INSTITUTO DE SISTEMAS E ROBÓTICA
Departamento de Engenharia Electrotécnica – Universidade de Coimbra
Polo II da Universidade de Coimbra
3030-290 COIMBRA, PORTUGAL
Tel: +351 239 796 200

Manual Técnico da RobChair

Luís Filipe Rodrigues Alves

Coimbra, Junho de 2008

Conteúdo

Lista de Figuras

Capítulo 1

1 Introdução

A ***ROBCHAIR*** é um robô móvel diferencial sobre a forma de uma cadeira de rodas, destinado ao teste e desenvolvimento de novas plataformas de controlo com vista ao melhoramento das condições de locomoção das pessoas. Esta poderá ser totalmente autónoma, permitindo assim uma maior capacidade de autonomia aos seus utilizadores.

A sua arquitectura é baseada num sistema distribuído e modular, tanto ao nível do hardware como do software. Basicamente esta assenta sobre um barramento de comunicação CAN, onde se encontram ligados os diversos módulos de hardware/software que irão ser descritos mais em detalhe nos próximos capítulos.

Capítulo 2

2 Descrição do Sistema

Para além de uma estrutura muito semelhante a uma cadeira de rodas motorizada convencional, a RobChair foi dotada de outros sensores e o modo de interacção do utilizador com a mesma é também diferente, podendo ser semi-autónoma ou autónoma.

2.1 Estrutura Física

A sua estrutura assenta numa típica cadeira de rodas motorizada (baterias, motores, drives de potência, joystick), à qual foram efectuadas algumas alterações, que permitissem comportar a integração de todo os novos sistemas, tais como: computador, dispositivos sensoriais e os diversos módulos de CAN para controlo do sistema. O seu aspecto actual, pode ser verificado nas imagens seguintes:

Figura 2.1: Vista frontal da RobChair

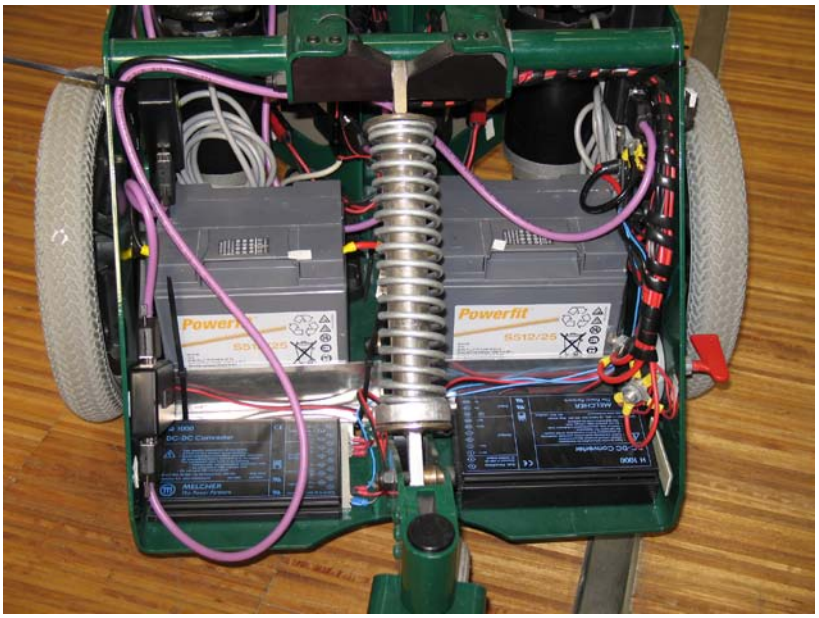

Figura 2.2: Vista da retaguarda da RobChair

Em termos de dimensões esta apresenta aproximadamente 66cm de largura, 105cm de comprimento e 90cm de altura, estando o seu peso distribuído simetricamente. A cadeira assenta em duas rodas motrizes e possui ainda três rodas castores, duas frontais e outra na retaguarda acoplada a um sistema de mola, para permitir a estabilidade do sistema. É actuada recorrendo a dois motores DC de 24V com respectivo desmultiplicador de velocidade com factor de "1:10", alimentados por duas baterias de 12V e controlados por dois *drivers* de potência que garantem o seu efectivo controlo de forma independente, obtendo um binário sensivelmente de 29.3 Nm em cada roda.

2.2 Constituição do Sistema

De seguida apresenta-se o sistema da RobChair, em termos de hardware, ou seja, são apresentados todos os componentes que constituem o sistema de navegação autónoma e a forma como estes estão interligados. É ainda descrito o fluxo de dados entre estes.

2.2.1 Constituintes do Sistema

O sistema de navegação autónoma é então constituído por:

- ✓ 1 - PC Embutido Advantech PCM9577

Figura 2.3: PC Embutido

- ✓ 1 - Placa de CAN PCM-3680

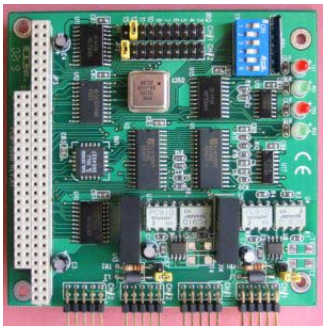

Figura 2.4: Placa de CAN

- ✓ 2 - Bateria 12V (30 Ah)

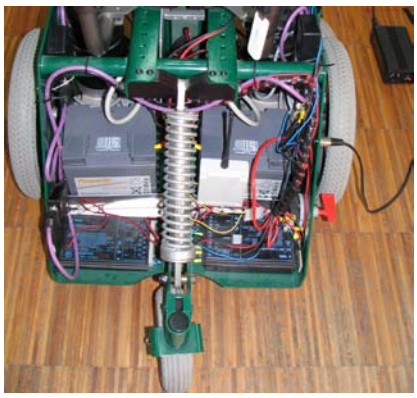

Figura 2.5: Baterias

- ✓ 6 - Módulos de interface com PIC18F258 (PIC_Base)

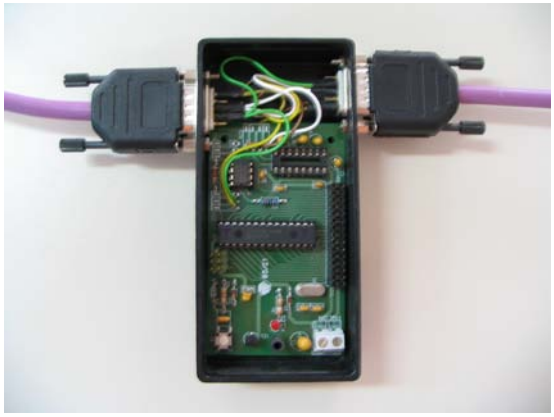

Figura 2.6: Pic Base

- ✓ 2 - Motor / Driver de Potência /Módulo de Hardware de interface com PIC_Base

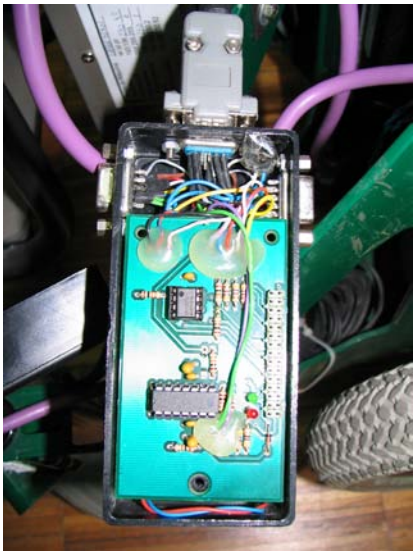

Figura 2.7: Módulo de interface com o *driver* de potência

- ✓ 2 - Encoder / Módulo de Hardware de interface com PIC_Base

Figura 2.8: *Encoder* do motor

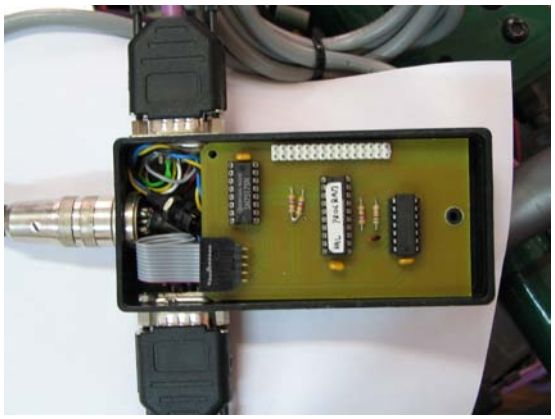

Figura 2.9: Módulo de interface com o *encoder*

✓ 1 - Joystick / Módulo de Hardware de interface com PIC_Base

Figura 2.10: Joytick

Figura 2.11: Módulo de interface com o *joystick*

2.3 Fluxo de Dados

A Figura 2.12 mostra a forma como se interligam os vários constituintes que foram apresentados anteriormente e a forma como se faz a troca de informação entre estes.

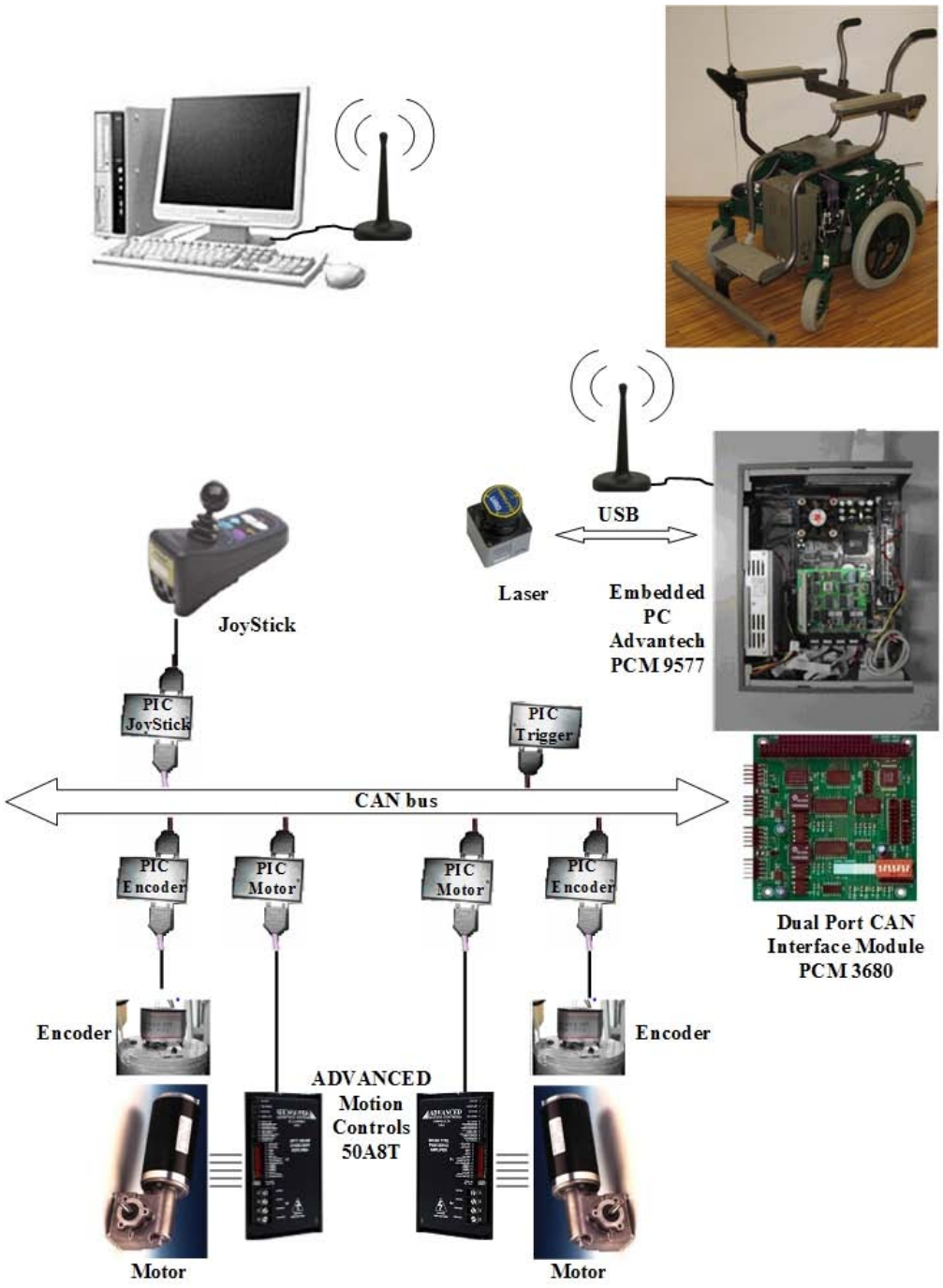

Figura 2.12: Fluxo de Dados

Como é perceptível, toda a informação circula entre os vários PIC's (*Programmable Interface Controller*) e o PC (*Personal Computer*), através de um barramento CAN (*Controller Area Network*). Funcionando o PC como supervisor/planeador, este

encontra-se num nível hierárquico superior ao dos PICs, recebendo informação dos mesmos e transmitindo-lhes ordens para controlo de todo o sistema.

Neste caso como temos uma placa de CAN com duas portas no PC, temos uma disposição do barramento de CAN em forma de anel, começando e terminando no PC. Neste momento é possível usar apenas uma porta CAN, para leitura e escrita do sistema e seu controlo. A outra é usada mais para *debug* e futuramente poderá ser usada num sistema mais complexo em que se pretenda mais robustez e fiabilidade do sistema.

Como a identificação de cada dispositivo de CAN é feita de forma inequívoca e respectiva prioridade de comunicação no barramento a sua ordem de ligação nesse mesmo barramento é irrelevante.

Neste momento essa disposição é a seguinte:

PC(CAN_0)[1]→Magnetos[2]→Sincronismo[3]→EncoderEsquerdo[4]→Alimentação[5]→EncoderDireito[6]→Joystick[7]→PowerDriveDireito[8]→PowerDriveEsquerdo[9]→PC(CAN_1)[10]

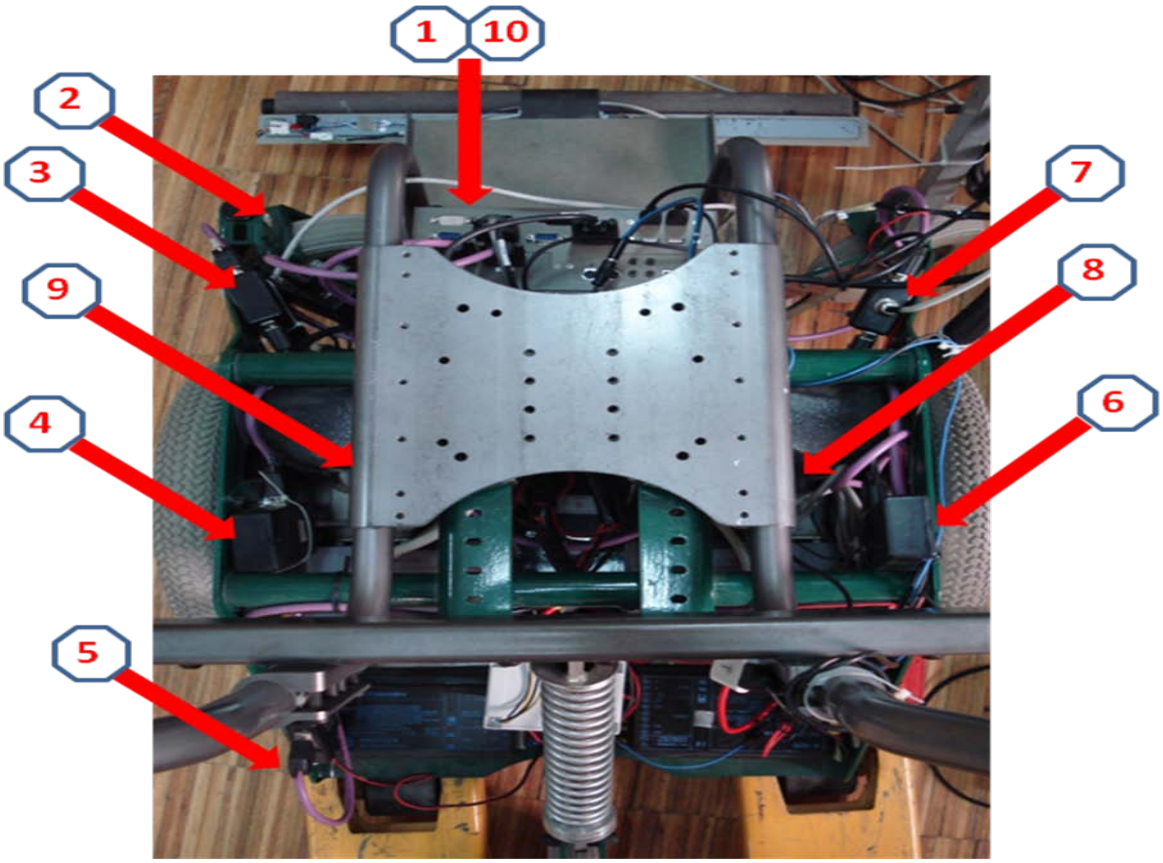

Figura 2.13: Disposição dos componentes na RobChair

2.4 Sistema de Alimentação

O sistema de alimentação é composto por três níveis de tenções distintas. Como fonte de energia temos portanto as duas baterias de 12V em série para obter os 24V utilizados pelos *PowerDrives* que alimentam e controlam os motores respectivamente assim como o PC e os dois *PowerConverters* e depois disponibilizam os 12V e 5V.

Os 12V ficam logo disponíveis após que seja ligado o corte geral, enquanto que para ter os 5V disponíveis é necessário ainda ligar um outro interruptor mais pequeno preto ligeiramente acima do de corte geral que corresponde ao *enable* do respectivo *powerdrive*. Uma vez que este botão é de difícil acesso quando estamos sentados na RobChair, existe ainda um outro de pressão que se encontra na consola do joystick para casos de emergência (botão redondo vermelho), estando este nomeadamente em série com o outro.

Os dispositivos que usam os 5V, são os lasers, e todos os sistemas que estão ligados ao barramento de CAN. Dado que é possível que a RobChair seja totalmente autónoma e que ninguém vá nela para a parar em caso de necessidade, o sistema mais básico de segurança que existe é que ela para de andar no caso de corte dos 5V. Por esse motivo, existe um interruptor local onde se dá entrada da alimentação para o barramento de CAN, e ainda um outro controlado remotamente via rádio frequência.

Os 12V servem para alimentar nomeadamente, o dispositivo de rádio frequência, o mini-tft e o sensor inércial.

No esquema eléctrico da RobChair, que é descrito como blocos na figura seguinte, pode-se visualizar a ligação dos componentes. Cada bloco é representado por um numero que corresponde às imagens dos dispositivos.

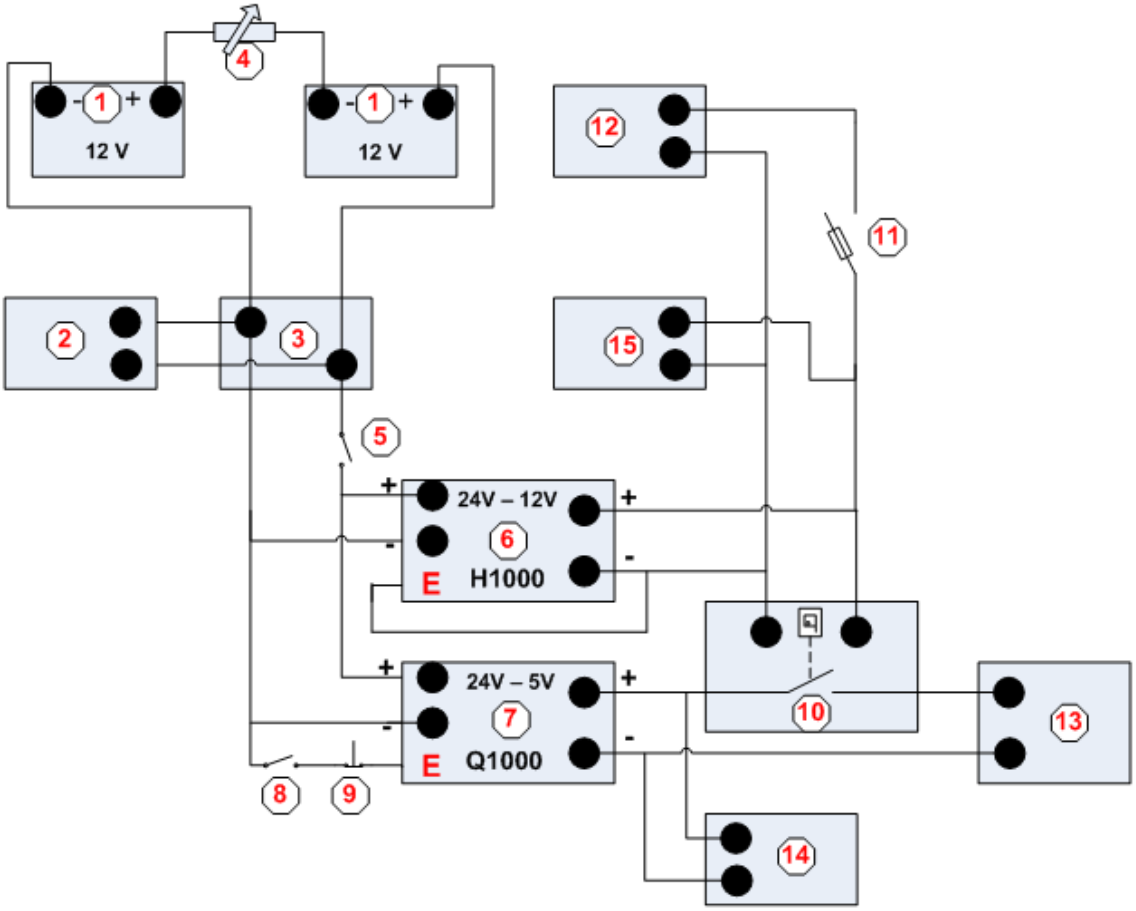

Figura 2.14: Esquema eléctrico

➢(1) →Bateria 12V

Figura 2.15: Bateria

➢(2) →Carregador Baterias

Figura 2.16: Carregador

➢(3) →Barramento de ligações (24V)

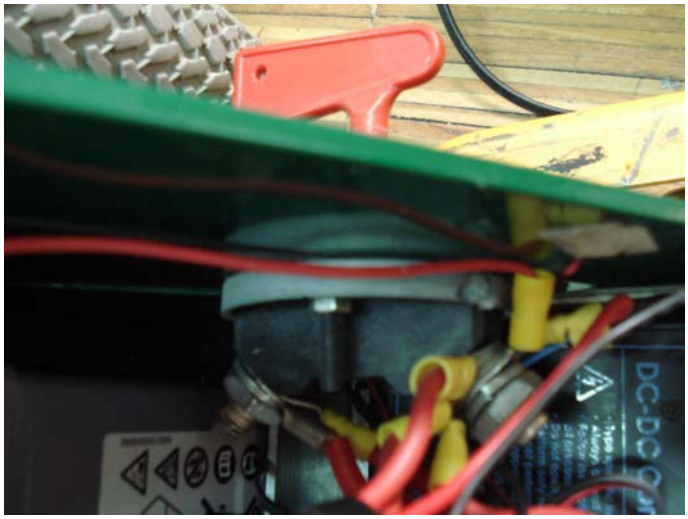

Figura 2.17: Barramento de ligações

➢(4) →Corta-Fusível com rearme manual

Figura 2.18: Corta fusível

- (5) →Corte Geral (Interruptor rotativo vermelho)

Figura 2.19: Corte geral

- (6) →Conversor DC-DC 24V-12V (H1000)

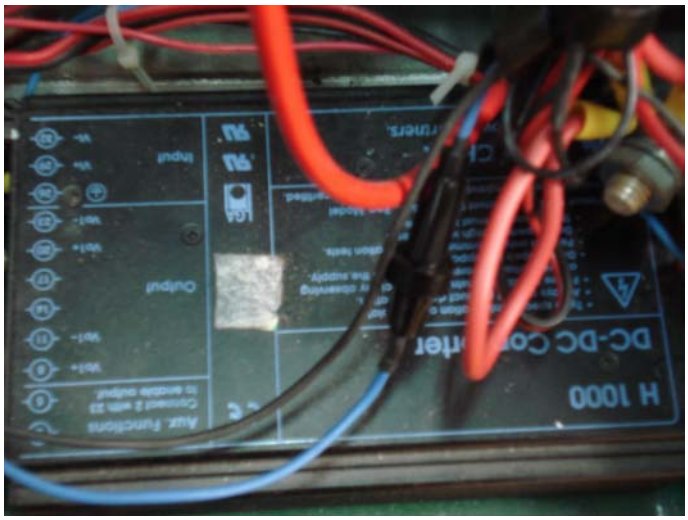

Figura 2.20: Conversor dc-dc H1000

- (7) →Conversor DC-DC 24V-5V (Q1000)

Figura 2.21: Conversor dc-dc Q1000

- (8) →Activação do conversor DC-DC Q1000 (Interruptor simples)

Figura 2.22: Interruptor simples e ligação do carregador

- (9) →Botão de emergência vermelho

Figura 2.23: Joystick e botão de emergência

- (10) →Rádio controle (controlo remoto)

Figura 2.24: Controlo remoto

- (11) →Fusível protecção do monitor (2A)

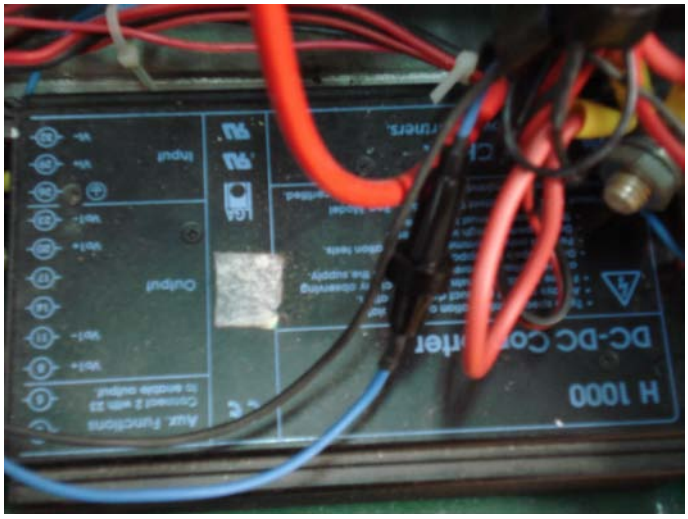

Figura 2.25: Porta fusível

- (12) →Monitor (mini-tft)

Figura 2.26: Mini TFT

- (13) → Alimentação do Bus de CAN e dispositivos presentes nele

Figura 2.27: Interruptor de alimentação dos módulos de CAN

- (14) → Alimentação suplementar de 5V (Laser)

Figura 2.28: Ligação da alimentação do laser

- (15) → Alimentação suplementar de 12V (Sensor Inércial)

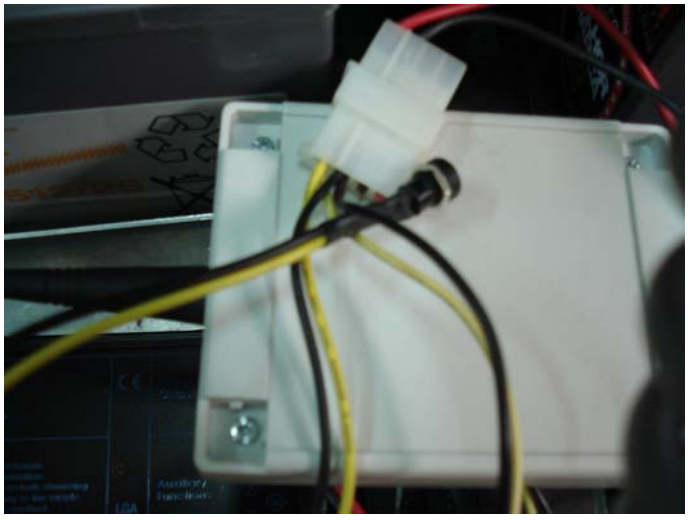

Figura 2.29: Ligação da alimentação do sensor inércial

2.5 Recomendações de Uso

Dado que existem duas baterias em série no sistema para se obter os 24V necessários, e uma vez que o carregamento dessas mesmas baterias é feito também com estas em série, ocorre uma situação que embora possível não é ideal. Essa situação é o facto de uma das baterias carregue menos que a outra, o que resulta numa descarga também assimétrica. Perante tal situação é recomendável que as baterias sejam trocadas de posição de 3 em 3 meses, ou menos caso se verifique um número de recargas considerável durante esse tempo.

É também preferível que todo o sistema esteja desligado no interruptor geral (alavanca vermelha), antes de se proceder à recarga das baterias, apenas como segurança.

Dado que o sistema tem vários interruptores (três em locais distintos), para os diversos níveis de tensão/módulos, como em qualquer instalação eléctrica, é recomendável que se ligue o geral e depois os restantes. Quando se desliga o sistema é conveniente proceder pela ordem inversa.

2.6 Colocação em funcionamento/Diagnóstico do sistema

Para colocação do sistema em funcionamento deve-se proceder à ligação da alimentação de todo o sistema (ligando para tal os três interruptores existentes). Em condições normais o sistema distribuído está neste momento operacional, sendo possível verificar esse funcionamento fazendo a leitura simples do barramento de CAN, tanto na porta can0 como na can1 do PC. Em situação normal os módulos de CAN e RTAI são iniciados conjuntamente no arranque do Linux.

O sistema de CAN funciona a 250 kbits/s, como essa é a baud-rate por defeito da placa de CAN do PC, não é necessário reconfigurá-la. Basta portanto executar o programa "./receive can0" ou "./receive can1" para que surja no ecrã várias mensagens, uma por cada pic presente no sistema. Existem seis mensagens com id's de mensagem diferentes e que se repetem ciclicamente. Caso isso não ocorra significa que algo não está correcto.

Na possibilidade de por algum motivo haver a omissão desta mensagem inicial do pic, que corresponde ao seu estado de boot-loader, como é explicado no capítulo 3.1.2 mais em pormenor, tal pode significar em último caso a necessidade de substituir/reprogramar o boot-loader do pic. A programação do boot-loader do pic, é feita recorrendo ao programdor da microchip ICD2 e o interface gráfico MPLab, fazendo a importação do ficheiro ".hex" correspondente ao BootFirmware do pic pretendido, que se encontra na pasta referente ao *firmware*.dos pic's. Não fazer confusão com o programa principal, o *firmware* serve apenas de identificação do pic, para posterior programação do código principal via CAN, existindo dois programas distintos para cada pic.

Outra situação possível de ocorrer é o código que programamos via CAN (código principal), necessitar de ser reprogramado novamente, esta situação é facilmente

detectável da seguinte forma, caso após a ordem via CAN de mudar o estado de "boot" para "run" do pic, este continuar a apresentar a mensagem inicial de boot.

Para programação via CAN e dar ordem ao pic para mudar o seu estado de "boot" para "run", tal é explicado no capítulo 3.1.2.

Estando tudo operacional no barramento de CAN inicia-se o programa de RTAI, para tal entra-se numa consola (user: root, passwd: rob03) e na directoria *"cd /home/robchair/exec"*.está o programa de leitura de CAN e o programa de RTAI. Ao executar seleccionar a opção 1 (correspondente ao modo de joystick), para constatar a operacionalidade do sistema.

Existem outras opções não tão óbvias em termos de funcionamento. Para novos algoritmos o modo de joystick é a melhor base, pois contem a parte de inicialização do CAN e a interacção com os controladores de velocidade dos motores.

O código fonte encontra-se na pasta *"cd /home/robchair/RTrobchair"*, para compilar execute *"./autogen.sh;make"*, sendo o executável gerado dentro da pasta *"src"* e tem o nome de "rtrobchair". Para correr o programa basta o comando *"./rtrobchair"*.

Capítulo 3

3 Arquitectura Distribuída

A Arquitectura Distribuída é composta por vários componentes que visam a sua modularidade e versatilidade, levando no entanto à necessidade de obedecer a algumas regras para que haja determinismo e coerência na informação recebida e transmitida por cada módulo do sistema, sendo para tal a arquitectura baseada num sistema de tempo real.

O sistema global é composto por três camadas, presentes na Figura 3.1, que são: comunicação de baixo nível, camada de controlo de baixo nível e no topo a camada responsável pela navegação e percepção do meio em seu redor.

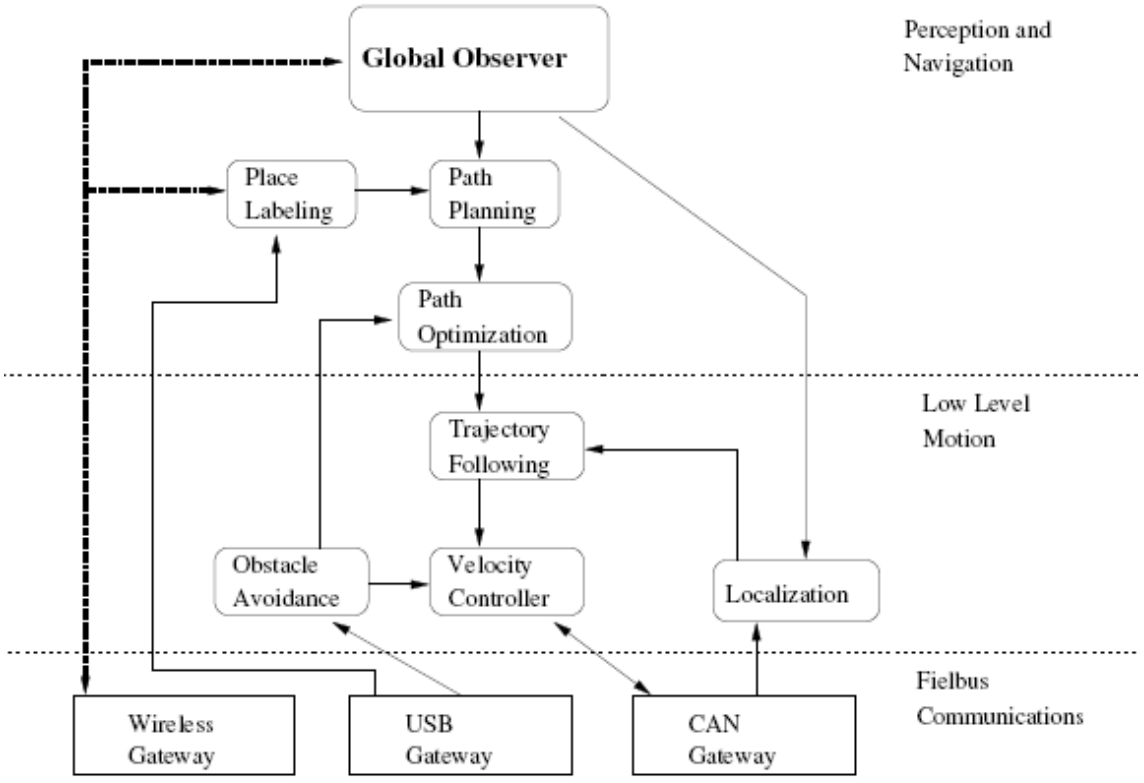

Figura 3.1: Arquitectura de Software

Dada a modularidade da arquitectura tanto a nível de software como de hardware, existe portanto diversos módulos. Cada módulo é composto por hardware projectado para

determinada função e um microcontrolador para permitir a interacção com todo o sistema via CAN e o hardware que tem a seu cargo. A arquitectura de hardware é ilustrada na Figura 3.2, onde é perceptível os diversos componentes de hardware que a compõem.

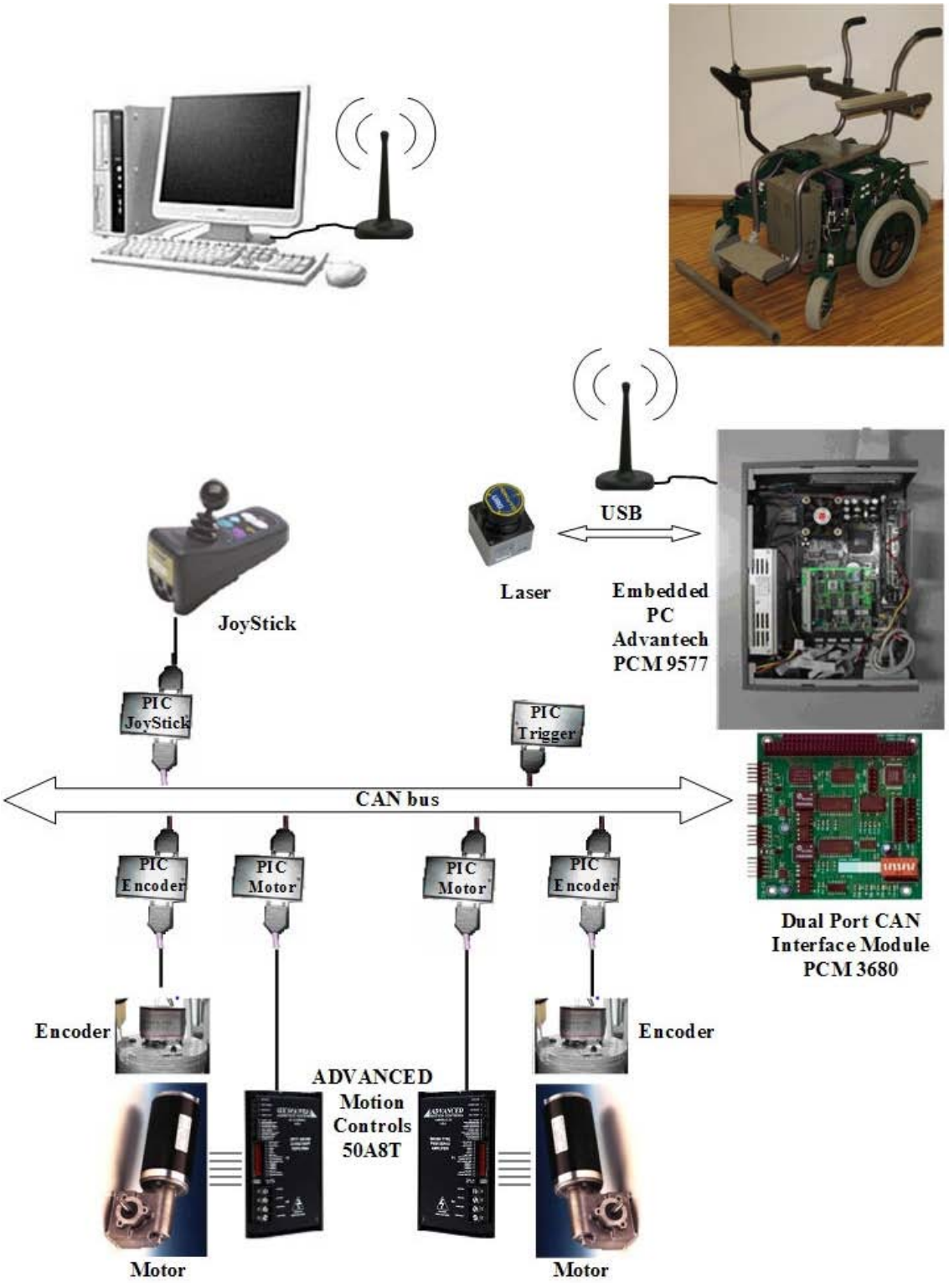

Figura 3.2: Arquitectura de hardware

O Programa utilizado para desenvolver o código para os microcontroladores é o HITECH, uma vez que este trabalha em Linux, ao contrário do MPLab®, utilizando-se a linguagem C, como linguagem de programação. Na camada superior de tempo real presente no PC é usada a linguagem C++, sendo cada módulo uma classe de forma a ter o sistema modular.

3.1 Módulo de Processamento "PIC_Base"

O módulo de processamento "PicBase", é composto pelos componentes necessários ao funcionamento do microcontrolador, à comunicação no barramento de CAN e à expansão das suas capacidades através do interface com o módulo de hardware para o qual está programado para gerir.

O uso dos microcontroladores traz muitas vantagens para este tipo de sistemas onde existe a necessidade de constante evolução, permitindo flexibilidade. Estes oferecem inúmeras capacidades tais como de tratamento, análise e comunicação de dados/sinais, e uma panóplia de periféricos integrados que evita assim muita electrónica externa. Este facto confere mais flexibilidade às aplicações que integram, recaindo sobre este tipo de dispositivo a escolha para o desenvolvimento dos vários componentes que constituem o sistema.

3.1.1 Características Gerais do PIC Utilizado

Os microcontroladores utilizados neste sistema são da família PIC18FXX8, sendo o modelo utilizado neste trabalho o PIC18F258, o qual apresenta já bastantes periféricos integrados, como comunicação via RS232, CAN, SPI, ou mesmo ADC e geradores de PWM, (os periféricos utilizados serão apresentados mais à frente). Pode ver-se na Figura 3.3 o diagrama de pinos do Microcontrolador:

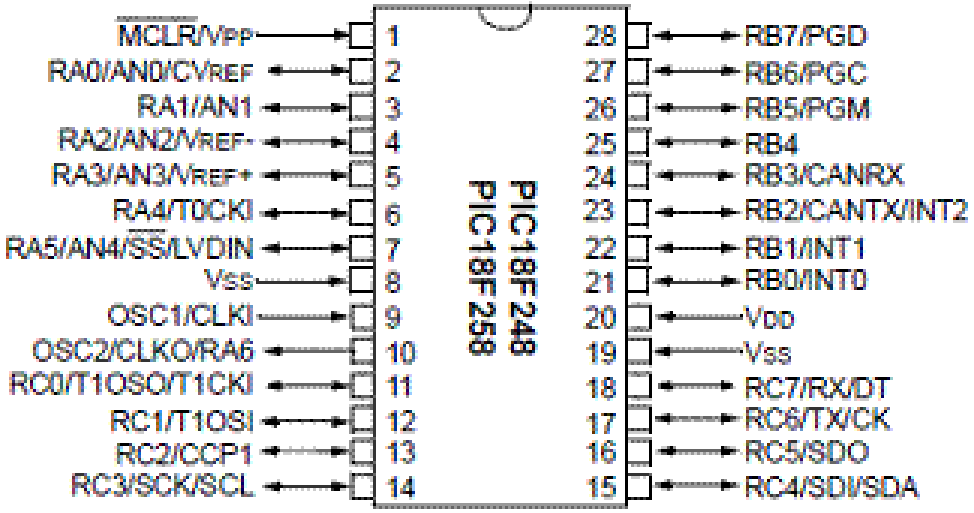

Figura 3.3: Diagrama de pinos do PIC18F2x8

Como é perceptível pelo "*pin diagram*", este microcontrolador apresenta 3 portos de I/O (Porto A, B e C), sendo que estes se encontram quase na sua totalidade multiplexados com outras funções associadas aos vários periféricos presentes no microcontrolador. Regra geral quando um determinado periférico está activo os pinos associados a este não podem ser utilizados como pinos genéricos de I/O.

3.1.2 Módulo de Hardware "*PIC_Base*"

A "*PIC_Base*" é um módulo de hardware, utilizado para fazer a interface entre os PIC's e os vários módulos de hardware desenvolvidos. Este módulo de interface tem várias capacidades. Destaca-se, a possibilidade de programação do PIC através do programador disponível da Microchip(R), o MPLab ICD 2(R) sendo este utilizado na fase inicial para programação do *boot-firmware* de cada PIC, que incorpora a sua identificação única no sistema. Para além da possibilidade de programação, o módulo dispõe ainda de uma ligação fácil a todos os portos por meio de uma ficha de 20 pinos e do hardware periférico necessário para comunicação RS232 e CAN, sendo esta última também explorada para a programação do microcontrolador, recorrendo para tal ao programa "canbootmngr_v2", uma vez já residindo no módulo o seu respectivo *boot-firmware.*

A identificação dos diversos módulos nesta fase é expressa de seguida assim como um exemplo de utilização.

Tabela 3.1: ID’s utilizados pelo programa “canbootmngr_v2”

CanBootManager	Group	PIC	Msg Boot
PDriveNode	1	1 (Left) / 2 (Right)	0x21/0x22
EncoderNode	2	1 (Left) / 2 (Right)	0x41/0x42
JoystickNode	4	1	0x81
TriggerNode	6	1	0xC1

O programa “canbootmngr_v2” é usado essencialmente para programação das funcionalidades de cada PIC via CAN, permitindo desta forma programar todos os PICs sem ter de conectá-los um a um ao programador da Microchip. Para além de programar é possível transmitir uma ordem aos PIC´s para saírem do estado *“Boot Loader Mode”* e passarem para *“Idle Mode”* através do programa “canbootmngr_v2”. O comando completo para programar um determinado PIC com o ficheiro “.hex” criado pelo compilador, neste caso o TriggerNode é o seguinte: “./canbootmngr_v2 -g 6 -p 1 -d -f TriggerNode.hex”. A interpretação do comando significa fazer o *download* do código para o PIC com a identificação expressa.

Os microcontroladores presentes em cada módulo possuem três modos de operação, sendo dois deles já enumerados anteriormente, nomeadamente *“Boot Loader Mode”*, *“Idle Mode”* e também *“Running Mode*”. O *“Boot Loader Mode”* é o primeiro estado em que este fica quando é alimentado, permitindo a sua reprogramação de forma simples e acessível sem ter de mexer no hardware. Passando para *“Idle Mode”* este fica num estado de espera, em que o seu normal funcionamento é suspenso. Enquanto em *“Running Mode*” este efectua o devido controlo do que o rodeia e adquire a devida informação do sistema para posterior envio para o sistema central. Existindo no entanto uma ordem pela qual os diferentes estados podem passar, a qual é expressa no diagrama seguinte.

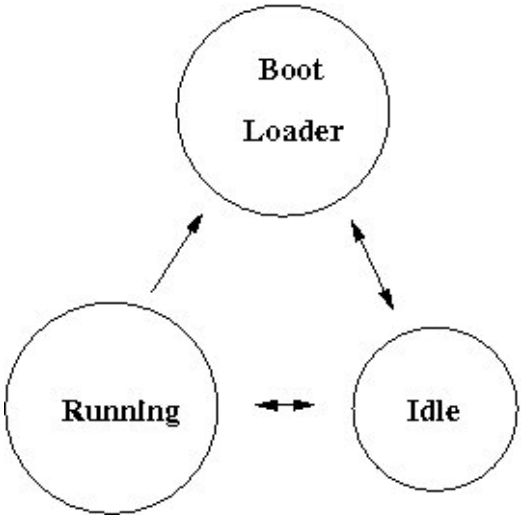

Figura 3.4: Modos de operação

3.1.3 Detalhes da Comunicação CAN

Para que os vários dispositivos troquem a informação correctamente sob o protocolo CAN, existe a necessidade da definição da estrutura das mensagens. Sendo a estrutura global de uma mensagem segundo o protocolo CAN como se retrata na Figura 3.2, definiu-se a estrutura específica das mensagens, a qual é adequada às necessidades existentes neste sistema.

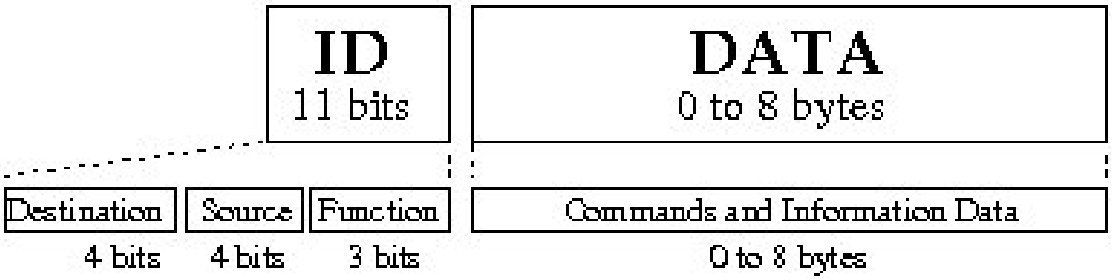

Figura 3.5: Protocolo CAN utilizado

Como pode ser observado, o ID da mensagem é único uma vez que é composto pelo destino, origem e a função pretendida para o módulo de destino. Em termos de ID dos módulos estes estão descritos na tabela seguinte, no caso da função esta é dependente do módulo em questão sendo portanto apresentada na secção do módulo a que respeita, assim como o campo de dados, cujo tamanho também ira depender dessa mesma função.

Tabela 3.2: ID's de *"Destination/Source"* usados no protocolo CAN

ID	Name Node
0	PC
1	Right PDrive
2	Left PDrive
3	Both PDrive
4	Right Encoder
5	Left Encoder
6	Both Encoder
10	Joystick
15	syncMCU

Como se poderá verificar mais à frente, existem duas funções comuns a todos os módulos, que são nomeadamente o "*Turn Node OFF → Idle Mode*" e "*Turn Node ON → Running Mode*", as quais são essenciais, correspondendo o ID "0" e "1" respectivamente.

Para além da identificação de cada módulo no barramento de CAN, para que tenhamos um sistema fiável e determinista é necessário que todas ou a sua maioria obedeça a determinados instantes de tempo na comunicação dos dados e operações de controlo.

Existe portanto uma sequência temporal das várias acções a serem executadas no sistema, como é ilustrado na figura seguinte.

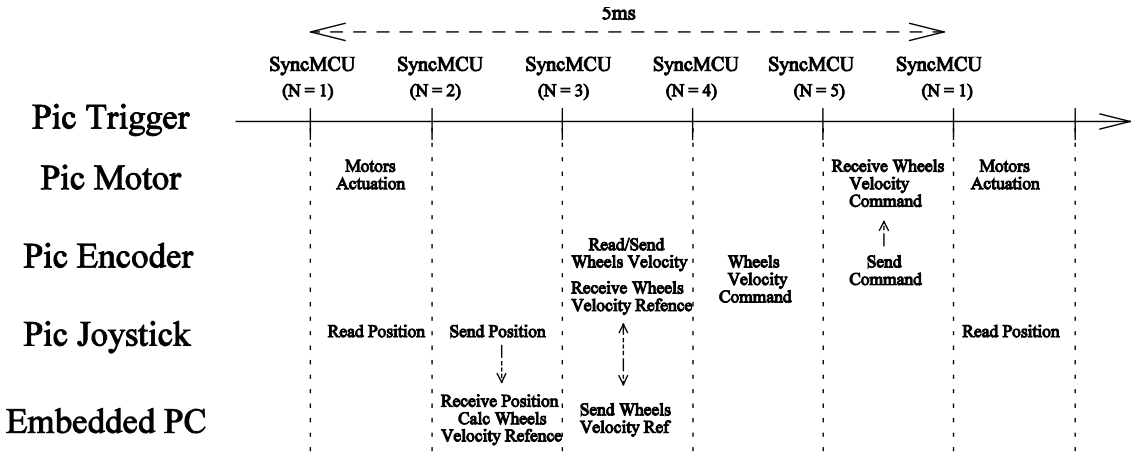

Figura 3.6: Diagrama temporal de acções

3.2 Módulo de Software/Hardware "PDrive_Interface"

3.2.1 Software

De seguida apresentam-se as funções de CAN para este módulo:

Tabela 3.3: ID's das funções presentes no módulo PDrive

OperationNumber	Function
0	Turn Node Off
1	Turn Node On
2	Set DAC Command
3	Set Control Mode
4	Data from Motor

A função "*Set DAC Command*", é usada para definir o valor a transmitir ao DAC que por sua vez irá corresponder a uma velocidade do motor, sendo esse valor composto por 2 bytes (DAC de 10 bits de resolução).

Existem dois modos de funcionamento de comandos para o motor: modo directo e modo de velocidade. Estes são comutados pela função "*Set Control Mode*". No primeiro caso o comando do motor pode vir directamente do joystick, por exemplo, não sendo garantido que esse comando corresponda sempre à mesma velocidade (controlo em malha aberta). No segundo caso isso é possível uma vez que existe um controlador PI no módulo *EncoderNode* que irá assegurar o controlo de velocidade. Os comandos do motor são então enviados por este módulo e não pelo PC ou joystick. Os dados da mensagem serão "0" ou "1", de acordo com o modo de funcionamento desejado.

Os valores monitorizados pelo módulo totalizam seis bytes, em que dois bytes correspondem ao valor lido pelo ADC, que corresponde à saída do sinal de comando, seguido de mais dois bytes referentes à corrente do motor e por fim mais dois bytes correspondendo ao valor de comando recebido pelo módulo. Estes são os dados transmitidos sobre a função "*Data from Motor*".

Módulo do PDrive

O fluxograma seguinte mostra resumidamente a estrutura do código desenvolvido para o PIC responsável pelo envio de comandos para o motor, o qual irá corresponder a uma determinada velocidade.

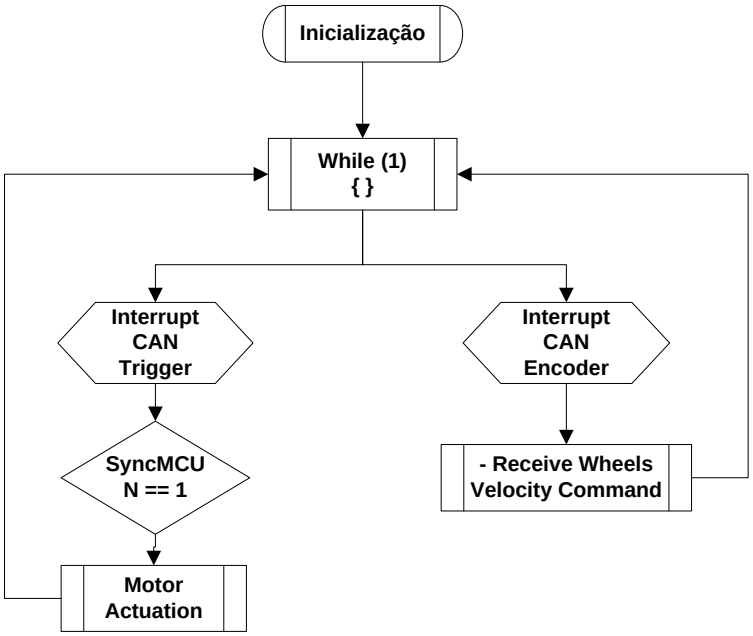

Figura 3.7: Fluxograma do código implementado no *PDrive Node*

Como é visível pelo fluxograma anterior, o *PDrive* recebe através do barramento CAN os comandos de controlo do processo provenientes da camada de alto nível. Visto isto, será apresentado apenas a parte do código que permite a recepção desses mesmos comandos.

O código responsável pela descodificação das mensagens de CAN com destino ao *PDrive*, é o seguinte.

```
void interrupt low_priority LowPriorityInterrupts(void)
{

    //! Process received CAN message
    //! Process Id
    CANMsgIn.stdID = (*ptr++) << 3;
    CANMsgIn.stdID |= ( (*ptr++) >>5 ) & 0x07;

    //! Process Data
    ptr += 3;
    CANMsgIn.len = ( *(ptr-1) & 0x0F );
    for ( i=0; i<CANMsgIn.len; i++ )      // copy of can message to buffers
      CANMsgIn.data[i] = ptr[i];

    //! Check if it is a Trigger Message
    if (CANMsgIn.stdID == 0x0707)
    {
```

```
      //! If Pdrive is ON update DAC value
      if (PDriveState == ON)
        Send_PDrive_Comm(DAC_Command.w);
    }

    if((((CANMsgIn.stdID  & 0x780) >> 7) == BOTH_PDRIVES) || // CAN Message sent to both
       (((CANMsgIn.stdID  & 0x780) >> 7) == PDRIVE))  {      // PDrive or to this one

   // Sets a reference command to module

     if ((((CANMsgIn.stdID  & 0x780) >> 7) == BOTH_PDRIVES) && (PDRIVE == LEFT_PDRIVE))
     {
       DAC_Command.b.b0 = CANMsgIn.data[2];
       DAC_Command.b.b1 = CANMsgIn.data[3];
     }
     else {
       DAC_Command.b.b0 = CANMsgIn.data[0];
       DAC_Command.b.b1 = CANMsgIn.data[1];
     }


   ...

  }
}
```

Quando é recebida uma mensagem de CAN esta provoca uma interrupção no programa. È então identificado o *ID*, o seu tamanho e conteúdo e copiado para uma estrutura de dados. Com base no *ID* é verificado se é uma mensagem vinda do *trigger*, e caso seja, é feita a actualização do valor de saída do DAC para actuação do motor. Caso não seja uma mensagem do *trigger* é verificado o destino da mensagem, se for para o respectivo módulo é efectuada a operação, nomeadamente pode ser para actualização do valor da variável que guarda o valor de comando desejado para o DAC.

3.2.2 Hardware

Dadas as necessidades que se apresentam na adequação/transformação dos sinais de controlo do driver de potência, o módulo PDrive_Interface foi desenvolvido de modo a servir de interface entre o driver de potência e o módulo "*PIC_Base*", que faz o interface com o PIC responsável pelo envio do sinal de comando para o motor.

As funções principais deste módulo são então:

- Tratar o sinal de comando enviado pelo PIC segundo o protocolo SPI, de modo a transformar posteriormente num sinal analógico de -10V a 10V com recurso a um DAC, para colocar na entrada de comando do "drive de potência". É ainda enviado para o PIC um sinal analógico de 0V a 5V, resultado da conversão D/A, que serve de monitorização do sinal de saída.

- Tratar os sinais de *Enable* a enviar da PIC_Base para o drive de potência, de forma a controlar a inibição do driver de potência, nomeadamente activar ou não

os andares de potência do conversor (Ponte em H), que permite a operação do motor num só sentido, em ambos ou em nenhum.

Tratamento do Sinal de Comando

O sinal de comando (velocidade pretendida) é enviado pelo PC via CAN. Depois de recebido pelo microcontrolador é tratado e enviado usando o protocolo SPI para o DAC. Existe então a necessidade da utilização de um conversor Digital/Analógico (DAC) para a conversão desse sinal digital num sinal analógico. Utilizou-se o DAC TLV5616 (*12-bit voltage output digital-to-analog converter*), como se pode observar na Figura 3.9. Este coloca na saída um sinal analógico VDAC, de 0V a 5V. Este sinal é reenviado para o PIC para o porto RA0, onde vai ser utilizado um ADC interno de modo ao sinal poder servir de auto regulação.

Figura 3.8: Conversão A/D do sinal de comando

Este sinal, tem ainda de ser tratado antes de ser enviado para o módulo de potência, uma vez que o sinal de comando a enviar, tem de ser um sinal analógico de -10V a 10V. Recorre-se para o efeito a um andar amplificador com a configuração que se pode observar na Figura 3.10, obtendo o sinal CMD_ANL à saída.

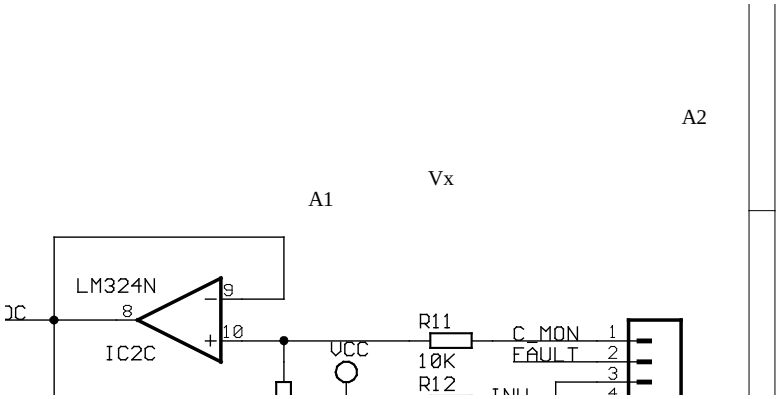

Figura 3.9: Andar amplificador para sinal de comando

O ganho da montagem amplificadora presente na Figura 3.10 é obtido recorrendo a relações presente na Equação 3.1.

$$Vx = -\frac{R2}{R1} \cdot VDAC$$
$$CMD\_ANL = -\left(\frac{R5}{R3} \cdot Vx + \frac{R5}{R4} \cdot 10 \right) \quad (3.1)$$

Módulo de Hardware "PDrive_Interface"

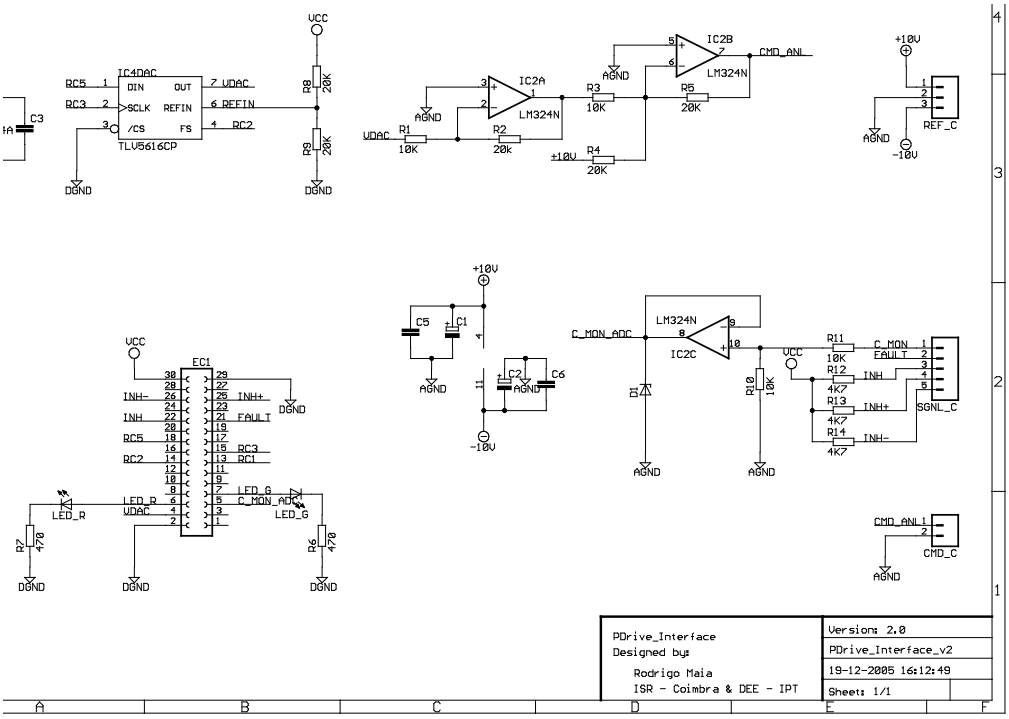

Figura 3.10: Esquemático do módulo

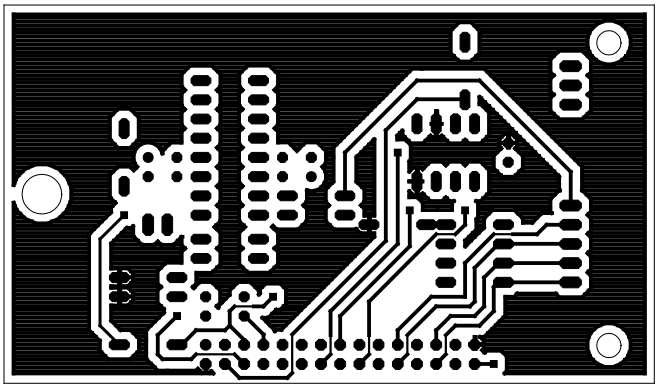

Figura 3.11: Layout do PCB – Top Layer

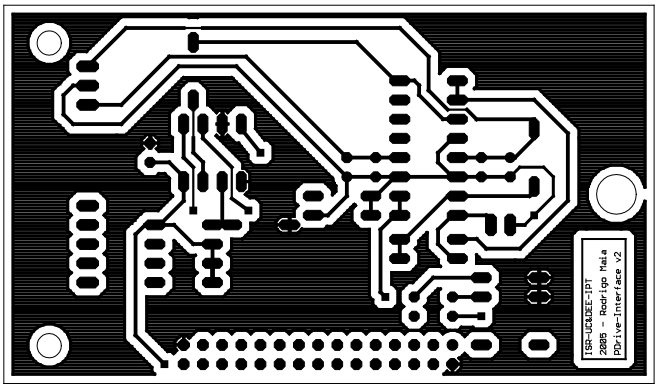

Figura 3.12: Layout do PCB – Bottom Layer

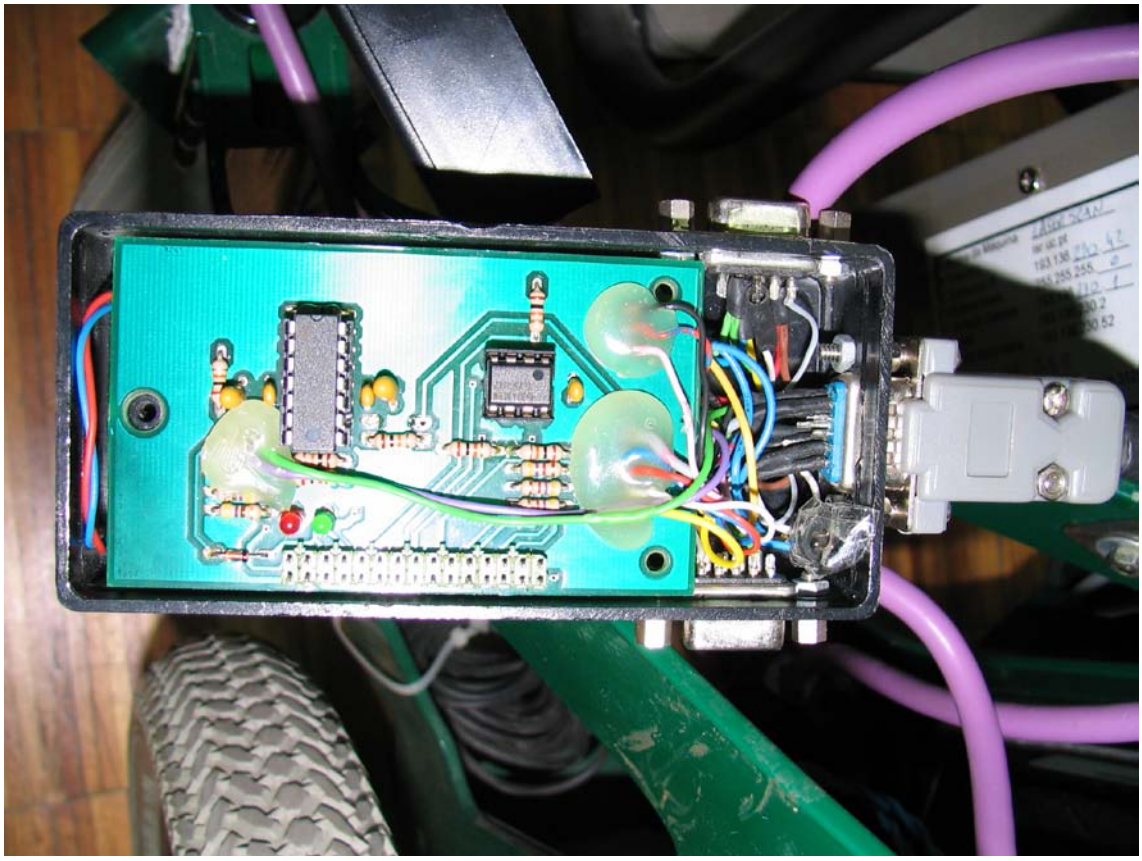

Figura 3.13: Vista real da placa

*3.3 **Módulo de Software/Hardware** "Encoder_Interface"*

3.3.1 Software

Em termos de funções de CAN para este módulo, este apresenta as seguintes:

Tabela 3.4: ID's das funções presentes no módulo Encoder

OperationNumber	Function
0	Turn Node Off
1	Turn Node On
2	Set Velocity Value
3	Set Control Mode
4	Set PI Control Values
5	Reset Encoder Information
6	Data from Encoder

A função "*Set Velocity Value*", é usada para definir o valor de referência da velocidade desejada para o motor, sendo esta controlada com base no controlador PI presente neste

módulo. Caso este não esteja em modo de velocidade, a sua activação é possível através da função "*Set Control Mode*" de igual forma como descrito no *PDriveNode*. Para definição da velocidade são usados dois bytes de dados

É também possível alterar os valores do controlador, sendo para tal necessário quatro bytes para o ganho proporcional e outros quatro para o ganho integral. Estes são valores inteiros e devem estar referenciados às milésimas de unidade. Tal alteração é apenas válida enquanto o microcontrolador estiver activo (valores voláteis), voltando depois aos valores originais na sua inicialização.

Os valores monitorizados pelo módulo totalizam seis bytes, em que quatro bytes correspondem ao valor da posição do motor, seguido de mais dois bytes referentes à velocidade do motor. Estes dados são transmitidos sobre a função "*Data from Encoder*". No entanto o valor da posição pode ser reinicializado, sendo para tal usada a função "*Reset Encoder Information*".

Módulo do Encoder

O fluxograma presente na Figura 3.14 mostra resumidamente a estrutura do código desenvolvido para o PIC responsável pela recepção do valor de referência para a velocidade do motor. A leitura da velocidade real é lida por este módulo. O controlo de velocidade é realizado recorrendo a um controlador PI.

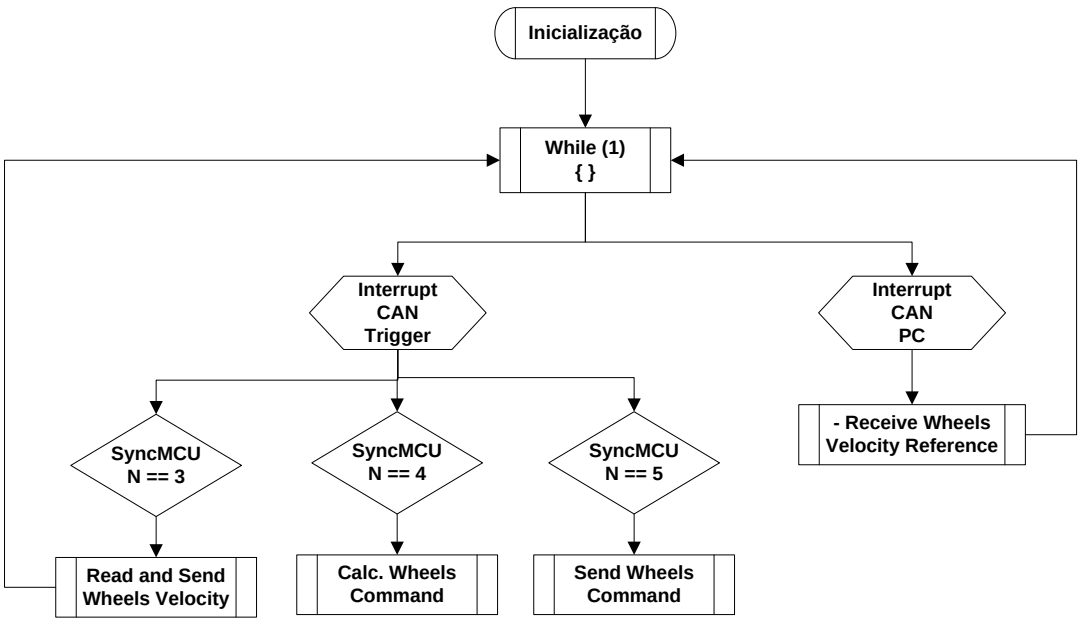

Figura 3.14: Fluxograma do código implementado no *Encoder Node*

Como é visível pelo fluxograma anterior, o Encoder recebe por CAN da camada de alto nível, a referência de velocidade para controlo do processo. Visto isto, será apresentado apenas a parte do código que permite a recepção dessa mesma referência.

O código responsável pela descodificação das mensagens de CAN com destino ao Encoder, é o seguinte.

```
void interrupt low_priority LowPriorityInterrupts(void)
{
   ...
   if((((CANMsgIn.stdID  & 0x780) >> 7) == BOTH_ENCODERS) || // CAN Message sent to both
    (((CANMsgIn.stdID  & 0x780) >> 7) == ENCODER))  {        // Encoders or to this one
   switch (CANMsgIn.stdID & 0x0007){
   case 0x00:   // Turn off module
     if (EncoderState==ON){
       TMR0ON = 0;  // Stop Timer0
       TMR1ON = 0;  // Stop Timer1
       EncoderState=OFF;
     }
    StateChanged = TRUE;
     break;
   case 0x01:   // Turn on Module
     if (EncoderState==OFF){
       TMR0ON = 1;  // Start Timer0
       TMR1ON = 1;  // Start Timer1
       //! Initialize reference value
       VelocityControl.Reference     = 0;
       //! Initialize reference value
       VelocityControl.ReferenceOld  = 0;
       //! Initialize old DAC command value
       VelocityControl.DACCommandOld = 0;
       //! Initialize old error value
       VelocityControl.ErrorOld      = 0;
       //! Initialize DACCommand
       DACCommand =  DAC_MIDLE_SCALE;
       EncoderState=ON;
     }
    StateChanged = TRUE;
     break;
   case 0x02:   // Sets a reference command to module
     if( EncoderState == ON){
       if ((((CANMsgIn.stdID  & 0x780) >> 7) == BOTH_ENCODERS) && (ENCODER ==
LEFT_ENCODER)){
         cmd.b.b0 = CANMsgIn.data[2];
         cmd.b.b1 = CANMsgIn.data[3];
       }
       else {
         cmd.b.b0 = CANMsgIn.data[0];
         cmd.b.b1 = CANMsgIn.data[1];
       }
       switch (Control_Mode) {
       case DIRECT_MODE:
         DACCommand = cmd.w;
         break;
       case VELOCITY_MODE:
         TXREG='V';
         VelocityControl.Reference=cmd.w;
         break;
       }
      ...
    }
}
```

Visto ser neste módulo que é feito o controlo da velocidade de seguida segue o código que implementa o controlador PI responsável por tal.

```
int Wheel_Velocity_Controller(int Velocity, VelCntrStruct *VelocityControl){

  int DACVelocityCommand; // New DAC command calculated
  int VelocityError;
  long auxLongDACCommand;

  // Limiting reference to the maximum and minimum values
  if (VelocityControl->Reference > MAX_VELOCITY_REFERENCE)
    VelocityControl->Reference = MAX_VELOCITY_REFERENCE;
  else
    if (VelocityControl->Reference < MIN_VELOCITY_REFERENCE)
      VelocityControl->Reference = MIN_VELOCITY_REFERENCE;

  //  Limiting the reference variation
  if (VelocityControl->Reference > (VelocityControl->ReferenceOld + MAX_VAR_REFERENCE))
    VelocityControl->Reference = VelocityControl->ReferenceOld + MAX_VAR_REFERENCE;
  else
    if (VelocityControl->Reference < (VelocityControl->ReferenceOld -
MAX_VAR_REFERENCE))
      VelocityControl->Reference = VelocityControl->ReferenceOld - MAX_VAR_REFERENCE;

  VelocityControl->ReferenceOld = VelocityControl->Reference;

  // Calculates new velocity error
  VelocityError =  VelocityControl->Reference - Velocity;

  // Calculate the DACVelocityCommand in long format
   auxLongDACCommand = (long)(VelocityControl->DACCommandOld) + (long)((VelocityControl-
>KP + VelocityControl->KI * 0.005) * VelocityError) / 10000 + (long)(((VelocityControl-
>KI * 0.005 - 2*VelocityControl->KP)/2) * VelocityControl->ErrorOld)/10000;

  // Stores DACCommand in an int format
  DACVelocityCommand = (int)auxLongDACCommand;

  // Limits the DACCommand value
  if (DACVelocityCommand > MAX_DAC_COMMAND)
    DACVelocityCommand = MAX_DAC_COMMAND;
  else
    if (DACVelocityCommand < MIN_DAC_COMMAND)
    DACVelocityCommand = MIN_DAC_COMMAND;

  // Stores Values for next sample
  VelocityControl->DACCommandOld = DACVelocityCommand;
  VelocityControl->ErrorOld = VelocityError;

  return (DACVelocityCommand);
}
```

3.3.2 Hardware

Dadas as necessidades que se apresentam na leitura dos valores dados pelo encoder e sua adequação, o módulo Encoder_Interface foi desenvolvido de modo a servir de interface entre o encoder (gerador de pulsos função da velocidade) e o módulo "*PIC_Base*". Este faz o interface com o PIC responsável pelo envio da contagem de pulsos do encoder, de forma a obter a velocidade e posteriormente a posição respeitante a cada roda, para posterior cálculo da odometria do robô.

As funções principais deste módulo são então:

- Tratar os sinais vindo do encoder de forma a contabilizar o nº de pulsos que o motor deu no sentido directo e também no sentido inverso. Sendo para tal feito uso da quadratura dos sinais do encoder.

- Mediante os valores lidos do encoder, nomeadamente a velocidade do motor, este possui um controlador PI responsável por controlar a velocidade do mesmo de acordo com a velocidade de referência dada pelo PC ou pelo Joystick.

Módulo de Hardware "Encoder_Interface"

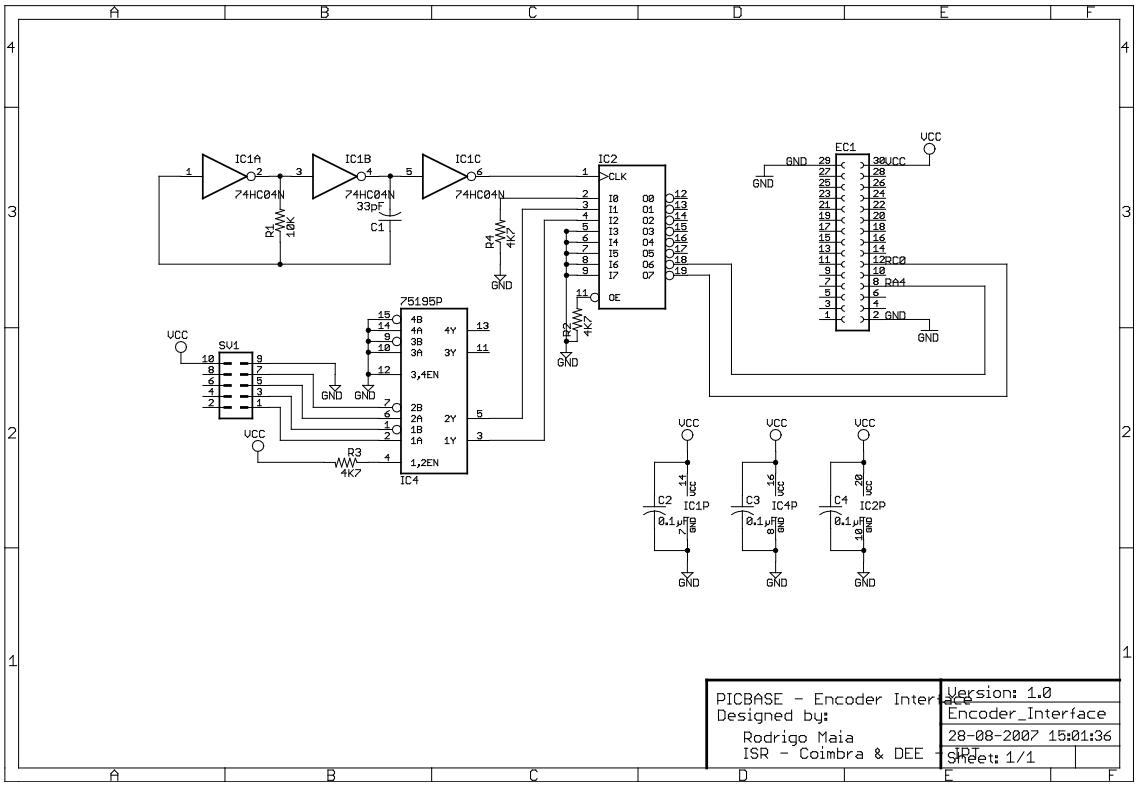

Figura 3.15: Esquemático do módulo

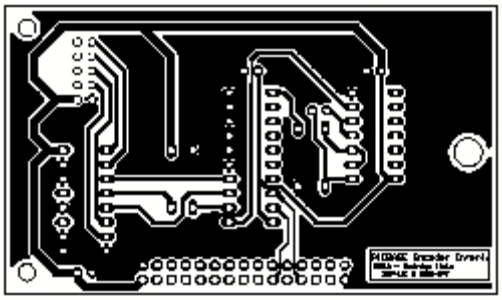

Figura 3.16: Layout do PCB – Bottom Layer

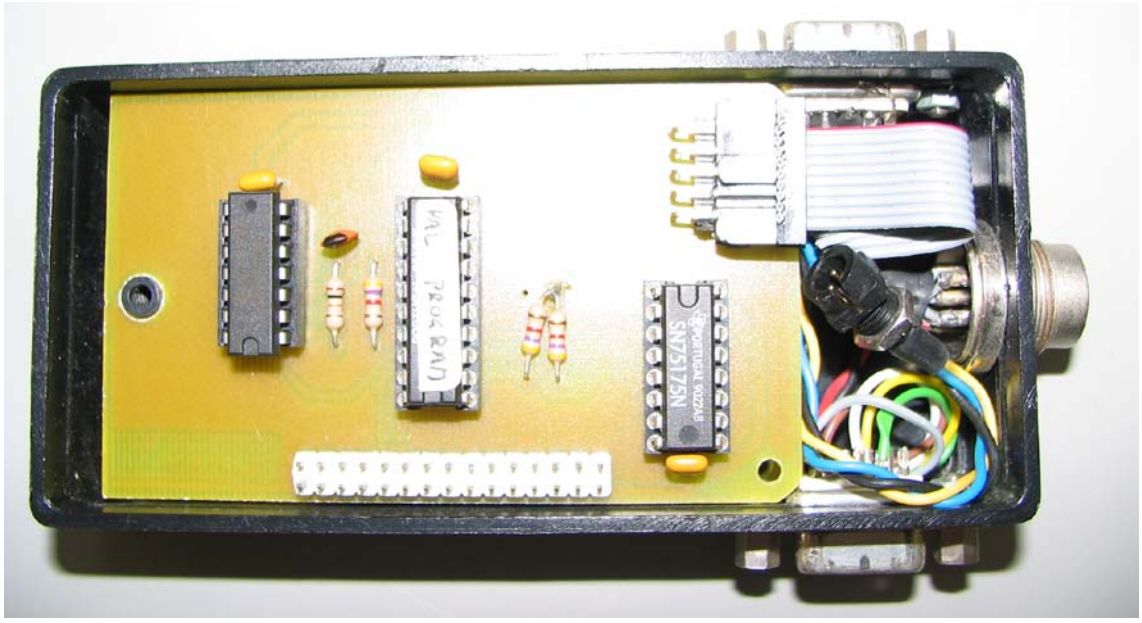

Figura 3.17: Vista real da placa

*3.4 **Módulo de Software/Hardware** "Joystick_Interface"*

3.4.1 Software

Em termos de funções de CAN para este módulo, este apresenta as seguintes:

Tabela 3.5: ID's das funções presentes no módulo Joystick

OperationNumber	Function
0	Turn Node Off
1	Turn Node On
4	Data from Joystick

Os valores monitorizados pelo módulo totalizam seis bytes, em que os dois primeiros bytes correspondem ao valor desejado para a velocidade linear, seguido de mais dois bytes referentes ao valor desejado da velocidade angular e por fim outros dois bytes de referencia do centro dos valores medidos. Estes dados são transmitidos sobre a função "*Data from Joystick*".

Módulo do Joystick

O fluxograma presente na Figura 3.18 mostra resumidamente a estrutura do código desenvolvido para o PIC responsável pelo envio dos comandos de referência dados pelo joystick.

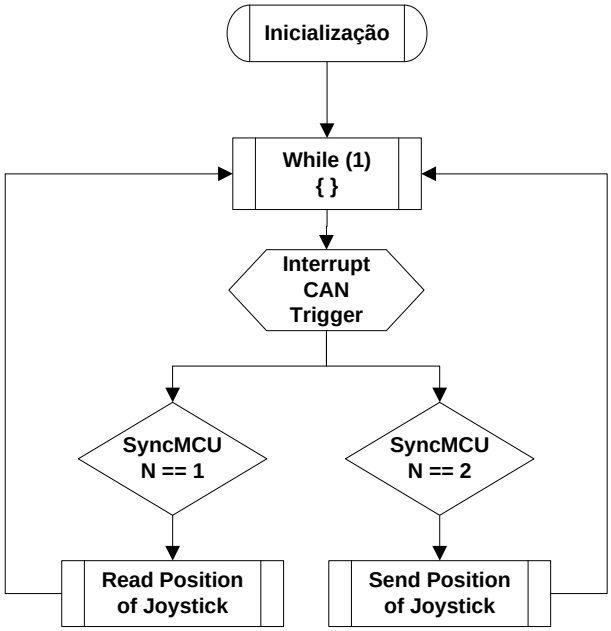

Figura 3.18: Fluxograma do código implementado no *Joytick Node*

Como é visível pelo fluxograma anterior, o Joystick envia por CAN para a camada de alto nível presente no PC, o valor da sua posição em X e Y e referência da posição central, de forma periódica. É com base nestes valores que é calculada a velocidade linear e angular desejada para o controle do robô. Visto isto, será apresentado apenas a parte do código que permite a leitura e envio dessas posições. O código responsável pela codificação das mensagens de CAN com origem no Joystick e destino o PC, é o seguinte.

```
void interrupt HighPriorityInterrupts(void){
{
  if(ScanCycle == 1)
  {
    WORD aux = 0;
    aux = Get_Joystick_X();
    Joystick2Can[0] = (char)(aux & 0x0FF);
    Joystick2Can[1] = (char)((aux >> 8) & 0x0FF);

    aux = Get_Joystick_Y();
    Joystick2Can[2] = (char)(aux & 0x0FF);
    Joystick2Can[3] = (char)((aux >> 8) & 0x0FF);

    aux = Get_Joystick_Ref();
    Joystick2Can[4] = (char)(aux & 0x0FF);
    Joystick2Can[5] = (char)((aux >> 8) & 0x0FF);

  }
```

```
      if(ScanCycle == 2)
      {
       CANMsgOut.stdID = 0;
       CANMsgOut.stdID = JOYSTICK << 3;
       CANMsgOut.stdID += 4;

       // Transmission of message CAN
       CANMsgOut.data[0]  = Joystick2Can[0]; //First  Byte of X
       CANMsgOut.data[1]  = Joystick2Can[1]; //Second Byte of X

       CANMsgOut.data[2]  = Joystick2Can[2]; //First  Byte of Y
       CANMsgOut.data[3]  = Joystick2Can[3]; //Second Byte of Y

       CANMsgOut.data[4]  = Joystick2Can[4]; //First  Byte of Ref
       CANMsgOut.data[5]  = Joystick2Can[5]; //Second Byte of Ref

       CANMsgOut.len = 6;
       sendCanMessage(CANMsgOut);
    }
}
```

A operação de leitura e envio dos valores da posição do joystick é feita de forma cíclica e em tempos distintos com base nas mensagens de sincronismo vindas por parte do módulo do *trigger* .

3.4.2 Hardware

Dadas as necessidades que se apresentam na leitura dos valores dados pelo joystick e sua correcta aquisição, o módulo Joystick_Interface foi desenvolvido de modo a servir de interface entre o joystick e o módulo "*PIC_Base*". Por sua vez, este faz o interface com o PIC responsável pelo envio do valor desejado da velocidade linear e angular do robô para o PC, de modo a que depois seja transmitida a devida velocidade desejada a cada motor.

Módulo de Hardware "Joystick_Interface"

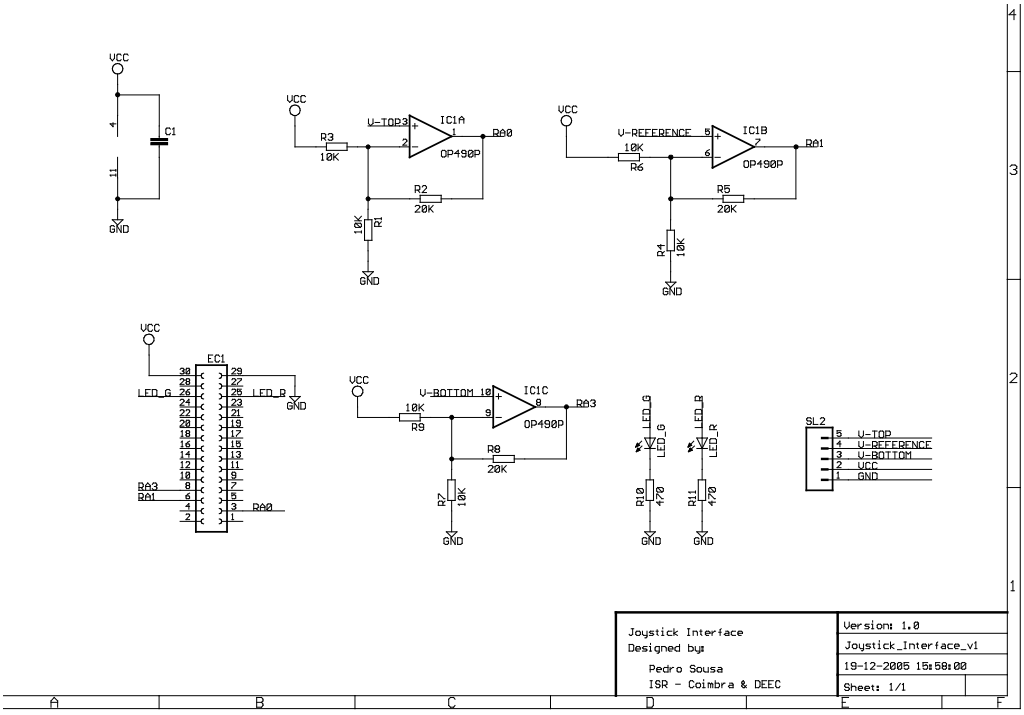

Figura 3.19: Esquemático do módulo

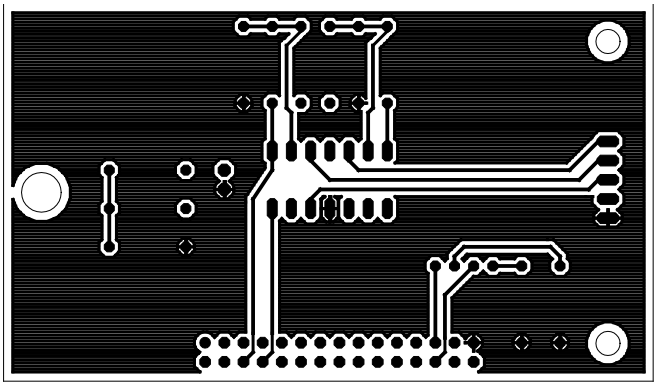

Figura 3.20: Layout do PCB – Top Layer

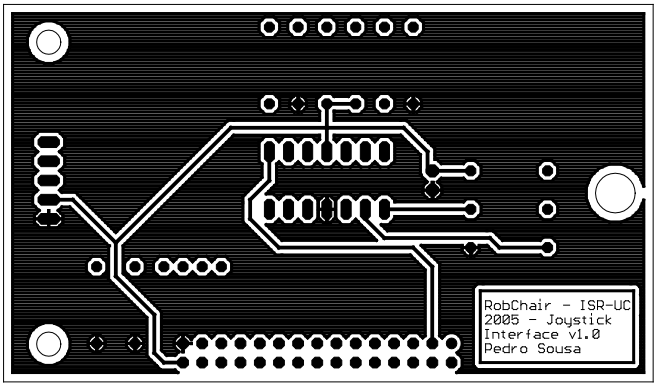

Figura 3.21: Layout do PCB – Bottom Layer

*3.5 **Módulo de Software/Hardware** "Trigger"*

3.5.1 Software

Em termos de funções de CAN para este módulo, este apresenta as seguintes:

Tabela 3.6: ID's das funções presentes no módulo Trigger

OperationNumber	Function
0	Turn Node Off
1	Turn Node On
15	Synchronize All Nodes

A função "Synchronize All Nodes", é usada para sincronizar a malha de controlo do sistema. Sendo o tempo do ciclo de controlo de 5ms, é enviada uma mensagem de sincronismo a cada 1ms, com a indicação da fase do ciclo em que o sistema se encontra, para que, sejam efectuadas todas as acções dos diversos módulos de uma forma cíclica e sincronizada. Tal mensagem é elaborada da seguinte forma:

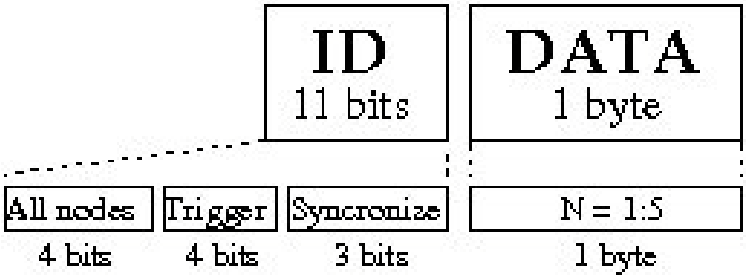

Figura 3.22: Mensagem de sincronização do sistema

Módulo do Trigger

Como foi possível ver nos outros módulos, estes efectuam as suas acções com base numa determinada ordem e respeitando determinado ciclo, sendo este módulo o responsável por essa sincronização, a qual é feita de 1 em 1ms, sendo o ciclo total de 5ms como referido anteriormente.

Visto isto, apresenta-se de seguida o código que permite esta sincronização dos módulos.

```
void interrupt HighPriorityInterrupts(void){
  if(TMR3IF)
    {
      TMR3IF = 0;
      // WriteTimer3( 55535 );  55535d = D8EFh  // Resets Cycle Timer (1ms/40Mhz)
```

```
    TMR3H = 0xD8;  // Write high byte to Timer3
    TMR3L = 0xEF;  // Write low byte to Timer3

    if (Cycle > 4)
    {
       Cycle = 0;
    }
    Cycle++;

    if (TriggerState==ON)
    {
      CANMsgOut.stdID   = Sincronize;
      CANMsgOut.len     = 1;
      CANMsgOut.data[0] = Cycle;
      sendCanMessage(CANMsgOut);             // Send Sincronization message to CAN BUS

      if (StateChanged == TRUE ){
        CANMsgOut.stdID = 0;                 // Informs robcan that
        CANMsgOut.stdID = TRIGGER << 3;      // trigger node is now on
        CANMsgOut.stdID += 1;
        CANMsgOut.len = 1;
        sendCanMessage(CANMsgOut);
        StateChanged = FALSE;
      }
    }
    else
    {
      if (StateChanged == TRUE ){
        CANMsgOut.stdID = 0;                // Informs robcan that
        CANMsgOut.stdID = TRIGGER << 3;     // trigger node is now off
        CANMsgOut.len = 1;
        sendCanMessage(CANMsgOut);
        StateChanged = FALSE;
      }
    }
  }
```

Este módulo envia uma mensagem de CAN de forma periódica, baseado num temporizador interno que conta 1ms, quando o módulo se encontra no estado activo. Na mensagem de sincronismo temos um byte que determina a fase do ciclo de operações. Temos cinco fases distintas de operações correspondentes a diferentes tarefas dos módulos, perfazendo o ciclo de 5ms e que se repete de forma cíclica.

3.5.2 Hardware

Este módulo é essencialmente apenas o módulo de processamento PIC_Base, uma vez que este não necessita de mais nenhum hardware de interface, utilizando apenas a comunicação de CAN do módulo, para efectuar a sincronização (tipo maestro numa banda) de todo o sistema e sinalizando todas as acções dos outros módulos no devido e exacto momento em que tal deve ser efectuado, como foi descrito anteriormente.

Módulo de Hardware “PIC_Base”

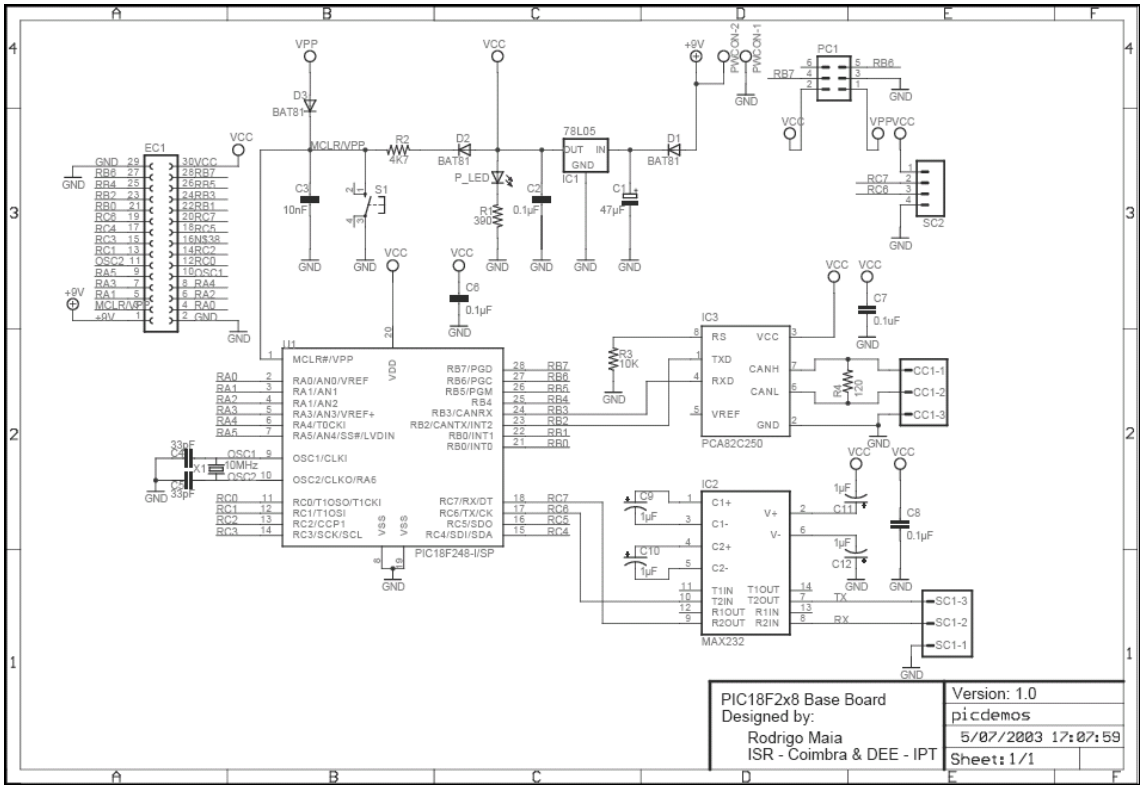

Figura 3.23: Esquemático do módulo

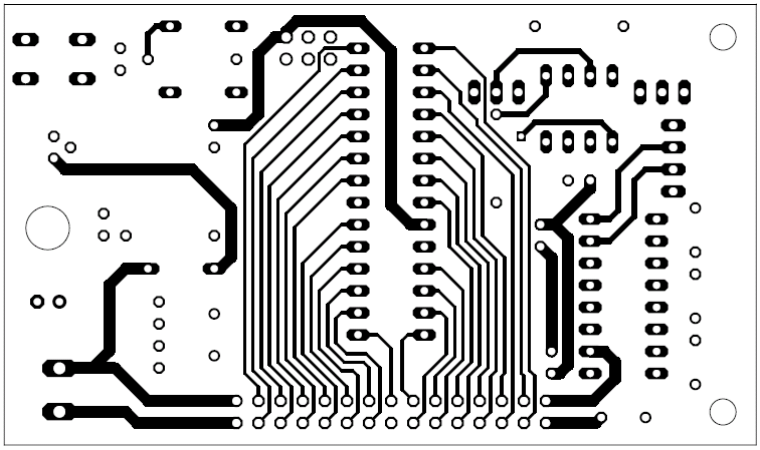

Figura 3.24: Layout do PCB – Top Layer

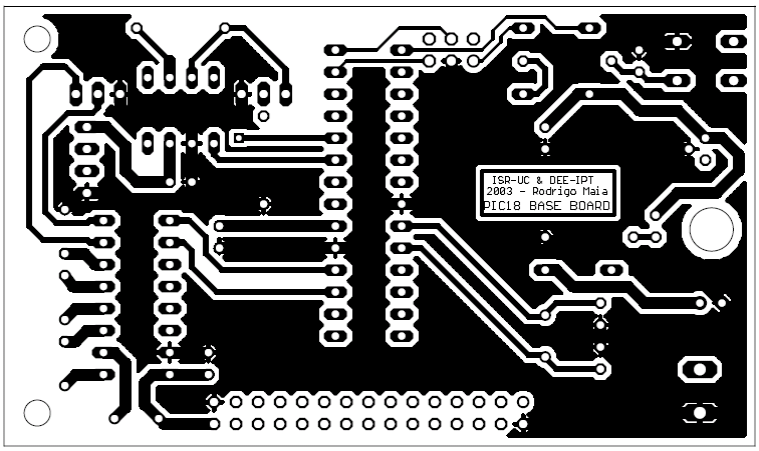

Figura 3.25: Layout do PCB – Bottom Layer

3.6 Módulos de Software Presentes no PC

O fluxograma seguinte representa resumidamente a estrutura de código implementada para o controlo do robô no PC, estando este a operar apenas em controlo por joystick.

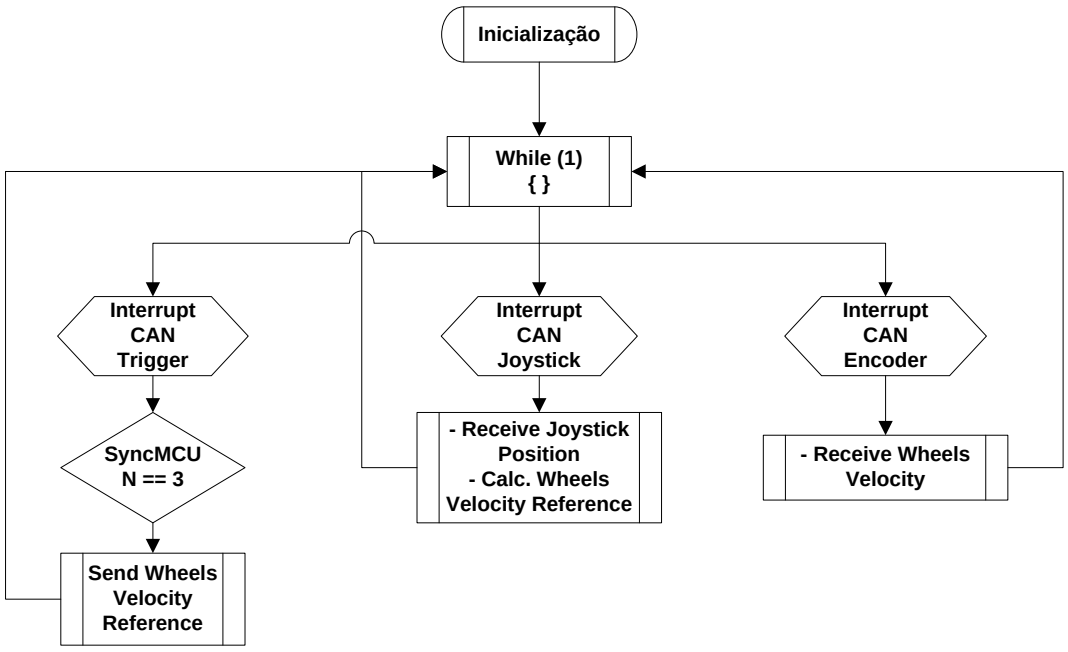

Figura 3.26: Fluxograma de módulos de controlo no PC

Como é visível pelo fluxograma anterior, o PC recebe por CAN da camada de baixo nível, informação do processo o qual este tem de controlar, estando a recolha da informação respeitante a cargo da camada de interligação.

Capítulo 4

4 Conteúdo do CD

- Documentos

  Diversos documentos de trabalhos anteriores e outros que descrevem outras partes do sistema, nomeadamente a parte de software de médio/alto nível presente no PC e que dá a autonomia/controle da plataforma móvel que é a RobChair.

- Hardware

  - Dispositivos

  Manual dos diversos componentes para consulta de informação mais detalhada.

  - Pic's

  Esquemático das diversas placas que estão ligadas ao barramento de can e que servem de interface entre a placa do microcontrolador e os dispositivos de hardware. É usado o programa Eagle para visualizar e alterar os esquemáticos. Ficheiros Grebber também presentes para a replicação das placas caso necessário.

- Filmes

  Resultados experimentais da RobChair em acção.

- Imagens

  Imagens dos diversos componentes existentes na RobChair.

- Software

- Código PC

Programa de RTAI, necessário ter privilégios de super-utilizador para executar e carregar os módulos de RTAI e CAN.

- Código Pic's

Programas dos diversos microcontroladores, para compilar é usado o compilador da Hitech "picc18". Para programar é necessário usar o programador por CAN "canbootmngr_v2". Modo de uso "./canbootmngr_v2 -h" para ver a ajuda. Modo de uso mais aprofundado no capitulo 3.1.2.