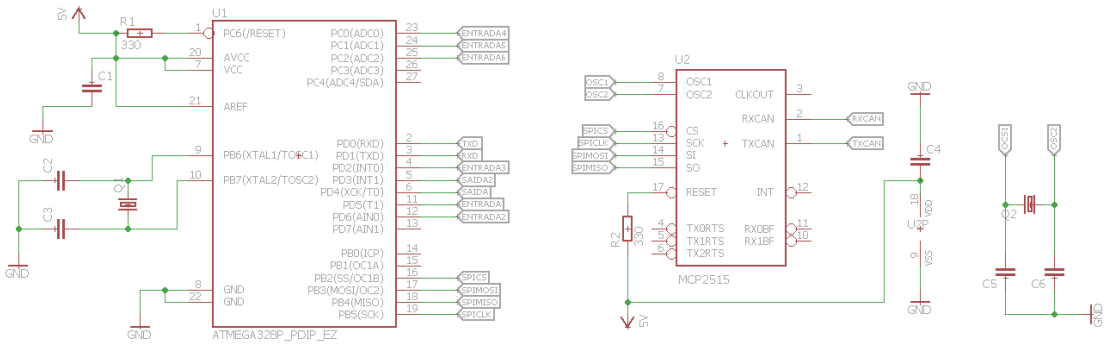

Figura 9 - ATMega328 e MCP2515[5][4]

Relativamente à programação do microcontrolador ATMega328 foi utilizada a IDE do Arduíno[6], baseada em programação em linguagem C, para criar as propriedades da rede CAN, envio e receção de dados por SPI, controlo de entradas e saídas, envio de mensagens entre módulos, etc. Tendo isto, para fazer a comunicação entre o PC e a placa através da porta USB, foi necessário utilizar um FTDI FT232RL que é um módulo conversor RS232 TTL para USB. Através do FTDI basta apenas utilizar 3 pinos do módulo ligados ao ATMega328, nomeadamente o RXD (Receiving Asynchronous Data Input), TXD (Transmit Asynchronous Data Output) e a partilhar as massas entre o PC e a placa pelo GND, sendo a sua alimentação feita via USB (5V). Assim na PCB foi implementado um socket de 18 pinos para fixação do módulo FTDI como podesmo ver na figura seguinte:

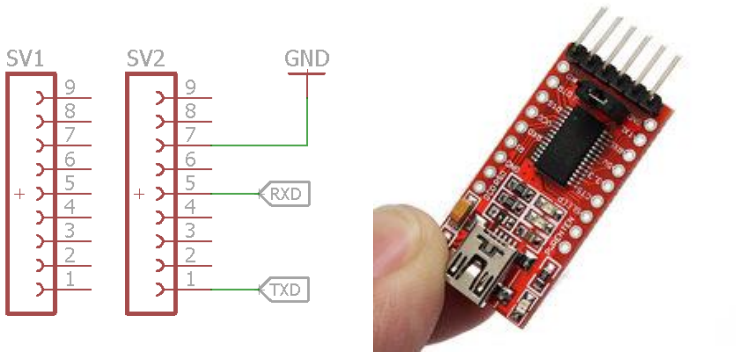

Figura 10 – Módulo FTDI

Relativamente às saídas, tanto para o módulo Master como para o módulo de expansão, foram utilizados os relés SMD com isolamento galvânico recomendados, os CPC1002N. Uma vez que a versão de Eagle utilizada não possuía este componente foi utilizado um semelhante com as mesmas medidas, nomeadamente AQY41SOP. Tal como requirido no módulo master teremos duas saídas a relé e seis no módulo de expansão.

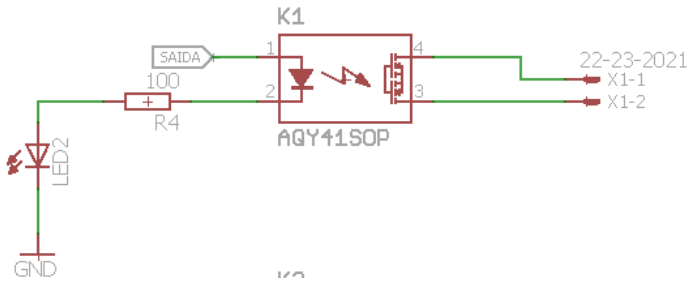

Figura 11 - Saídas com relé estado sólido

As saídas dos relés são conectadas posteriormente conectadas a terminais para poderem ser utilizadas. Os relés são atracados pelo ATMega, e que quando existe uma diferença de potencial aplicada na entrada do relé, é ativo o díodo que consequentemente faz com que o transístor conduza, podendo a saída ser utilizada. Para ser mais perceptível o estado das saídas foram colocados LED'S em cada uma delas juntamente com uma resistência de 100 Ohm, valor este obtido a partir de cálculos efetuados para que o relé tivesse a corrente necessária para atracar.

Quanto às entradas foi optamos por utilizar o opto acoplador PC817. Á semelhança das saídas, também para as entradas foram utilizados terminais para que possam conectar entradas de 12V. Nesta configuração utilizamos uma resistência e um díodo em série. Tal como para os relés, também esta resistência é dimensionada para limitar a corrente de entrada para o opto acoplador, já que o datasheet indica que a Forward Current máxima é de 50mA. O díodo serve para evitar correntes no sentido oposto que possam danificar a própria entrada que se ligará a esse opto acoplador.

O principio de funcionamento do opto acoplador é semelhante ao do relé. Assim que é aplicada uma tensão ao foto-díodo do PC817, o transístor começa a conduzir porque existe uma corrente aplicada na base e desta forma iremos ter os +5V na entrada corresponde do ATMega. Também aqui foi colocado um led na saída do opto acoplador para saber em que estado estará a entrada. Por outro lado, para evitar que a entrada fique no estado de alta-impedância foi implementado uma resistência de pull - down de 4.7kOhm.

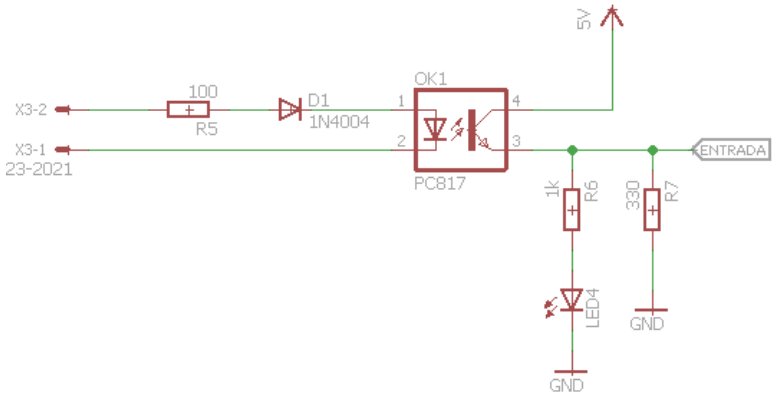

Figura 12 - Entradas com optoacoplador

As PCB's a desenvolver serão alimentadas a 12V por um jack. Tendo isto e uma vez que todos os componentes do circuito são alimentados a 5V, foi necessário implementar um regulador de tensão. Foi então utilizada então a seguinte montagem:

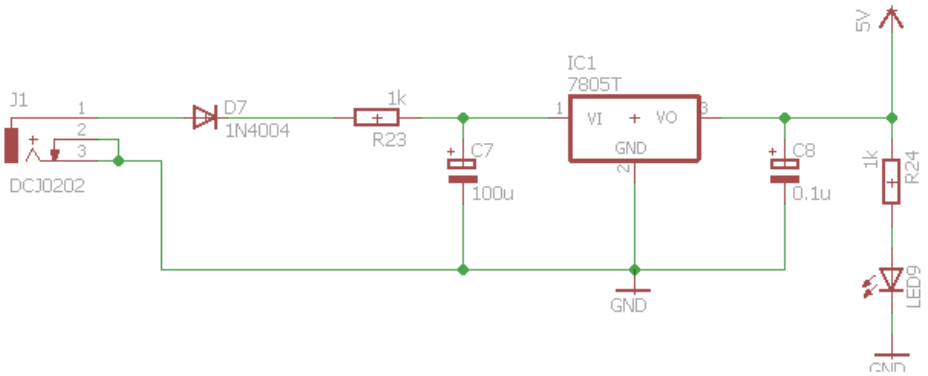

Figura 13 - Regulador de tensão

Na montagem do esquemático anterior, além do LM7805 foi necessário utilizar um díodo para evitar correntes inversas que danifiquem o circuito e dois condensadores de acoplamento. Para termos feedback do estado da de alimentação do circuito foi igualmente colocado um led para saber o estado da mesma.

A interface para a rede CAN requer que o MCP2515 tenha um driver MCP2551[7]. O MCP2515 liga ao driver através do TXCAN e RXCAN sendo este driver que irá enviar as mensagens para o barramento CAN através do CANH e CANL. Uma vez que irão ser desenvolvidas duas PCB's (uma para o master e outra para o módulo slave - expansão), estas saídas de CAN irão estar ligadas a terminais para que se possa ligar aos módulos de expansão pretendidos. De salientar que por cada módulo de expansão ligado a

estas saídas, estes deverão ter nas suas entradas de CAN uma resistência de 120Ohm em paralelo entre as linhas de CANH e CANL.

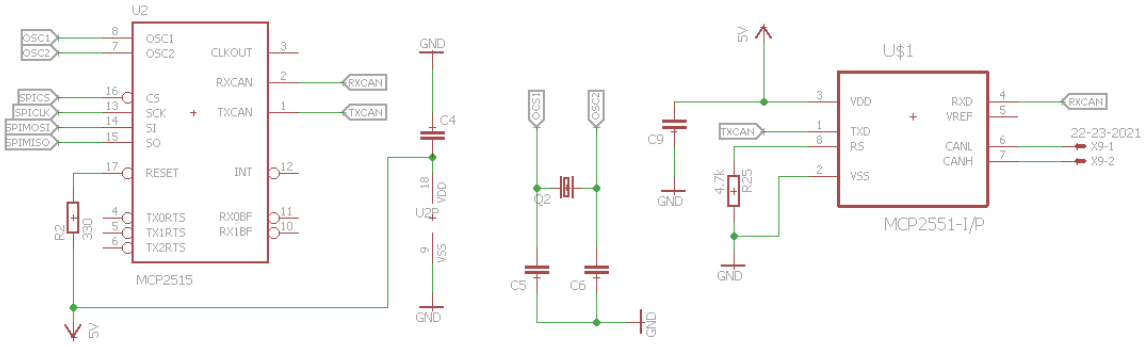

Figura 14 - MCP2515 e MCP2551[5], [7]

O esquemático do módulo Master resultou na construção de uma PCB de 10.4x8.3 cm. Esta foi desenvolvida recorrendo também ao Eagle tendo o cuidado de seguir os principios de desenho de uma PCB de forma a evitar ruídos eletromagnéticos. Foi adicionado também um plano de massa GND.

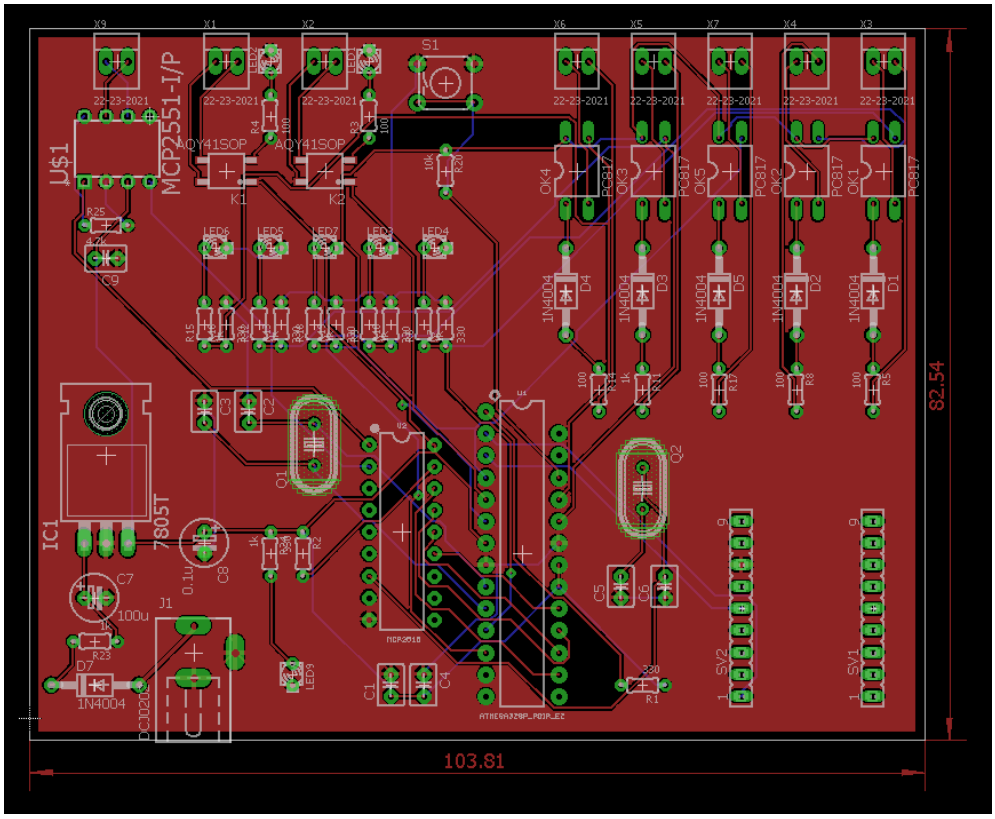

Figura 15 - PCB Master

As figuras 16 e 17 ilustram em 3D a PCB do módulo master concebida.

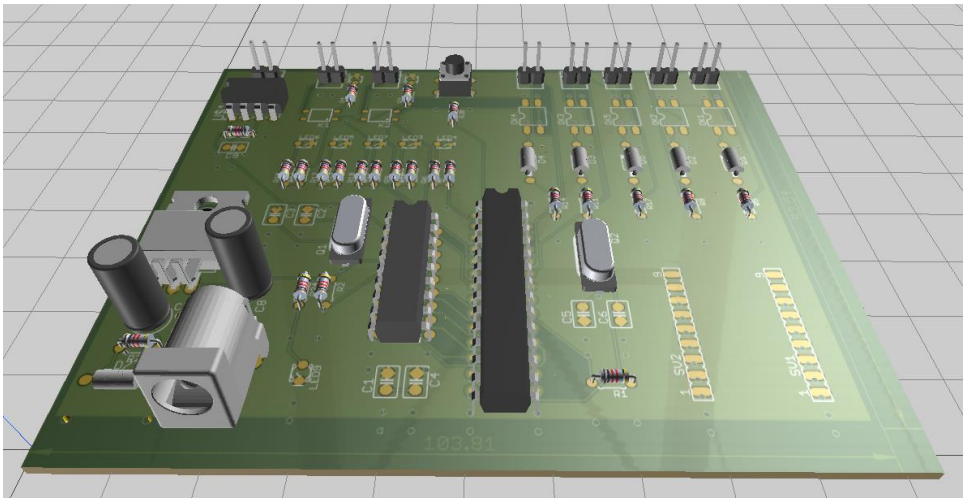

Figura 16 - PCB Master 3D(1)

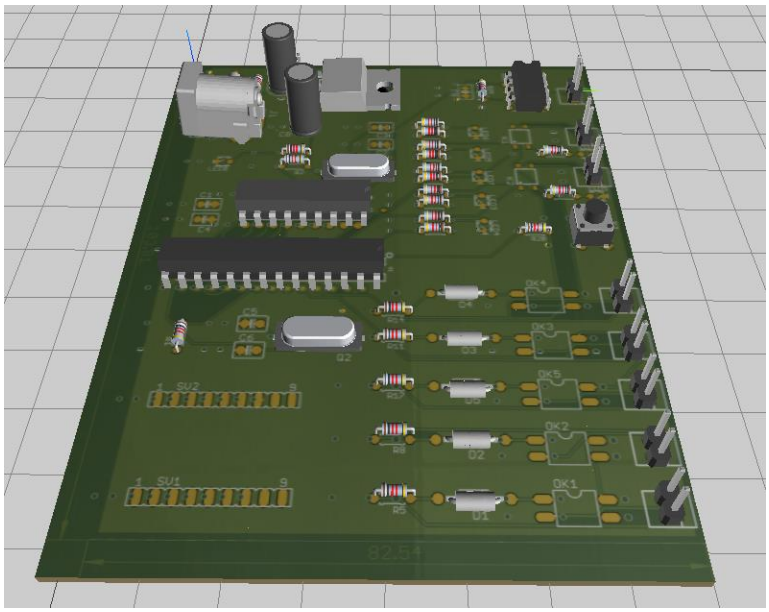

Figura 17 - PCB Master 3D(2)

3.1.2. Módulo de expansão I/O CAN

Este módulo é composto essencialmente por dois componentes principais, o MCP2551 e o MCP25050[8]. O MCP25050 É um expansor de entradas e saídas para trabalhar com redes CAN, este possui 8 I/O digitais, 4 conversores A/D e 2 saídas PWM. Desta forma permite que um nó da rede CAN seja implementado sem a necessidade de recorrer a mais um microcontrolador.

Na figura 18 podemos ver o esquemático geral do módulo de expansão. O MCP2551, tal como já referido, terá a função regularização dos níveis de tensão para a rede CAN tanto no módulo de expansão como no Master. De realçar que no MCP25050 é utilizado igualmente o oscilador de 16MHz com os condensadores de 22pF utilizado no esquemático do master mantendo assim o sistema em sincronismo.

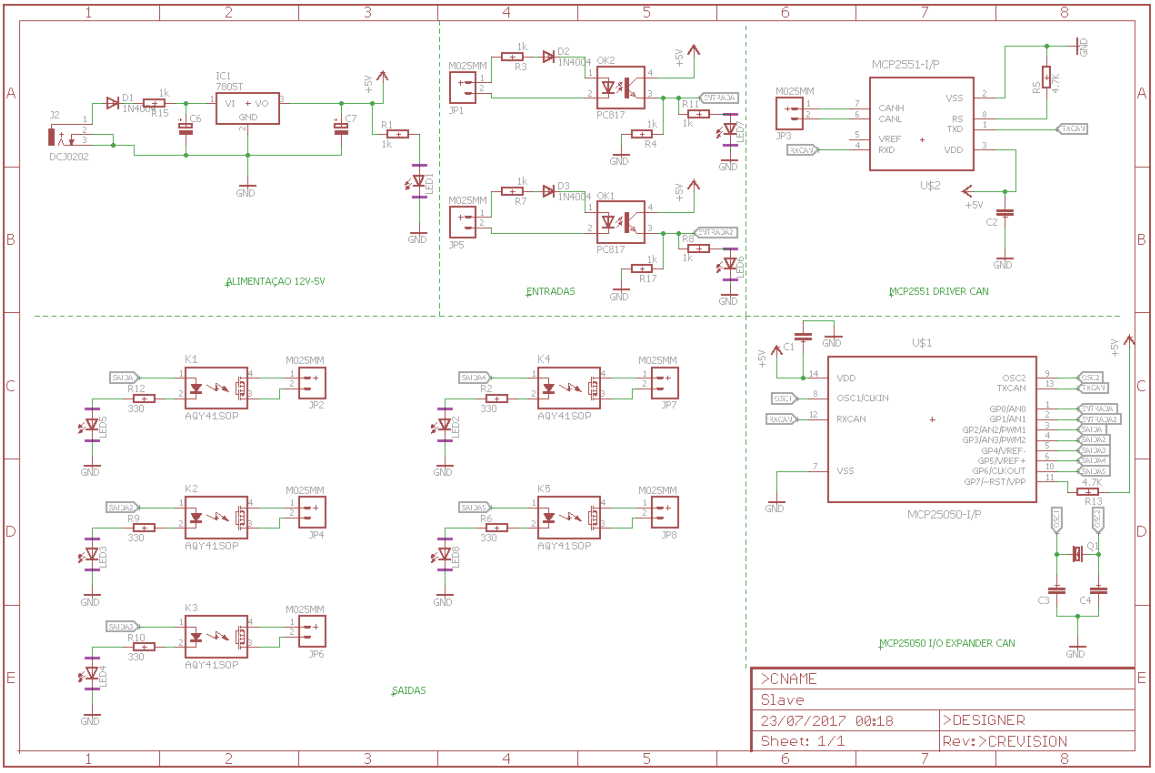

Figura 18 - Esquemático módulo de expansão

Tendo isto, passou-se ao desenvolvimento da PCB para o módulo de expansão. O resultado foi uma placa de 9.2x5.8 cm, tendo-se igualmente cumprido os critérios utilizados na conceção do módulo Master.

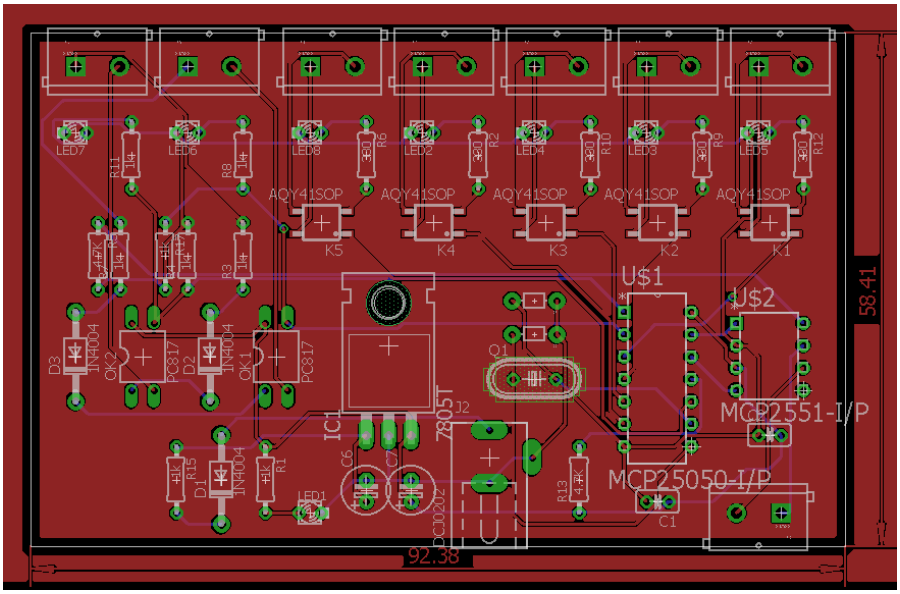

Figura 19 - PCB módulo expansão

As figuras 20 e 21 ilustram em 3D a PCB do módulo de expansão concebida.

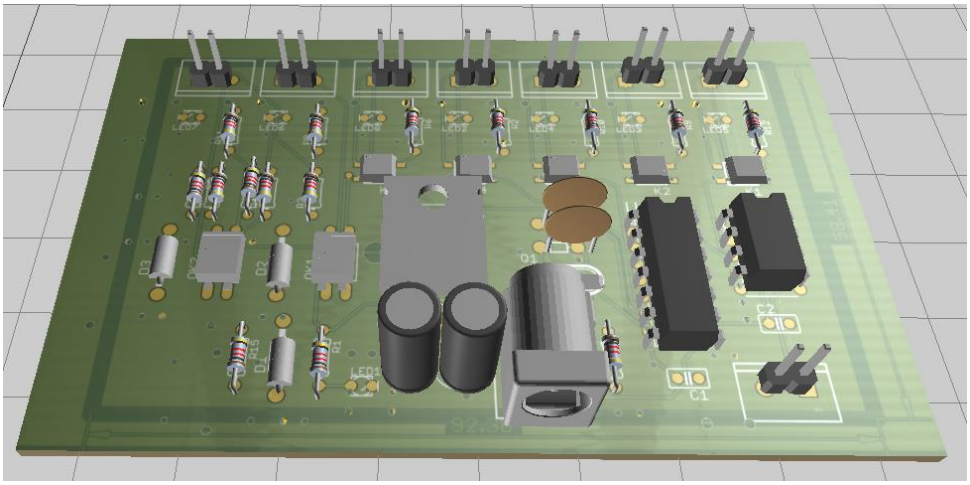

Figura 20 - PCB módulo expansão 3D (1)

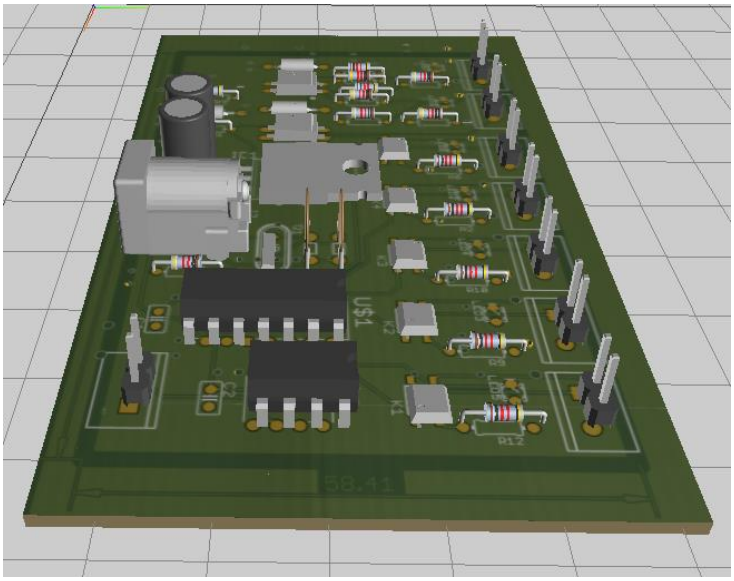

Figura 21 - PCB módulo expansão 3D (2)

3.2. Sofware

3.2.1. Programação MCP25050

Enquanto que o ATMega328 pode ser programado constantemente através do IDE do Arduíno, o MCP25050 apenas pode ser programado uma única vez na ROM do integrado. Este integrado foi programado recorrendo ao software MCP250xxProgrammer. O MCP25050 foi programado para ser utilizado com um clock de 16MHz, tendo sido usado também para configurar as suas 8 GPIO's. Tal como requerido estas foram configuradas duas entradas e seis saídas.

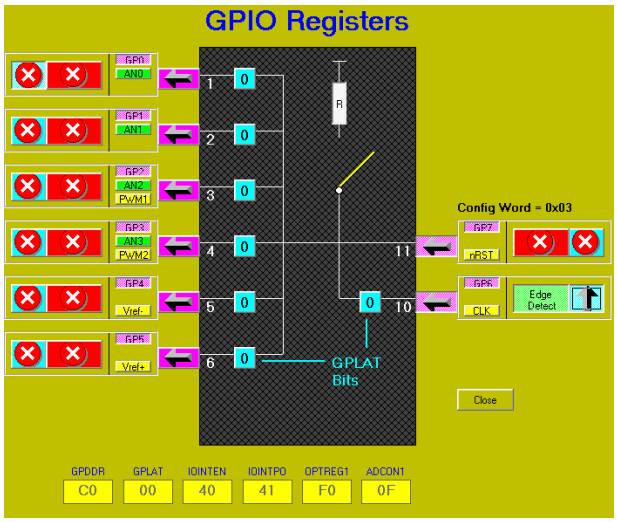

Figura 22 - GPIO Registers MCP25050

Para fazer a comunicação para o MCP25050 utilizou-se dois endereços distintos. Os endereços escolhidos foram o 0x580, para aplicar configurações e ler dos registos, e o endereço 0x680 para envio de mensagens que necessitem de resposta por parte do integrado. Assim, quando é criada uma interrupção recebemos o valor de 0x580.

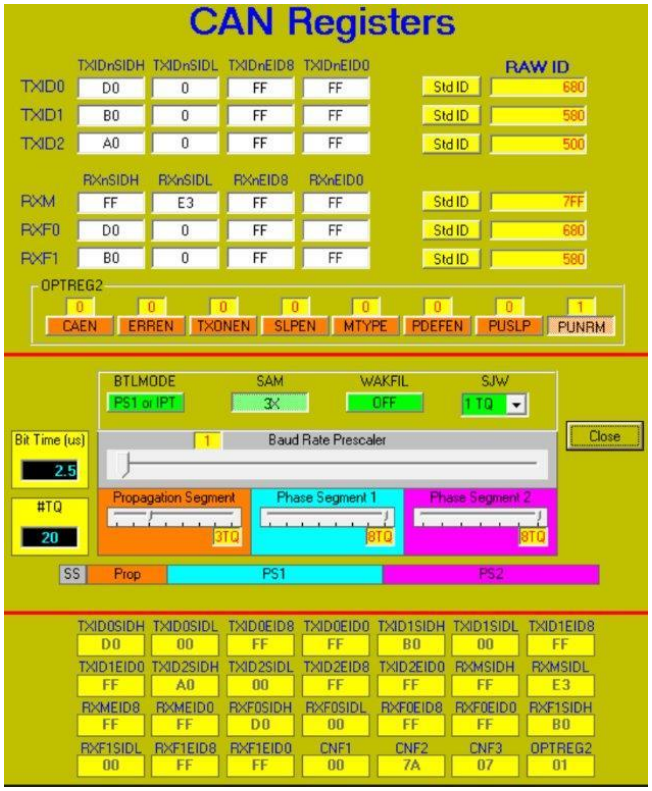

Figura 23 - CAN Registers MCP25050

Para “queimar” estas configurações, isto é, o enviar para o integrado o ficheiro .hex, utilizou-se um PICKIT2 disponibilizado pelo IPCA. Basicamente, a programação é feita a partir de um nível de tensão elevado aplicado num pino, para que desta forma seja possível entrar no modo de programação. A ligação entre o MCP25050 e o PICKIT2 obedece ao seguinte esquemático:

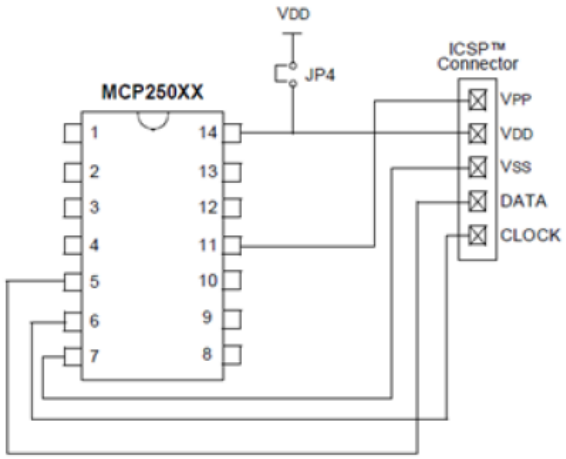

Figura 24 - Esquemático PICKIT2

Uma vez criado o ficheiro *.hex* e montado o circuito anterior, utilizou-se o programa da Microchip PICKIT2 para programar o MCP25050.

3.2.2. Comunicação entre a interface gráfica e o módulo master CAN

Para a comunicação entre a interface gráfica e o módulo master CAN foi utilizado uma porta série e o protocolo inspirado em formato JSON mas customizado para melhor leitura do módulo master. Como príncipio do protocolo JSON, é utilizado o método key: value, ao qual o key é o GPIO a controlar, e o value o estado desse GPIO. Na parte do módulo, este ficará à espera de receber os últimos bytes *'\n\0'*. Após a recepção destes bytes o módulo procede à repartição da trama através dos divisores ';' e ':'. Deixando assim no final apenas o estado dos GPIOS a controlar. A figura 25 demonstra um exemplo desta trama construída na interface gráfica.

```
serial_data = String.Format("{ User: {0} ; GPIO1: {1} ; GPIO2: {2} ; GPIO3: {3} ; GPIO4: {4}} \r\n", names_detected[0], gpio_state_1,
```

Figura 25 - Construção da trama de envio

De mencionar que na falta de face para reconhecer, a aplicação cria uma trama com os GPIO's todos a *low.* Também na caso de existir uma face detetada mas a falta desta na base de dados, a aplicação irá ordenar para que seja escrita os IO's todos a *high.*

3.2.3. Comunicação entre módulo master e módulo slave (CAN)

Na programação utilizamos 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.

Na programação do ATMega328 utilizou-se o comando CAN.sendMsgBuf (ID,Standard/Extended Frame, Tamanho da Data, Data, RTR). Assim, quando queremos comunicar com o módulo de expansão basta enviar o ID x580 (RTR=0) ou x680(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.

```
Serial.println("Modo de Configuracao");
String array[4];
int endereco,mascara,valor;

array[0] = inputString.substring(5,9); //endereco
array[2] = inputString.substring(10,12); //mascara
array[3] = inputString.substring(13,15); //valor
Serial.println(array[0]);
Serial.println(array[2]);
Serial.println(array[3]);
endereco = array[0].toInt();
 mascara = array[2].toInt();
valor = array[3].toInt();

unsigned char buf[3] =  {0x1E,mascara,valor};
 CAN.sendMsgBuf(endereco, 0,3, buf,0); //ID,Standard,tamanho,buffer
 inputString = "";
```

Figura 26 - Envio de comandos CAN

3.2.4. Interface gráfica

Devido às funcionalidades da interface gráfica, este tópico fica repartido por os diversos tópicos. Começando por a ordem de funcionamento do algoritmo, inicialmente demonstra-se a detecção de faces em tempo real. Depois a implementação de algoritmos de reconhecimentos de faces com demonstrações. Após isto é apresentado o método de treino de faces. Por fim é apresentado a forma como é guardado e analisado as faces e também as preferências dos utilizadores.

3.2.4.1. Detecção de faces

A detecção de faces passa por o carregamento de dois ficheiros de classificadores em cascata de características haar. Estes dois ficheiros estão em formato .xml e foram fornecidos por o site oficial das librarias do OpenCV[9]. Os classificadores em cascata de características haar, são treinados para detetar faces e olhos, e são carregados estes classificadores para detetar as faces.

Neste projeto, a detecção de faces inicialmente não era robusta, por isso foi criada uma condição para detetar uma face e dois olhos. Assim, com esta condição reduz-se a quantidade de falsos positivos. A figura 27 demonstra a detecção de uma face com também a detecção de dois olhos dentro desta.

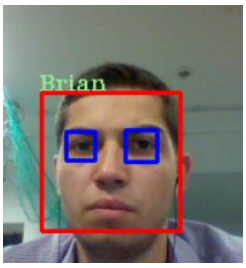

Figura 27 - Exemplo da detecção de uma face

3.2.4.2. Reconhecimento de faces

Para o reconhecimento de faces é utilizado o método de Eigenfaces, ao qual consiste numa comparação de distancia de vetores entre as faces da base de dados e a face apresentada em tempo real.

No algoritmo implementado de reconhecimento de faces é chamada uma função que recebe como parâmetros as faces treinadas, os nomes dos utilizadores treinados, uma constante *eigenDistanceThreshold* e um critério de iteração.

A constante *eigenDistanceThreshold* é um valor que relaciona a probabilidade da face a reconhecer está inserida na base de dados. Caso este valor esteja muito baixo a probabilidade da face a reconhecer ser tratada como uma face desconhecida é muito alta. Se esta constante for muito alta a probabilidade da face a reconhecer estar inserida na base de dados é mais alta. A constante *eigenDistanceThreshold* toma valores entre 0-10000. Neste projeto está a ser utilizado um valor constante de 2500, resultado de recomendação do EmguCv[10] e algum ajuste por problemas de luminosidade local.

O critério de iteração é resultado de uma função também incluída nas librarias do EmguCv. Este critério avalia o tamanho da base de dados e determina quando a pesquisa nesta deve parar com a ajuda de uma constante epsilon.

A figura 28 contém à esquerda os parâmetros necessários parao funcionamento deste *recognizer* e um exemplo da implementação desta à direita.

```
public EigenObjectRecognizer(
        Image<Gray, byte>[] images,
        string[] labels,
        double eigenDistanceThreshold,
        ref MCvTermCriteria termCrit
)
```

```
//Eigen face recognizer
EigenObjectRecognizer recognizer = new EigenObjectRecognizer(
    trainingImages.ToArray(),
    labels.ToArray(),
    2500,
    ref termCrit);
name = recognizer.Recognize(result);
```

Figura 28 - Exemplo da função "EigenObjectRecognizer" à esquerda e a implementação desta à direita

Nas figura 29 é demonstrado dois exemplos diferentes de reconhecimento de faces utilizando este método.

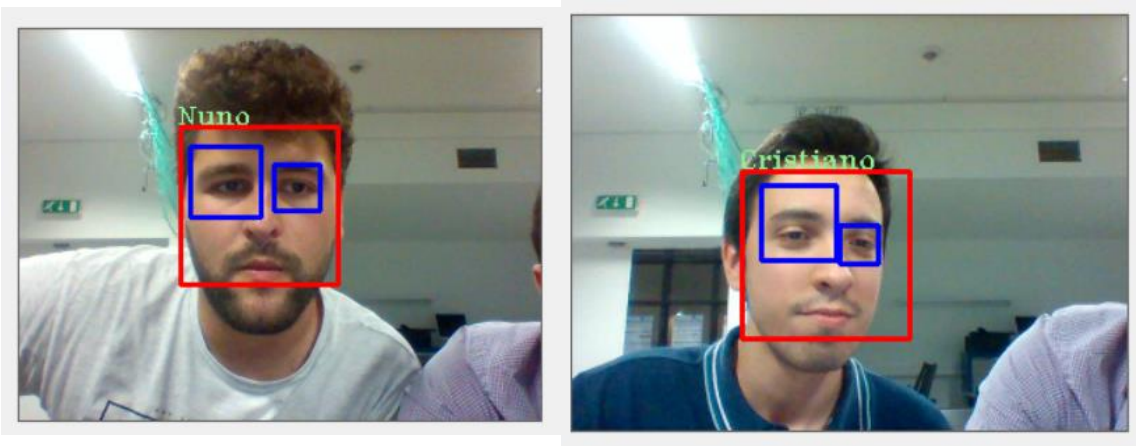

Figura 29 - Exemplos de reconhecimento de faces

3.2.4.3. Treino de faces

Para o treino de faces foram utilizados vários métodos, mas o que prevaleceu foi a extracção de múltiplas imagens de várias expressões faciais principais. Das expressões faciais existentes, foram selecionadas cinco mais determinantes no reconhecimento facial, (i) expressão normal, (ii) expressão sorridente, (iii) expressão cabisbaixa, (iv) expressão de surpresa e (v) expressão mais sonâmbula. Segues-se um exemplo desta implementação na figura 30.

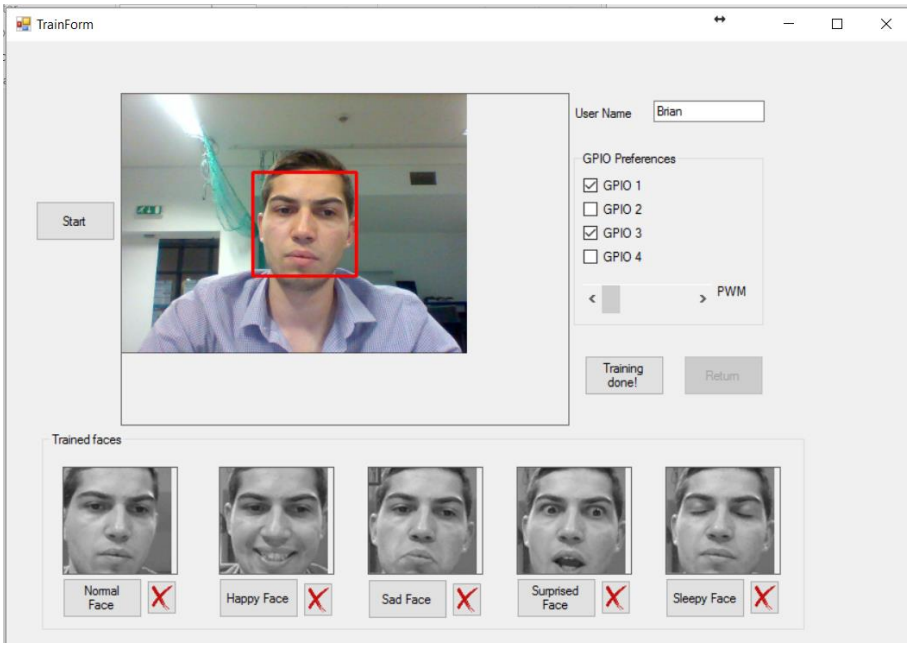

Figura 30 - Interface gráfica para treino de um novo utilizador

3.2.4.4. Base de dados das faces

A base de dados implementada, está dividida em três vertentes. A primeira consiste numa pasta com as imagens de treino guardadas em formato .bmp. Estas faces são lidas de modo sequencial e guardadas numa lista de `new Image<Gray, byte>`. A figura 31 ilustra algumas faces guardadas na pasta. O nome das imagens ajuda no carregamento ordenado destas. Para futura edição da base de dados.

Figura 31 - Exemplo de faces guardadas localmente

A segunda vertente consiste no nome dos utilizadores guardados. Estes nomes estão guardados por ordem das faces guardadas. Tendo as variáveis divididas por um caracter (‘%’), guarda-se o primeiro valor com a quantidade de utilizadores guardados e de seguida o nome correspondente a cada face guardada. A figura 32 demonstra um excerto das informações guardadas.

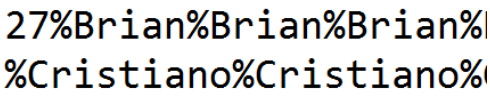

Figura 32 - Informações dos nomes dos utilizadores guardados

A terceira vertente consiste nas preferências guardadas por utilizador. Estas preferências correspondem respetivamente ao estado dos leds a controlar no slave CAN. A figura 33 demonstra um excerto das informações guardadas no ficheiro.

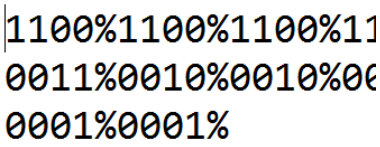

Figura 33 - Informações das preferências dos GPIO's dos utilizadores guardados

4. Conclusões

Através da implementação do trabalho prático foi possível adquirir e aprofundar conhecimentos inerentes à utilização do protocolo de comunicação CAN, bastante utilizado em sistemas embebidos para a indústria automóvel por se tratar de um protocolo robusto e pela gestão de comunicação (taxas de transmissão compatíveis e deteção de falhas).

Pelo facto do trabalho ser composto pela componente de *hardware* e *software*, o mesmo seguiu todas as etapas que constituem o desenvolvimento de um sistema embebido de raiz, desde a fase de projeto à execução final.

A implementação do *hardware* em EAGLE compreendeu a parte do desenho esquemático e a respetiva PCB do circuito do Master e do Slave. Apesar disto não foi possível conceber as PCB's pelo facto de não termos tido êxito na implementação do circuito em *breadboard*. O circuito foi montado e testado em *breadboard*, tendo sido ligado a um Atmega328p e testadas as entradas e saídas com LEDs. Foi utilizada uma ferramenta 3D para a visualização do ficheiro .brd do EAGLE, a 3D BRD Viewer onde é possível ter uma visão completa 360º da placa de circuito gerada.

A programação do MCP25050 foi efetuada no programa MCP250xxProgrammer. A interface gráfica amigável permitiu a definição de entradas e saídas que foram utilizadas, bem como a configuração dos registos necessários de transmissão e receção de dados via CAN. Para a programação do MCP25050 foi utilizado o programador PicKit2 através do ficheiro .hex gerado pelo programa.

Apesar de todas as tentativas não foi possível obter uma comunicação CAN entre o módulo Master e o módulo de expansão a partir do protocolo de comunicação criado. Assim foi criada uma comunicação CAN entre o cirucito do módulo Master e um Arduino a funcionar como slave. Com isto, podemos concluir que o sistema desenvolvido carece de várias melhorias para que se possa tornar um sistema estável.

Relativamente à parte de software, nomeadamente ao nível do reconhecimento facial, foram sentidas algumas dificuldades devido ao pouco suporte disponível online para

EmguCV quando comparado com o suporte disponível para OpenCV. Quanto à interface gráfica desenvolvida e ao método de reconhecimento facial, para obter um sistema com mais confiança, seria necessária a utilização de métodos mais robustos sem ser o EigenFaces, como por exemplo Machine Learning, e também uma base de dados mais estrutura, como por exemplo em SQL.

Em suma, podemos concluir que, apesar de alguns objetivos não terem sido alcançados com sucesso, a realização deste trabalho prático contribuiu em muito para o enriquecimento das nossas competencias não só a nível de conhecimentos de eletrónica, mas também ao nível do protocolo de comunicação CAN e de algoritmos de reconhecimento facial.

5. Bibliografia

[1] Git, “Git.” [Online]. Available: https://git-scm.com/. [Accessed: 15-Feb-2017].

[2] Trello, “Trello.” [Online]. Available: https://trello.com/. [Accessed: 14-Feb-2017].

[3] Autodesk, “PCB Design & Schematic Software | EAGLE | Autodesk.” [Online]. Available: https://www.autodesk.com/products/eagle/overview. [Accessed: 27-Jul-2017].

[4] Atmel, “Datasheet Atmega328p.”

[5] Microchip, “Datasheet MCP2515.”

[6] Arduino, “Arduino - Home.” [Online]. Available: https://www.arduino.cc/. [Accessed: 27-Jul-2017].

[7] Microchip, “Datasheet MCP2551.”

[8] Microchip, “MCP25050 pdf, MCP25050 description, MCP25050 datasheets, MCP25050 view ::: ALLDATASHEET :::” [Online]. Available: http://pdf1.alldatasheet.com/datasheet-pdf/view/90160/MICROCHIP/MCP25050.html. [Accessed: 27-Jul-2017].

[9] OpenCV, “OpenCV library.” [Online]. Available: http://opencv.org/. [Accessed: 27-Jul-2017].

[10] Emgu CV, “Emgu CV: OpenCV in .NET (C#, VB, C++ and more).” [Online]. Available: http://www.emgu.com/wiki/index.php/Main_Page. [Accessed: 27-Jul-2017].