Universidade do Minho
Escola de Engenharia
Departamento de Electrónica Industrial

Bruno Guilherme Gonçalves de Matos

Controlador e Accionador para Motor DC em Malha Fechada

Maio de 2008

Universidade do Minho
Escola de Engenharia
Departamento de Electrónica Industrial

Bruno Guilherme Gonçalves de Matos

Controlador e Accionador para Motor DC em Malha Fechada

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Electrónica Industrial e Computadores

Dissertação realizada sob a orientação cientifica do
Professor António Fernando Macedo Ribeiro
Professor associado do Departamento de Electrónica Industrial da Universidade do Minho

Maio de 2008

“Penso 99 vezes e nada descubro; deixo de pensar, mergulho no silêncio – e eis que a verdade se revela!”

(Einstein)

Agradecimentos

No desenvolvimento deste projecto, várias pessoas e entidades contribuíram tornando-o possível, às quais quero agradecer.

Quero agradecer aos meus familiares em especial aos meus pais António Matos e Maria Gonçalves, por todo o esforço e apoio que me deram ao longo de todos estes anos como estudante.

Ao Departamento de Electrónica Industrial (DEI) da Escola de Engenharia da Universidade do Minho pela formação e condições que me foram dadas, pois sem estas era impensável realizar este projecto.

Ao meu orientador, Doutor Fernando Ribeiro, por todo o apoio e orientação dada, nunca pondo em causa as minhas capacidades para a conclusão deste projecto.

Aos meus colegas e amigos, em especial (André Oliveira, Carlos Ribeiro, Cristiano Santos, João Guimarães, Luís Pacheco, Paulo Carvalho, Pedro Amorim, Sílvia Reis), pelas ajudas dadas, questões construtivas e companheirismo em todos os momentos.

A todas as empresas (Intersil, Microchip, Maxim, Texas Instruments, STMicroelectronics) que não estando ligadas a este projecto, forneceram amostras, tornando possível criar protótipo fiável de custo reduzido.

Resumo

Controlador e Accionador para motor DC em Malha Fechada

Os motores CC (corrente contínua) têm actualmente como principal aplicação a motorização de plataformas móveis, onde a fonte de energia é feita com o recurso a baterias. O controlo da velocidade de rotação destes motores vinha a ser feito com recurso a reóstatos variando a corrente do estator, sendo um processo eficaz no caso em que a carga e velocidade de rotação variem pouco. Com o desenvolvimento da electrónica surgiram controladores de velocidade, com gamas de variação de velocidade elevadas, fazendo compensações às flutuações de carga.

O presente trabalho, trata do projecto e implementação de um controlador e accionador em malha fechada, para aplicação em motores DC (*direct current*), fazendo com que estes rodem a uma velocidade desejada, independente da carga aplicada.

É feito um estudo do mercado, seleccionando alguns dos controladores existentes e analisando se o projecto actual é uma solução necessária para os objectivos pretendidos. Seguido do projecto completo, desde a análise topológica das estruturas escolhidas até a concepção de toda a electrónica necessária (accionadores, sensores, barramentos de dados e protecções), referindo os cuidados tidos no desenho do PCB (*Printed Circuit Board*). Na fase de projecto, é focado o estudo a várias técnicas de controlo em malha fechada, indicando o porquê, da técnica escolhida. É descrito todo *software* utilizado e programado, desde a linguagem de programação e compiladores utilizados até aos algoritmos realizados para o *software* desenvolvido. No final é apresentado o protótipo construído em laboratório e resultados alcançados, tendo como referência os objectivos pretendidos.

Palavras-Chave – *Encoders*, PID (*proportional–integral–derivative*), PWM (*pulse width modulation*), MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*), microcontroladores PIC, I2C (*Inter Integrated Circuit*), Motores DC, Ponte H, Sensores.

Abstract

Controller and drives for DC motors in closed loop

DC (Direct Current) motors have nowadays as main application the drive of mobile platforms, where the energy source comes from batteries. These motors rotation speed control was carried out using rheostat varying the stator current, being an efficient process when the load applied and rotation speed have little variation. With the actual electronics development speed controllers came up, with a high ranges of speed changes, compensating load changes.

The present work, develops the implementation of a controller in a closed loop, to be used in DC motors, allowing them to rotate at a certain speed and completely independent of the load applied.

A market survey is carried out, selecting some of the existing controllers and analyzing if the actual project is a desired solution to the defined objectives. After completing the project, from a topological analysis of the chosen structures up to the conception of the required electronics (drives, sensors, data bus and electronic protections), referring the care taken on the PCB (Printed Circuit Board) drawing. On the design phase, several closed loop control techniques were studied, pointing out the chosen technique. All the used and developed software is described, from the programming language to the used compilers, up to the developed algorithms for this project. In the end a prototype is developed, built up and presented, as well as the achieved results, having as reference the objectives stated for the project.

Key-Words – Encoders, PID (proportional–integral– derivative), PWM (pulse width modulation), MOSFET (Metal Oxide Semiconductor Field Effect Transistor), microcontrollers PIC, I2C (Inter Integrated Circuit), DC Motors, H Bridge, Sensors.

ÍNDICE

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

CC	→ Corrente Contínua
DC	→ *Direct Current*
CD's	→ *Compact Disc*
CA	→ Corrente Alternada
I2C	→ *Inter Integrated Circuit*
W	→ *Watt*
PIC	→ *Peripherical Interface Controller*
PWM	→ *Pulse Width Modulation*
V	→ *Volt*
PID	→ *Proportional Integral Derivative*
Hz	→ *Hertz*
RS232	→ *Recommended Standard 232*
CAN	→ *Controller Area Network*
TWI	→ *Two Wire Interface*
MOSFET	→ *Metal Oxide Semiconductor Field Effect Transistor*
IGBT	→ *Insulated Gate Bipolar Transistor*
MCT	→ *MOS Controlled Thyristor*
CMOS	→ *Complementary Metal Oxide Semiconductor*
TVS	→ *Transient Voltage Supressor*
TTL	→ *Transistor-Transistor Logic*
SMD	→ *Surface Mount Technology*
MIPS	→ *Millions of Instructions Per Second*
RISC	→ *Reduced Instruction Set Compute*
SRAM	→ *Static Random Access Memory*
EEPROM	→ *Electrically Erasable Programmable Read Only Memory*
SOIC	→ *Small Outline Integrated Circuit*
ADC	→ *Analog To Digital Converter*
LED	→ *Light Emitting Diode*
PLL	→ *Phase Locked Loop*
QEI	→ *Quadrature EncoderIinterface*
PCB	→ *Printed Circuit Board*
PC	→ *Personal Computer*

Capítulo 1 - Introdução

Sabia que?

Pode construir um motor eléctrico em 30 segundos, veja aqui: [1].

1.1 Introdução

A primeira máquina eléctrica considera-se que surgiu em 1886 pelas mãos do cientista *Werner von Siemens* construindo um gerador de corrente contínua auto-induzido [2]. Desde então vários tipos de máquinas eléctricas foram desenvolvidas e aperfeiçoadas até aos dias de hoje, sendo utilizadas em quase tudo que nos rodeia, desde um simples leitor de CD's até grandes geradores de energia.

O motor CC (corrente contínua) foi uma das primeiras máquinas eléctricas inventadas, actualmente com o desenvolvimento de técnicas em accionamentos de corrente alternada (CA) e viabilidade económica têm sido substituídos por motores de indução accionados por inversores de frequência. Devido às suas características, o motor CC ainda se mostra vantajoso no uso de inúmeras aplicações (movimentação e elevação de cargas, elevadores, prensas, extrusoras) entre outras [3].

Quase todas as aplicações deste motor requerem um controlo de velocidade, actualmente é feito através do uso de um controlador electrónico. Este dependendo da aplicação varia a sua complexidade, funções, potência e custo, sendo necessário para algumas aplicações o desenvolvimento de novos controladores de velocidade.

1.2 Objectivos do Trabalho

A Fig. 1, é um exemplo para aplicação destes controladores, consiste numa plataforma omnidireccional de 3 eixos, constituída por um conjunto de 3 rodas omnidireccionais desfasadas de 120° acopladas a 3 motores DC que proporcionam o movimento da plataforma. A fonte de energia para estas plataformas é normalmente feita com recurso a baterias, não tendo qualquer alimentação externa.

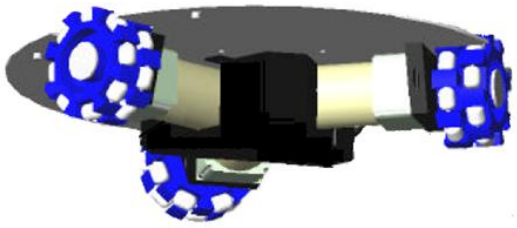


Fig. 1 – Plataforma omnidireccional de 3 rodas

O uso de motores CC torna-se eficaz devido às suas características, binário de arranque elevado e funcionamento em corrente contínua, o mesmo tipo de corrente fornecida pelas baterias. Cada um destes motores roda a uma velocidade, a soma dos vectores velocidade de cada motor vai originar um outro responsável pela direcção e velocidade da plataforma, sendo necessário um controlador para variar a velocidade em cada um dos 3 motores.

Esta tese propõe o estudo, projecto e implementação de um controlador de velocidade em malha fechada para 3 motores CC com os seguintes objectivos:

- Comunicação com o exterior via barramento I2C (*InterIntegrated Circuit*).
- Alimentação a 24 volts DC
- Accionamento 3 motores de 24 volts DC de ímanes permanentes, para 200 W (Watts) no mínimo, nos dois sentidos de rotação.
- Frequência de PWM dos motores não audível.
- Baixa emissão de ruído electromagnético.
- Ler três *encoders* ópticos (um por motor) simultaneamente com discriminação do sentido de rotação.
- Accionamento de travão em cada um dos motores.
- Medir a corrente em cada um dos motores em tempo real, para impedir correntes de curto-circuito nos motores.
- Disponibilizar pelo barramento I2C, dados de telemetria (correntes, tensão de alimentação, velocidades, presença ou não de motores…).
- Aplicar uma lei de controlo de forma a garantir com precisão a velocidade de rotação pedida para cada um dos motores.
- Todo o sistema terá de ser projectado com a intenção de máxima redução de custos mas com a máxima performance.

A Fig. 2, referencia a localização da placa controladora de velocidade no sistema a controlar.

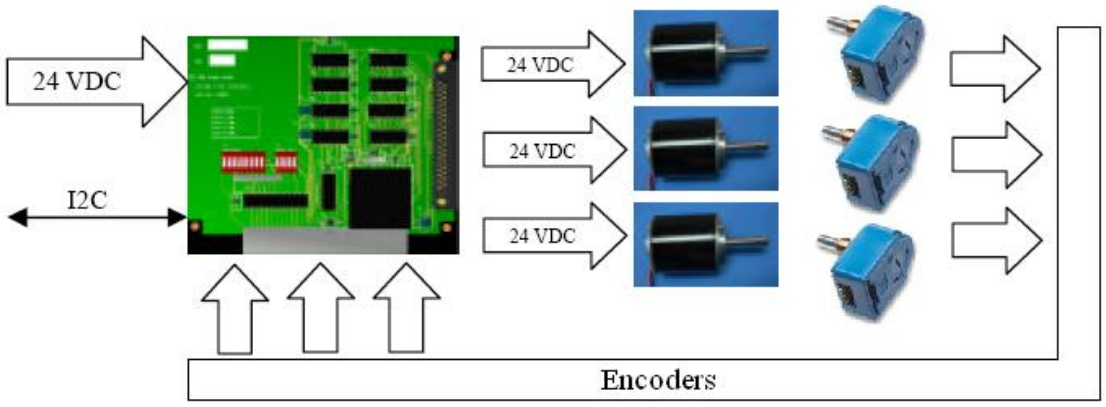


Fig. 2 – Diagrama inicial das partes constituintes do sistema a controlar

1.3 Estrutura do Trabalho

A Fig. 2, mostra um diagrama de uma solução apresentada, em que o controlo e accionamento dos motores são feitos numa única placa controladora. Esta solução é viável para aplicações em que são utilizados 3 motores, caso se pretenda usar 2 ou 4 motores são necessárias 1 ou 2 placas ficando subaproveitada.

A Fig. 3, demonstra a estrutura utilizada, a placa controladora é para um único motor. Esta solução é vantajosa em relação à solução apresentada (Fig. 2). Permite o controlo desde 1 até 128 motores por barramento, permitindo ainda que quando

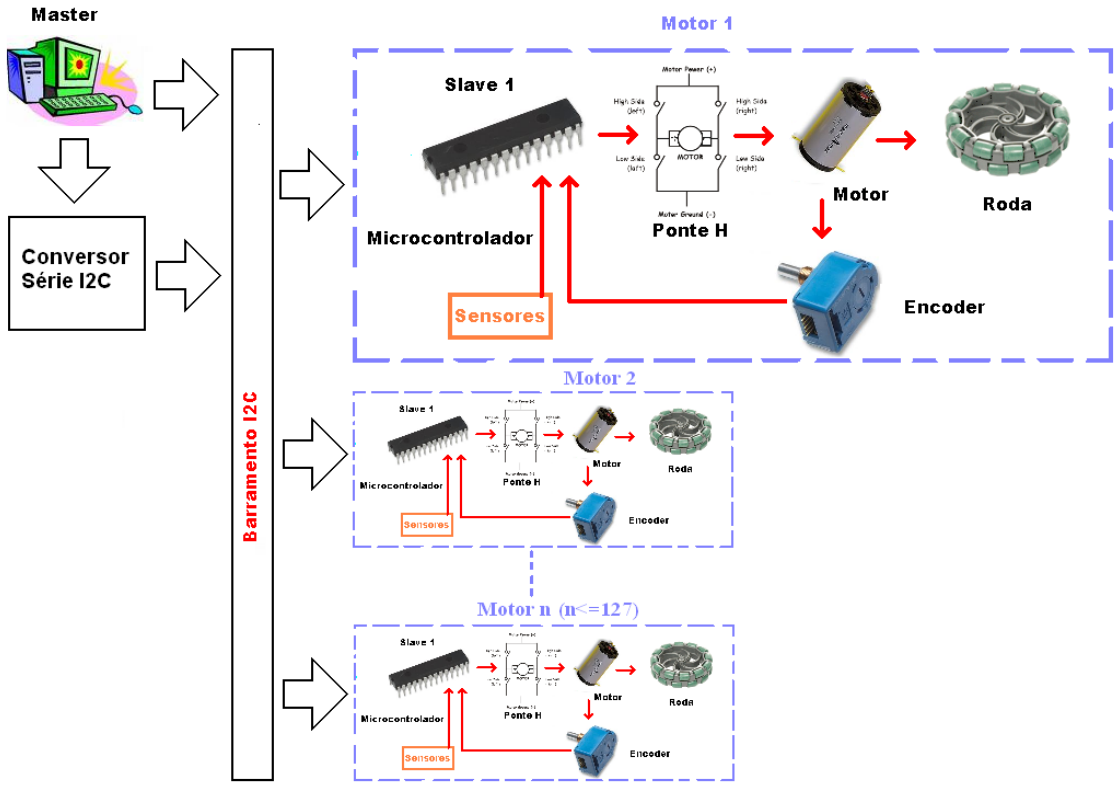


Fig. 3 – Diagrama das partes constituintes da solução estudada

usada em plataformas móveis omnidireccionais estas se desloquem em caso de falha de um controlador.

Cada placa controladora é constituída por um microcontrolador, uma ponte H e alguns sensores (temperatura, corrente, tensão). Esta recebe ordens (tramas) através de um barramento I2C de um *Master*, estas tramas podem ser de leitura ou de escrita (ex. configurações, velocidade desejada, telemetria).

O microcontrolador é o componente chave, este lê constantemente a velocidade real, quando recebe uma velocidade desejada, faz diferença das duas velocidades e aplica a um algoritmo de controlo PID (*proportional – integral - derivative*) que retorna um valor de correcção. Este valor é aplicado num sinal PWM (*Pulse Width Modulation*) que opera a uma frequência de 19500 Hz (hertz). O sinal PWM é utilizado para controlar os MOSFETs da ponte H (accionador), corrigindo a velocidade real para a desejada. Os sinais provenientes dos sensores de temperatura, corrente e tensão, também são processados pelo microcontrolador que actua caso os valores obtidos ultrapassem os limites predefinidos.

A placa controladora suporta uma potência de 500W com uma alimentação de máxima 35V (*volts*). A programação dos microcontroladores é desenvolvida na plataforma MPLAB C18 e CCS, na linguagem C e assembler.

A solução acima descrita refere-se a uma placa controladora, sendo necessárias 2 outras iguais para cumprir os objectivos do projecto. Nos próximos capítulos será descrito o porquê das decisões tomadas para cada componente da solução apresentada, assim como o respectivo funcionamento.

Capítulo 2 - Estado da Arte

2.1 Introdução

Actualmente o principal desenvolvimento de controladores de velocidade para motores DC é feito pela indústria automóvel, fabricantes de motores DC e pela robótica. Cada uma destas indústrias desenvolve os seus controladores para aplicações específicas. A indústria automóvel desenvolve os seus controladores para funcionarem com uma tensão de 12V, sendo os controladores dos elevadores dos vidros diferentes dos controladores dos limpa-pára-brisas. Os controladores dos fabricantes de motores DC são maioritariamente projectados para aplicações em que a velocidade é ajustada na instalação ou com poucas variações (esteiras rolantes). Na robótica os controladores são utilizados para maiores variações de velocidade sendo controlados de forma digital, estes estão ligados a um computador central que envia informação sobre a velocidade desejada a cada controlador. A movimentação de plataformas integra uma grande percentagem dos controladores de velocidade construídos na robótica.

De seguida são analisados alguns dos controladores de velocidade existentes no mercado, estes foram escolhidos tendo em conta a equivalência com o projecto pretendido.

2.2 TMC200

TMC200 ilustrada na Fig. 4, é uma placa controladora desenvolvida pelo instituto *Fraunhofer AIS (Autonomous Intelligent Systeme)*, capaz de controlar 3 motores DC de 200W cada em malha fechada. É controlada por um único microcontrolador de 16 bits, implementa o algoritmo de controlo PID, efectua leitura de velocidades através de *encoders*, comunicação CAN (*Controller Area Network)* e RS232 (*Recommended Standard 232*). A tensão de alimentação pode variar entre 12 e 35V com uma corrente contínua máxima de 8A, o PWM funciona com uma frequência de 20 kHz. Possui funções de odometria, monitorização de tensões e correntes [4].

Fig. 4 – Controlador de Velocidade TMC200 [4]

Sendo bastante funcional e integrada, esta placa no entanto possui desvantagens em comparação com o projecto estudado. Não possui comunicação I2C, não permite a expansão do número de motores, em caso de falha todos os motores são afectados. Recentemente e como principal inconveniente este controlador saiu de produção.

2.3 VolksBot® Motor Controller VMC

O controlador de velocidade da *Fraunhofer IAIS (Intelligent Analyse und Informations Systeme) VMC Motor controller*, representado na Fig. 5, contém algumas semelhanças com o controlador anterior. Controla 3 motores DC em simultâneo com uma potência até 150W por motor, possui um algoritmo de controlo PID para cada motor fazendo a leitura dos respectivos *encoders*, a comunicação é feita através de RS232 permitindo diversas configurações [5].

Fig. 5 – Controlador de Velocidade VolksBot® Motor Controller VMC

Como inconvenientes não possui comunicação I2C, em caso de falha compromete todo o sistema, actualmente o seu preço é muito significativo. Este tipo de controladores, 1 controlador para 3 motores, traz vantagens a nível de espaço em relação à solução estudada.

2.4 ADS 50/10

ADS 50/10 representada na Fig. 6, é um controlador de velocidade da Maxon, controla um único motor até 500W, estando a corrente limitada a 10A (Amperes). O valor de velocidade é ajustado através de um potenciómetro, possui uma entrada para um *encoder* e algoritmo de controlo é unicamente proporcional, sendo também ajustado por um potenciómetro. Possui protecções tais como, limitador de corrente, de temperatura, e protecção contra curto-circuito [6].

Fig. 6 – Controlador ADS 50/10 da Maxon

Este tipo de controlador tem como principal desvantagem (para o projecto em causa) o ajuste de velocidade (potenciómetro), tendo inúmeras aplicações nos casos em o valor da velocidade é variado poucas vezes (ex. esteiras rolantes), ou só mesmo na instalação. Como vantagens salienta-se a eficiência de 95% e a frequência de operação do PWM (50kHz)

2.5 MD03

A Fig. 7 ilustra a MD03, é mais um controlador de velocidade existente no mercado, tem a capacidade de controlar um motor DC até 1000W, possui um barramento I2C para envio de tramas de controlo e recepção de tramas informativas (correntes, temperatura), permite a comunicação com um dispositivo de radiofrequência e contém circuitos de protecção contra curto-circuitos e sobre-temperaturas. A tensão de alimentação do motor pode variar entre 5 e 50V, sendo a corrente máxima de 20A, necessita de uma alimentação de 5V para alimentação de circuitos lógicos [7].

A desvantagem desta placa para o projecto proposto, é a ausência de um circuito de controlo em malha fechada, não garantindo que a velocidade real atinja a desejada. Este problema pode ser contornado criando um circuito, que implemente uma

Fig. 7 – Placa controladora MD03

lei de controlo em malha fechada, não sendo esta solução viável, o circuito final seria pouco integrado e mais dispendioso.

2.6 S24-15A-30V

A placa controladora S24-15A-30V da Acroname, representada na Fig. 8, é semelhante à MD03, no entanto esta não possui barramento I2C, a potência é menor (450W). Controla um motor não possuindo circuito de controlo em malha fechada, a tensão de alimentação vária de 5 a 30V com uma corrente máxima permanente de 15A, possui um PWM a operar a uma frequência de 100 kHz, sensores de temperatura e corrente [8].

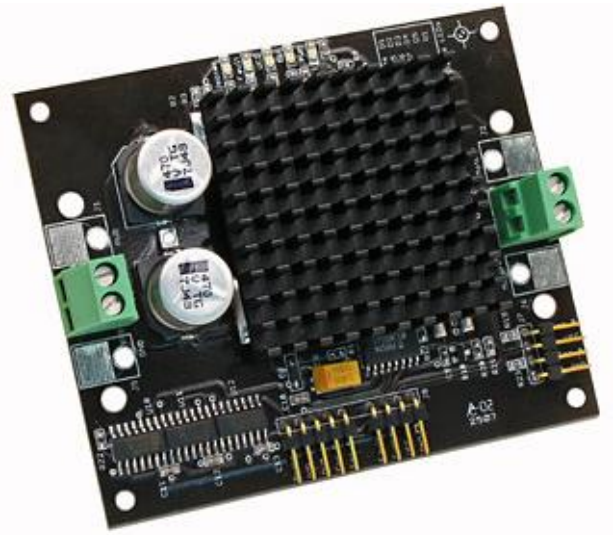


Fig. 8 – Placa controladora S24-15A-30V

2.7 µM-H-Bridge

O controlador de velocidade µM-H-Bridge em baixo representado, Fig. 9, não está comercialmente disponível, no entanto o acesso aos planos de construção são livres, desde que não utilizados para fins lucrativos.

Este controla um motor com uma potência máxima de 290W, sendo a corrente máxima de 16A, a tensão de alimentação pode variar entre 7V a 18V. Possui um barramento de comunicação TWI (Two Wire Interface), sendo semelhante ao I2C este possui uma nomenclatura diferente devido às patentes sobre o barramento I2C. Rampas de aceleração/desaceleração, travagem forçada/livre e controlo de corrente, são mais algumas das funcionalidades desta placa [9].

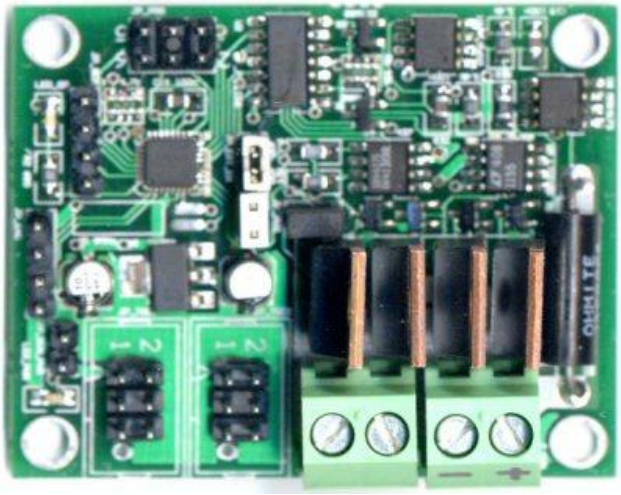


Fig. 9 – Controlador de velocidade µM-H-Bridge

A µM-H-Bridge tem bastantes funcionalidades, no entanto esta possui 2 inconvenientes, sendo o primeiro a não comercialização desta placa controladora e o segundo, deve-se à tensão de alimentação máxima ficar abaixo dos requisitos mínimos (24V).

2.8 RoboteQ AX1500

RoboteQ AX1500 (Fig. 10) é um controlador de velocidade com uma potência de 1200W, consegue controlar 2 motores com uma corrente máxima de 30A cada e uma tensão que pode variar desde os 12V até aos 40V. Possui controlo em malha fechada para velocidade e posicionamento. A comunicação é via RS232, possuindo uma entrada analógica e outra de rádio frequência para controlo de velocidade [10].

Fig. 10 – Controlador de Velocidade RoboteQ AX1500

Como inconvenientes salienta-se o facto de não possuir barramento I2C e o PWM opera a uma frequência audível 16kHz.

2.9 MD22, MD23, Simple-H

Existem no mercado muitos outros controladores, a MD22 (Fig. 11-a), MD23 (Fig. 11-b), Simple-H (Fig. 11-c), são mais alguns controladores existentes, tendo em comum a ausência de controlo em malha fechada. O MD22 e MD23 só suportam 5A e 3A respectivamente, sendo o Simple-H o controlador que suporta mais corrente (20A).

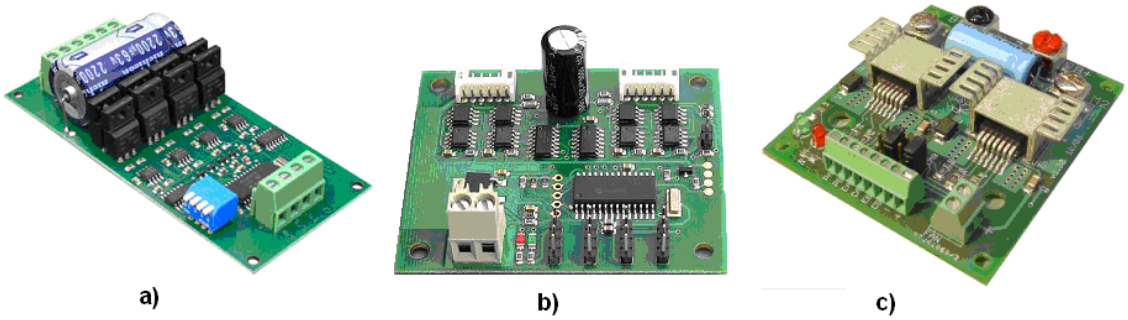


Fig. 11 – Controlador, a) MD22, b) MD23, c) Simple-H

A comunicação com o MD22 [11] e MD23 [12] é feita através de I2C, o controlador Simple-H [13] não possui comunicação sendo necessário ejectar pulsos PWM para funcionar.

Este tipo de placas é bastante útil em aplicações onde não seja necessário efectuar um controlo de velocidade, ou onde este seja feito com recurso a circuitos auxiliares.

2.10 Conclusões

Nos subcapítulos anteriores foram referidas as principais características de dez controladores de velocidade existentes no mercado. Desta pequena amostra conclui-se que nenhum deles possui as mesmas características, cada aplicação requer controladores com características únicas. Nenhuma das placas estudadas possui a totalidade dos requisitos propostos.

A Tabela I indica as principais características dos controladores analisados.

	Tensão	Corrente	Nº Motores	Controlo	PWM	Comunicação
Controlador Desenvolvido	**35V**	**15A**	**1**	**PID**	**19,5kHz**	**I2C**
TMC 200	**35V**	**8A**	**3**	**PID**	**20kHz**	**CAN-RS232**
VMC	**24V**	**5A**	**3**	**PID**	**-------**	**RS232**
ADS 50/10	**50V**	**10A**	**1**	**P**	**50kHz**	**Não Possui**
MD03	**50V**	**20A**	**1**	**Não Possui**	**15kHz**	**I2C**
S24-15A-30V	**30V**	**15A**	**1**	**Não Possui**	**100kHz**	**Não Possui**
µM-H-Bridge	**18V**	**16A**	**1**	**PID**	**-------**	**TWI-RS232**
RoboteQ AX1500	**40V**	**30A**	**2**	**PID**	**16kHz**	**RS2332**
MD22	**50V**	**5A**	**2**	**Não Possui**	**------**	**I2C**
MD23	**12V**	**3A**	**2**	**Não Possui**	**------**	**I2C**
Simple-H	**28V**	**20A**	**1**	**Não Possui**	**20kHz**	**Não Possui**

Tabela I – Principais caracteristicas dos controladores analisados

Fazendo uma selecção de todos controladores que possuem controlo em malha fechada, tensão de alimentação igual ou superior a 24V, potência igual ou superior a 200W, algum tipo de comunicação digital e uma frequência de PWM não audível, (próximo de 20kHz) obtêm-se o controlador TMC 200. Tendo o controlador TMC200 sido descontinuado, não existe das soluções analisadas nenhuma outra capaz de cumprir os requisitos sem desenvolvimento de *hardware* e *software* auxiliar.

O desenvolvimento da solução estudada torna-se necessário, para a obtenção de um controlador capaz de cumprir todos objectivos propostos estando preparado para outras aplicações. Outra vantagem do desenvolvimento deste controlador é o desenvolvimento de tecnologia nacional.

Capítulo 3 - Motores DC e Ponte H

3.1 Introdução

No capítulo anterior foi feito o estudo do mercado dos controladores de velocidade para motores DC. Como o nome indica estes controladores servem para variar a velocidade. Sendo o controlo de velocidade feito num motor DC, é necessário estudá-lo, de modo a descobrir quais os parâmetros, que quando alterados fazem com que o motor mude de velocidade e direcção.

Neste capítulo serão estudados os motores de corrente contínua ou motores DC, analisando o seu princípio de funcionamento, especificações, tipos, mostrando como se pode controlar a velocidade e direcção de rotação. No controlo de velocidade e direcção a ponte H (actuador) é essencial, sendo abordada neste capítulo. Serão explicados todos os passos desde o princípio de funcionamento até à função de cada componente da ponte H.

3.2 Motores DC

Os Motores de corrente contínua comuns consistem na forma mais utilizada de se converter energia eléctrica em energia mecânica, sendo por esse motivo amplamente utilizados como principal meio de tracção das partes móveis de robôs, automatismos e diversos tipos de dispositivos de Mecatrónica [14]. A Fig. 12 representa uma vista em corte de um motor DC da *Maxon*, semelhante ao utilizado neste projecto.

Fig. 12 – Vista em corte de um motor DC da Maxon [17]

3.2.1 Constituição, Princípio de Funcionamento

Os motores DC são constituídos por duas partes, uma parte móvel que se chama Rotor ou Armadura (Fig. 13-a) e uma parte fixa a que se dá o nome de Estator ou Campo (Fig. 13-b).

Fig. 13 – Partes constituintes de um motor DC. a) Rotor, b) Estator

O Rotor (armadura) é a parte girante, montado sobre o eixo da máquina, construído de um material ferromagnético envolto em um enrolamento chamado de enrolamento de armadura, este enrolamento liga ao anel comutador [15].

O Estator (campo) como indica o nome é a parte estática da máquina, montada em volta do rotor, de forma que o mesmo possa girar internamente, também constituído de material ferromagnético envolto em um enrolamento chamado de enrolamento de campo, que tem a função de produzir um campo magnético fixo para interagir com o campo da armadura. Dependendo do tipo de motor DC este campo magnético pode ser produzido por ímanes permanentes [15].

Existem vários tipos de motores CC, estes podem ser de ímanes permanentes com ou sem escovas, com excitação série, *shunt* ou paralelo, composto e excitação independente, sendo comum o seu princípio de funcionamento.

A Fig. 14, representa uma estrutura simplificada de um motor DC que servirá de base para compreensão do seu princípio de funcionamento.

Entre os pólos do íman em forma de ferradura está colocada uma bobina rectangular, montada sobre um eixo giratório. Os terminais desta bobina estão ligados a um sistema comutador, formado por anéis colectores e escovas, permitindo que a bobina seja alimentada durante a rotação do rotor.

Durante uma rotação existem quatro situações que acontecem sequencialmente, estas estão representadas na Fig. 15.

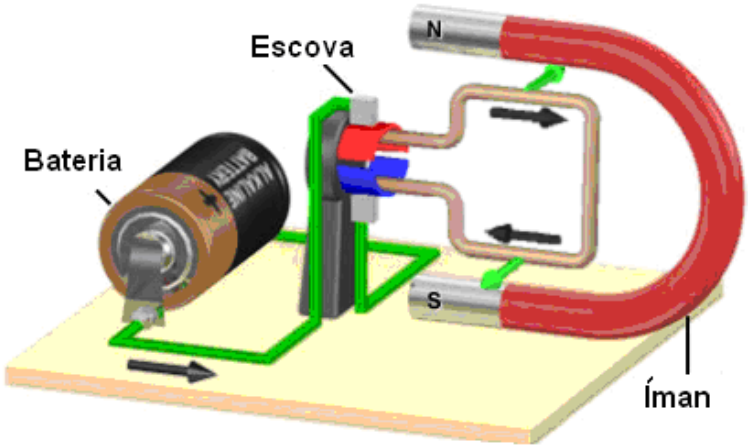


Fig. 14 – Imagem para compreensão do inicio de funcionamento de um motor DC [16]

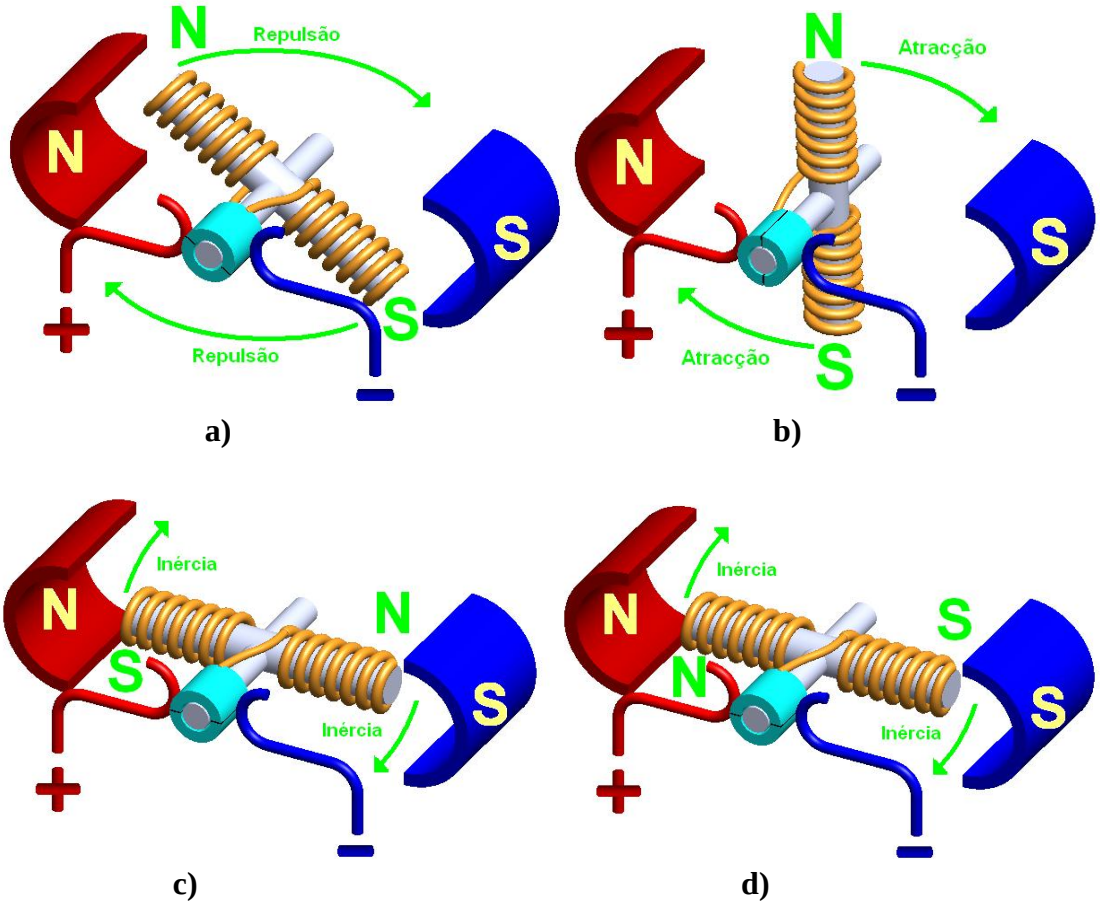


Fig. 15 – Estados possíveis do rotor durante uma rotação

Começando pela situação representada na Fig. 15–a) a bobina do rotor é alimentada com uma tensão fazendo com que circule uma corrente na mesma, esta produzirá um campo magnético, que interage com o campo magnético criado pelos ímanes do estator, originando uma força de atracção ou repulsão dependendo do sentido

da corrente e do sentido do enrolamento na bobina do rotor. Nesta situação é uma força de repulsão capaz de fazer girar o rotor no sentido indicado pela Fig. 15–a).

Na situação da Fig. 15–b) o movimento continua com a diferença que a força predominante passa a ser a de atracção, até que o campo magnético produzido pelo íman se alinha com o campo magnético da bobina do rotor deixando de existir forças de atracção ou repulsão.

Nas situações seguintes Fig. 15–c) e Fig. 15–d) cabe à inércia do motor fazê-lo girar até que os contactos troquem de posição ocorrendo uma inversão do sentido da corrente, por sua vez uma inversão do campo magnético. Ao inverter o sentido do campo magnético repete-se a primeira situação, continuando a rotação até alcançar um novo ponto de equilíbrio, como os contactos estão sempre a trocar e o motor possui inércia este ponto de equilíbrio não será atingido continuando a rotação.

A Fig. 15–c) e Fig. 15–d) representam dois pontos de equilíbrio momentâneos, estes podem ser eliminados com a inserção de um terceiro pólo, como o rotor representado na Fig. 13.

Ao inverter a fonte de alimentação do rotor, o sentido de circulação da corrente na bobina também inverterá e esta tenderá a girar em sentido contrário. Logo o sentido de rotação do motor depende do sentido da corrente.

Até ao momento foi descrito o princípio de funcionamento do motor, o passo seguinte consiste em estudar as características do mesmo, encontrando uma forma para variar a velocidade.

3.2.2 Características do Motor DC

Os motores DC possuem quatro características principais:

- Tensão nominal
- Corrente nominal
- Potência / binário
- Velocidade

A **tensão nominal**, medida em volts (V) é a tensão para qual o motor pode operar permanentemente sem se danificar, podendo operar abaixo desta tensão quando se pretender reduzir a velocidade ou a sua potência.

A **corrente nominal**, medida em amperes (A), é a corrente para a qual o motor opera em condições nominais, esta depende directamente da carga aplicada ao rotor,

deve-se garantir que o motor rode num regime em que a carga não exija mais corrente que a máxima recomendada pelo fabricante.

Potência, medida em Watts (W), é a força que um motor pode exercer em determinado momento, determinada pelo produto (tensão x corrente). Conhecer a potência, corrente e tensão é importante para dimensionar os circuitos de potência.

Binário, medido em (Newton x Metro) (N.m) é a força que o motor pode exercer a uma distância do eixo do motor. Este pode ser determinado através da equação (1) [18].

$$T = K_T \times I \quad \textbf{(1)}$$

Onde:

T→ Binário (N.m);

K_T → Constante de binário do motor, relação entre o binário produzido pelo motor e a corrente que nele circula (N.m/A);

I → Corrente que circula no motor (A);

A **Velocidade**, é medida em rotações por minuto (RPM), esta depende da corrente, da tensão, e da carga aplicada, qualquer um destes factores influenciam a velocidade do motor. Com a variação do fluxo magnético produzido no estator também se pode variar a velocidade, no entanto o motor utilizado é de ímanes permanentes sendo o fluxo magnético constante. Através da equação (2) pode-se determinar a velocidade (n) em função das suas variáveis, como K_n e R são constantes resta Vi e i para variar a velocidade, sendo R (resistência do enrolamento do rotor) muito baixa, desprezável em certos casos, a variação de velocidade em função corrente é mínima, controlar a velocidade através da tensão torna-se a melhor solução [18].

$$n = K_n (Vi - R \times i) \quad \textbf{(2)}$$

Onde:

Vi → Tensão de entrada aplicada aos terminais do rotor (V);

i → Corrente que circula no rotor (A);

K_n → Constante de velocidade do motor, relação entre a velocidade e a força electromotriz induzida na bobina (RPM/V);

R → Resistência do enrolamento do rotor (Ω);

3.2.3 Conclusões

Através deste breve estudo obteve-se a base necessária para iniciar o projecto do controlador, sendo salientadas algumas conclusões até ao momento nos seguintes pontos.

- O motor DC tem como principais características, tensão nominal, corrente nominal, potência, binário e velocidade.
- A velocidade varia directamente com a tensão de alimentação.
- O sentido de rotação depende do sentido da corrente no enrolamento do rotor.
- A potência do motor varia directamente com a velocidade.
- O binário nominal deve ser aplicado à velocidade nominal. Se o motor rodar abaixo da velocidade nominal e a tensão de alimentação é abaixo da nominal, o motor para manter o binário consome uma corrente acima da nominal podendo danificar-se.

3.3 Ponte H

Alterando a tensão aplicada no rotor varia-se a velocidade, alterando o sentido da corrente altera-se a direcção. Estas são as duas características fundamentais para o desenvolvimento de um controlador de direcção e velocidade.

Geralmente as fontes de alimentação são de tensões fixas, não permitindo um controlo por *software* nem sentido da corrente, nem da tensão de alimentação, tornando-se necessário o desenvolvimento de *hardware* para o fazer.

Podendo existir outras soluções, foram analisadas três para controlar a tensão e o sentido da corrente. Sendo o princípio da variação da tensão comum às três este será descrito no ponto 3.3.2.

A primeira solução consiste em desenvolver uma fonte de tensão variável que gere tensões positivas e negativas, alterando assim o sentido da corrente no rotor, logo o sentido de rotação. Sendo a fonte de alimentação usualmente baterias, a complexidade do circuito necessário para gerar tensões negativas torna inviável esta solução face à solução escolhida.

Outra solução seria inverter os terminais do motor com o uso de comutadores mecânicos, desta forma a fonte de tensão gera tensões positivas variáveis para controlo

de velocidade e circuito comutador mecânico alterava o sentido de rotação. Este circuito comutador não é prático nem fiável podendo ser até perigoso, ao comutar correntes elevadas em cargas indutivas num período de tempo muito pequeno, geram-se tensões elevadas, criando arcos eléctricos, que resultam em temperaturas elevadas, capaz de fundir ou vaporizar os contactos.

A solução escolhida consiste na utilização de uma ponte H, Fig. 16, esta possui este nome devido à forma como os componentes comutadores e carga estão dispostos no circuito. Os componentes comutadores são estáticos não sendo necessário desconectar terminais, não são necessárias tensões negativas, permite o controlo de velocidade e direcção com um único circuito.

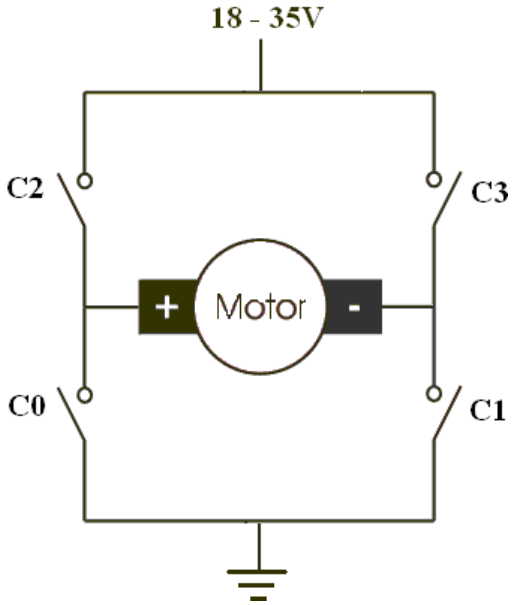


Fig. 16 – Diagrama simplificado de uma Ponte H

3.3.1 Princípio de Funcionamento

A Fig. 16, representa o diagrama simplificado de uma ponte H, esta é constituída por quatro componentes comutadores estáticos, cada um com dois estados possíveis (Ligado, Desligado). Estes comutadores ou estão todos desligados ou então ligam aos pares (três ou quatro haveria um curto-circuito e um haveria um circuito aberto). No momento que dois estão ligados existem dois desligados, desta forma podem-se formar sete circuitos diferentes. A Tabela II mostra os estados possíveis da ponte H, indicando o efeito sobre o motor assim como na ponte (Estado 5 e 6).

Nos parágrafos seguintes é descrito o funcionamento da ponte H em função do estado(s) em que se encontra.

	Comutador				
Estado	**C0**	**C1**	**C2**	**C3**	**Motor / Ponte H**
1	**Desligado**	**Desligado**	**Desligado**	**Desligado**	**Parado**
2	**Desligado**	**Ligado**	**Ligado**	**Desligado**	**Roda - Direita**
3	**Ligado**	**Desligado**	**Desligado**	**Ligado**	**Roda - Esquerda**
4	**Ligado**	**Ligado**	**Desligado**	**Desligado**	**Travado**
5	**Desligado**	**Desligado**	**Ligado**	**Ligado**	**Travado**
6	**Ligado**	**Desligado**	**Ligado**	**Desligado**	**Curto-circuito**
7	**Desligado**	**Ligado**	**Desligado**	**Ligado**	**Curto-circuito**

Tabela II – Estados da ponte H

No **estado 1,** (Fig. 17), todos os comutadores encontram-se em aberto (estado desligado), o circuito resultante não permite que circule uma corrente no motor, estando ou ficando o motor parado. Este estado é utilizado quando se pretende parar o motor, de modo que este efectue uma travagem livre (não forçada).

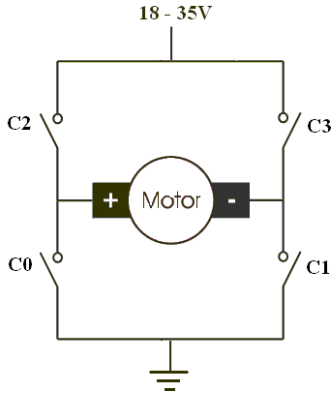


Fig. 17 – Estado 1, todos os comutadores em aberto

No **estado 2,** (Fig. 18), os comutadores C1 e C2 encontram-se fechados (estado ligado) estando C0 e C3 abertos, nesta configuração é aplicada uma tensão directa aos terminais do motor, fazendo com que este rode para a direita. A Fig. 18 ilustra os comutadores fechados e abertos, o sentido da corrente que circula na ponte e no motor (verde) assim como o sentido de rotação do motor (seta vermelha).

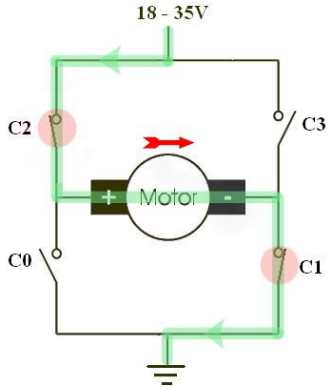


Fig. 18 – Estado 2, comutadores C1 e C2 fechados, motor roda para a direita

No **estado 3,** (Fig. 19), é o inverso do estado 2, os comutadores C1 e C2 encontram-se abertos estando C0 e C3 fechados, desta forma a tensão aplicada no motor é uma tensão inversa circulando uma corrente no sentido inverso, fazendo com que este rode para a esquerda.

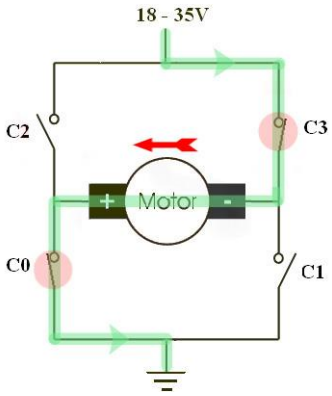


Fig. 19 – Estado 3, comutadores C0 e C3 fechados, motor roda para a esquerda

No **estado 4 e 5, (**Fig. 20**),** se o motor estiver a rodar este efectua uma travagem forçada. Tomando como exemplo o estado 4, os comutadores C2 e C3 estão abertos e C0 e C1 fechados, unindo os terminais do motor. Este como possui inércia devido à sua rotação, deixa de funcionar como motor e passa a funcionar como gerador. A equação (3) mostra como varia a tensão gerada em função da velocidade e da corrente, ao unir os terminais do motor a tensão gerada V_o passa para 0 volts, sendo dissipada toda a energia do motor pela sua resistência interna, como esta é normalmente muito pequena a energia é dissipada rapidamente, parando o motor.

$$Vo = (Kn \cdot n) - (R \cdot i) \quad \textbf{(1)}$$

Onde:

V_o → Tensão gerada aos terminais do motor (V);

K_n → Constante de velocidade do motor, relação entre a velocidade e a força electromotriz gerada na bobina (V/RPM);

n → Velocidade de rotação do motor dc (RPM);

R → Resistência interna do motor (Ω);

i → Corrente que circula nos enrolamentos do rotor (A);

Deve-se ter em atenção ao valor da resistência interna do motor é normalmente muito baixa, o que gera correntes altas nos enrolamentos e dissipação de energia sob a forma de calor o que provoca o aquecimento do motor [19].

No **estado 4**, a diferença entre a Fig. 20 – a) e Fig. 20 – b), é no sentido de rotação antes da travagem, este faz com que o sentido em que circula a corrente após a travagem seja diferente.

O **estado 5**, tem um funcionamento semelhante ao estado 4, a única diferença é que os terminais do motor são unidos pelos comutadores C2 e C3 e não pelos C0 e C1 como acontece no estado 4.

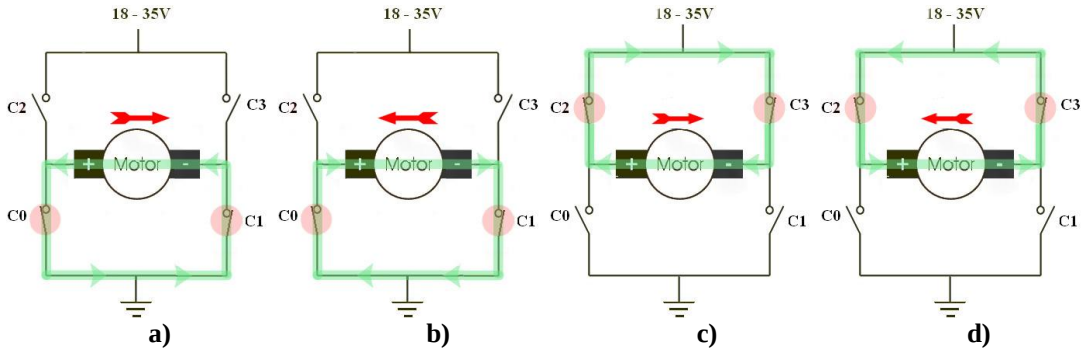
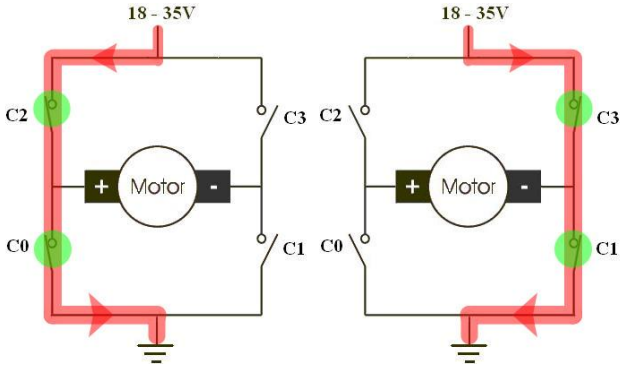


Fig. 20 – Estado 4 e 5, nestes estados o motor efectua uma travagem forçada

O **estado 6 e 7,** (Fig. 21), é um estado indesejado, também chamado de *shoot through,* acontece quando dois comutadores do mesmo lado da ponte são fechados em simultâneo, no caso da Fig. 21 – a) são os comutadores C0 e C2 que estão a provocar o curto-circuito, sendo os comutadores C1 e C3 no caso da Fig. 21 – b).

Estes curto-circuitos têm como única resistência a resistência interna da fonte de alimentação (bateria) e a resistência dos comutadores quando estão fechados. Como esta resistência é normalmente muito baixa circulam correntes elevadas nos comutadores fazendo com que estes sejam completamente destruídos.

Existem técnicas para evitar que estes estados aconteçam, que serão referidas na descrição do controlador da ponte H [19].



a) b)

Fig. 21 – Estado 6 e 7, nestes estados a ponte H fica em curto-circuito

3.3.2 Controlo de Velocidade através do controlo PWM (Pulse Width Modulation)

Até ao momento foram referidos dois tipos de travagem e como se muda o sentido de rotação, não tendo sido abordado como se varia a velocidade do motor.

Existem duas tecnologias básicas para fazer o controlo de velocidade, o controlo linear e o controlo PWM ou *Pulse Width Modulation* (modulação de largura de pulso), sendo seguidamente descritas.

O **controlo linear de potência** consiste em variar linearmente a corrente ou a tensão aplicada numa carga. A forma mais simples de o fazer e de perceber o método é com o uso de um reóstato, exemplo na Fig. 22.

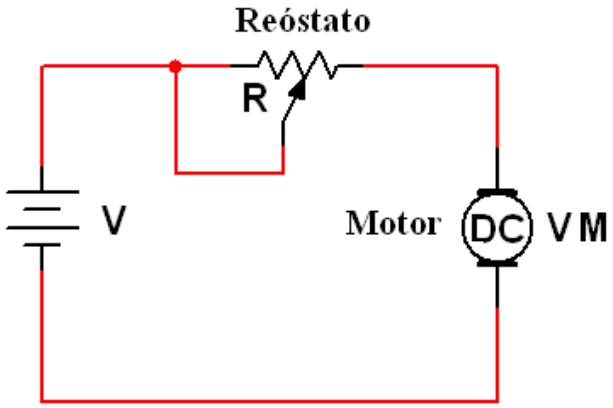


Fig. 22 – Circuito de controlo linear

Analisando a Fig. 22, verifica-se que a tensão aplicada no motor (VM) é igual à tensão de alimentação (V) menos a queda de tensão no reóstato (R). Aumentando a resistência do reóstato a tensão do motor VM para um V constante é menor diminuindo

a velocidade do motor como mostra o gráfico da Fig. 23, onde (n) é a velocidade nominal, (V) a tensão da fonte e (VM) a tensão aplicada no motor.

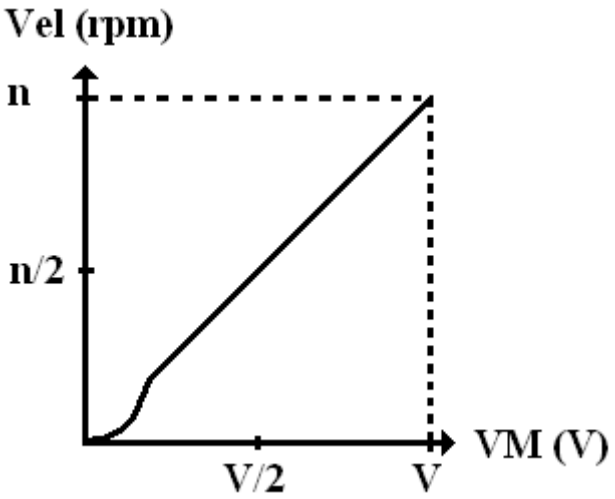


Fig. 23 – Gráfico que representa a velocidade do motor em função da tensão aplicada ao motor através de controlo linear

O controlo linear de potência é normalmente implementado com recurso a componentes electrónicos, tais como transístores bipolares, transístores *darlingtons,* MOSFETs, transístores de efeito de campo, operando numa zona linear (zona entre a zona de corte e a saturação).

Este tipo de controlo possui essencialmente duas desvantagens pelas quais não foi utilizado. Uma desvantagem é o facto de não se conseguir um controlo preciso para baixas rotações, devido a inércia do motor a sua velocidade não acompanha linearmente a tensão de alimentação (ver Fig. 23), tendo um arranque irregular e de difícil controlo. A outra desvantagem encontra-se no elemento linear utilizado no controlo, este para provocar uma queda de tensão dissipa a energia sob a forma de calor, a titulo de exemplo, quando o motor gira a metade da velocidade nominal, a tensão aos terminais deste é aproximadamente metade o que implica que a energia consumida pelo motor é igual à energia dissipada, sendo estas as desvantagens que inviabilizam o uso deste tipo de controlo [20], [21].

O **controlo PWM** ou *Pulse Width Modulation* (modulação de largura de pulso), é um controlo mais preciso, permite um controlo de baixa velocidade com precisão, e uma eficiência energética acima dos 90%.

O método de controlo PWM consiste em ligar e desligar uma carga, neste caso o motor, a uma frequência fixa através de um comutador, fazendo com que seja

aplicada no motor uma tensão média proporcional à relação entre o intervalo de tempo em que o comutador está fechado (Ton), e o período da frequência de comutação (T). A esta relação dá-se o nome de Duty Cycle (D) [19].

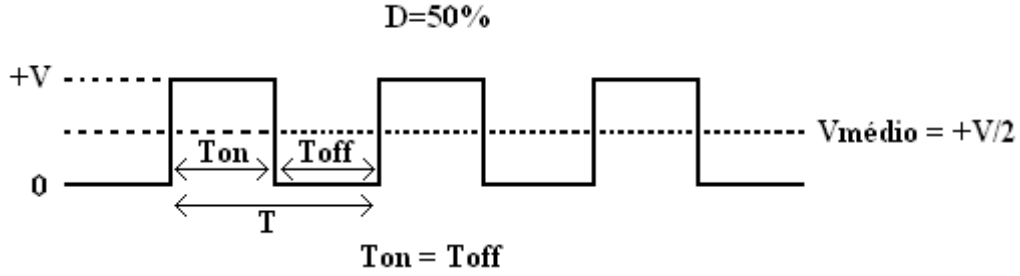


Fig. 24 – Sinal PWM

Na Fig. 24, está representado um sinal PWM com um *duty cycle* de 50% este pode ser calculado através da equação (4), sendo o valor médio da tensão aplicada no motor igual ao produto da tensão de alimentação (+V) pelo *duty cycle* (D), equação (5).

$$D = \frac{Ton}{T} \times 100 \quad \textbf{(2)}$$

$$Vm = +V \times D \quad \textbf{(3)}$$

A Fig. 25 é um exemplo de aplicação do PWM no controlo de velocidade de um motor DC, o sinal PWM abre e fecha o comutador a uma frequência fixa, a tensão dos pulsos aplicada ao motor é a tensão nominal eliminando o problema de controlo a baixas rotações, sendo a tensão média proporcional ao *duty cycle*.

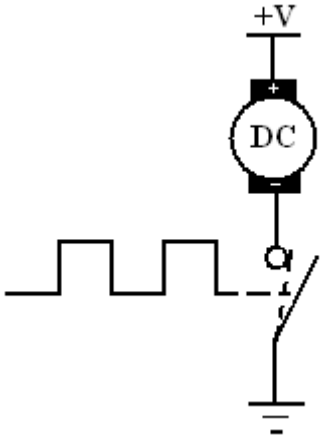


Fig. 25 – Implementação do controlo PWM

Na figura abaixo (Fig. 26) existem três formas de sinais PWM diferentes, todas tem a mesma frequência mas *duty cycles* diferentes. A forma da esquerda (A) quando aplicada ao circuito da Fig. 25, o motor rodaria com uma velocidade muito baixa, o

tempo em que fica desligado é muito maior do que o tempo em que está ligado, sendo a tensão média aplicada ao motor baixa. Na forma de PWM (B), o tempo que está ligado com uma tensão nominal é igual ao tempo em que está desligado logo o valor médio de tensão é aproximadamente metade da tensão nominal sendo a velocidade também aproximadamente metade. A forma C é aquela em que o motor roda aproximadamente à velocidade nominal pois o valor médio de tensão aproxima-se do valor nominal.

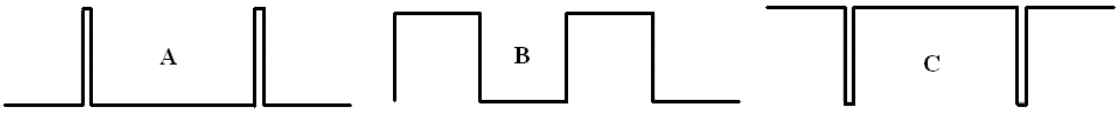


Fig. 26 – Sinais PWM com mesma frequência e *duty cycles* diferentes

Neste momento já se sabe como variar a velocidade faltando integrar o variador de velocidade na ponte H. A Fig. 27 demonstra como o fazer. Tomando como exemplo a Fig. 27 – a) o comutador C1 é mantido fechado sendo aplicado um sinal PWM ao comutador C2 ficando um circuito equivalente ao da Fig. 25. Para variar a velocidade no sentido inverso é fechado C0 em vez de C1 e o sinal PWM é aplicado no comutador C3 em vez de C2 como mostra a Fig. 27 – b).

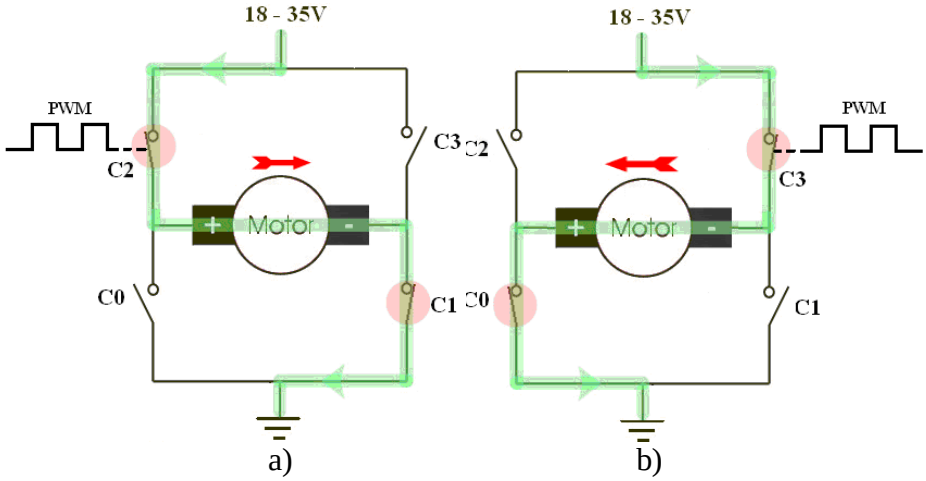


Fig. 27 – PWM integrado na ponte H a) roda para a direita b) roda para a esquerda

Na realidade o sinal PWM também é aplicado no comutador C0 e C1 (Fig. 28), sendo o sinal aplicado em C0 o inverso de C2 e em C1 o inverso de C3, ou seja, quando C2 ou C3 liga, C0 ou C1 desliga, e vice-versa.

Esta situação deverá acontecer devido à indutância existente no motor, a corrente tende a circular de forma constante mesmo quando C2 ou C3 estão desligados, até deixar de existir energia armazenada na indutância do motor. Quando se liga C0 ou

C1 dependendo do sentido de rotação (circuito da Fig. 20) está-se a criar um trajecto de baixa impedância para a corrente fluir. Apesar de travar o motor momentaneamente é este efeito de ligar e desligar alternadamente um braço da ponte H que gera uma tensão média através de pulsos e não uma tensão média contínua, originando as vantagens do controlo PWM na variação de velocidade [19].

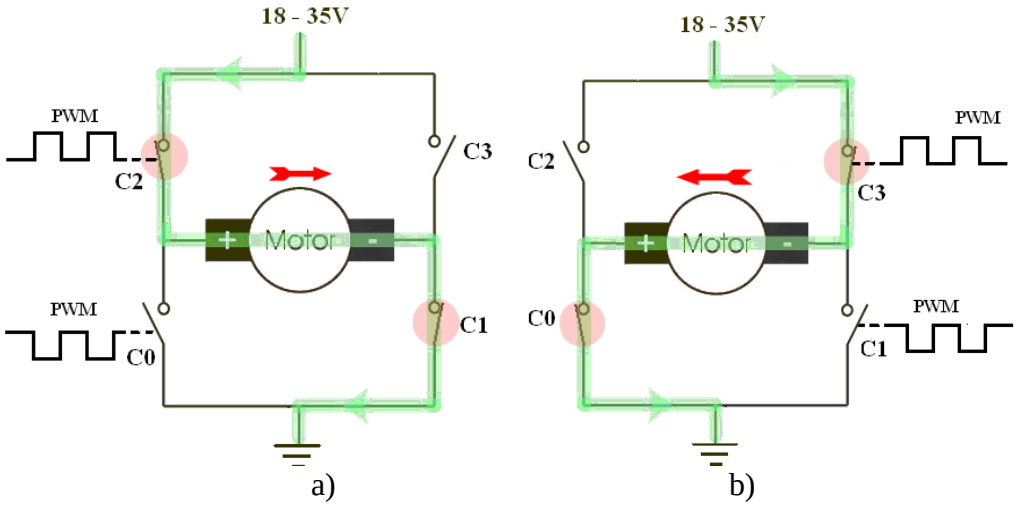


Fig. 28 – PWM aplicado numa ponte H, situação real

A desvantagem dos controlos PWM, deve-se à sua frequência de comutação, normalmente muito alta dependendo do tipo de aplicação. A comutação rápida dos comutadores gera transientes e sinais de alta-frequência, que provocam interferências electromagnéticas (EMI) nos aparelhos próximos que usem ondas de rádio. No entanto podem-se implementar filtros que evitem a propagação destes sinais.

A fase de estudo teórico para controlo de direcção e velocidade do motor DC encontra-se concluída, sendo a fase seguinte a projecção da ponte H utilizada.

3.3.3 Projecto da Ponte H.

Ao iniciar o projecto da Ponte H surgiram duas questões que serviram de base para dar início ao projecto da mesma:

- Existirão no mercado Pontes H já implementadas?
- Caso se tenha que desenvolver que comutadores escolher?

Como surgiram duas perguntas iniciais, procuraram-se respostas, tendo-se encontrado as seguintes:

Existirão no mercado Pontes H já implementadas?

A resposta é sim. Fazendo uma pesquisa encontram-se muitos circuitos integrados que contém pontes H, especialmente no ramo automóvel. Estas vêm integradas com circuitos de potência, lógica, e protecção, no entanto para a aplicação em causa, as pontes H pesquisadas não satisfazem os requisitos.

O integrado BTS 7960B constitui meia ponte H, desenvolvida pela *infineon* esta ponte suporta uma corrente de 43A, uma tensão de alimentação máxima de 27,5V e um PWM que poderá oscilar até 25kHz [22],[23].

Fig. 29 – Ponte H, BTS 7960B [22]

Este integrado cumpre os requisitos para a ponte H a projectar, sendo o único inconveniente a compra, este circuito integrado (CI) só é vendido em grandes quantidades.

O circuito integrado da Fig. 30, VNH2SP30-E, constitui uma ponte H, é desenvolvida pela *STMicroelectronics* para aplicações da indústria automóvel.

A VNH2SP30-E suporta correntes nominais de 30A contínuos e tensões máximas de 16 Volts suportando picos de tensão de 41 V, o seu PWM funciona a 20KHz, possui sensor de corrente e protecções de sobre-temperatura, sobre-corrente e sobre-tensão [25].

Fig. 30 – Ponte H, VNH2SP30-E [24]

O VNH3SP30-E é outro circuito integrado (ponte H) desenvolvido pela *STMicroelectronics*, suporta tensões até 43Ve correntes de 30A contínuos. O seu PWM

opera a uma frequência de 10KHz, igualmente ao circuito integrado VNH2SP30-E este também possui sensor de corrente e protecções de sobre-temperatura, sobre-corrente e sobre-tensão [26].

Tendo em vista os requisitos a cumprir estas pontes H quase os satisfazem, no entanto a ponte H VNH2SP30-E está limitada na tensão de alimentação (requisito mínimo de 24V) e a ponte H VNH3SP30-E está limitada na frequência do PWM (requisito, frequências não audíveis). Estas limitações inviabilizam o uso destas no projecto.

LMD18201 é mais um circuito com uma ponte H integrada, desenvolvida pela *National Semiconductor*. As características desta ponte não permitem que seja usada no projecto, a corrente nominal desta é somente 3A [28].

Fig. 31 – Ponte H LMD 18201, da *National Semiconductor* [27]

Para além dos circuitos integrados anteriormente mostrados foram analisadas as pontes H L298 [29], e MSK 4225 [30], sendo a corrente nominal da ponte H L298 insuficiente para aplicar no projecto e o método de controlo analógico da ponte H MSK 4225 inconveniente para quem pretende controlar a ponte através de um microcontrolador (digital).

A resposta à pergunta inicialmente feita "**Existirão no mercado Pontes H já implementadas?"** é sim, no entanto não foi encontrada nenhuma ponte H que satisfaça os requisitos do projecto.

Não encontrada nenhuma ponte H desenvolvida, é necessário desenvolver uma, para tal, visto o funcionamento desta, é necessário escolher uns comutadores, dando origem à segunda pergunta.

Caso se tenha que desenvolver uma ponte H que comutadores escolher?

A resposta a esta pergunta requer o conhecimento dos semicondutores de potência que permitem comutação, assim como os seus limites de funcionamento.

A Fig. 32, representa os limites (tensão, corrente e frequência) de operação de vários semicondutores de potência (dados de 1994).

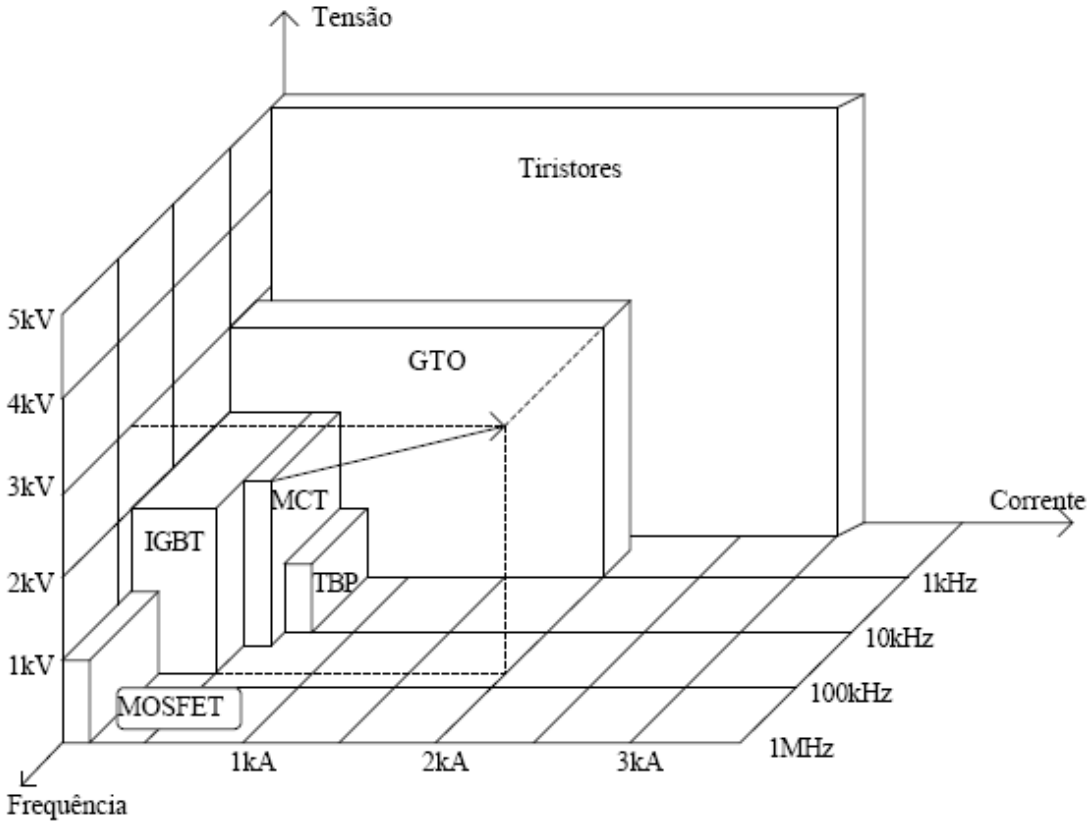


Fig. 32 – Limites de operação de semicondutores de potência [31]

Para que a frequência de comutação dos comutadores da ponte H não seja audível, é necessário que esta opere a uma frequência próxima dos 20kHz. A tensão de alimentação da ponte H será inferior a 1KV e a corrente inferior a 100A.

Partindo da Fig. 32 e das restrições descritas no parágrafo anterior conclui-se que os MOSFET (*Metal Oxide Semiconductor Field Effect Transístor*), IGBT (*Insulated-gate bipolar transístor*) e o MCT(*MOS Controlled Thyristor*) são os semicondutores de potência que se adequam para a construção da ponte H.

Tendo em vista o projecto qualquer um dos semicondutores de potência anteriormente referidos serviria, no entanto foi optado preferencialmente como comutador o MOSFET pelas seguintes razões:

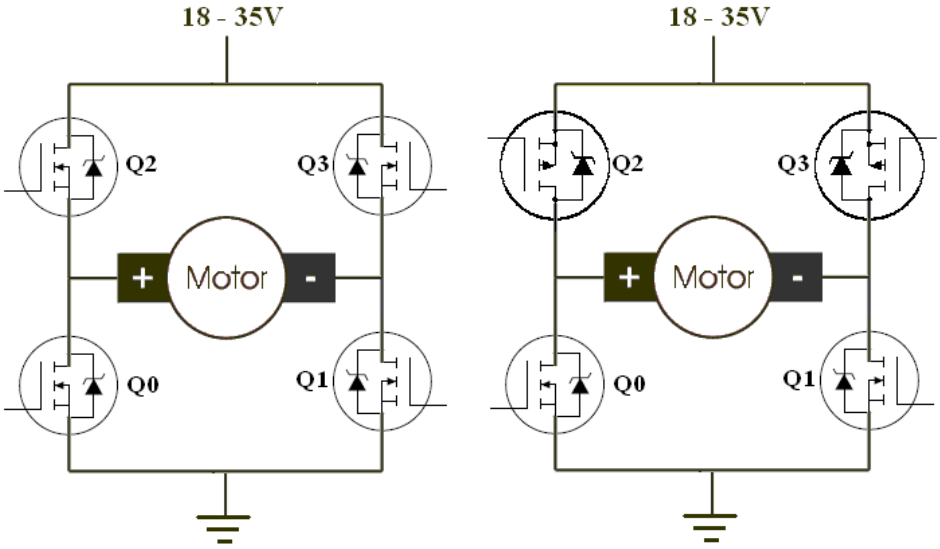
- Baixo custo;
- É facilmente encontrado no mercado;
- É controlado por tensão o que simplifica o circuito de controlo;
- Atinge elevadas frequências de comutação;
- Não drena corrente para o circuito de controlo o que resulta numa protecção do mesmo;
- Adequa melhor às características (tensão, corrente, frequência, preço) para o projecto pretendido.

MOSFETs

Os MOSFETs operam em três regiões, (Corte, Tríodo ou Saturação), sendo apenas usadas para este projecto as regiões de Saturação e Corte (ON - OFF). Quando operam no estado de saturação estes possuem uma resistência (resistência de canal), quanto mais elevado for o valor desta resistência mais elevadas serão as perdas na condução do MOSFET, devendo-se escolher um MOSFET que tenha uma resistência de canal o menor possível.

Outra decisão a ser tomada é quanto ao tipo de MOSFET a ser utilizado, existe MOSFETs de canal N e canal P. Normalmente os de canal N possuem uma menor resistência de canal (menos perdas), no entanto o seu controlo é mais complexo quando utilizados na parte superior da ponte H, ao contrário dos MOSFETs de canal P na mesma condição. A Fig. 33 representa duas soluções, uma [Fig. 33-a)] em que todos os MOSFETs são de canal N sendo as perdas menores e o controlo mais complexo e a outra [Fig. 33-b)] em que os MOSFETs da parte superior da ponte H são de canal P e os da parte inferior de canal N, esta solução possui um controlo menos complexo sendo as perdas maiores [32].

Para se optar por uma solução é necessário conhecer qual a complexidade do controlo dos MOSFETs da primeira solução, sendo necessário conhecer as principais características de comutação dos MOSFETs.



a) b)

Fig. 33 – a) Ponte H com MOSFETs de canal N, b) Ponte H com MOSFETs de canal N e P

O **MOSFET de canal P** encontram-se na região de corte (desligado) quando a tensão entre a *gate* e a *source* (Vgs) é maior que a tensão de *threshold* (Vt) (limiar de condução do MOSFET), equação (6) [33].

$$Vgs > Vt \tag{6}$$

Para que o mesmo passe para a região de saturação é necessário cumprir a equação (7), onde Vds é a tensão entre o *drain* e a *source* [33].

$$Vds \leq Vgs - Vt \tag{7}$$

A Fig. 35 representa um circuito de controlo simples capaz de ligar e desligar os MOSFETs Q2 e Q3 da ponte H, Fig. 33 - b). Tipicamente, para os MOSFETs de canal P a tensão Vgs varia de -10 a -15V e Vt de -2 a -4V.

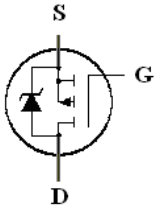


Fig. 34 – Símbolo de um MOSFET de canal P.

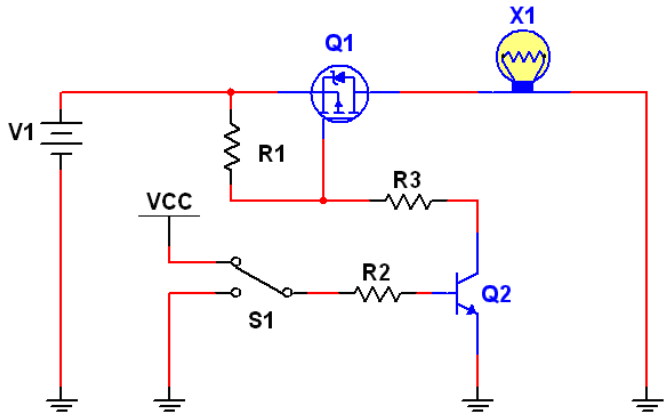


Fig. 35 – Circuito de controlo de MOSFETs de canal P

Os **MOSFETs de canal N** tem um funcionamento semelhante, sendo as tensões inversas. Este encontra-se na região de corte quando a tensão entre a *gate* e a *source* (Vgs) é menor que a tensão de *threshold* (Vt) (limiar de condução do MOSFET), equação (8) [33].

$$Vgs < Vt \tag{8}$$

Para que o mesmo passe para a região de saturação é necessário cumprir a equação (9) (9), onde Vds é a tensão entre o *drain* e a *source* [33].

$$Vds \geq Vgs - Vt \tag{9}$$

A tensão Vgs típica para MOSFETs de canal N varia de 10 a 15V. Para os MOSFETs Q0 e Q1 das duas soluções apresentadas Fig. 33-a) e Fig. 33-b) estas tensões de controlo dos MOSFETs são facilmente obtidas, já que a tensão de alimentação da ponte H varia entre 18 e 35V.

O problema de controlo reside na solução da Fig. 33-a) nos MOSFETs Q2 e Q3, como já foi referido a tensão Vgs para que o MOSFET sature é de 10 a 15 V, no entanto a *source* destes MOSFETs não estão ligadas a 0 V mas sim através do motor e do MOSFET Q0 ou Q1, ignorando a queda de tensão no MOSFET (muito baixa em relação ao motor), resta a queda de tensão do motor. A tensão entre a *gate* dos MOSFETs Q2 e Q3 e a massa (0V) será a tensão Vgs mais a queda de tensão no motor (24V neste caso), que será 34 a 39V. Como a tensão de alimentação varia entre 18 e 35V não é possível saturar os MOSFETs.

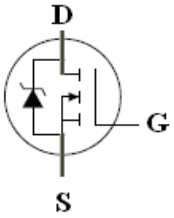


Fig. 36 – Símbolo de um MOSFET de canal N

A solução para este problema consiste em utilizar um elevador de tensão de modo a obter as tensões necessárias para saturar os MOSFETs.

Actualmente existem no mercado circuitos integrados (CI) de controlo para pontes H, estes elevam as tensões para os níveis necessários ao controlo dos MOSFETs de toda a ponte H, possuindo ainda circuitos de protecção. Estas e outras características serão referidas na descrição do mesmo CI posteriormente.

Conclui-se então que o controlo com o uso de todos os MOSFETs de canal N é mais complexo no entanto com o uso de um CI existente para o controlo, esta solução torna-se a melhor, a dificuldade de controlo desaparece sendo a eficiência energética melhor.

Sabe-se até ao momento que o comutador a usar é um MOSFET de canal N faltando determinar a corrente e tensão de modo a procurar no mercado um MOSFET que se adeqúe.

A tensão mínima de operação dos MOSFETs será de 35V no entanto como estes comutam cargas indutivas serão gerados picos de tensão, devendo-se escolher um MOSFET com uma margem de segurança acima dos 35V.

Na maioria dos datasheets de MOSFETs de canal N o valor da resistência de canal é para a temperatura de 25°C, no entanto esta aumenta com a temperatura (Fig. 37), formando um ciclo fechado. Aumentando a temperatura a resistência de canal aumenta, fazendo com que as perdas por efeito de Joule aumentem, aumentando novamente a temperatura. Este ciclo fechado pode levar à destruição do MOSFET, devendo ser escolhido o MOSFET de acordo com a sua resistência interna, corrente, tensão, temperatura de funcionamento e elementos dissipadores. A temperatura de junção (típica ≈170°C) é a principal limitação de potência do MOSFET.

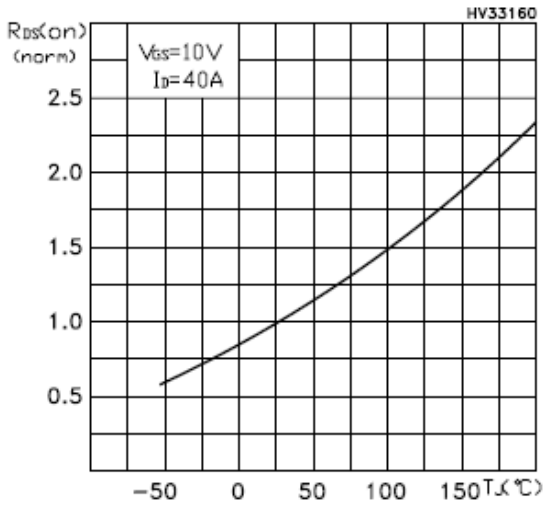


Fig. 37 – Resistência de canal versus temperatura de junção do MOSFET stb140nf55 [34]

Como já referido, a temperatura limita o funcionamento do MOSFET, dependendo das perdas por efeito de joule (comutação e condução) a corrente máxima pode ficar abaixo da corrente nominal do MOSFET, podendo-se ter que sobredimensionar o valor da corrente nominal do MOSFET.

Como a resistência interna, temperatura de junção e tempos de comutação são diferentes para cada MOSFET escolheu-se um MOSFET que cumprisse os requisitos, sendo atribuídas margens de segurança abrangentes, podendo ser ajustadas posteriormente como trabalho futuro.

O MOSFET escolhido é o stb140nf55 da *ST Microelectronics* possuindo as seguintes características:

STB140nf55

Tensão	55V
Corrente	80A
Resistência máxima	8mΩ
Temperatura junção	175°C
Package	TO-220

Dimensionamento Térmico

Escolhido o MOSFET é necessário dimensionar e verificar o sistema térmico, este sistema é responsável por dissipar a energia calorífica de modo que o MOSFET opere dentro de uma temperatura de funcionamento sem se danificar.

Para dimensionar o dissipador é necessário saber qual a energia que vai dissipar, esta energia provem das perdas de comutação e perdas de condução do MOSFET.

As perdas na condução dependem da resistência interna do MOSFET e da corrente que nele circula. Sabendo que i é a corrente que circula no MOSFET e Ron a resistência interna, obtém-se a potência de perdas na condução [35]:

$$Pcondução = Ron \times i^2 \quad \textbf{(4)}$$

Neste caso e para o pior caso (resistência máxima, corrente máxima):

$$Pcondução = 0.008 \times 20^2 = 3.2W$$

Sendo *tr, tf, ton, toff* as temporizações relativas ao CMOS (*Complementary metal-oxide-semiconductor*) e representadas abaixo (Fig. 38), v a tensão de alimentação máxima e f a frequência de comutação, as perdas por comutação podem ser calculadas da seguinte forma [35]:

$$Pcomutação = \left(\frac{tr + tf + ton + toff}{2} \right) \times i \times v \times f \quad \textbf{(5)}$$

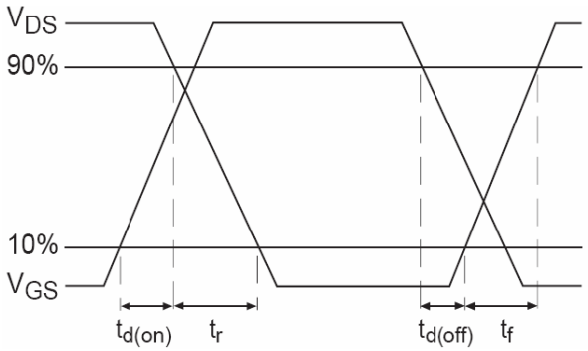


Fig. 38 – Temporizações relativas ao CMOS [35]

Os tempos *tr, tf, ton, toff* são fornecidos no data sheet do MOSFET, a tensão máxima é de 35V e a frequência de comutação é de 19500Hz, de modo que:

$$Pcomutação = \left(\frac{150 \times 10^{-9} + 45 \times 10^{-9} + 30 \times 10^{-9} + 125 \times 10^{-9}}{2} \right) \times 20 \times 35 \times 19500 = 2.39W$$

A potência dissipada pelo dissipador será:

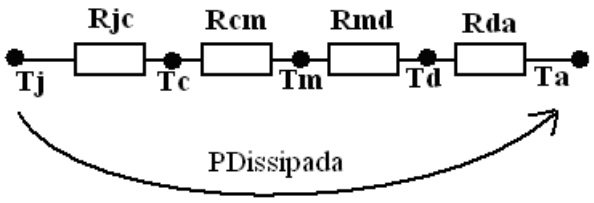
$$Pdissipador = Pcondução + Pcomutação = 3.2 + 2.39 = 5.59W \text{ por MOSFET}$$

Como anteriormente descrito no funcionamento da ponte H, existem 2 MOSFETs que estão sempre a comutar, um outro que está permanentemente ligado e outro que permanece desligado. Dos 2 MOSFETs que comutam quando um liga outro desliga, sendo as perdas na condução equivalentes a 1 MOSFET permanentemente ligado.

Logo as perdas totais serão:

$$Ptotais = 2 \times Pcomutação + 2 \times Pcondução = 2 \times 3.2 + 2 \times 2.45 = 11.18W$$

A temperatura é dissipada desde a junção do MOSFET até à temperatura ambiente, no entanto existem diversos materiais envolvidos que possuem resistências térmicas próprias. O esquema abaixo representa as várias resistências existentes desde a temperatura de junção até à temperatura ambiente.



Tj > Temperatura de junção
Tc > Temperatura da caixa to-220
Tm > Temperatura da mica
Td >Temperatura dissipador
Ta > Temperatura ambiente
Rjc > Resistência junção-caixa
Rcm > Resistência caixa-mica
Rmd > Resistência mica-dissipador
Rda > Resistência dissipador-ambiente

Fig. 39 – Esquema eléctrico equivalente da transferência de energia calorífica

A partir do esquema anterior pode-se obter que:

$$Tj - Ta = Rja \times P \quad \textbf{(6)}$$

Rja > Resistência junção_ambiente

Definindo uma temperatura ambiente máxima de 80°C e uma temperatura de junção máxima de 175°C obtém-se:

$$175 - 80 = Rja \times 11.18 \Leftrightarrow Rja\_máx = 8.50 \text{ °C/W}$$

Do datasheet do MOSFET podem-se retirar os valores das seguintes resistências térmicas:

Rjc (junção - caixa) → 0.5 °C/W

Rja (junção - ambiente)→ 62.5 °C/W

Para o encapsulamento TO-220 com isolamento de mica e pasta térmica, a resistência térmica é aproximadamente 2,5 ºC/W [35]:

Rcd (caixa - dissipador) ≈ 2,5 ºC/W

Através do circuito eléctrico equivalente é possível determinar a resistência térmica do dissipador para o ambiente.

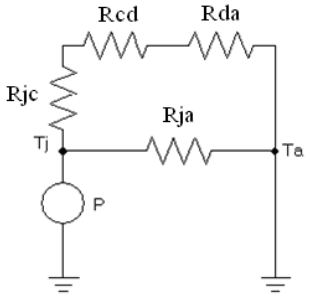


Fig. 40 – Circuito eléctrico equivalente da transferência de energia calorífica [35]

$$\mathrm{Re}\, q = (Rjc + Rcd + Rda) // Rja = 8.50$$

$$8.50 = \frac{[(0.5 + 2.5 + Rda) \times 62.5]}{(0.5 + 2.5 + Rda) + 62.5}$$

$$Rda = 6.85\ ^{\circ}\mathrm{C/W}$$

Deste modo pode ser escolhido um dissipador para todos os MOSFETs cuja resistência térmica seja inferior a 6.85 ºC/W. Feita uma pesquisa foi encontrado o dissipador da Fig. 41, este possui uma resistência de 1.30 ºC/W. O facto desta resistência ter um valor menor aumenta a margem de segurança, já que o cálculo foi feito com a temperatura de junção máxima.

Fig. 41 – Dissipador ventilado da AAVID com resistência de 1.30 ºC/W [36]

Díodos Schottky

O díodo é um componente constituído por uma junção de dois materiais semicondutores (em geral silício ou gerânio dopados), um do tipo n e o outro do tipo p, ou de um material semicondutor e de um metal, sendo usualmente representado pelo símbolo da Fig. 42. Aos terminais A e K dão-se respectivamente os nomes de Ânodo e Cátodo.

Este dispositivo permite a passagem de corrente, com facilidade, num sentido, e oferece uma grande resistência à sua passagem no sentido contrário [37].

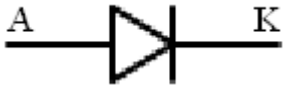


Fig. 42 – Símbolo de um díodo.

Os díodos de schottky diferem na sua construção, é utilizado um metal em vez do material do tipo P, tirando partido do efeito de schottky. O principal destaque do díodo schottky é o menor tempo de recuperação, pois não há recombinação de cargas do díodo de junção. Outra vantagem é a maior densidade de corrente, o que significa uma queda de tensão directa menor que a do díodo comum de junção. A contrapartida é uma corrente inversa maior, o que pode impedir o uso em alguns circuitos. Sendo utilizados principalmente em circuitos de alta-frequência e de alta velocidade de comutação [38].

Na ponte H estes díodos são utilizados para dissipar a energia que provém da inércia do motor para as baterias.

Quando está acoplada uma carga ao motor (ex. robot) e este roda à velocidade máxima com a corrente máxima da ponte ≈20A, existem 2 modos de travagem (excepto rampas de desaceleração), travagem brusca e travagem livre.

Na travagem brusca os terminais do motor são ligados e toda a energia é dissipada nos enrolamentos do motor.

Na travagem livre todos os MOSFETs ficam em aberto, o motor passa a funcionar como gerador sendo a energia dissipada para as baterias através dos díodos internos dos MOSFETs da ponte H, Fig. 43, sendo geradas correntes que podem atingir ≈20A durante alguns segundos (depende da massa acoplada ao motor e da velocidade). De forma a proteger os díodos internos dos MOSFETs (preparados para grandes correntes durante curtos espaços de tempo) foram adicionados em paralelo 4 díodos

(STTH3002CT) que suportam correntes de 30 A contínuos com tempos de recuperação de 17ns.

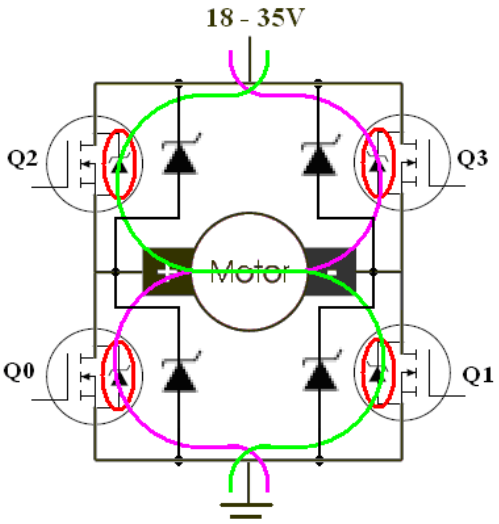


Fig. 43 – Ponte H com MOSFETs de canal N

Condensador Electrolítico

O condensador é um componente que armazena cargas eléctricas, onde a capacidade eléctrica (*C*) relaciona a tensão aos terminais com a respectiva carga armazenada. Os formatos típicos consistem em dois eléctrodos ou placas que armazenam cargas opostas, as placas são condutoras e separadas por um isolante ou por um dieléctrico sendo a carga armazenada na superfície, no limite com o dieléctrico [39].

Os condensadores electrolíticos possuem polaridade, são normalmente utilizados em baixas frequências e possuem grande capacidade comparativamente aos outros tipos de condensadores. A inversão da polaridade neste tipo de condensadores pode levar à destruição do mesmo com risco de explosão, devendo-se ter cuidado ao manusear os mesmos, Fig.44.

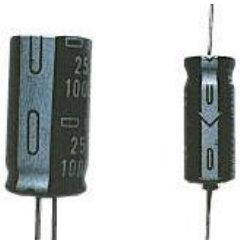


Fig. 44 – Condensador electrolítico [40]

Em paralelo com a alimentação da ponte H é utilizado um condensador electrolítico de grande capacidade, que auxilia a fonte de energia (baterias) nos arranques do motor, mantendo a tensão estável e reduzindo as quedas de tensão nos cabos de alimentação devido às impedâncias dos mesmos.

Para tal utilizou-se um condensador de 1500µF 63V. A capacidade do condensador pode ser reduzida com o aumento da secção dos cabos de alimentação e com o aumento da rapidez de resposta das baterias a pulsos de corrente elevados.

TVS (Transient voltage supressor)

O TVS ou supressor de transientes de tensão é um componente parecido com o díodo zener, este pode ser unidireccional ou bidireccional, tendo como função a absorção dos picos de tensão com valores acima da tensão nominal, possuindo capacidades de absorção de picos de grandes potências (1500W TVS utilizado).

No circuito da ponte H existem 3 TVS (1.5KE46CA) bidireccionais de 46V dispostos de forma que a tensão na fonte e nos MOSFETs não ultrapasse os valores nominais evitando se danifiquem, reduzindo também o ruído electromagnético [19].

A figura seguinte representa o circuito até agora descrito, Fig. 45.

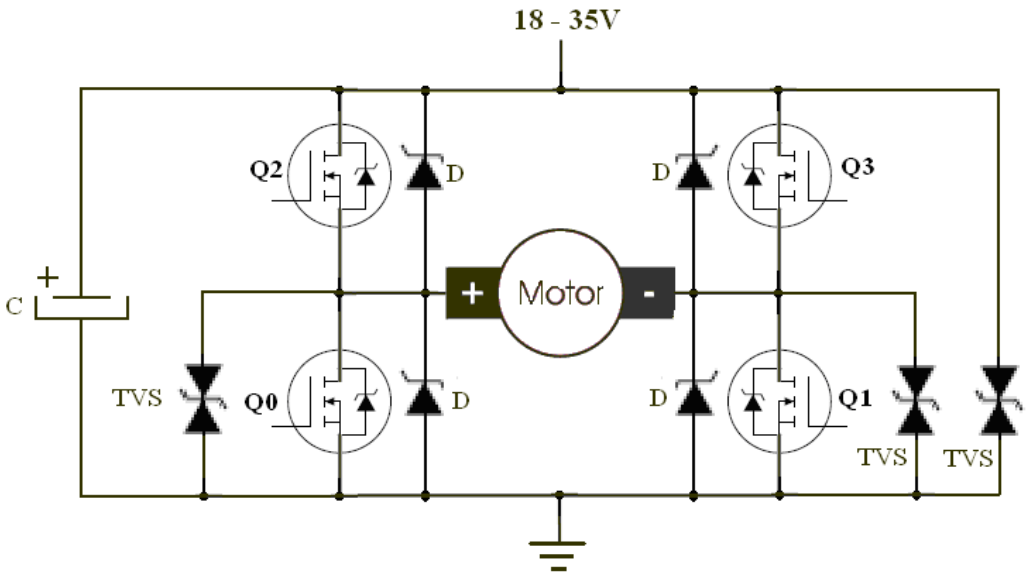


Fig. 45 – Circuito eléctrico analisado até ao momento

Controlador HIP4081A

Os MOSFETs são accionados por tensão ao contrário dos transístores, este facto facilita o desenvolvimento de um circuito de controlo. No entanto, existe uma característica que cria algumas dificuldades, para o MOSFET entrar em condução, é necessário injectar uma carga eléctrica na *gate*, de modo que a tensão Vgs (*gate-source*) seja superior à tensão mínima necessária para que o MOSFET entre em plena condução (10- 15V). A necessidade de injectar cargas no MOSFET pode ser modelada como um condensador em paralelo com a *gate* do MOSFET, este efeito nomeia-se de capacitância parasita [19].

O controlador HIP4081A é ideal para carregar essa capacitância pois este é capaz de fornecer até 2A para os quatro MOSFETs da ponte H. Dependendo dos MOSFETs utilizados a capacidade da *gate* é diferente, exigindo correntes diferentes ao controlador, de forma que as correntes do controlador não excedam os 2A é colocado em série com a *gate* de cada MOSFET uma resistência. Quanto maior for esta resistência menor será a corrente e maior será o tempo que o MOSFET leva para entrar em condução, sendo necessário encontrar uma constante corrente versus velocidade que satisfaça as duas condições.

Com as resistências pode-se controlar o tempo que cada MOSFET demora a entrar em condução, evitando que dois MOSFETs do mesmo lado da ponte H (braço) sejam ligados em simultâneo. Para além desta possibilidade, o controlador HIP4081A possui um tempo programável através de duas resistências, que é introduzindo entre as comutações dos MOSFETS de cada braço da ponte H, Fig. 46.

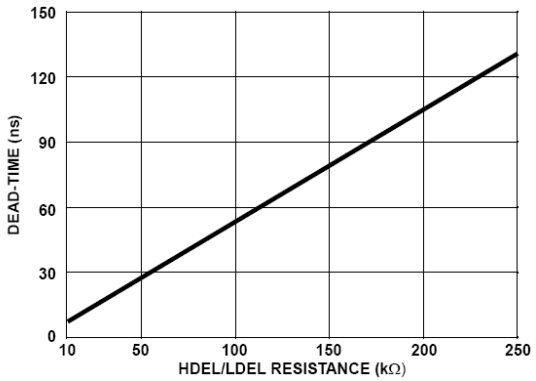


Fig. 46 – Variação do tempo entre as comutações em função das resistências [41]

Com a adição de diodos rápidos na *gates* dos MOSFETs, em paralelo com as resistências existentes, é possível diminuir o tempo que este demora a desligar. Quando

o MOSFET é desligado a energia armazenada na capacitância da *gate* é rapidamente dissipada através do díodo e não da resistência, diminuindo o tempo para ficar ao corte [19].

O CI (circuito integrado) HIP4081A tem a função de accionar os MOSFETs da parte inferior e superior da ponte H, para accionar estes últimos o controlador possui um circuito elevador de tensão.

A Fig. 47 representa um diagrama de aplicação do controlador HIP4081A, este pode ser alimentado por uma tensão desde 9,5 a 15V, gerando pulsos paras as gates dos MOSFETs em pontes H, cuja tensão de alimentação não ultrapasse os 80V. Se a tensão for inferior a 9,5V as saídas para os MOSFETs são desligas, caso a tensão ultrapasse os 16V o controlador pode-se danificar [19].

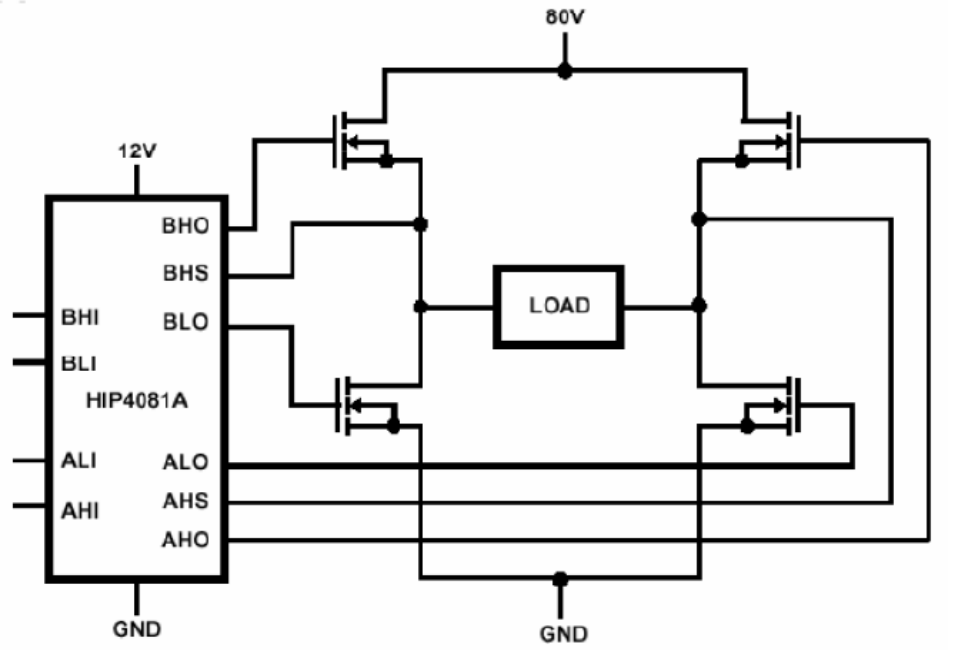


Fig. 47 – Diagrama simplificado de aplicação do controlador HIP4081A [19]

O HIP4081A possui quatro entradas digitais, AHI, ALI, BHI, e BLI, correspondendo às saídas que accionam cada MOSFET, AHO, ALO, BHO e BLO, respectivamente (Fig. 47). Quando uma entrada for activa, a saída correspondente faz com que o MOSFET entre em condução. O microcontrolador posteriormente descrito é responsável por enviar sinais PWM e direcção para as entradas do controlador HIP4081A de forma a controlar a ponte H. Esses sinais digitais são compatíveis com a lógica TTL, sendo qualquer tensão acima de 3V reconhecida como sinal de nível alto [19].

Para além das protecções mencionadas o controlador possui ainda uma protecção interna, que evita comutações simultâneas desligando o MOSFET superior quando o MOSFET inferior do mesmo braço é activado. O pino DIS (disable) permite que todas as saídas para os MOSFETs sejam desablitadas desligando a ponte H [19].

Devido à natureza dos MOSFETs utilizados (canal N), a tensão na *gate* deve ser 10 a 15V superior à tensão da *source*. Para gerar tensões acima da tensão de alimentação do controlador, possibilitando o accionamento dos MOSFETs superiores, este possui um sistema elevador de tensão, que, com a ajuda de um díodo rápido e um condensador externos, gera a tensão necessária para o accionamento dos respectivos MOSFETs [19].

A Fig. 48, representa a tabela de verdade das entradas lógicas do controlador HIP4081A a partir da qual se obtêm os sinais necessários em função da operação pretendida.

ALI, BLI	AHI, BHI	DIS	ALO, BLO	AHO, BHO
X	X	1	0	0
1	X	0	1	0
0	1	0	0	1
0	0	0	0	0

X = DON'T CARE 1 = HIGH/ON 0 = LOW/OFF

Fig. 48 – Tabela de Verdade das entradas lógicas do HIP4081A [41]

A partir da Fig. 48 chega-se à Fig. 49 que representa as entradas aplicadas ao controlador através do microcontrolador em função da acção pretendida.

AHI	BHI	ALI	BLI	DIS	Função
PWM	X	$\overline{PWM}$	1	0	Direita
X	PWM	1	$\overline{PWM}$	0	Esquerda
X	X	1	1	0	Travado
1	1	0	0	0	Travado
X	X	X	X	1	Livre

X= Irrelevante 1 = Ligado 0 = Desligado

Fig. 49 – Tabela de Verdade das entradas aplicadas em função da função pretendida

Buffer 74HCT373N

O circuito integrado 74HCT373N consiste num conjunto de 8 *latchs* do tipo D de 3 estados que forma um *buffer* entre o microcontrolador e o controlador HIP4081A, Fig. 57. A sua principal função é proteger o microcontrolador (encapsulamento SMD) de possíveis avarias no circuito de potência, podendo danificar este CI de fácil substituição mas não o microcontrolador.

Este circuito integrado possui 20 pinos (Fig. 50 esquerda), permite funções de leitura/escrita e colocar as saídas em alta impedância. Neste caso é somente utilizado como saída estando os pinos de *lacth enable* e *output enable* sempre ligados a 5V e 0V respectivamente, desta forma todos os níveis lógicos presentes nas entradas Dn reflectem-se nas saídas Qn correspondentes. Através da Fig. 50 à direita e da Fig. 51, é possível saber todos os estados possíveis e desejáveis para este circuito integrado.

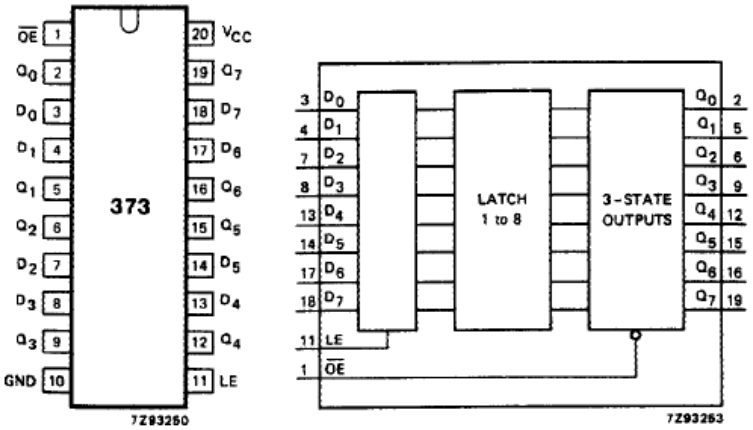


Fig. 50 – À esquerda está representado a configuração dos pinos e à direita o diagrama funcional [42]

OPERATING MODES	INPUTS			INTERNAL LATCHES	OUTPUTS
	$\overline{OE}$	LE	D_n		Q_0 to Q_7
enable and read register (transparent mode)	L	H	L	L	L
	L	H	H	H	H
latch and read register	L	L	l	L	L
	L	L	h	H	H
latch register and disable outputs	H	X	X	X	Z
	H	X	X	X	Z

Notes

1. H = HIGH voltage level
   h = HIGH voltage level one set-up time prior to the HIGH-to-LOW LE transition
   L = LOW voltage level
   l = LOW voltage level one set-up time prior to the HIGH-to-LOW LE transition
   X = don't care
   Z = high impedance OFF-state

Fig. 51 – Tabela de verdade de funcionamento do CI, 74HCT373N [42]

Outra particularidade que se deve salientar é o facto de se usar o circuito integrado 74HCT373N e não o 74LS373N, os dois fazem a mesma função no entanto os níveis de tensão em que se difere o nível lógico alto e baixo é diferente sendo o circuito integrado 74LS373N incompatível com os níveis de tensão do controlador HIP4081A.

Fontes de alimentação internas

A tensão de alimentação do controlador HIP4081A não deve exceder os 15V correndo o risco de se danificar. Como a tensão de alimentação varia de 18 e 35V é necessário criar uma fonte de alimentação interna, de modo que a tensão de saída se mantenha nos 15V independentemente do valor da tensão de alimentação (qualquer valor entre 18V e 35V). Outro factor tido em atenção é o consumo do sistema, a corrente que circula no sistema com o motor desligado varia entre 110 e 170mA (diferença depende se o ventilador está ligado ou desligado). No pior dos casos quando a alimentação é de 35V a queda de tensão será de 20V sendo a potência dissipada (fonte linear) de ≈3.4W. A maioria dos reguladores de tensão lineares aquecem muito sendo necessário dissipadores logo mais espaço.

A solução encontrada, Fig. 58, recorre ao uso de uma fonte comutada da *Texas Instruments* (PTN78000W), Fig. 52 (esquerda). Esta possui uma eficiência de 95%, a tensão de entrada pode variar de 15 a 36V sendo a tensão de saída ajustável, fornecendo uma corrente até 1.5A. Para além das vantagens já descritas a sua implementação é bastante simples requerendo unicamente três componentes externos, Fig. 52 (direita).

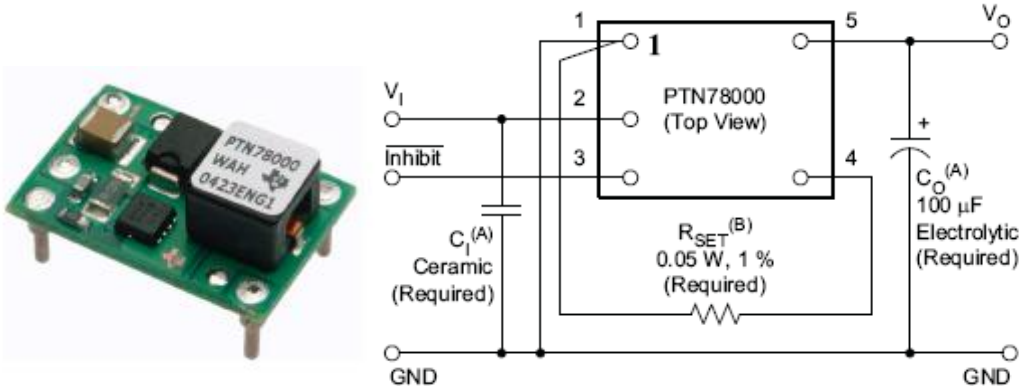


Fig. 52 – Fonte comutada PTN78000W da Texas Instruments (esquerda), esquema de ligações standard (direita) [43]

Solucionado o problema de alimentação do controlador HIP4081A é necessário outra fonte de alimentação para os circuitos integrados (*buffer*, microcontrolador). Esta

será de apenas 5V, neste caso, é utilizado o regulador de tensão linear LM117 da National tendo como tensão de entrada os 15V da fonte anterior, Fig. 53. Em semelhança ao regulador anterior este também permite o ajuste da tensão de saída e correntes até 1.5A.

O facto que levou a utilizar um regulador linear em vez de utilizar novamente a solução anterior foi a relação custo eficiência, na fonte de 5 volts as perdas são muito menores não se justificando a implementação de um regulador comutado, Fig. 58.

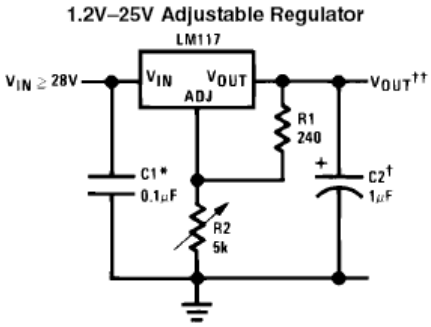


Fig. 53 – Circuito básico de ligações do regulador LM317 [44]

Accionador do ventilador do dissipador

O dissipador escolhido contém um ventilador, Fig. 41. Este funciona com uma tensão próxima dos 12V não sendo possível accioná-lo directamente através do microcontrolador (saídas 5V). O circuito abaixo representado consiste na utilização de um transístor que funciona como interruptor de forma a poder ligar e desligar o ventilador através do microcontrolador.

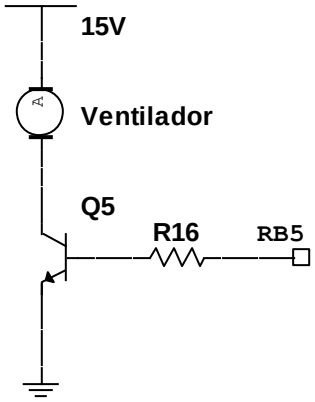


Fig. 54 – Circuito auxiliar de controlo do ventilador

Circuito Final da ponte H

Depois de descrito todo o hardware utilizado na projecção e construção da ponte H são apresentados abaixo os circuitos eléctricos com o respectivo material utilizado.

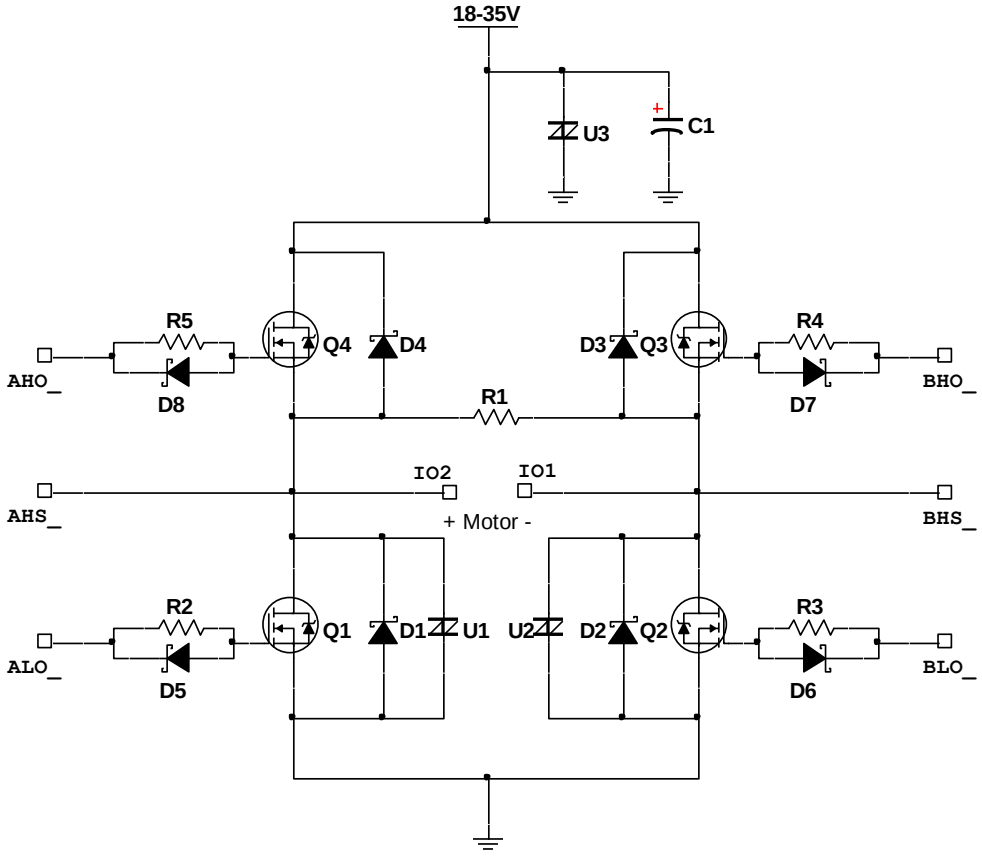


Fig. 55 – Circuito de potência da ponte H

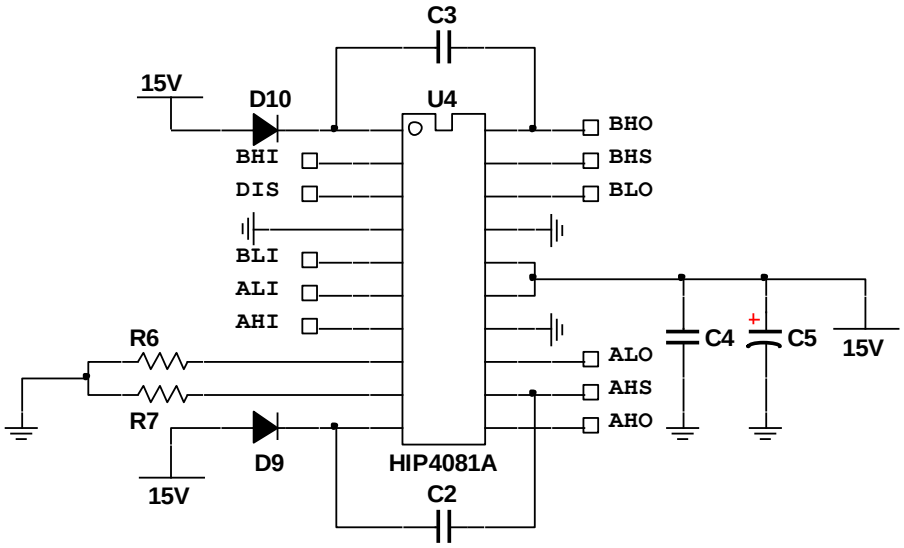


Fig. 56 – Circuito do controlador HIP4081

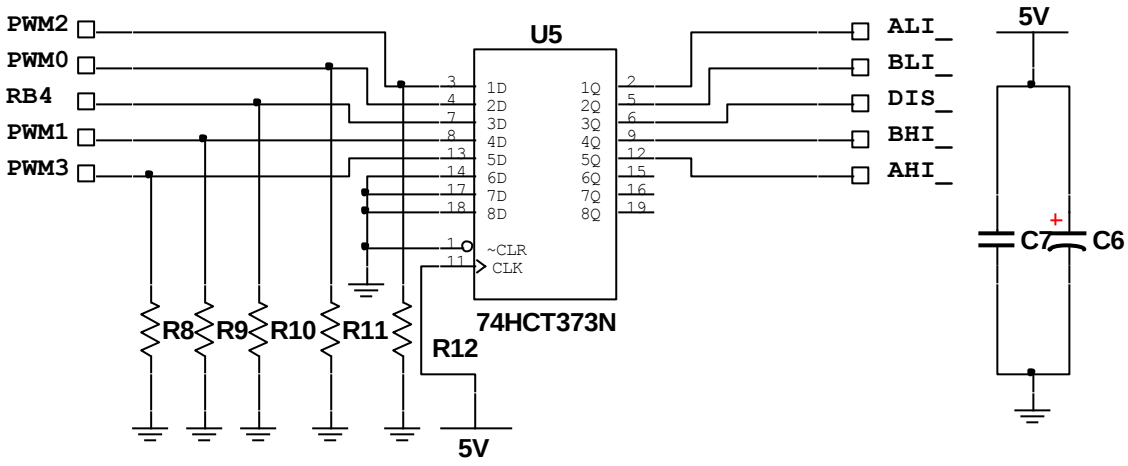


Fig. 57 – Circuito do *buffer* de protecção do microcontrolador

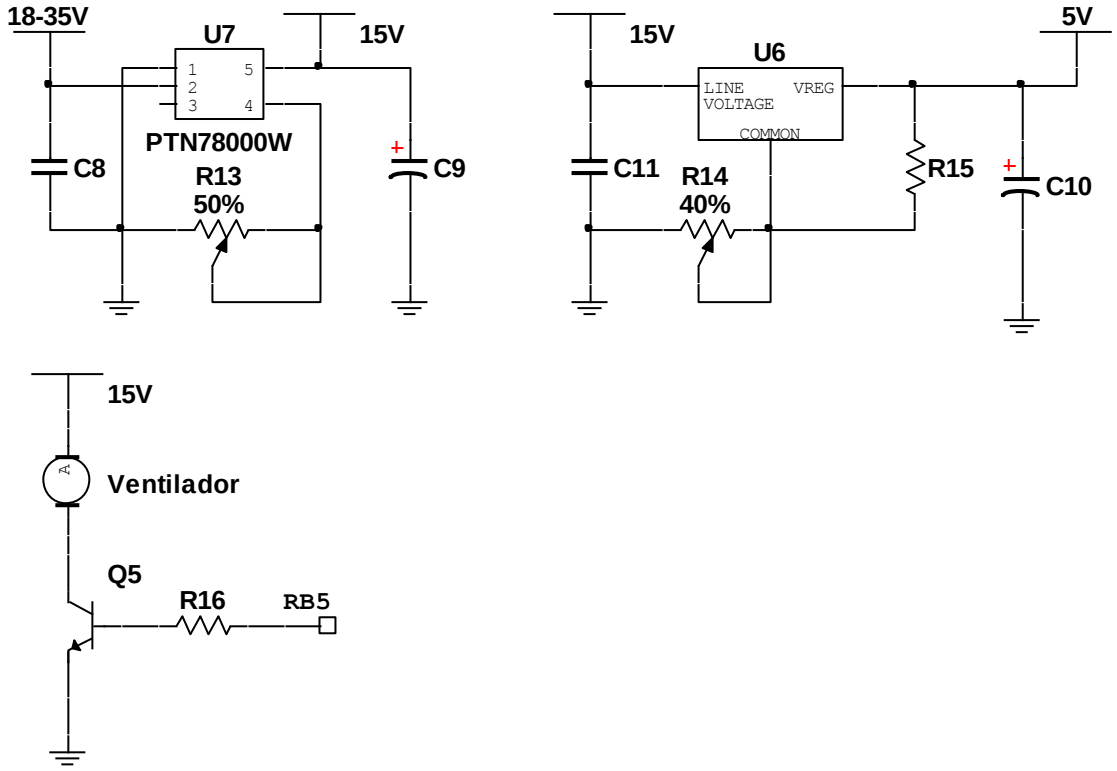


Fig. 58 – Circuito das fontes de alimentação internas e do accionador do ventilador

Componentes

Quantidade	Descrição	Referencia no Circuito
4	SCHOTTKY_DIODE, STTH3002CT	D4, D3, D2, D1
4	SCHOTTKY_DIODE, 1N5711	D8, D7, D6, D5
2	DIODES UF4002	D10, D9
1	CONTROLLER HIP4081A	U4
1	74HCT373N	U5

3	VOLTAGE_SUPPRESSOR, P6KE47CA	U3, U2, U1
4	POWER_MOS_N, stb140nf55	Q4, Q3, Q2, Q1
1	BJT_NPN, 2N2222A	Q5
1	VOLTAGE_REGULATOR, LM117H	U6
1	STEP_DOWN, ptn78000w	U7
1	CAP_ELECTROLIT, 1500uF	C1
2	CAP_ELECTROLIT, 100uF	C10, C9
2	CAP_ELECTROLIT, 10uF	C5, C6
3	CAPACITOR, 1uF	C3, C2, C8
3	CAPACITOR, 100nF	C4, C7, C11
2	RESISTOR, 240kΩ	R7, R6
1	RESISTOR, 10kΩ	R1
6	RESISTOR, 1kΩ	R12, R11, R10, R9, R8, R16
1	RESISTOR, 240Ω	R15
4	RESISTOR, 47Ω	R5, R4, R3, R2
1	POTENTIOMETER, 10kΩ	R14
1	POTENTIOMETER, 50kΩ	R13

Tabela III – Tabela do material utilizado nos circuitos anteriores

3.4 Conclusões

O estudo e projecção da ponte H, assim como o estudo do motor DC, geraram conhecimentos, salientando-se os mais importantes nos seguintes pontos:

- A velocidade varia directamente com a tensão de alimentação.
- O sentido de rotação depende do sentido da corrente no enrolamento do rotor.
- O binário nominal deve ser aplicado à velocidade nominal. Se o motor rodar abaixo da velocidade nominal, a tensão de alimentação for abaixo da nominal, para manter o binário nominal este consome uma corrente acima da nominal podendo danificar o motor.
- Comparativamente aos outros sistemas analisados a ponte H tem as seguintes vantagens. Os componentes comutadores são estáticos não sendo necessário desconectar terminais, não são necessárias tensões negativas, permite o controlo de velocidade e direcção com um único circuito.
- O controlo PWM também oferece vantagens em comparação com o controlo linear, este é um controlo mais preciso permite um controlo de baixa velocidade com precisão e uma eficiência energética acima dos 90%.

- No mercado encontram-se muitos circuitos integrados que contêm pontes H, especialmente no ramo automóvel. Estas vêm integradas com circuitos de potência, lógica, e protecção, no entanto para a aplicação em causa, as pontes H pesquisadas não satisfazem os requisitos.
- Os MOSFETs tornaram-se uma boa opção, são relativamente baratos, facilmente se encontram, são controlados por tensão e comutam a frequências elevadas. Os MOSFETs de canal N comparativamente com os MOSFETs de canal P possuem menor resistência de canal e comutam a frequências mais elevadas.
- Dos vários controladores testados o HIP4081A foi o que demonstrou maior fiabilidade, tendo a vantagem de que um único CI controla todos os MOSFETs da ponte H.
- Quando se encontrava em fase de testes, existiu uma situação em que o *buffer* foi danificado cumprindo o seu objectivo de proteger o microcontrolador.
- Toda a electrónica adicional à ponte H revelou-se totalmente eficaz.
- Devido há indutância do motor o tempo de intervalo entre comutações de MOSFETs do mesmo lado da ponte H não foi o suficiente. Este problema resolveu-se configurando os tempos de espera entre os pulsos PWM no microcontrolador (dead time).
- O sistema dissipador de calor e os MOSFETs mostraram-se muito eficazes, em plena carga a variação da temperatura em relação à temperatura ambiente foi aproximadamente 15° C.

Capítulo 4 - Microcontrolador

4.1 Introdução

O capítulo anterior descreve para além do motor DC o projecto da ponte H e todas as suas funcionalidades. No entanto, esta sem adição de algum componente que lhe envie sinais de controlo, não tem muita utilidade.

Analisando os requisitos para este projecto verifica-se a existência de sensores, algoritmos de controlo, comunicações digitais (I2C), leitura de *encoders* e sinais de controlo para a ponte H, Fig. 59. Para executar todas as funções é utilizado um microcontrolador, evitando o uso de muita electrónica discreta para o mesmo fim.

O microcontrolador é um dispositivo electrónico, capaz executar vários milhões de tarefas por segundo, possuindo módulos integrados que permitem a execução de funções específicas. Ao contrário dos componentes tradicionais a compra não define a sua a tarefa final, este permite a programação das tarefas a ser executadas.

Este capítulo descreve a escolha do microcontrolador utilizado, hardware envolvente ao funcionamento do mesmo, assim como, as linguagens de programação e funções programadas. Função *main* (função principal), função leitura de sensores/protecção, função gerador de sinais PWM e função de comunicação I2C. Existem outras duas funções implementadas no microcontrolador no entanto devido há sua complexidade serão descritas no próximo capítulo (algoritmo de controlo e leitura do *encoder*).

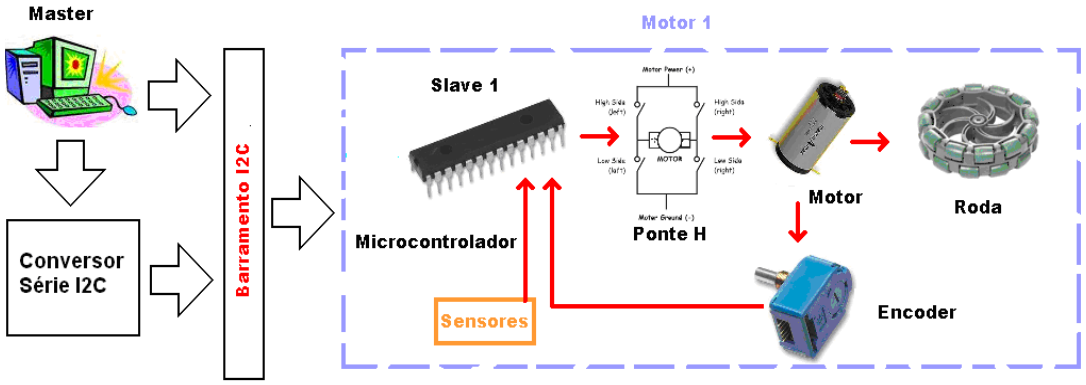


Fig. 59 – Diagrama de ligações da solução projectada

4.2 Escolha do Microntrolador

Sendo o microcontrolador o cérebro do projecto é necessário escolher um que seja capaz de executar todas as funções pretendidas de forma fiável e eficaz. Surgindo a seguinte pergunta:

Sendo o mercado tão vasto que microcontrolador escolher?

Não sendo uma resposta fácil, pois o mercado nesta área é enorme havendo vantagens e desvantagens entre os microcontroladores relativamente a este projecto.

O primeiro passo era escolher uma marca de microcontroladores. Como já havia trabalhado com microcontroladores e estava familiarizado com uma marca, tornou-se fácil a escolha, sendo escolhido os microcontroladores PIC da Microchip. O envio de amostras grátis, variedade enorme de microcontroladores, linguagem de programação acessível, compiladores livres e fórum de ajuda, tornam a escolha do fabricante Microchip muito atractiva.

O segundo passo era escolher dentro das centenas de microcontroladores que a Microchip possui, um microcontrolador que satisfaça os requisitos. Para tal o site da Microchip é uma grande ajuda, pois possui uma ferramenta que aconselha um microcontrolador em função da aplicação, sendo o microcontrolador aconselhado e utilizado o PIC18f2431.

4.2.1 Microcontrolador PIC18f2431

Dentro das várias famílias de microcontroladores da Microchip, a família PIC18f encontra-se no topo dos microcontroladores de 8 bits e aproximadamente a meio de todas as famílias tendo em vista a performance e as funcionalidades, Fig. 60.

O microcontrolador escolhido, PIC18f2431, possui dentro de muitas funcionalidades, algumas interessantes para o projecto, das quais se salienta a função de leitura de *encoder,* 6 canais PWM de 14 bits que funcionam em modo complementar utilizados no controlo da ponte H, interface de comunicação I2C que permite comunicar com um *master* e alguns sensores e 5 canais ADC utilizados para leitura de sensores. O seu cristal funciona até 40MHz executando 10 milhões de instruções por segundo (MIPS). Este microcontrolador possui 28 pinos, sendo os encapsulamentos disponíveis PDIP e SOIC.

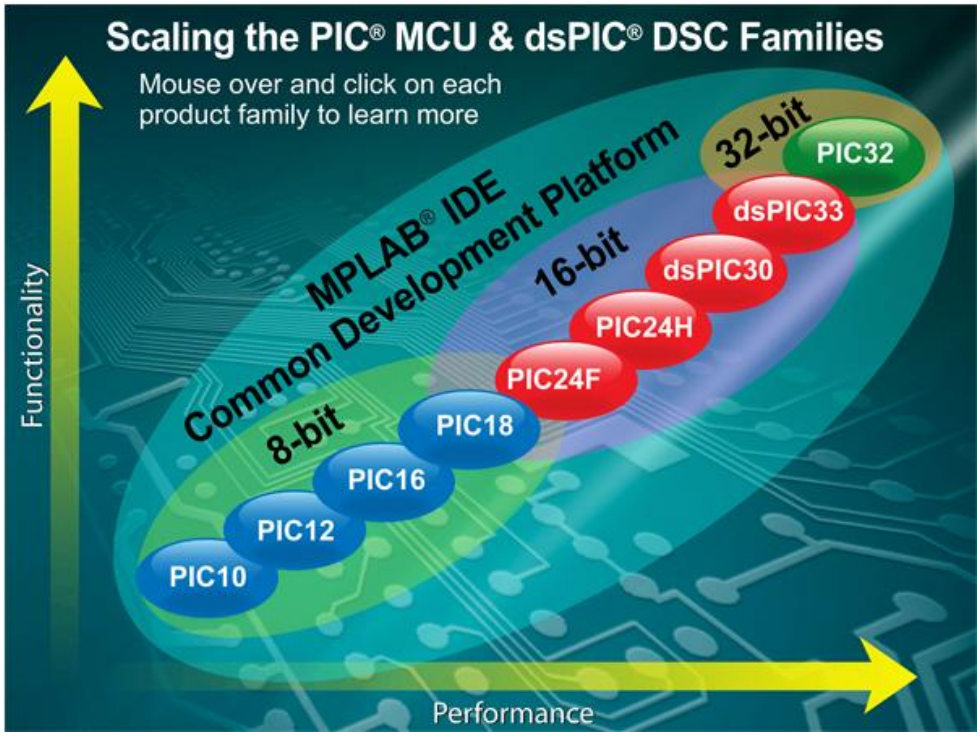


Fig. 60 – Funcionalidades e Performances de todas as famílias de microcontroladores da Microchip, [45]

Podendo-se encontrar no microcontrolador PIC18f2431, todas as características referidas abaixo:

- Arquitectura *Havard*
- Conjunto de instruções RISC (75 instruções)
- 16KB de memória *Flash* (memória de código)
- 768 Bytes de memória SRAM (memoria de dados)
- 256 Bytes de memória EEPROM
- 22 fontes de interrupção
- 4 Timers
- Comunicação série, paralela, I2C
- 6 canais PWM de 14 bits
- 5 canais ADC
- Leitura de encoder
- 24 pinos de entrada e saída

Todas as informações anteriores referentes ao microcontrolador e muitas outras indicando o funcionamento detalhado do mesmo podem-se encontrar no manual do microcontrolador PIC18f2431, [46].

4.3 Linguagens de Programação, Compilador e Programador

4.3.1 Linguagens de Programação

Os microcontroladores necessitam de ser programados de modo a realizarem as funções pretendidas, executando sequencialmente uma lista de instruções guardadas na memória de código. Estas instruções são programadas numa linguagem que se chama *assembler*, sendo considerada de baixo nível, que está ao nível da manipulação de registos e endereços, tendo os microcontroladores PIC o seu próprio conjunto de instruções (*Instruction Set*), (75 instruções de 16bits para a família 18f).

O uso desta linguagem traz vantagens quando se pretende executar funções próximas do *bit*, sendo (quando bem programada) melhor que outras linguagens de mais alto nível, pois o código torna-se muito optimizado reduzindo espaço em memória e aumentando a rapidez de execução. No entanto quando a tarefa a executar é complexa o risco de falha na programação é muito maior não sendo a melhor solução para programadores menos experientes.

A linguagem escolhida para executar este projecto foi a **Linguagem C**, que sendo uma linguagem de alto nível permite a manipulação próxima do bit em semelhança à linguagem *assembler*. Operações de multiplicação e divisão são um exemplo das vantagens da linguagem C, nesta linguagem o operador de multiplicação e divisão é '×' e '/' respectivamente, na linguagem *assembler* dependendo do número de bits a multiplicar e dividir, seriam necessárias dezenas de instruções para o mesmo fim.

Estes e outros factos fazem da linguagem C uma linguagem mais abrangente e atractiva, sendo a linguagem de programação escolhida para este projecto.

4.3.2 Compilador

Os compiladores são programas que convertem o código escrito em linguagem C ou *assembler* num ficheiro hexadecimal, sendo o conteúdo deste ficheiro enviado para o microcontrolador através de um programador.

O compilador utilizado foi o MPLAB C18 V3.15 instalado sobre uma plataforma de programação dos microcontroladores da Microchip (MPLAB IDE V8.00).

Este compilador (MPLAB C18 V3.15) permite compilar código C com enxertos de código em *assembler*, neste caso as funções de acesso há interrupção foram programadas em código *assembler* proporcionando um acesso mais rápido às interrupções.

Após a compilação, é necessário enviar para o microcontrolador o ficheiro hexadecimal. A plataforma MPLAB IDE V8.00 possui interface com um dispositivo externo de programação para o mesmo fim.

4.3.3 Programador

Após obter o ficheiro hexadecimal é necessário programar o microcontrolador. Este passo é feito com recurso a um programador, que lê o ficheiro hexadecimal e envia a informação para a memória de código do microcontrolador.

O Programador utilizado é o PICKit2 fabricado pela Microchip, capaz de programar a maioria dos microcontroladores da Microchip, através de um barramento ICSP (*In-Circuit Serial Programming*).

Na Fig. 62, está representado o esquema de ligações entre programador e o microcontrolador. Para mais informações sobre como efectuar a programação ou actualização do software no microcontrolador, aconselha-se a consulta do manual do controlador de velocidade desenvolvido, Anexo I.

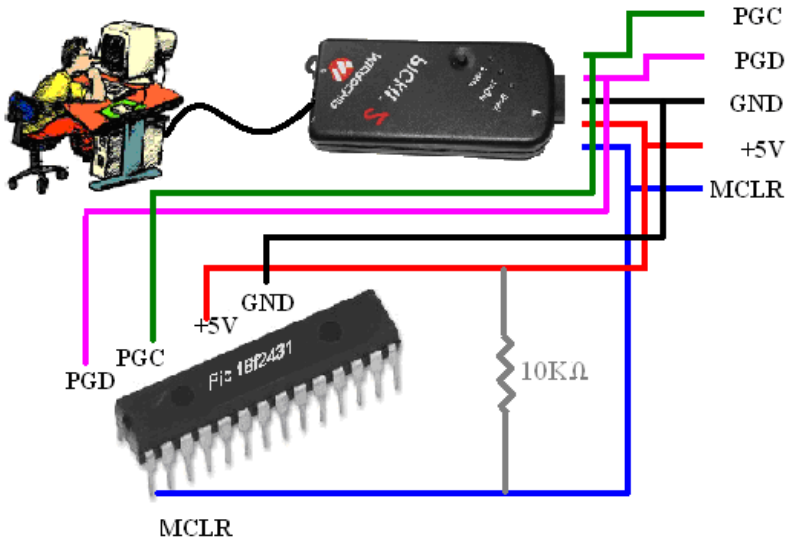


Fig. 61 – Esquema de ligações entre programador e microcontrolador

A Fig. 62, representa toda a sequência necessária desde o código C até à programação do microcontrolador.

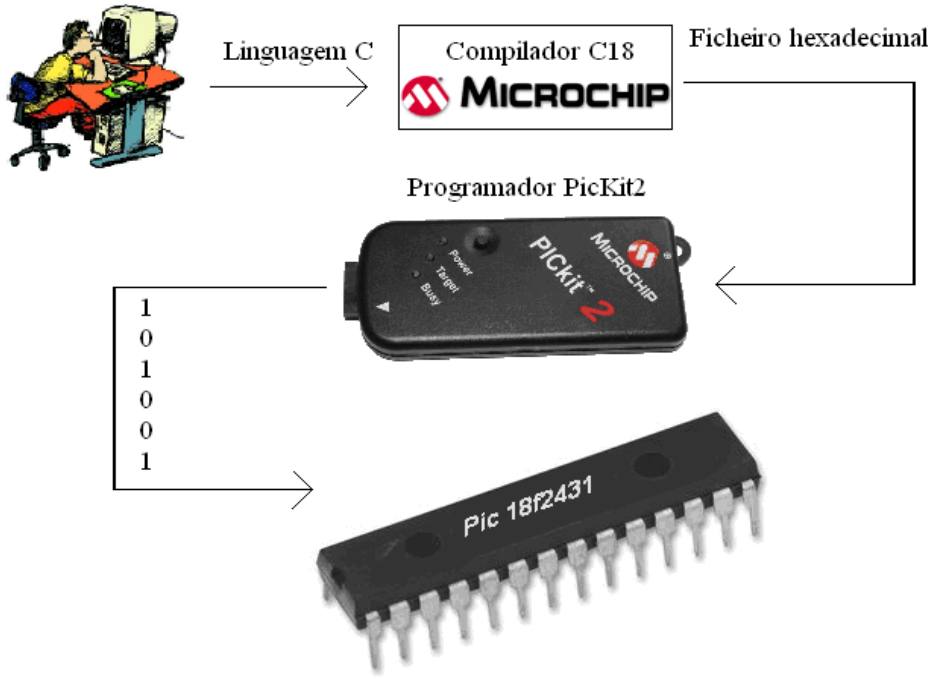


Fig. 62 – Sequência desde a programação do código em linguagem C até a programação do Microcontrolador

4.4 Ficheiro main.c

Para uma melhor organização do código C usaram-se 13 ficheiros, 7 ficheiros com extensão '.c' e 6 com extensão '.h'. Nos ficheiros com extensão '.h' declararam-se as variáveis e funções programadas nos ficheiros '.c'.

A Fig. 63 representa o diagrama dos ficheiros e a forma como estão associados.

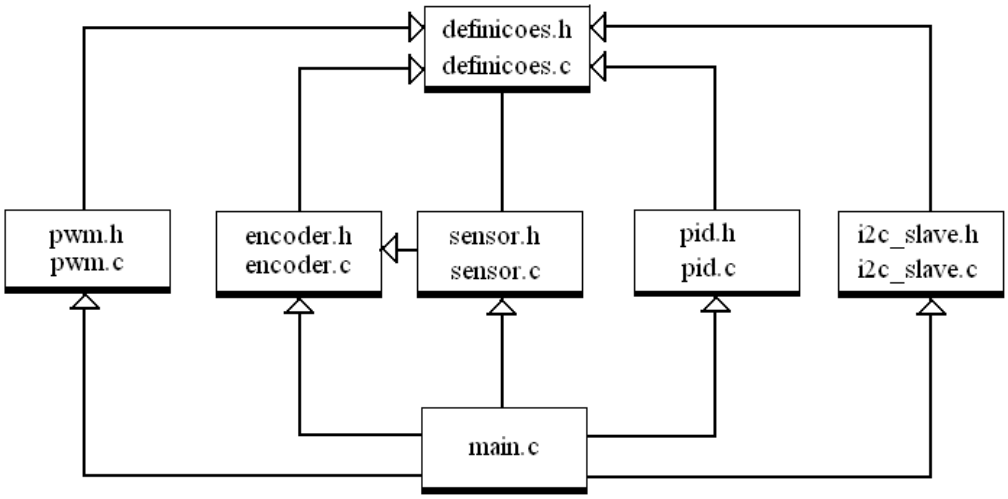


Fig. 63 – Diagrama de ficheiros e ligações entre os mesmos

O ficheiro **definições(.c, .h)** está no topo do diagrama anterior, não por executar as funções mais importantes mas por conter dados comuns a todos os outros ficheiros.

Este ficheiro contém a declaração e inicialização de todas as variáveis e definições globais, assim como, as funções de leitura/escrita na EEPROM e cálculo de parâmetros iniciais.

Devido à sua simplicidade este ficheiro não será descrito em pormenor, as suas funções são chamadas noutros ficheiros sendo ai descritas. O código deste ficheiro encontra-se no anexo II devidamente comentado.

Os ficheiros **pwm(.c, .h), encoder(.c, .h), sensor(.c, .h), pid(.c, .h) e i2c_slave(.c, .h)** são constituídos por variáveis internas e funções específicas, tendo acesso a todas as variáveis, definições e funções do ficheiro **definições(.c, .h).** Como executam funções mais complexas serão descritos posteriormente em pormenor.

O ficheiro **main.c** é o ficheiro que une todas as funcionalidades dos outros ficheiros. Quando o microcontrolador entra em funcionamento começa por executar as instruções deste ficheiro, que pode ser dividido em 3 partes, configurações, interrupções e função main.

Nas configurações são inseridas informações para o compilador, inserção de bibliotecas e configurações de funcionamento do microcontrolador.

As interrupções são funções, chamadas quando algum evento predefinido acontece, no código presente são utilizadas interrupções para a comunicação I2C, leitura dos ADCs, *timer_1* para execução do algoritmo PID, *timer_0* para execução da leitura dos sensores e *timer_5* quando é atingido *overflow* na leitura do *encoder*.

As funções de interrupção estão divididas em 2 tipos, interrupções de alta prioridade e interrupções de baixa prioridade, estas quando chamadas em simultâneo com as interrupções de alta prioridade serão executadas posteriormente. A única interrupção de alta prioridade usada é na comunicação I2C, quando uma comunicação começa com n bytes a serem transmitidos não deve ser interrompida por uma outra, correndo o risco de perda de dados.

Todas as interrupções enunciadas estão ligadas a outras funções principais sendo descritas no descrever dessas mesmas funções.

Por último, encontra-se a função main(), sendo por esta função que são iniciadas as primeiras instruções do programa. A Fig. 64 representa o algoritmo utilizado na

implementação da função main(), que tem como função principal a configuração dos vários módulos utilizados.

Inicialmente são lidos os valores guardados na EEPROM (valores de parâmetros configuráveis), sendo utilizados alguns desses valores para calcular parâmetros iniciais evitando que se faça o cálculo sempre que se pretende utilizar esses parâmetros. Os passos seguintes consistem na configuração de todos os módulos utilizados e na activação das interrupções. A função main() termina com um programa a fazer *reset* ao *watchdog* num ciclo infinito até que uma interrupção ocorra, voltando ao mesmo estado no final da execução.

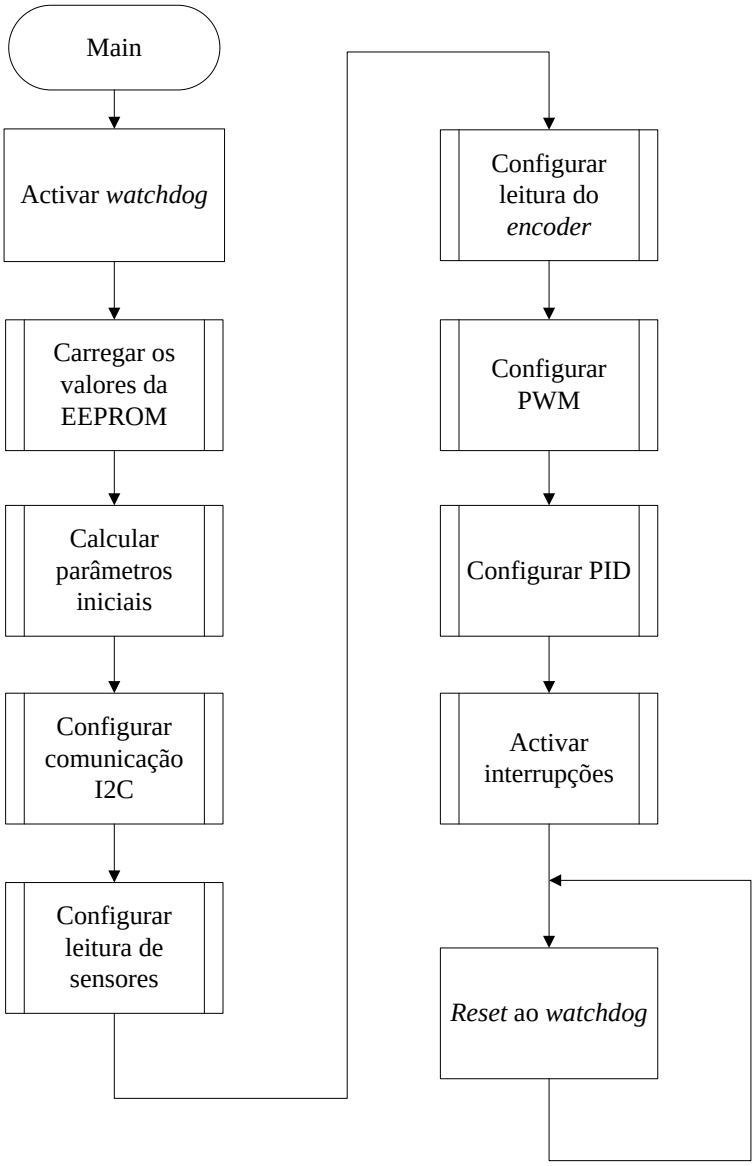


Fig. 64 – Algoritmo da função main()

O *watchdog* é uma espécie de temporizador, sendo configurado com um tempo pré-definido (neste projecto utilizou-se um tempo de 2 segundos), no início do programa é activo começando a contagem de tempo.

Em situações de bom funcionamento do programa, é feito o *reset* ao contador do watchdog no ciclo infinito, que se encontra no final da função main().

Quando ocorre uma irregularidade no programa (bug), não é executado o *reset* ao *watchdog*, o contador atinge o limite pré-definido, sendo feito o *reset* ao microcontrolador retomando novamente o funcionamento normal.

Nos sub-capítulos e capítulos seguintes são descritas as funções de configuração correspondentes aos módulos utilizados, encontrando-se todo o código C do ficheiro main.c no anexo II, devidamente comentado.

4.5 Comunicação I2C

4.5.1 Introdução ao I2C

O uso do microcontrolador traz muitas vantagens. A comunicação é uma característica dos microcontroladores, através da qual é possível transferir informação entre inúmeros dispositivos, sensores, memórias externas, visores, computadores pessoais e outros microcontroladores, sendo estes apenas alguns exemplos.

O microcontrolador utilizado, PIC18f2431, possui vários protocolos de comunicação, dos quais foi utilizado o protocolo I2C (*Inter-Integrated Circuit*) para efectuar a comunicação entre as placas controladoras de velocidade e o dispositivo que as controla.

O barramento I2C criado pela Philips é um barramento série síncrono com apenas 3 linhas de controlo: SDA (dados), SCL (relógio) e massa (para referência) [47].

"O princípio de funcionamento do barramento baseia-se no conceito de um dispositivo *master* (dispositivo que gere a comunicação, gera o sinal de relógio e transmite os dados) e outro *slave* (dispositivo que recebe os dados e confirma a recepção através do envio de um sinal de *acknowledge*)" [47].

4.5.2 Topologia do barramento I2C

A Fig. 65 corresponde à forma como são ligadas as placas controladoras de velocidade (*slave*) a um dispositivo controlador (PC, *master*). Com o uso do barramento I2C é possível ligar até 128 motores e respectivos controladores de velocidade, podendo-se configurar e controlar a velocidade de cada um de forma independente, com velocidades de transferência de dados até 400kbs. Quando for necessário acrescentar uma nova placa controladora, basta ligá-la ao barramento e atribuir-lhe um endereço, diferente dos endereços existentes, ficando pronta a funcionar.

O barramento I2C necessita de duas resistências de *pull-up* ligadas na linha SDA e SCL, estas resistências servem para criar um terceiro estado lógico no barramento.

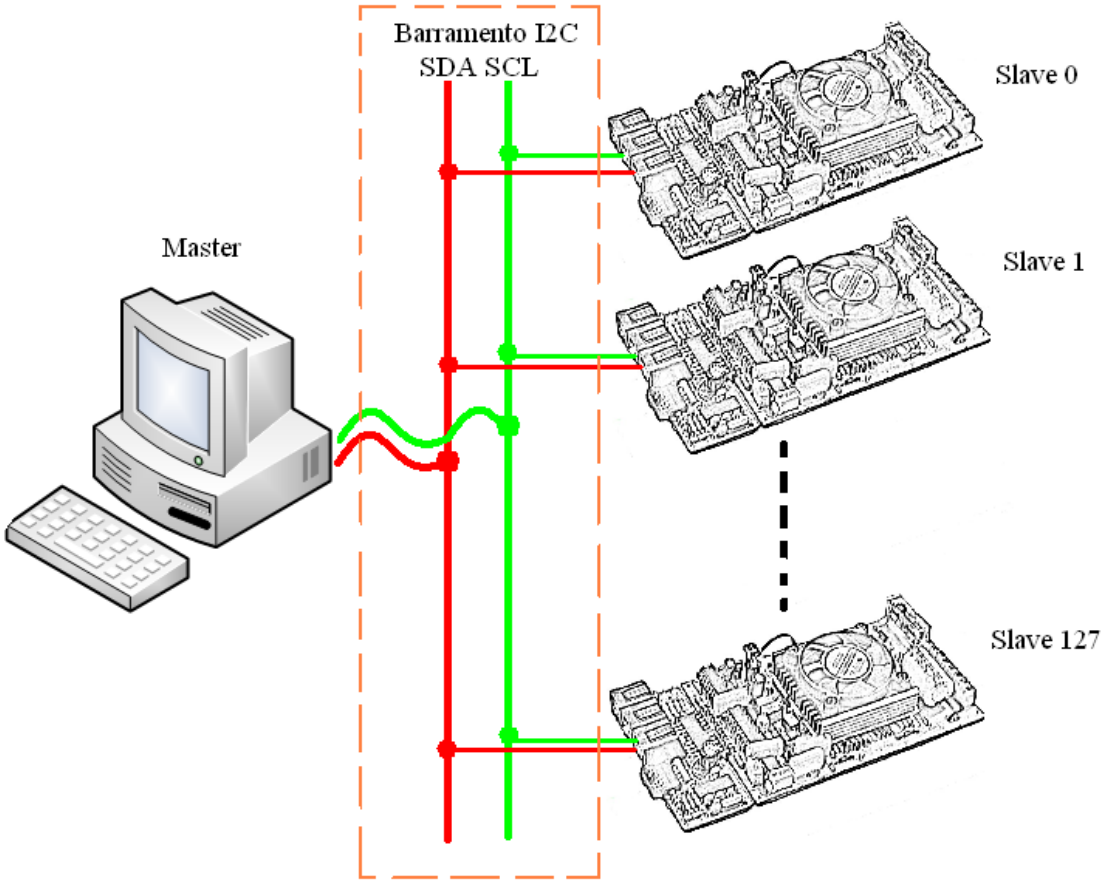


Fig. 65 – Configuração do barramento I2C utilizado

Sendo o *master* quem gere a ligação, a melhor forma de saber como o *slave* tem de responder ao *master* é conhecer o protocolo I2C de forma a proceder à programação do microcontrolador (*slave*).

4.5.3 Comunicação I2C

A Fig. 66 representa uma comunicação I2C, esta é sempre iniciada com envio por parte do *master* de um sinal *Start Bit* indicando o início da comunicação, seguido de oito *bits*, sete dos quais indicam o endereço do *slave* para o qual se pretende comunicar, o bit restante (bit menos significativo), indica ao *slave* se a operação é de leitura ou escrita. Se existir um dispositivo *slave* com o endereço indicado este retornará um sinal de *acknowledge* indicando uma comunicação bem sucedida, podendo o *master* enviar ou ler dados para/do *slave*. No final de cada dado transmitido com sucesso é novamente enviado o sinal de *acknowledge* por parte do dispositivo que está a receber os dados. No caso do envio de dados por parte do *master*, a comunicação termina quando este envia um sinal de *Stop bit*, caso contrário, se o *master* estiver a receber dados e quiser terminar a comunicação, este deixa de enviar o sinal de *acknowledge*, enviando de seguida um Stop bit para o *slave*.

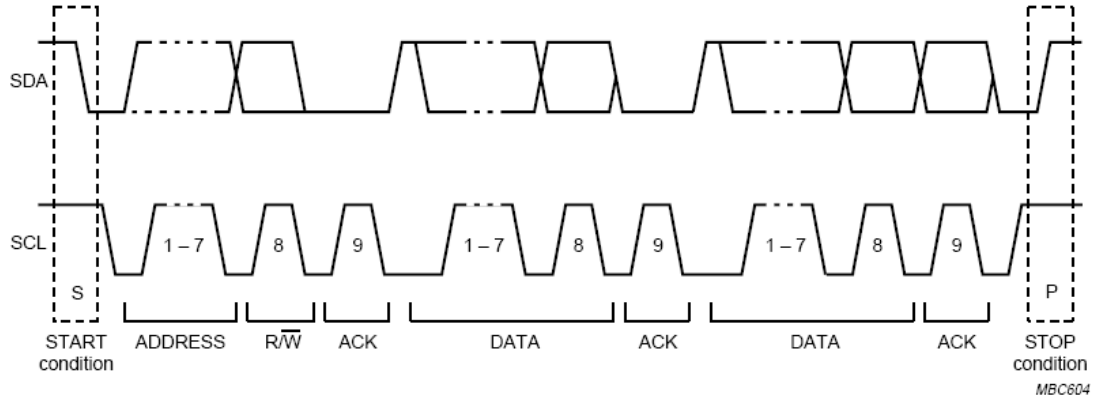


Fig. 66 – Exemplo de comunicação através do protocolo I2C [48]

O microcontrolador é utilizado como dispositivo *slave*, possuindo este: registos internos que permitem definir o seu endereço, *buffers* para leitura e escrita de dados com *flags* que indicam seu estado, *flags* que indicam se a operação é de escrita ou leitura, assim como *flags* que informam a ocorrência de *start bits* e *stop bits*, se o *byte* recebido é um endereço ou dado. O sinal de *acknowledge* é enviado automaticamente após a recepção.

Quando é feita uma comunicação com o microcontrolador este internamente verifica se o endereço é o dele, gerando uma interrupção caso o seja, após esta interrupção novas interrupções são geradas sempre que um byte é recebido até que o *master* envie um *stop bit* terminando a comunicação com o dispositivo. Dentro da

função de interrupção é necessário verificar em que estado se encontra a comunicação procedendo à leitura ou escrita de dados dependendo da mesma.

O documento AN734 "*Using the PICmicro® SSP for Slave I²C™ Communication*" [49] da Microchip ajudou na compreensão dos possíveis estados que poderiam ocorrer numa comunicação I2C.

Numa comunicação I2C por parte de um dispositivo *slave* existem 5 estados descritos abaixo:

Estado 1

O estado 1 ocorre quando o *master* quer escrever no *slave* sendo o último byte recebido o endereço do dispositivo *slave.*

As *flags* do registo SSPSTAT que indicam este estado, são:

- S = 1 (A *flag start bit* está activa t)
- R/W = 0 (*Master* pretende escrever no *slave*)
- D/A = 0 (Último byte foi o endereço do *slave*)
- BF = 1 (*Buffer* está cheio)

Neste estado é somente necessário limpar o buffer para poder receber novos dados.

Estado 2

O estado 2 ocorre quando o *master* quer escrever no *slave* sendo o último byte recebido um dado.

As *flags* do registo SSPSTAT que indicam este estado são:

- S = 1 (A *flag start bit* está activa)
- R/W = 0 (*Master* pretende escrever no *slave*)
- D/A = 1 (Último byte é um dado)
- BF = 1 (*Buffer* está cheio)

Neste estado é necessário ler o dado do buffer preparando-o para a recepção de novos dados, o dado lido poderá ser guardado ou processado.

Estado 3

O estado 3 ocorre quando o *master* quer ler um dado do *slave* sendo último byte recebido o endereço do dispositivo *slave.*

As *flags* do registo SSPSTAT que indicam este estado são:

- S = 1 (A *flag start bit* está activa)
- R/W = 1 (*Master* pretende ler do *slave*)
- D/A = 0 (Último byte foi o endereço do *slave*)
- BF = 0 (*Buffer* está vazio)

Neste estado é escrito o dado no buffer e enviado para o *master*.

Estado 4

O estado 4 ocorre quando o *master* quer ler do *slave* sendo o último byte lido um dado.

As *flags* do registo SSPSTAT que indicam este estado são:

- S = 1 (A *flag start bit* está activa)
- R/W = 1 (*Master* pretende ler do *slave*)
- D/A = 1 (Último byte é um dado)
- BF = 0 (*Buffer* está vazio)

Neste estado é escrito o dado no buffer e enviando para o master.

Estado 5

O quinto e último estado ocorre quando o *master* não pretende receber mais dados do slave.

As *flags* do registo SSPSTAT que indicam este estado são:

- S = 1 (A *flag start bit* está activa)
- R/W = 0 (*Master* pretende escrever no *slave*)
- D/A = 1 (Último byte é um dado)
- BF = 0 (*Buffer* está vazio)

Este é o último estado possível na comunicação I2C, caso ocorra um outro estado diferente, será activado um sinal visual indicando que ocorreu um erro na comunicação I2C.

4.5.4 Tramas de comunicação utilizadas

As tramas seguintes, Fig. 67 e Fig. 68 foram desenvolvidas sobre o protocolo I2C, estas não efectuam somente o envio e recepção de dados, permitem especificar quantos dados serão escritos e onde são escritos, assim como a trama de leitura permite escolher qual a primeira posição a ser lida.

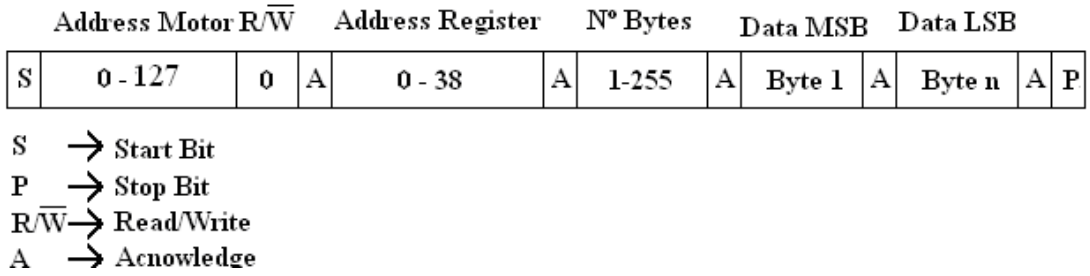


Fig. 67 – Trama utilizada na escrita de N *bytes* (0<n<256) a partir de um endereço especificado

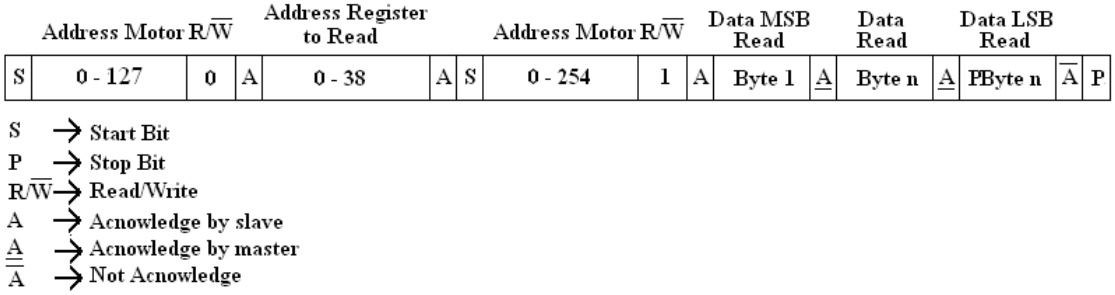


Fig. 68 – Trama utilizada na leitura de N bytes a partir de um endereço especificado

4.5.5 Software

O algoritmo seguinte, Fig. 69, promove todos os estados anteriormente descritos, assim como, as funções de leitura e escrita no microcontrolador. No código e manual da placa controladora de velocidade que se encontram no anexo I e anexo II, é possível analisar com mais detalhe o processo de escrita e leitura por I2C, assim como o código da função i2c_slave_interrupt() devidamente comentado.

Todos os dados lidos e escritos passam pelo *array* dados[], este contém 39 posições de 8 bits nas quais estão guardadas as configurações do controlador de velocidade (da posição 0 à 30), das posições 31 à 38 encontram-se as posições para

controlo de velocidade desejada e telemetria sobre o estado do controlador de velocidade (correntes, tensões, temperaturas e velocidade actual).

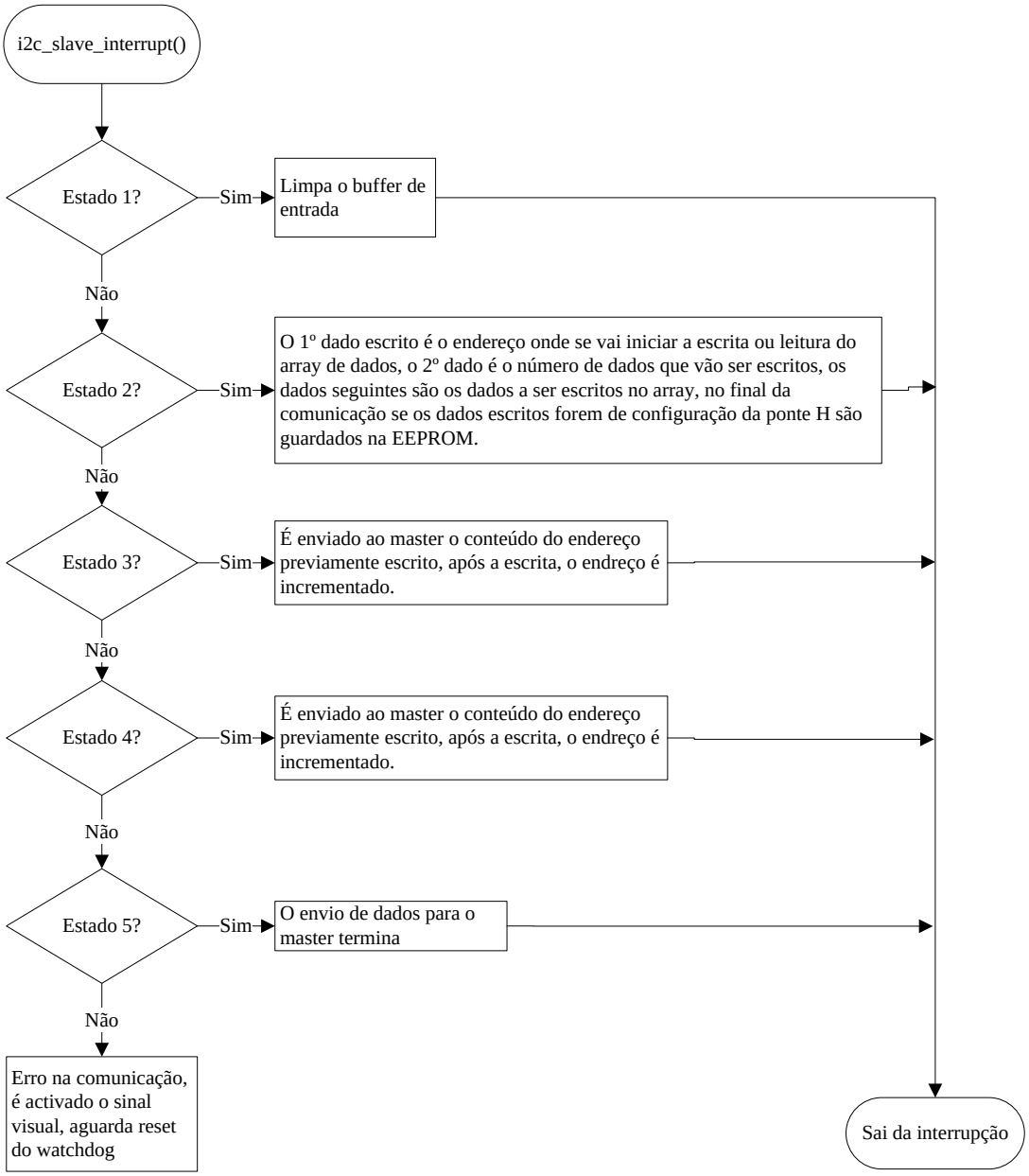


Fig. 69 – Algoritmo da função *slave* do protocolo I2C

As configurações do controlador de velocidade são copiadas para o *array* dados[] no início do programa, a partir da EEPROM (memória não volátil), sendo alteradas as informações na EEPROM quando existir alguma modificação na configuração do controlador.

Abaixo está descrito de forma resumida o que se encontra em cada uma das 39 posições do *array* dados[], estando descrito com mais pormenor no manual do controlador de velocidade que se encontra no anexo I.

0 -	MTR_ADRS	Endereço da placa controladora de velocidade
1 -	MTR_FLAG	Sinal indicador da presença de motor
2 -	KP	Ganho proporcional do controlo PID
3 -	KI	Ganho integrativo do controlo PID
4 -	KD	Ganho derivativo do controlo PID
5 -	DIV_INTV	Intervalo de tempo entre cálculo derivativo
6 -	INTEG_MAX	Limite máximo para o erro integrativo
7 -	AMSTR_INTV_H	Intervalo de tempo entre cálculo do algoritmo PID
8 -	AMSTR_INTV_L	Intervalo de tempo entre cálculo do algoritmo PID
9 -	AMP_LIM	Corrente máxima no motor
10 -	VOLT_LIM	Tensão mínima para a qual a ponte H deixa de funcionar
11 -	TEMP_P_LIM	Temperatura máxima na ponte
12 -	TEMP_M_LIM	Temperatura máxima no motor
13 -	PPR_ENC	Número de pulsos por rotação do *encoder*
14 -	LIM1_INF_M	Limites inferiores e superiores para mudança de escala na
15 -	LIM1_INF_L	leitura de velocidades
16 -	LIM2_INF_M	
17 -	LIM2_INF_L	
18 -	LIM3_INF_M	
19 -	LIM3_INF_L	
20 -	LIM1_SUP_M	
21 -	LIM1_SUP_L	
22 -	LIM2_SUP_M	
23 -	LIM2_SUP_L	
24 -	LIM3_SUP_M	
25 -	LIM3_SUP_L	
25 -	PWM_FREQ_M	Frequência de oscilação de PWM
27 -	PWM_FREQ_L	
28 -	CRISTAL_H	Frequência do cristal do microcontrolador
29 -	CRISTAL_L	
30 -	BREAK	Sinal que indica se a travagem é forçada ou livre
31 -	VEL_DES_M	Velocidade desejada para o motor
32 -	VEL_DES_L	

33 - VEL_ACT_M Velocidade actual no motor

34 - VEL_ACT_L

35 - AMP_ACT Corrente actual no motor

36 - VOLT_ACT Tensão actual na ponte H

37 - TEMP_P_ACT Temperatura actual da ponte H

38 - TEMP_M_ACT Temperatura actual do motor

Na tabela abaixo estão presentes todos os valores configurados por defeito no *array* dados[].

Endereço	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Valor	0x00	0x01	0x08	0x04	0x00	0x05	0x0A	0x8C	0x2D	0x96
Endereço	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
Valor	0x6A	0x32	0x32	0x32	0x00	0x75	0x02	0xF6	0x05	0xFF
Endereço	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**
Valor	0x00	0xA7	0x03	0x5A	0x06	0xC7	0x4C	0x2C	0x27	0x10
Endereço	**30**	**31**	**31**	**33**	**34**	**35**	**36**	**37**	**38**	
Valor	00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	

Tabela IV – Valores por defeito do array dados[]

4.5.6 Conclusões sobre a comunicação I2C

O barramento I2C trouxe inúmeras vantagens, sendo o elo de ligação de todas as placas controladoras de velocidade. Os pontos abaixo descrevem o conhecimento adquirido relevante sobre o barramento I2C.

- O interface dos dispositivos com o barramento I2C é muito acessível, bastando ligar os pinos SCL e SDA às linhas correspondentes no barramento.
- O protocolo apesar de bastante simples, adequa-se muito facilmente a comunicações complexas, sendo muito eficaz e fiável.
- A interrupção gerada na comunicação I2C é configurada como sendo a única interrupção de alta prioridade, caso ocorra outra interrupção com a mesma prioridade seriam perdidos dados, dando origem a um erro na comunicação.

- As tramas de leitura e escrita desenvolvidas sobre o protocolo I2C são muito flexíveis, permitem executar leituras e escritas em posições exactas definidas pelo utilizador.
- Os dados de configuração guardados na EEPROM evitam a necessidade de configurar o controlador de velocidade todas as vezes que este seja reiniciado.
- O uso de um *array* contendo todas as configurações, controlo e telemetria simplificou a programação e a comunicação I2C, sendo fácil adicionar ou remover parâmetros.
- Caso ocorra um erro na comunicação um LED (*Light Emitting Diode*) vermelho é ligado mantendo-se até que se faça o *reset* ao microcontralador eliminando o erro de comunicação.

4.6 Leitura de sensores e protecção

No capítulo anterior foram descritas três situações que poderiam afectar de forma significativa o funcionamento do sistema, sendo estas:

- Corrente circula no motor, que quando acima da corrente nominal pode danificar o motor.
- Tensão de alimentação, sendo abaixo de um certo valor provoca o mau funcionamento do controlador da ponte H, podendo danificar os MOSFETs.
- Temperatura da ponte H, se esta for superior à nominal, a temperatura de junção dos MOSFETs também o será levando-os à destruição.

Existindo uma outra não falada até ao momento que é:

- Temperatura no motor DC, se esta for superior ao limite poderá resultar em dá-nos nos enrolamentos levando à destruição do motor.

Estas são quatro situações importantes que nunca deverão acontecer, sendo necessário algo, que caso aconteçam actue, de forma a proteger todos os elementos referidos.

Nestas quatro situações estão presentes três grandezas físicas, corrente, tensão e temperatura. Sabendo quais as grandezas, o passo para que estas nunca ultrapassem os valores pré-estabelecidos será medi-las constantemente de forma a actuar caso necessário.

O dispositivo que permite medir estas grandezas tem o nome de sensor. Um sensor composto por um transdutor (dispositivo que converte um tipo de energia noutra.) e uma parte que converte a energia resultante num sinal eléctrico.

Sendo necessários sensores para medir têz grandezas, temperatura, tensão e corrente, foi feita uma pesquisa ao mercado, optando-se pelos seguintes sensores.

4.6.1 Sensor de corrente ACS712ELCTR-20A-T (*Hardware*)

Para medir a corrente foi utilizado um sensor de efeito de hall, ACS712ELCTR-20A-T, fabricado pela ALLEGRO, que possui as seguintes características [50]:

- Isolamento 2100 Volts
- Resistência interna 1.2mΩ
- Tempo de resposta de 5µs

- Faz medições até 50kHz
- Tensão de alimentação 5V
- Mede correntes ±20A
- Sensor de efeito de Hall
- Sensibilidade 100mV/A

O uso deste circuito integrado traz vantagens, toda a electrónica de instrumentação está incluída, o facto de ser um sensor de efeito de hall garante um maior isolamento e quedas de tensão menores em comparação com o uso de resistências para medição de correntes.

A Fig. 70 (direita) representa uma implementação típica, este circuito integrado está preparado para medir correntes CC e AC colocando um valor na saída Vout proporcional à corrente que nele circula. O valor de saída varia de 2.5V a 5V quando a corrente circula de IP+ para IP- e de 2.5V a 0V quando circula no sentido oposto.

O condensador Cf é utilizado para filtrar o sinal de saída, a corrente consumida não é contínua, mas pulsada (PWM) sendo necessário um filtro para que a corrente do sinal de saída seja um valor constante proporcional ao valor médio da corrente.

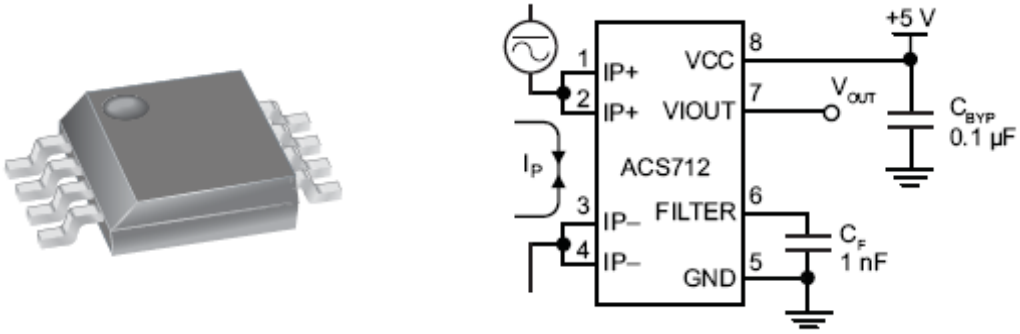


Fig. 70 – (Esquerda) encapsulamento SOIC utilizado, (direita) aplicação típica do sensor de corrente [50]

Como neste projecto a corrente circula somente num sentido, de IP- para IP+ a variação da tensão de saída é de 2.5V a 0V variando a corrente de 0A a 20A respectivamente. Para aumentar a sensibilidade no microcontrolador de modo a só se utilizarem 8bits na leitura do ADC, foi utilizado um circuito auxiliar que multiplica por 2 o valor da tensão de saída do sensor de corrente, de modo que esta varie de 5V a 0V variando a corrente de 0A a 20A.

A Fig. 71 representa todo o circuito utilizado para a medição da corrente, incluindo o circuito auxiliar de multiplicação por 2 e o estabilizador de tensão utilizado para eliminar o *ripple* criado pelo consumo pulsado da corrente.

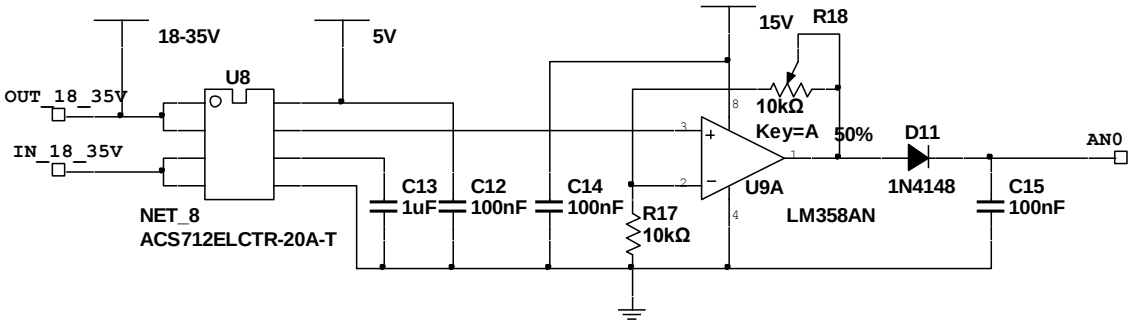


Fig. 71 – Circuito utilizado para medição da corrente

4.6.2 Sensor de tensão (*Hardware*)

O sensor de tensão serve para indicar o estado da fonte de alimentação (baterias), para tal utilizou-se um divisor de tensão de forma a ter na entrada do ADC (*analog-to-digital converter*) do microcontrolador uma tensão que varie de 0V a 5V correspondente à tensão das baterias.

O circuito utilizado, Fig. 72, é composto por um divisor de tensão e um filtro que permite eliminar possíveis picos de tensão. O divisor de tensão possui uma resistência variável permitindo o ajuste da tensão.

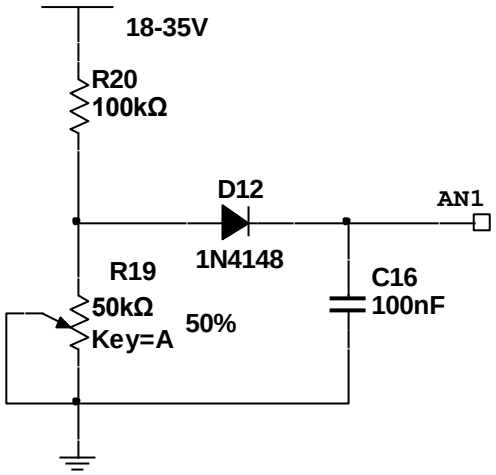


Fig. 72 – Circuito utilizado para medir a tensão nas baterias

4.6.3 Sensor de temperatura TMP100 (*Hardware*)

O sensor de temperatura TMP100 é um circuito integrado da Texas Instruments, muito compacto e bastante preciso, medindo temperaturas de -55 a 125°C.

Possui uma resolução máxima de 12 bits, contém toda a electrónica de instrumentação necessária, sendo de fácil implementação, Fig. 73. A saída é digital, comunica através de um barramento I2C que permite a leitura de temperaturas e configurações do próprio sensor.

Neste projecto são utilizados dois sensores, um para medir a temperatura na ponte H e outro no motor. Este facto faz com que os 2 sensores tenham de ter endereços diferentes, sendo configurados através dos pinos ADD0 e ADD1.

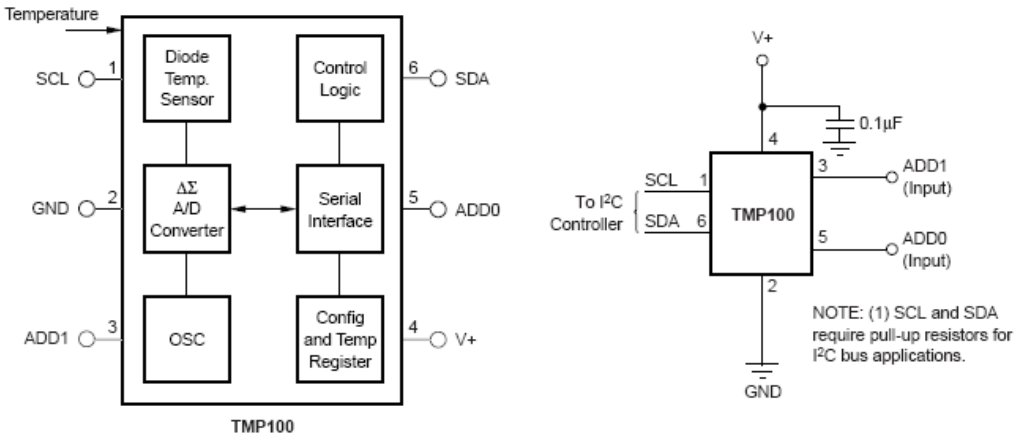


Fig. 73 – (Esquerda) pinos e diagrama interno do CI TMP100, (direita) aplicação típica [51]

A Fig. 74, representa os endereços possíveis para o sensor TMP100 em função dos pinos ADD0 e ADD1, neste projecto são utilizados os endereços 1001000 para o sensor de temperatura da ponte H e 1001010 para o sensor de temperatura do motor.

ADD1	ADD0	SLAVE ADDRESS
0	0	1001000
0	Float	1001001
0	1	1001010
1	0	1001100
1	Float	1001101
1	1	1001110
Float	0	1001011
Float	1	1001111

Fig. 74 – Endereços do TMP100 em função dos pinos ADD0 e ADD1 [51]

Na Fig. 75, estão representados os circuitos utilizados no sensor de temperatura do motor (esquerda) e no sensor de temperatura da ponte H (direita). Nestes circuitos são utilizadas resistências de *pull-up,* necessárias no barramento I2C e condensadores de desacoplamento para filtrar os ruídos na tensão de alimentação.

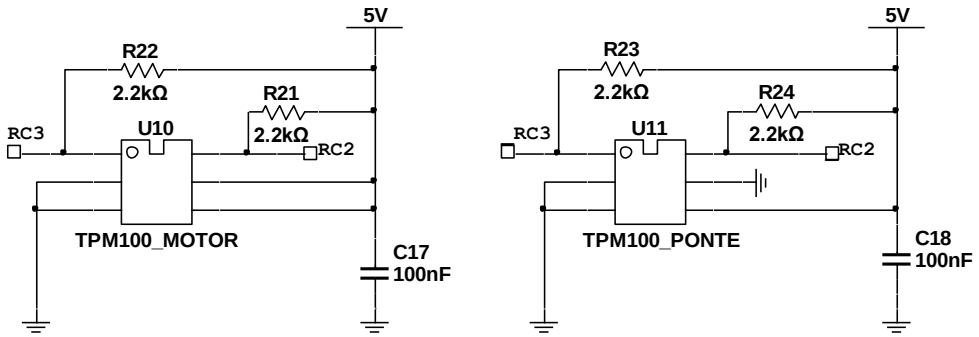


Fig. 75 – (Esquerda), circuito utilizado no sensor temperatura do motor, (direita), circuito utilizado no sensor de temperatura da ponte H

4.6.4 Software

Visto para o *hardware* dos sensores utilizados ser necessário ler o valor destes e actuar (proteger) caso necessário, utilizando-se o microcontrolador para esse efeito. Para tal foi programado no microcontrolador uma rotina que faz uma leitura e actuação a cada 500ms, sendo o mesmo sensor, lido de 2 em 2 segundos de forma a não causar paragens significativas no programa devido à execução da leitura dos sensores.

Para se entender o funcionamento das funções de leitura e actuação implementadas, é descrito de forma sequencial todo o processo.

No algoritmo da função main(), Fig. 64, os primeiros passos consistem nas configurações de funcionamento de todos os módulos, sendo a leitura de sensores um desses módulos. O algoritmo da Fig. 76, mostra exactamente esse passo, a configuração inicial do módulo de leitura dos sensores.

Nos primeiros dois passos deste algoritmo são configurados os sensores TMP100 colocando-os em aquisição contínua, com uma resolução de 9bits (0.5°C), diminuindo o tempo de aquisição para 40ms. As configurações e leituras são feitas através de um barramento I2C independente do descrito no subcapítulo anterior. Neste caso o microcontrolador é o *master*, para tal implementou-se a comunicação I2C por software utilizando-se funções fornecidas no compilador C18 da Microchip, tendo sido estas modificadas e compiladas de forma a utilizar outros pinos de *data* e *clock,* assim como intervalos de tempo definidos para outras frequências de oscilação. A referência, [51] corresponde ao manual do sensor TMP100 podendo-se obter aqui mais informações sobre o mesmo.

É no terceiro passo que são inicializadas as variáveis auxiliares necessárias às funções de leitura e actuação dos sensores.

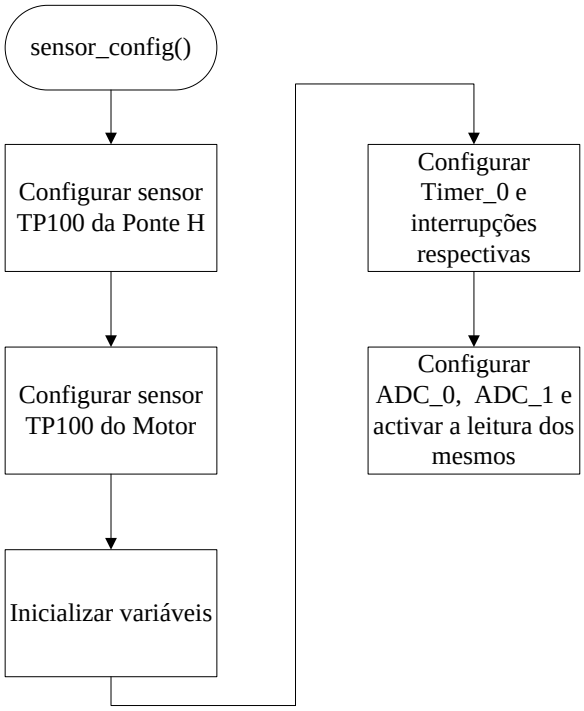


Fig. 76 – Algoritmo da função responsável pela configuração da leitura dos sensores.

No penúltimo passo são configurados; o *timer_0* e a interrupção respectiva de forma a gerar uma interrupção a cada 100ms, sendo esta a interrupção que chama as funções de leitura e actuação dos sensores. Para gerar este intervalo de tempo utilizou-se o timer_0 no modo de 16bits com um *prescale* de 1:256, sendo os valores de TMR0H e TMR0L, 0xF0 e 0xBD respectivamente. Estes valores foram calculados com a ajuda do manual do microcontrolador [46], podendo ser alterados nas definições, (ficheiro definições.h).

São configurados no quarto e último passo os ADCs utilizados para efectuar a leitura da corrente e tensão. Sendo um ADC um conversor que converte um sinal analógico num certo instante de tempo para um valor numérico digital correspondente.

Neste caso foram utilizados dois ADC que operam em modo contínuo lendo os dois canais simultaneamente, gerando uma interrupção de baixa prioridade na qual são lidos os valores destes para duas variáveis correspondentes à corrente e tensão.

A forma como os ADC estão configurados permite que sejam feitas leituras aproximadamente a cada 250µs, sendo este, o maior intervalo de tempo possível com o cristal utilizado. As referências dos ADCs são AVDD e AVSS sendo 5V e 0V respectivamente.

No final da função main() são activadas as interrupções entrando em funcionamento a interrupção do timer_0.

O algoritmo seguinte, Fig. 77 é referente à interrupção do timer_0 , que tem como tarefa chamar a função de leitura e a função que verifica os valores lidos para cada sensor a cada 5 incrementos do contador, ou seja, lê e verifica um sensor a cada 500ms.

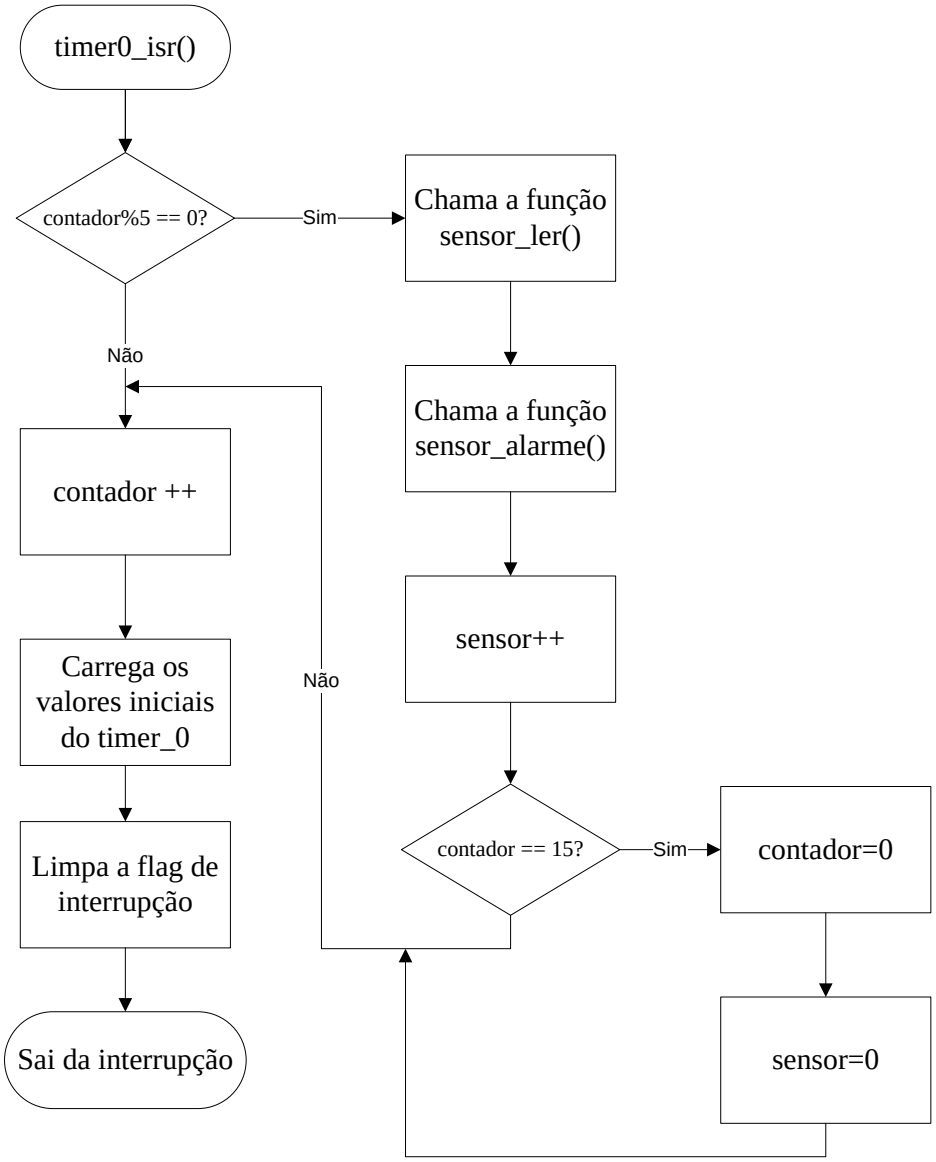


Fig. 77 – Algoritmo da interrupção do timer_0

Visto o funcionamento da interrupção do timer_0 o próximo passo é saber como funciona a função ler e posteriormente a função que verifica os dados lidos, correspondendo à Fig. 78, ao algoritmo da função ler.

Na interrupção é escolhido o sensor que vai ser lido sendo chamada a função de leitura, nesta, é verificado qual o sensor escolhido, lendo-o posteriormente. Todas as leituras actuais são guardadas no *array* dados[], nas posições correspondentes aos dados de telemetria. No anexo I encontra-se o manual da ponte H que contém informações sobre cada posição do *array* assim como os cuidados a ter ao modificá-las.

Neste algoritmo existe um "sensor" implementado por software, este activa um sinal (flag_motor), caso o motor se encontre parado quando deveria estar a rodar, ou seja, se a velocidade desejada for acima de 100RPM (*offset* mínimo) e o motor não rodar este encontra-se com algum problema sendo activado um sinal visual.

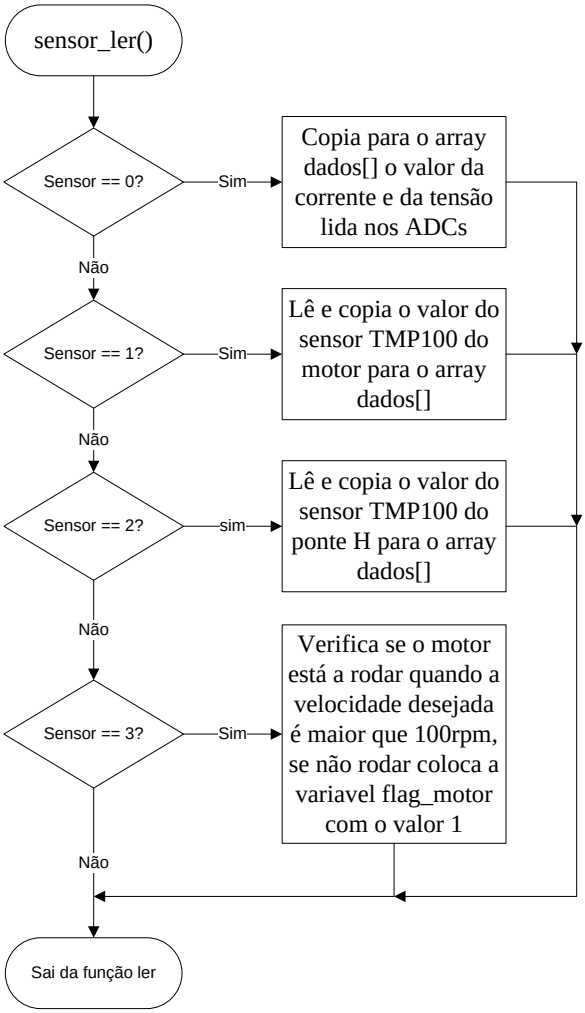


Fig. 78 – Algoritmo da função ler

Descrita a função de leitura falta a função que vai proteger os componentes, sendo esse o próximo algoritmo

Fig. 79.

A função sensor_alarme(), é executada após a função sensor_ler(), verificando o valor do sensor que leu. Em todas as situações quando ocorre uma anomalia é desligada a ponte H sendo indicado através de um LED (*Light Emitting Diode*) vermelho o erro. Este LED, dependendo do número de vezes que pisca e da velocidade com que pisca, indica um erro diferente (Tabela V).

Dependendo do erro ocorrido a ponte H pode ficar inactiva durante 10 segundos (corrente e tensão), por tempo indefinido no caso das temperaturas, só é activa após a temperatura descer abaixo da temperatura limite e permanentemente desligada caso occorra algum problema com o motor ou comunicação I2C.

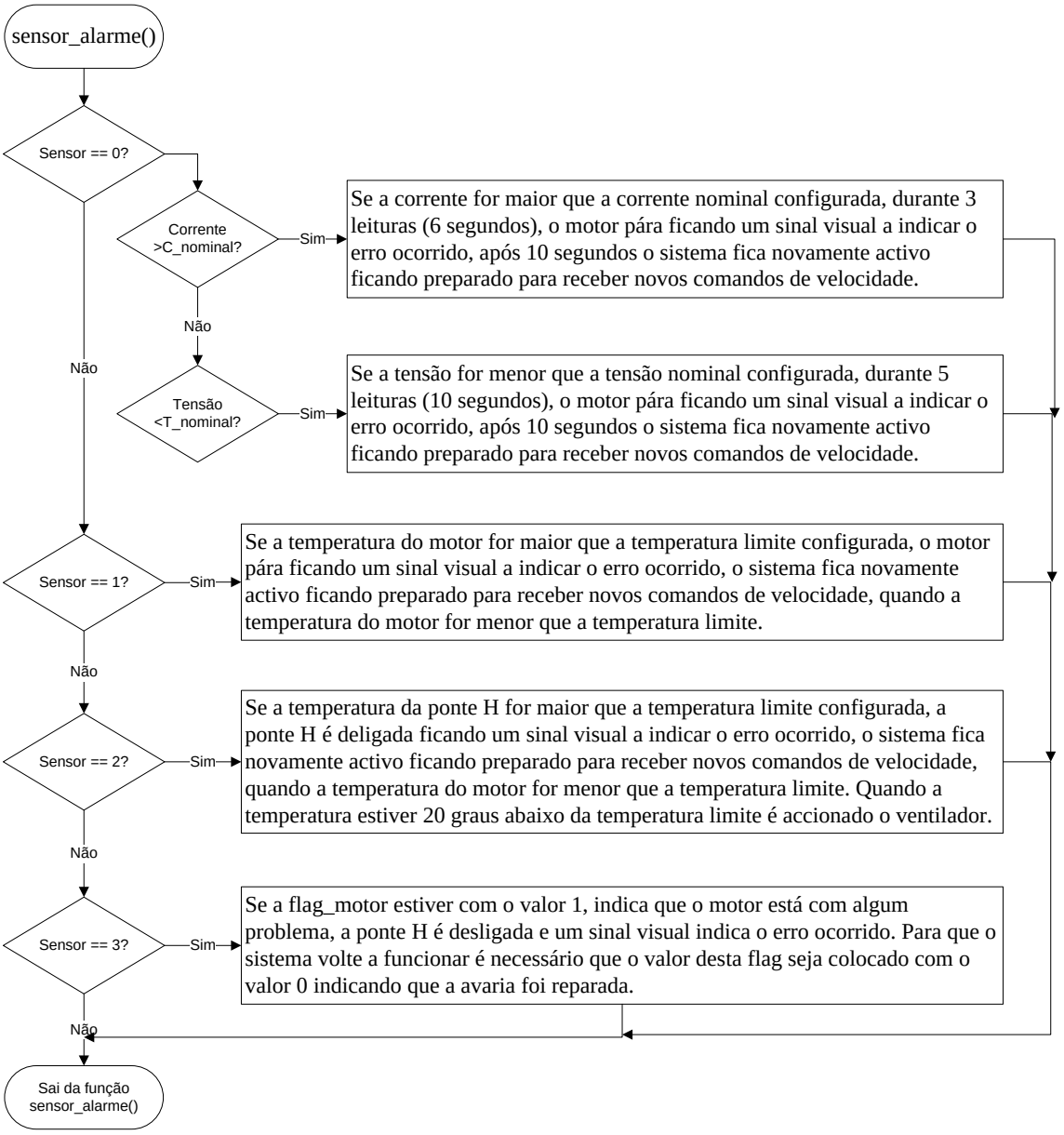


Fig. 79 – Algoritmo da função sensor_alarme()

A tabela seguinte indica o erro ocorrido em função do número de vezes e intervalo de tempo que o LED pisca.

Estado LED Vermelho	Erro
	Tensão inferior à tensão limite
	Corrente superior à corrente limite
	Temperatura do motor excedeu o limite
	Temperatura da Ponte H excedeu o limite
	Problema com o motor
	Problema de comunicação I2C

Tabela V – Sinais de Erro Visuais

4.6.5 Conclusões Sensores

Após a conclusão do hardware e software referentes à leitura e verificação dos sensores chegou-se às seguintes conclusões:

- Os sensores TMP100 são extremamente precisos e funcionais tendo como desvantagem a baixa protecção contra electricidade electrostática.
- Os sensores de corrente funcionam correctamente, no entanto é necessário ajustar o valor 0 ocasionalmente, sendo proposto como trabalho futuro o uso de um sensor com saída digital.
- No sensor de tensão ocorre o mesmo que no sensor de corrente com pequenas alterações do valor da tensão de alimentação é necessário ajustar o valor de referência.
- Os dois problemas descritos anteriormente não influenciam o funcionamento do sistema sendo apenas perfeccionismo.
- Como existem várias interrupções com a mesma prioridade, a leitura e actuação de um sensor de cada vez é muito eficaz tornando-se imperceptível, não causando atrasos noutros intervalos de tempo.

- A compilação das bibliotecas I2C da Microchip foi bastante dificultada e demorosa pela falta de informação sobre os comandos de compilação.
- O LED indicador mostrou-se bastante útil tornando possível saber o porquê do erro ocorrido de forma muito fácil.
- Como trabalho futuro fica a inserção de uma resistência menor que 2200Ω na entrada de cada ADC de forma dissipar a energia residual nos condensadores que estão ligados aos mesmos ADCs.
- Todo o algoritmo final mostrou-se funcional.

4.7 Gerador de Sinais PWM

4.7.1 Introdução ao PWM

No capítulo 3, ponto 3.3.2 descreve-se o funcionamento do controlo PWM indicando as suas vantagens em relação ao controlo linear.

Relembrando o princípio de funcionamento, o controlo PWM (*Pulse With Modulation*) consiste em fornecer a uma carga, energia pulsada. Estes pulsos são gerados a uma frequência fixa, sendo o valor médio da energia entregue à carga correspondente ao *duty cycle* dos pulsos gerados. Este método permite maior precisão e eficiência energética em relação ao método linear.

Os sinais PWM quando aplicados na ponte H permitem controlar a energia entregue ao motor variando a velocidade, mudando a forma de como são aplicados torna-se possível alterar o sentido de rotação e as formas de travagem.

O microcontrolador escolhido PIC18f2431 possui 3 canais PWM de 14 bits. Os PWM0, PWM1, PWM2 e PWM3 formam os dois pares de canais de PWM utilizados e configurados para operar no modo complementar (PWM0 e PWM2 geram sinais inversos dos sinais PWM1 e PWM3).

4.7.2 Sinais PWM aplicados ao controlador da ponte H

A partir do princípio de funcionamento dos PWM no modo complementar e da Fig. 80, que representa os sinais aplicados no controlador em função dos movimentos desejados para o motor, descritos na análise do controlador HIP4181A, obtém-se a correspondência entre os sinais PWM do microcontrolador e as entradas do controlador HIP4180A, Fig. 81. Para colocar o PWM0 ou PWM2 com *duty cycle* de 100% (estado

1 nas figuras abaixo) é necessário colocar o PWM1 ou PWM3 com *duty cycle* de 0% (estado 0 nas figuras abaixo) desaparecendo o estado "irrelevante".

AHI	BHI	ALI	BLI	DIS	Função
PWM	X	$\overline{PWM}$	1	0	Direita
X	PWM	1	$\overline{PWM}$	0	Esquerda
X	X	1	1	0	Travado
1	1	0	0	0	Travado
X	X	X	X	1	Livre

X= Irrelevante 1 = Ligado 0 = Desligado

Fig. 80 – Tabela de verdade das entradas aplicadas em função do movimento do motor

PWM3	PWM1	PWM2	PWM0		
AHI	BHI	ALI	BLI	DIS	Função
PWM	0	$\overline{PWM}$	1	0	Direita
0	PWM	1	$\overline{PWM}$	0	Esquerda
0	0	1	1	0	Travado
1	1	0	0	0	Travado
0	0	0	0	1	Livre

1 = Ligado 0 = Desligado

Fig. 81 – Sinais PWM aplicados ao controlador HIP4081A

A escolha da frequência dos sinais PWM consiste numa relação entre a resolução e frequência do PWM, sendo um dos requisitos deste projecto que os sons gerados pela frequência de ressonância das bobinas do motor não sejam audíveis. Isto indica, que a frequência deve ser proxima dos 20kHz, no entanto, através da análise à tabela da Fig. 82 que relaciona as frequências de operação e a resolução do PWM, conclui-se que a resolução diminui com o aumento da frequência. A frequência de 19500Hz é a frequência escolhida, tendo o melhor relacionamento encontrado entre resolução e frequência não audível. Os sons ouvidos pelo ouvido humano têm uma

frequência entre 20Hz e 20kHz, estando a frequência de 19500Hz dentro da gama audível, no entanto, para a maioria dos seres humanos esta é imperceptível.

PWM Frequency = 1/TPWM				
Fosc	MIPS	PTPER Value	PWM Resolution	PWM Frequency
40 MHz	10	0FFFh	14 bits	2.4 kHz
40 MHz	10	07FFh	13 bits	4.9 kHz
40 MHz	10	03FFh	12 bits	9.8 kHz
40 MHz	10	01FFh	11 bits	19.5 kHz
40 MHz	10	FFh	10 bits	39.0 kHz
40 MHz	10	7Fh	9 bits	78.1 kHz
40 MHz	10	3Fh	8 bits	156.2 kHz
40 MHz	10	1Fh	7 bits	312.5 kHz
40 MHz	10	0Fh	6 bits	625 kHz
25 MHz	6.25	0FFFh	14 bits	1.5 kHz
25 MHz	6.25	03FFh	12 bits	6.1 kHz
25 MHz	6.25	FFh	10 bits	24.4 kHz
10 MHz	2.5	0FFFh	14 bits	610 Hz
10 MHz	2.5	03FFh	12 bits	2.4 kHz
10 MHz	2.5	FFh	10 bits	9.8 kHz
5 MHz	1.25	0FFFh	14 bits	305 Hz
5 MHz	1.25	03FFh	12 bits	1.2 kHz
5 MHz	1.25	FFh	10 bits	4.9 kHz
4 MHz	1	0FFFh	14 bits	244 Hz
4 MHz	1	03FFh	12 bits	976 Hz
4 MHz	1	FFh	10 bits	3.9 kHz

Fig. 82 – Exemplos de frequências e resoluções [46]

O *dead time* é um tempo morto que ocorre entre as comutações de um canal PWM no modo complementar, este tempo impede que um MOSFET fique activo enquanto o outro fica inactivo, evitando curto-circuitos na fonte de alimentação e aquecimento ou até mesmo destruição dos MOSFETs (Fig. 83).

O *dead time* utilizado é de 1.5µs, sendo este valor encontrado na fase de testes da placa controladora.

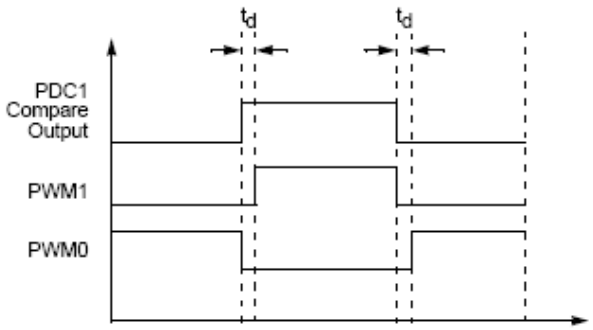


Fig. 83 – Exemplo de um *dead time* (td) aplicado a um canal PWM em módulo complementar [46]

4.7.3 Software

O software desenvolvido para o módulo PWM é composto por uma função de configuração do módulo PWM, chamada na função main() e pela função que actualiza o duty cycle e sentido de rotação do motor.

O algoritmo da função **pwm_config()** está abaixo representado, Fig. 84, juntamente com duas funções auxiliares. A função pwm_config() começa por calcular a base de tempo para a frequência de oscilação configurada, esta base de tempo consiste no número de ciclos de relógio necessários para obter o mesmo tempo que o período da frequência desejada (19500Hz).Após o passo anterior é calculado o valor máximo que o *duty cycle* pode obter, para este caso a resolução é de 11 bits podendo o *duty cycle* ter valores entre 0 e 2048. Os passos seguintes são configurações de funcionamento do PWM e inicialização.

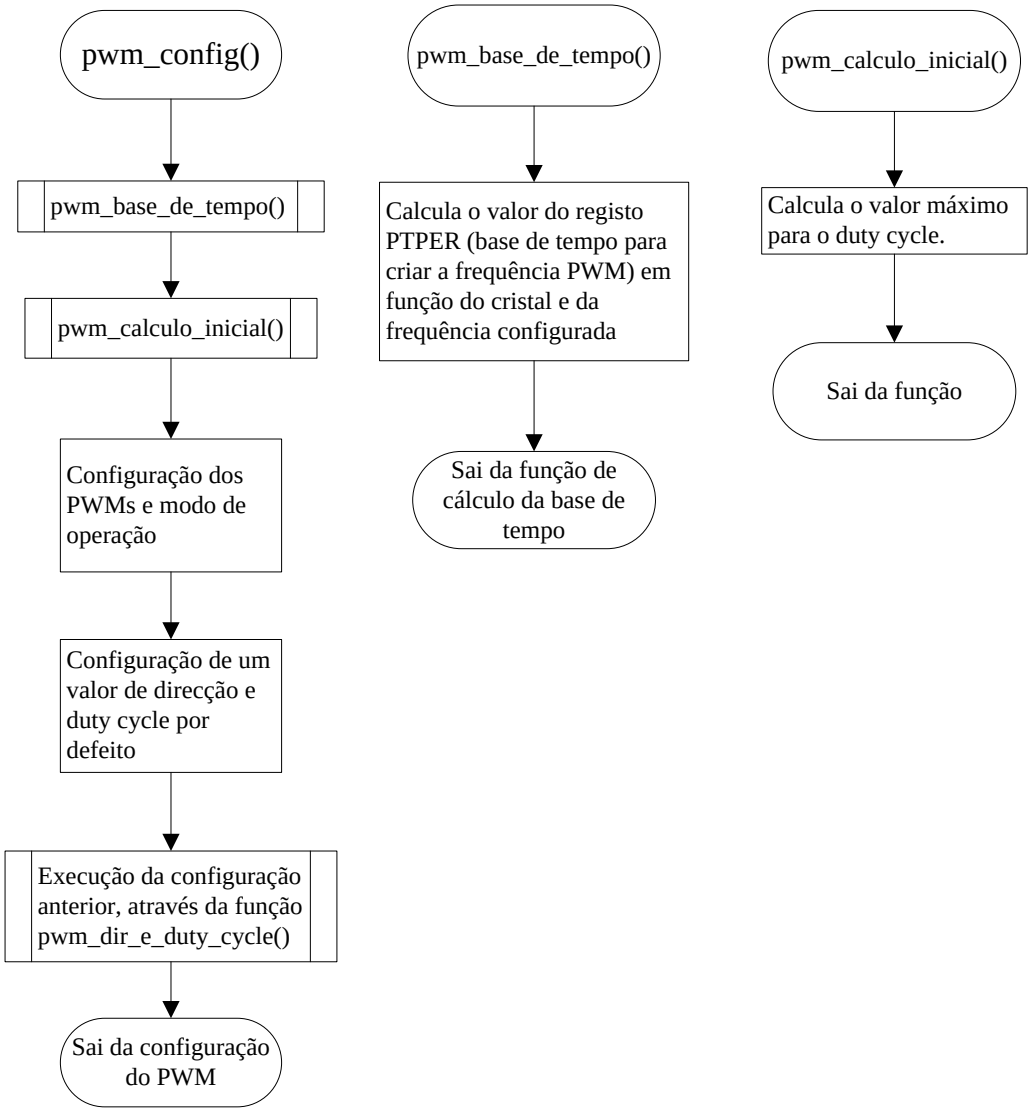


Fig. 84 – Algoritmo da função de configuração do PWM

A função **pwm_dir_e_duty_cycle()** actualiza o valor do *duty cycle* e direcção. O algoritmo em baixo representado, Fig. 85, corresponde à função pwm_dir_e_duty_cycle(). Esta, ao ser executada começa por verificar se o valor pretendido para o *duty cycle* não é superior ao valor máximo, caso o seja o *duty cycle* toma o valor máximo. A próxima verificação é na direcção pretendida, sendo actualizado o *duty cycle* nos PWM correspondentes à direcção desejada.

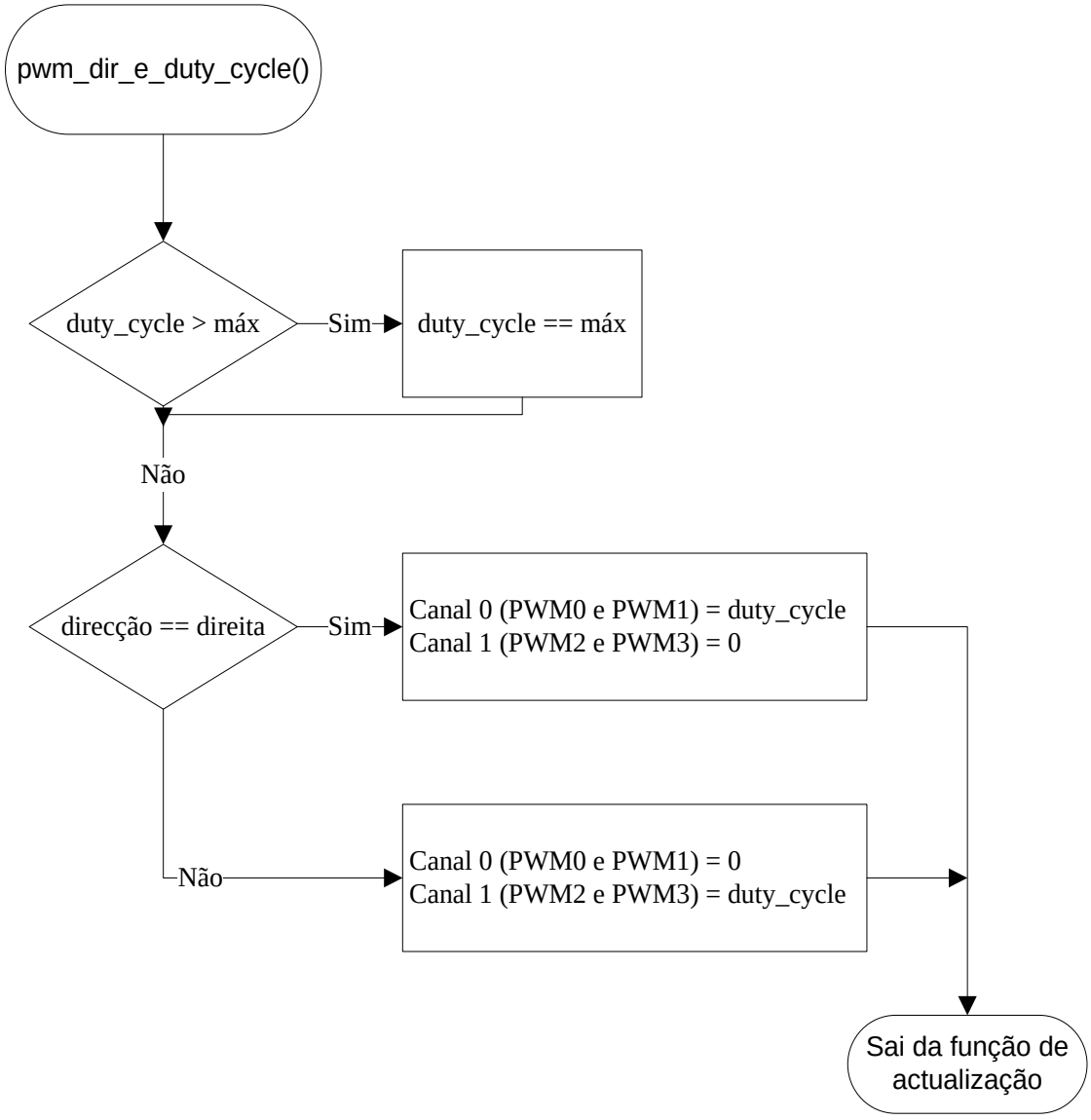


Fig. 85 – Algoritmo da função pwm_dir_e_duty_cycle()

O código destes algoritmos encontra-se no anexo II no ficheiro **pwm.c** e **pwm.h** estando devidamente comentado.

4.7.4 Conclusões gerador de sinais PWM

Durante a implementação do módulo PWM obtiveram-se algumas conclusões, descritas abaixo:

- A utilização dos canais PWM em módulo complementar adequa-se perfeitamente aos sinais necessários para controlar a velocidade e direcção da ponte H
- A frequência utilizada não é audível cumprindo os requisitos do projecto
- Para a frequência seleccionada a resolução dos PWMs é de 11bits obtendo-se precisões de 3.5RPM para o motor utilizado (7500RPM).
- As funções programadas permitem a configuração da frequência dos canais PWM.
- É necessário utilizar *dead time* para impedir que os MOSFETs comutem simultaneamente.

4.8 Hardware

Este subcapítulo descreve todo o *hardware* utilizado para o funcionamento do microcontrolador, incluindo as ligações para programação e LEDs indicadores de funcionamento.

A Fig. 86 representa o circuito utilizado, sendo agora descrito. O elemento chave deste circuito é o microcontrolador que necessita de 4 pinos de alimentação (AVSS, VSS, VDD, AVDD), estando ligados em paralelo com estes vários condensadores de desacoplamento que filtram o ruído da tensão de alimentação.

O cristal utilizado é de 10MHz sendo multiplicado no microcontrolador por quatro através de um PLL (*phase-locked loop*) interno, ficando o *clock* deste a oscilar a 40MHz, para o cristal oscilar é necessário dois condensadores de 15pf .

Os LEDs utilizados são informativos, LED verde quando ligado indica a presença da tensão de alimentação, o LED vermelho indica o erro quando ocorrem.

A resistência R27 (10KΩ) é uma resistência de *pull up* que evita que o pino de *reset* (MCLR) seja activado por casualidade.

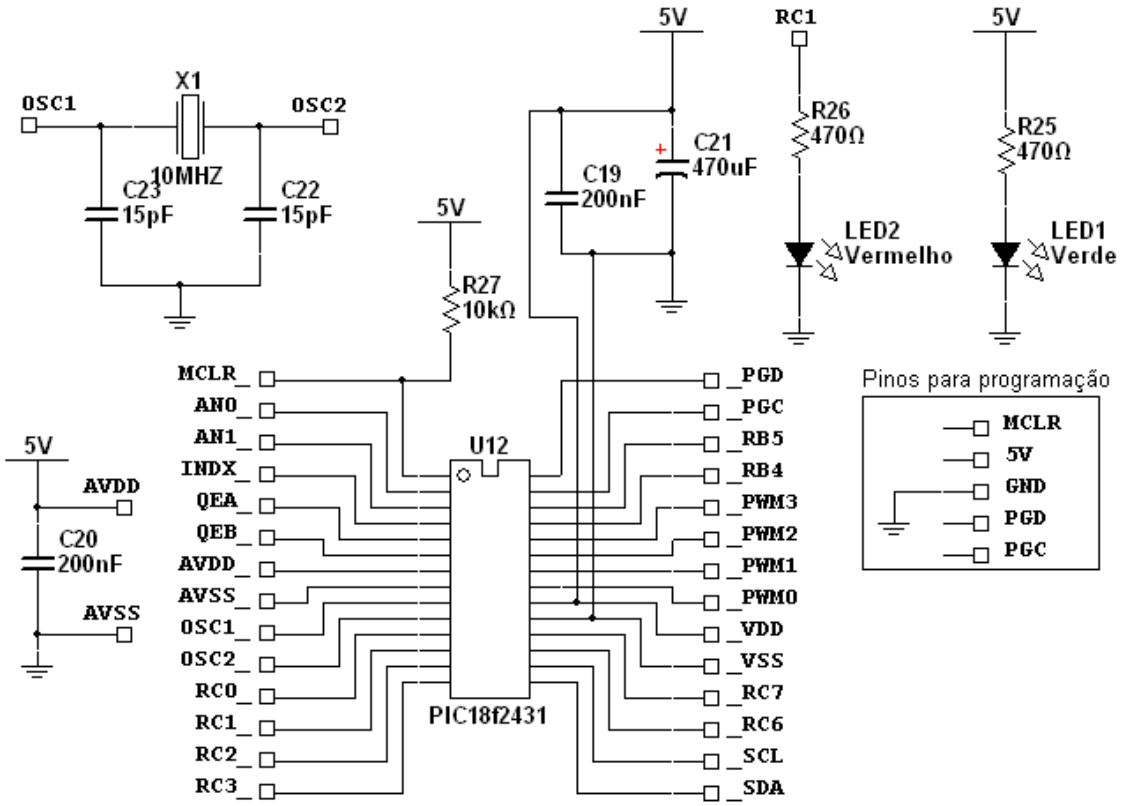


Fig. 86 – *Hardware* envolto ao microcontrolador PIC18f2431

4.8.1 Conclusões do hardware envolto ao microcontrolador

Deste subcapítulo retiram-se as seguintes conclusões:

- É necessário colocar alimentação nos pinos AVDD e AVSS sem os quais o microcontrolador não funciona, ao contrário da família PIC16Fxxx.
- É necessário colocar condensadores nos pinos de alimentação, sem os quais o módulo HSPLL do microcontrolador não funciona.
- Os condensadores aplicados no cristal devem ser os recomendados no manual do microcontrolador para um bom funcionamento.
- O cristal máximo que este microcontrolador suporta é de 20MHz sendo a sua frequência de operação máxima de 40MHz, utilizando-se um cristal de 10MHZ sendo multiplicado internamente por quatro.

4.9 Conclusões

Após a escolha do microcontrolador, compiladores, ferramentas de programação, implementação por *software* dos módulos PWM, comunicação I2C, leitura/actuação de sensores e implementação de *hardware* envolto ao microcontrolador resultaram as seguintes características e conclusões.

- O microcontrolador escolhido adequa-se perfeitamente aos módulos implementados.
- A utilização da linguagem de programação C diminui o tempo de programação permitindo implementar as mesmas funções que em linguagem assembler.
- O interface dos dispositivos com o barramento I2C é muito acessível, bastando ligar os pinos SCL e SDA às linhas correspondentes no barramento.
- O protocolo apesar de bastante simples adequa-se muito facilmente a comunicações complexas, sendo muito eficaz e fiável.
- As tramas de leitura e escrita desenvolvidas sobre o protocolo I2C são muito flexíveis, permitem executar leituras e escritas em posições exactas definidas pelo utilizador.
- Caso ocorra um erro na comunicação um LED vermelho é ligado mantendo-se até que se faça *reset* ao microcontralador eliminando o erro de comunicação.
- Os sensores TMP100 são extremamente precisos e funcionais tendo como desvantagem a baixa protecção contra electricidade electrostática.
- O sensor de corrente funciona bem no entanto é necessário ajustar o valor 0 ocasionalmente, sendo proposto como trabalho futuro o uso de um sensor com saída digital.
- No sensor de tensão ocorre o mesmo que no sensor de corrente com pequenas alterações do valor da tensão de alimentação é necessário ajustar o valor de referência.

- Como existe várias interrupções com a mesma prioridade, a leitura e actuação de um sensor de cada vez é muito eficaz tornando-se imperceptível, não causando atrasos noutros intervalos de tempo.
- O LED indicador mostrou-se bastante útil tornando possível saber o porquê do erro ocorrido de forma muito fácil.
- A utilização dos canais PWM em módulo complementar adequa-se perfeitamente aos sinais necessários para controlar a velocidade e direcção da ponte H
- A frequência utilizada não é audível cumprindo os requisitos do projecto.
- É necessário utilizar *dead time* para impedir que os MOSFETs comutem simultaneamente.
- É necessário colocar alimentação nos pinos AVDD e AVSS sem os quais o microcontrolador não funciona ao contrário da família PIC16Fxxx.
- É necessário colocar condensadores nos pinos de alimentação, sem os quais o módulo HSPLL do microcontrolador não funciona.

Capítulo 5 - Encoder óptico e controlador PID

5.1 Introdução

Com a implementação do software e hardware descritos nos capítulos anteriores é possível colocar o motor a rodar e até variar a sua velocidade e sentido, no entanto, as variações de carga e tensão de alimentação (descarga das baterias), alteram a velocidade não garantindo que este rode à velocidade desejada.

Para garantir que o motor rode a uma velocidade desejada é necessário conhecer a velocidade actual e aplicar um controlo que ajuste de forma automática a velocidade actual para a velocidade pretendida.

Neste capítulo é descrito o dispositivo utilizado que permite a leitura de velocidade, como funciona e como foi implementado, o *software/hardware* para efectuar a leitura de velocidade actual, assim como, a análise a vários métodos de controlo, descrição do método escolhido, implementação por software e resultados obtidos.

5.2 Encoder

O *encoder* óptico, Fig. 87, é o dispositivo electromecânico que converte um deslocamento angular em sinais eléctricos (analógicos ou digitais), a partir dos quais é possível determinar várias informações, tais como, posição ou velocidade do sistema, a qual o *encoder* se encontra acoplado.

Fig. 87 – Exemplo de um *encoder* incremental [52]

5.2.1 Princípio de funcionamento

Os *encoders* ópticos angulares são constituídos por um ou mais discos perfurados (ou máscaras) e por um ou mais conjuntos de emissores/receptores de infravermelhos, estando os emissores numa face do disco e os receptores na outra, Fig. 88.

Quando se inicia a rotação o feixe infravermelho ora passa pelo orifício do disco atingindo o receptor, ora não, o receptor como é um fototransístor, só conduz quando recebe o feixe infravermelho gerando um sinal correspondente à passagem de luz pelos orifícios

O sinal gerado aproxima-se a uma onda quadrada sendo a sua frequência proporcional à velocidade de rotação.

Quando contados os pulsos gerados é possível determinar a posição do sistema a qual o *encode*r se encontra acoplado

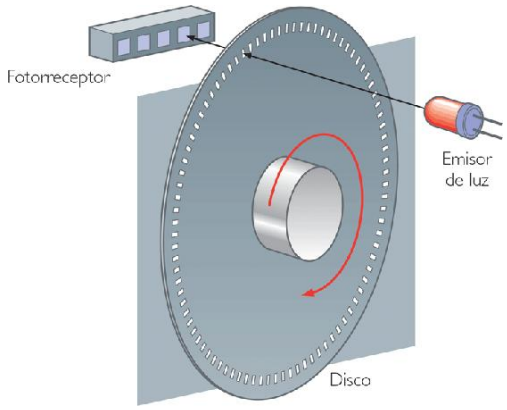


Fig. 88 – Exemplo da constituição interna do *encoder* incremental [52]

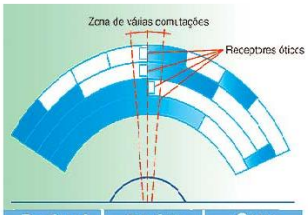
Quanto maior o número de pulsos em cada disco (PPR- pulsos por revolução) maior será a precisão da medição efectuada.

5.2.2 Tipos de encoder óptico

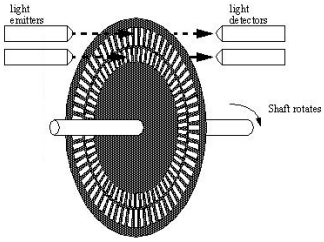
Os *encoders* ópticos dividem-se em dois tipos, *encoders* incrementais e *encoders* absolutos.

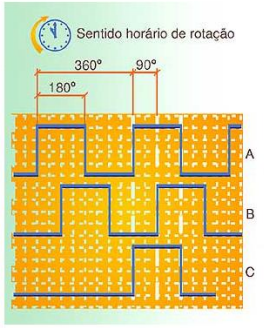
O *encoder* incremental pode gerar de um a três sinais, dependendo das suas funcionalidades. A figura anterior (Fig. 88) representa um *encoder* incremental simples apenas gera um sinal, através do qual, é possível determinar a velocidade e posição. A Fig. 89 (b) representa um *encoder* incremental de dois canais, este é constituído por um disco com duplos orifícios desfasados 90°, que para além da velocidade e posição, permite obter a direcção de rotação. O *encoder* incremental que gera três sinais possui um canal de posição zero (*index*) que indica quando o *encoder* efectuou uma rotação, os seus sinais de saída estão representados na Fig. 89 (b) sendo o sinal A e B correspondentes ao canal A e B respectivamente e o sinal C correspondente ao canal zero (*index*).

O *encoder* absoluto fornece um sinal digital na sua saída, o seu disco não contém orifícios, tendo sido substituído por uma mascará, através da qual é gerado um código binário único para cada posição do seu curso, como representado na Fig. 89 (a). Estes *encoders* estão limitados na sua resolução, cada bit é composto por um conjunto emissor/receptor de infravermelhos sendo o número de emissores/receptores limitados pela sua construção (*hardware).*



Decimal	Binário	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000





(a) (b)

Fig. 89 – (a) *Encoder* absoluto, (b) *encoder* incremental [54]

5.2.3 Encoder utilizado

O *encoder* utilizado foi o HEDS-5540#A11, idêntico ao representado na Fig. 90, este vem acoplado no motor DC utilizado, possui 500 PPR e 3 canais, dois canais utilizados para calcular o posicionamento, velocidade e direcção e um terceiro que é utilizado como posição zero ou indicador de rotações.

Fig. 90 – Imagem do *encoder* utilizado HEDS-5540#A11 [55]

A escolha de um *encoder* incremental deve-se principalmente à sua precisão sendo o mais apropriado para medições de velocidade e direcção, se o objectivo fosse medições de posicionamento o *encoder* absoluto teria vantagens.

5.2.4 Medições de velocidade e direcção

Neste projecto a utilização do *encoder* destina-se à leitura de velocidade e direcção de rotação.

A detecção do sentido de rotação pode ser executada de duas formas. Da forma utilizada o sentido de rotação é determinado pelo microcontrolador analisando o canal A e B do *encoder*, se o canal A fica a nível lógico 1 (5V) primeiro que o canal B roda no sentido horário, caso contrário roda no sentido anti-horário. O microcontrolador utilizado dispõe de um módulo de *hardware* que implementa a detecção da direcção, activando uma flag caso rode no sentido horário, desactivando-a caso rode no sentido inverso.

A segunda forma consiste na utilização de um flip-flop do tipo D como mostra a figura abaixo, Fig. 91, ficando a saída a nível lógico 1 (5V) se o *encoder* rodar no sentido horário e a nível lógico 0 (0V) caso rode no sentido inverso.

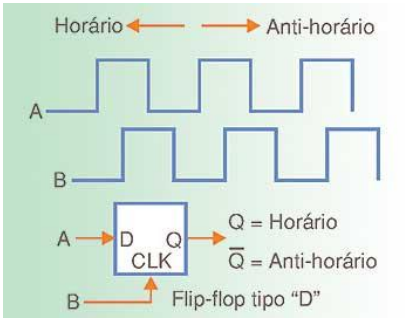


Fig. 91 – Circuito para determinar sentido de rotação [56]

A leitura da velocidade é mais complexa, tendo-se analisado dois métodos para o fazer.

O primeiro consiste na contagem de X pulsos num intervalo de tempo fixo Y. Em que Y, (13), é o tempo mínimo necessário para detectar 1 pulso a 1 RPM (rotação por minuto) com um *encoder* de K PPR (pulsos por revolução).

$$Y = \frac{60}{K} \qquad \textbf{(13)}$$

Sabendo que ocorreram X pulsos em Y tempo a velocidade será determinada pela equação (14).

$$RPM = X \tag{14}$$

Sendo Y o intervalo de tempo mínimo necessário entre actualizações de leituras de velocidade e este só diminui com o aumento do número PPR do *encoder*, esta solução torna-se inviável pois o custo do *encoder* é proporcional aos seus PPR.

Com este método e com *encoder* utilizado que possui 500 PPR o intervalo entre leituras mínimo é de 120ms (Y), tornando a resposta do sistema lenta.

A opção escolhida consiste em medir o tempo em que um pulso gerado por um canal do *encoder* está a nível lógico 1 ou 0.

A onda quadrada gerada por um canal do *encoder* possui um *duty cycle* de 50% o que torna irrelevante ler o tempo t1 ou t0, como representa a Fig. 92. Sendo a frequência do pulso proporcional à velocidade de rotação, os tempos t1 e t0 têm uma relação directa com a velocidade.

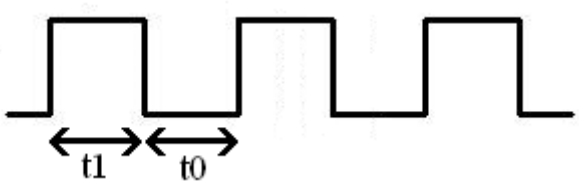


Fig. 92 – Tempos medidos para cálculo da velocidade

Esta implementação é mais complexa devido aos tempos a ser medidos, no entanto permite intervalos de tempo entre leituras de velocidade variáveis, estes são tão menores quanto maior for a velocidade.

A título de exemplo, se o *encoder* utilizado rodar a 1RPM, o tempo mínimo para medir um nível lógico alto ou baixo (t1 ou t2) é de 60ms, se este rodar a 10RPM esse mesmo tempo passa para 6ms.

O cálculo do tempo do nível lógico alto ou baixo (t1 ou t0) é feito através da equação (15), em que Y é o tempo t1 ou t0.

$$Y = \frac{60}{2 \times PPR \times RPM} \tag{15}$$

5.2.5 Software

O microcontrolador PIC18f2431 possui um módulo de leitura de tempos dos níveis lógicos do encoder, a implementação desse módulo é abaixo descrita.

O módulo QEI (*Quadrature encoder interface*) do microcontrolador possui 3 entradas, duas para os canais A e B e uma terceira para o canal zero (*index*), detecta de forma automática a direcção de rotação gerando uma interrupção caso pretendido, possui um contador de 16 bits que incrementa ou decrementa conforme o sentido de rotação indicando para posicionamento do *encoder,* permite fazer o posicionamento através da leitura de um ou dois canais aumentando a sua precisão, possui um módulo de leitura de velocidade que permite o ajuste para altas e baixas rotações

Neste tipo de leitura surgem dois problemas; o primeiro deve-se aos tempos a medir nas baixas rotações (tempos grandes), o segundo surge nas altas rotações sendo causado pela diferença mínima entre tempos, para rotações muito próximas.

Para verificar a existência destes dois problemas estão abaixo descritas duas situações e duas soluções, utilizando como modelo a aplicação em causa.

1º Problema

Para o primeiro problema a situação consiste em utilizar um encoder de 500 PPR a partir do qual é calculado o tempo de um nível lógico (alto ou baixo) para a rotação mínima de 1RPM.

Correspondendo 500 PPR a 500 níveis lógico alto e 500 níveis lógico baixo, existem 1000 níveis lógicos por rotação, quando o *encoder* roda a 1RPM, existem 1000 níveis lógicos a cada 60 segundos, o que dá um tempo de 60ms por nível lógico.

Como o microcontrolador opera a 40MHz e o seu contador de tempo (*timer_5*) funciona a ¼ da sua frequência, este incrementará o seu valor a cada 0.1µs. O valor resultante do *timer_5* do microcontrolador para 1RPM será de 600000, no entanto o *timer_5* é de 16bits, este só conta até 65535, o que dá origem ao problema 1.

Para resolver este problema utilizou-se o *prescaler* do *timer5* para as velocidades mais baixas, aumentando assim o seu tempo entre incrementos para 0.8 µs, permitindo a distinção e medição entre rotações muito baixas.

2º Problema

Para verificar a existência do segundo problema são calculadas as diferenças de tempo entre duas altas rotações muito próximas (6999RPM e 7000RPM), utilizando um encoder com 500PPR.

Para 6999 rotações o tempo de um nível lógico é de 8.5727µs, para 7000 é 8.5714µs, sendo a diferença entre as duas de 1.22ns.

No problema anterior foi referido que o *timer_5* incrementava a cada 100ns, sendo a diferença das duas rotações de apenas 1.22ns, desta forma não existe distinção entre 6999 e 7000 RPM.

Para resolver este problema aumentaram-se o número de níveis lógicos lidos, aumentando assim a diferença entre rotações muito próximas de forma a haver distinção entre as mesmas.

Em resumo, a solução para estes problemas consiste em aumentar o número de pulsos lidos para velocidades altas e diminuir o número de pulsos lidos e colocar um *prescaler* no *timer5* para velocidades mais baixas.

Os algoritmos abaixo descritos implementam todas as funções necessárias à leitura de velocidade (configuração, leitura, cálculo para RPM, ajuste de *prescaler* e *postscaler*).

O módulo de leitura de velocidade contém vários elementos que são necessários configurar, sendo configurados através da função **encoder_config()**, chamada no início do programa, na função **main()**.

Esta função começa por fazer o cálculo de algumas constantes utilizadas no cálculo da velocidade evitando operações repetidas a cada leitura, após calcular as constantes faz a inicialização de variáveis, configura o módulo QEI e o timer_5, o último passo consiste em activar a leitura e esperar o tempo necessário para a ocorrência da primeira leitura, Fig. 93.

As configurações anteriores consistem em activar o modo de leitura de velocidade, medir o tempo somente num canal e de um nível lógico (inicialmente o motor está parado), colocar *timer_5* com um *prescaler* de 1:8, associá-lo ao módulo de leitura de velocidade e por fim configurar a interrupção do *timer_5*.

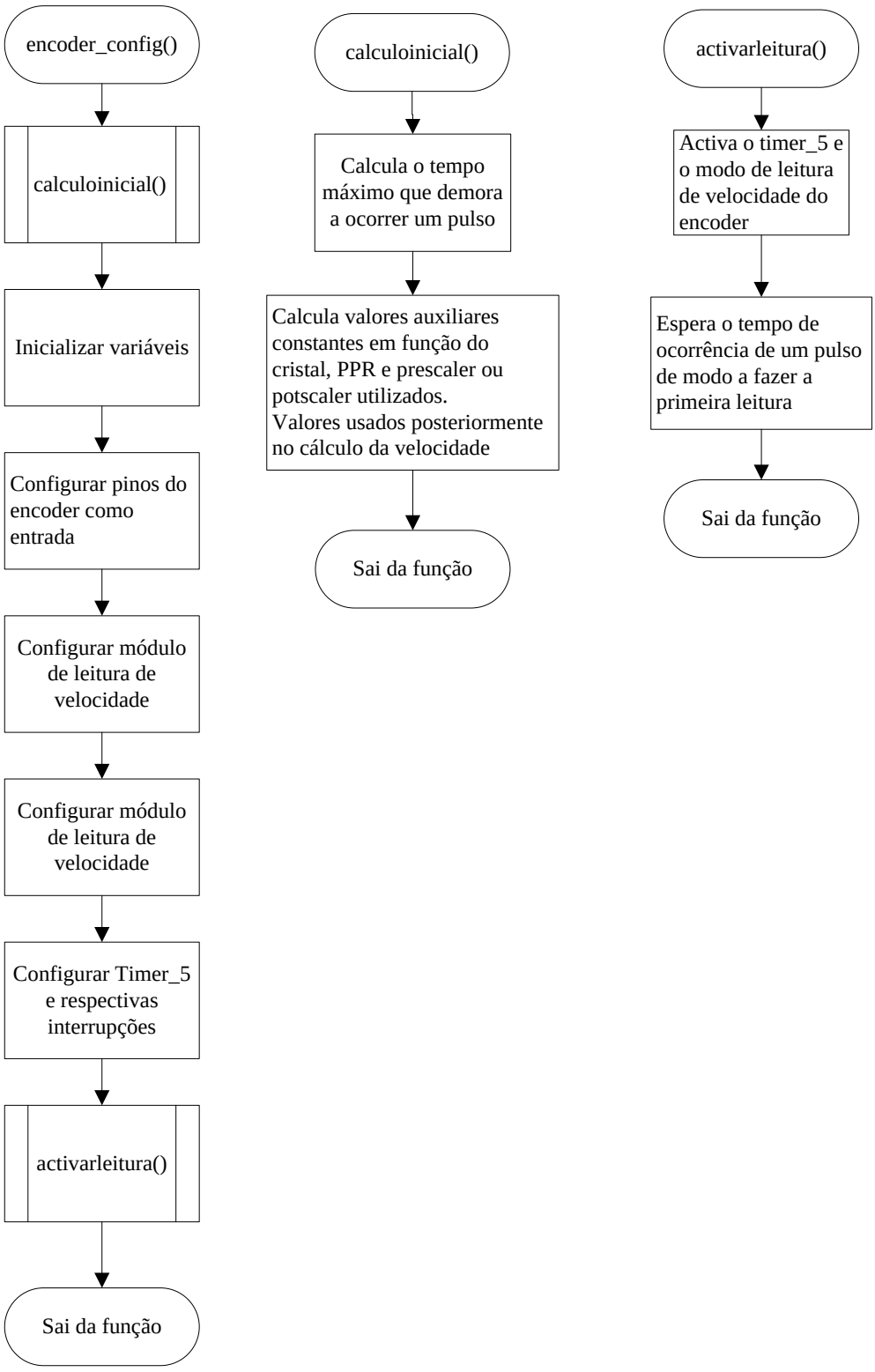


Fig. 93 – Algoritmo da função encoder_config() e das suas funções auxiliares

O módulo de leitura de velocidade é contínuo sendo o tempo do(s) pulso(s) guardado continuamente nos registos VLRH e VLRL, quando se pretende a velocidade actual é chamada a função **velocidade()** (chamada antes da execução do PID) que calcula o valor da velocidade em RPM.

O cálculo da velocidade depende das seguintes variáveis:

O *prescaler* do *timer_5* utilizado varia nas escalas de 1:1 e 1:8 sendo o valor do timer_5 incrementado a cada 100ns ou 800ns respectivamente.

O postscaler do número de níveis lógicos medidos, varia nas escalas de 1:1, 1:4, 1:16 e 1:64, dependendo da velocidade, sendo optimizado para máxima precisão.

O cristal pode ser alterado e configurado através do barramento I2C.

O encoder também pode ser alterado e os seus PPR configurado através do barramento I2C.

O tempo medido, em cada pulso ou pulsos, guardado nos registos VLRH e VLRL.

As variáveis são então, prescaler, postscaler, cristal, PPR, VLRH e VLRL.

O primeiro passo para calcular a velocidade é saber o tempo de um nível lógico. Através dos registos VLRH e VLRL determina-se o número de incrementos do *timer_5*, sabe-se que o timer_5 opera a ¼ da frequência de oscilação e esta por sua vez é 4 vezes a frequência do cristal, logo o timer_5 opera à frequência do cristal, no entanto é necessário multiplicar pelo *prescaler* do timer_5 e dividir pelo postscaler para obter o tempo de um nível lógico, a equação (16) representa o cálculo descrito.

$$Tnível = \frac{(VLRH \times 255 + VLR) \times prescaler}{potscaler \times cristal} \quad \textbf{(16)}$$

Sabendo o tempo de um nível lógico, é necessário calcular quantos níveis lógicos acontecem em 60 segundos, equação (17).

$$n^{o}\, niveis = \frac{60}{Tnível} \quad \textbf{(17)}$$

Como numa rotação os níveis lógicos são o dobro dos PPR do encoder a velocidade em RPM será dada pela equação (18).

$$Velocidade\_rpm = \frac{n^{o}\, níveis}{2 \times PPR} \quad \textbf{(18)}$$

A equação (19) representa todo o cálculo necessário para determinar a velocidade. Esta equação está dividida em 2 coeficientes; o primeiro é constituído por valores constantes sendo calculado na função de configuração, o segundo depende do valor lido, a multiplicação deste coeficientes dará a velocidade actual em RPM.

$$Velocidade\_rpm = \left(\frac{30 \times potscaler \times cristal}{PPR \times prescaler} \right) \times \left(\frac{1}{(VLRH \times 255 + VLRL)} \right) \quad \textbf{(7)}$$

O algoritmo da função de cálculo da velocidade está abaixo representado, a interrupção do timer_5 foi usada para indicar que o motor se encontra parado, quando a contagem deste for superior a 65535 a interrupção é activa indicando que não ocorreu nenhuma medição de nível lógico. Como o motor está parado, é esperado o tempo correspondente ao necessário para ocorrer uma transição de níveis lógicos indicando que este está a rodar, podendo-se calcular novamente a velocidade.

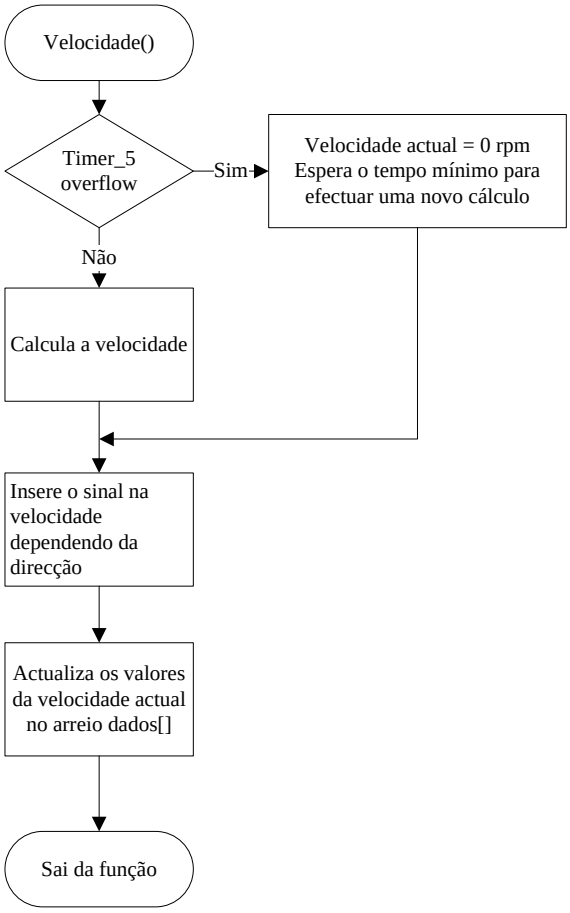


Fig. 94 – Algoritmo da função que calcula a velocidade actual

A função **ajustar_escala()** é executada após o cálculo da velocidade ajustando o *prescaler* do *timer_5* e o *postscaler* do número de pulsos lidos de forma a que a leitura de tempos dos níveis lógicos seja o mais precisa possível.

Esta função é constituída por quatro escalas, variando o *prescaler* e *postscaler* da seguinte forma:

Escala 1→ *Prescaler* do *timer_5* é de 1:8 e o *postscaler* ajustado para a leitura de 1 nível lógico.

Escala 2→ *Prescaler* do *timer_5* é de 1:1 e o *postscaler* ajustado para a leitura de 4 níveis lógicos.

Escala 3→ *Prescaler* do *timer_5* é de 1:1 e o *postscaler* ajustado para a leitura de 16 níveis lógicos.

Escala 4→ *Prescaler* do *timer_5* é de 1:1 e o *postscaler* ajustado para a leitura de 64 níveis lógicos.

A escolha da escala depende da velocidade actual e dos limites configurados, Fig. 95, para cada escala, existem 6 registos de 16 bits no *array* de dados para configurar os limites.

Escala / Limites	**Escala 4**	**Escala 3**	**Escala 2**	**Escala 1**
Vel_act <= LIM1_INF				
LIM1_SUP< Vel_act <=LIM2_INF				
LIM2_SUP< Vel_act <=LIM3_INF				
LIM3_SUP< Vel_act				

Fig. 95 – Mudança de escala em função dos limites

São utilizados dois limites para mudança de escala, um limite superior para mudar para a escala seguinte e um limite inferior para mudar para a escala anterior criando assim uma margem que evita a mudança contínua de escala quando a velocidade actual se encontrar próxima dos limites. Se só fosse utilizado um limite e a velocidade actual oscilasse em torno desse limite, haveria uma mudança contínua de escala.

Os valores dos limites foram escolhidos da seguinte forma:

1. Para todas as escalas foi calculado o valor do timer_5 desde 1RPM até 10000RPM, tendo em conta os PPR do encoder e a frequência do cristal.
2. Para cada escala foi retirado o valor mínimo e máximo de RPM em que é possível distinguir velocidades com precisão de 1RPM.

3. Com os valores máximo e mínimo foi calculado um ponto médio para cada escala de modo que a mudança de escala seja 50% abaixo do limite de distinção.
4. De modo a evitar mudanças de escala contínuas foram criados 2 limites em torno do ponto médio de cada escala.

Todo o processo de modificação dos limites no microcontrolador através do barramento I2C encontra-se descrito no manual do controlador desenvolvido, que se encontra no anexo I.

O algoritmo da função **ajustar_escala()** encontra-se representado na Fig. 96, a variável new_vel corresponde à velocidade actual, e vel_postscaler corresponde ao índice da escala utilizada. Este índice é utilizado para evitar ajustar para a mesma escala continuamente e para utilizar o valor constante, correspondente à escala utilizada calculado na função **encoder_config()** no cálculo da velocidade.

Todas as funções descritas estão implementadas e devidamente comentadas nos ficheiros **encoder.c** e **encoder.h,** no anexo II.

A função de posicionamento do encoder também foi implementada, não tendo sido utilizada, ficando como trabalho futuro a implementação de funções de odometria.

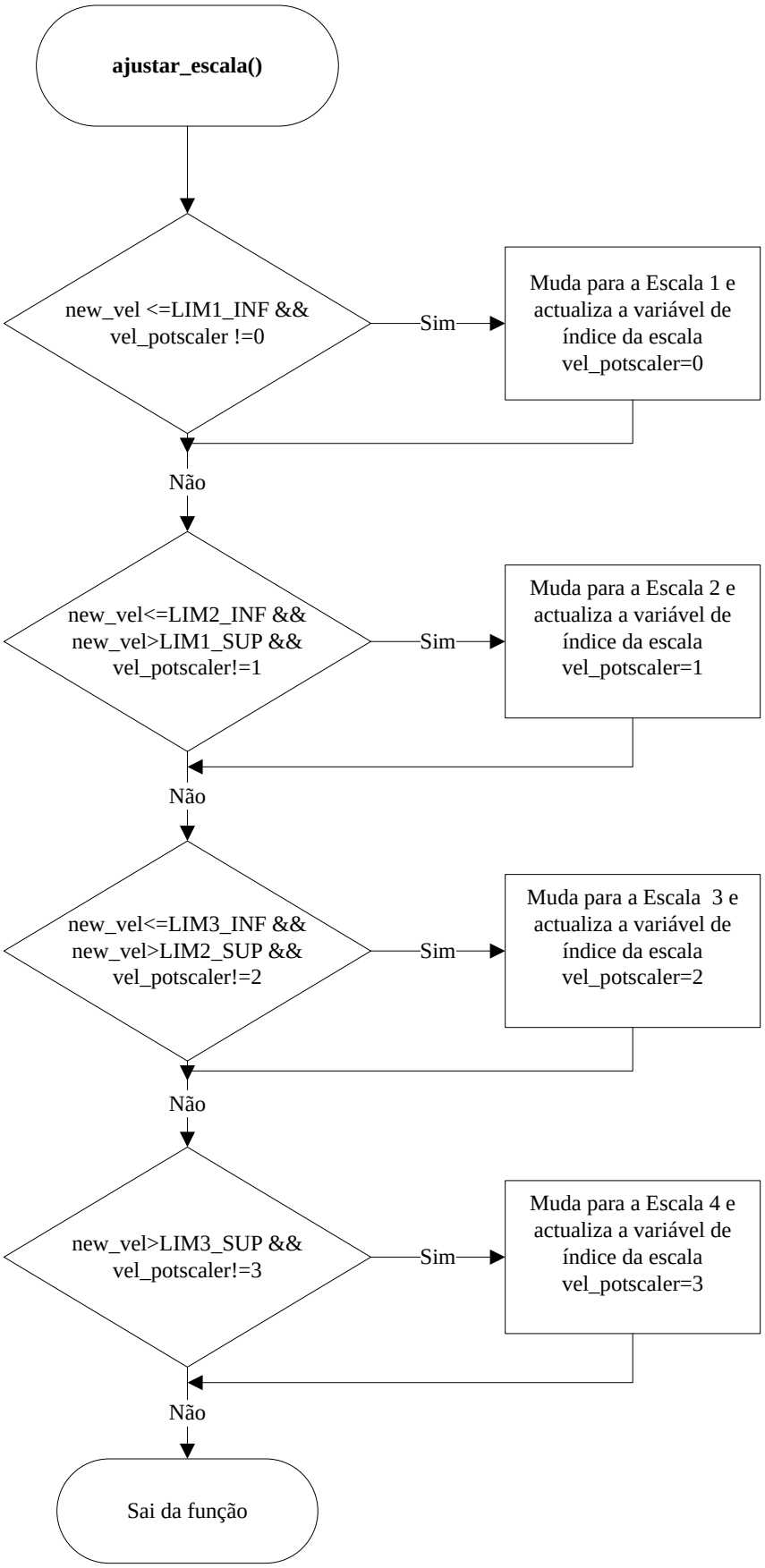


Fig. 96 – Algoritmo da função ajustar_escala()

5.2.6 Hardware

A Fig. 97, descreve os pinos de ligação do *encoder* utilizado. Este precisa de uma tensão de alimentação de 5V (GND, +5V) e de 3 resistências de pull-up nos canais INDEX, CH.A e CH.B para funcionar correctamente, como representa o circuito da Fig. 98.

Pino	Designação
1	GND
2	INDEX
3	CH.A
4	+5V
5	CH.B

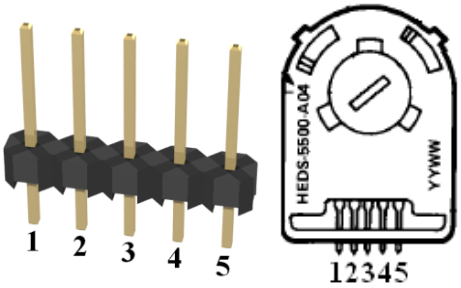


Fig. 97 – Descrição dos pinos do encoder utilizado

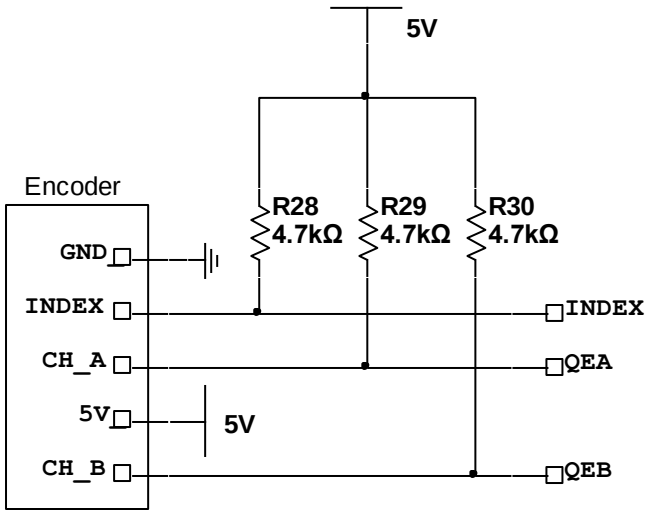


Fig. 98 – Circuito necessário ao funcionamento do encoder

5.2.7 Conclusões

A leitura de velocidade através do encoder óptico é dificultada pela precisão dos tempos medidos, tendo-se obtido as seguintes conclusões na implementação da mesma.

- Após cada leitura de tempo de nível lógico não deve ser gerada nenhuma interrupção, esta provoca atraso na leitura dos próximos tempos originando uma leitura incorrecta (tendo sido esta a primeira versão implementada).

- A leitura de tempos deve ser contínua, sendo o cálculo da velocidade feito com o último valor de tempo lido.
- Com o método utilizado a leitura de velocidade varia de 1RPM a 6249RPM com precisão de 1 RPM, de 6250RPM a 8994RPM com precisão de 2RPM e de 8995RPM a 1000RPM com uma precisão de 3 RPM.
- O tempo mínimo de leitura é 384µs a 10000RPM e o máximo é de 60ms a 1RPM.
- O método de mudança de escala tornou a leitura muito eficaz.

5.3 Algoritmo de controlo (PID)

5.3.1 Controlo em malha fechada

O controlador em malha fechada é a parte responsável por executar realmente o que se deseja, corrigindo perturbações que possam existir no sistema.

Os primeiros controladores deste tipo surgiram no século XVIII, em 1788, James Watt utilizou um controlador flyball, Fig. 99, para controlar a velocidade de um motor a vapor [62].

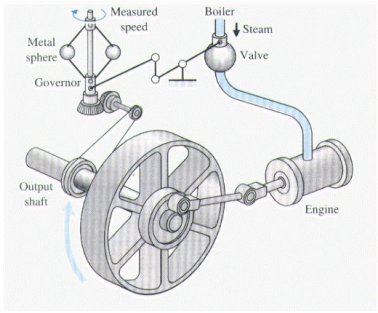


Fig. 99 – Controlador Watt flyball [57]

A Fig. 100 representa um diagrama, do controlo em malha fechada, de realimentação negativa. Este é constituído pelo **processo** que se pretende controlar (velocidade do motor) e por um **controlador** que corrige continuamente o erro, causado pela diferença entre o **valor desejado** (velocidade) e o **valor actual** (velocidade actual), medido através de um sensor.

O controlo em malha fechada pode ser contínuo quando é feito de forma contínua no tempo, sendo implementado com dispositivos analógicos, ou discreto quando as leituras, controlo e actuação são executados em intervalos de tempo fixos, este tipo de implementação usa normalmente componentes digitais. O controlo discreto

é a melhor solução para este projecto, permitindo ajustes de forma digital sendo a sua implementação mais fácil e de custo mais reduzido.

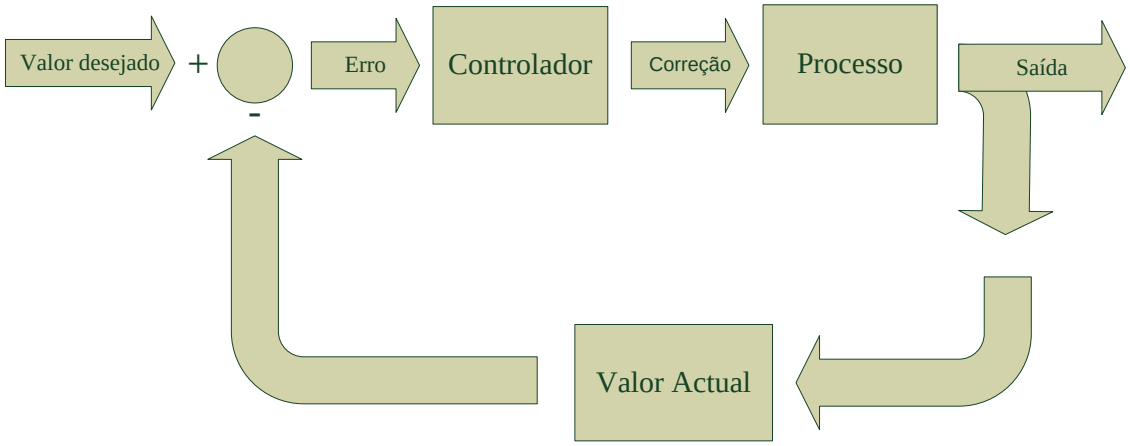


Fig. 100 – Controlo em malha fechada

5.3.2 Controladores

O controlador como já referido é o dispositivo que através do erro entre os valores desejados e reais, corrige o valor real de forma a alcançar o desejado.

As técnicas de controlo implementadas nos controladores, são as formas de como a correcção é executada, existindo várias técnicas de controlo. Para o projecto em causa foram analisadas as técnicas PLL (*Phase-Locked-Loop*), ON-OFF, P (*Proportional*), PI (*Proportional–Integral*) e PID (*Proportional–Integral–Derivative*).

PLL (*Phase-Locked-Loop*)

O PLL é um controlo em malha fechada, cujo funcionamento é baseado na detecção da diferença de fases entre sinais de saída e de entrada controlados através de um oscilador [58]. Este tipo de controlo é muito preciso, no entanto o seu custo é o mais elevado requerendo electrónica auxiliar ao microcontrolador para o implementar.

ON - OFF

O controlo ON-OFF é um controlo mais simples que o PLL, consiste em ligar e desligar o sistema a controlar, tornando as variações na saída bruscas, e menos precisas, a sua implementação é a de menor custo e de fácil implementação, sendo a precisão, a pior.

P (*Proportional*)

O controlo proporcional é mais suave que o controlo ON-OFF, a sua saída varia de forma linear. Este controlo aplica à diferença entre o valor desejado e o real (erro) um factor (ganho proporcional), que quando aplicado ao sistema tende a corrigi-lo.

Quanto maior o ganho proporcional menor é o tempo de resposta, no entanto, maior será a resposta oscilatória do sistema, devendo-se escolher um ganho que proporcione uma resposta satisfatória com oscilações mínimas.

O controlo proporcional não consegue corrigir totalmente o erro em regime permanente, sendo utilizado, nas situações em que o erro é elevado corrigindo-o quase totalmente.

PI (*Proportional – Integral*)

O controlo PI, une o controlo proporcional ao controlo integrativo. Foi referido no controlo proporcional que este não consegue corrigir totalmente o erro em regime permanente, e a forma como o controlo integrativo é calculado permite eliminar o erro em regime permanente. Ao longo do tempo todos os erros são acumulados e multiplicados em cada iteração por um ganho integrativo, quando somado ao controlo proporcional o erro é eliminado. O controlo PI é dos mais utilizados sendo muito eficaz, a sua única desvantagem deve-se ao aumento do *overshoot* devido ao número de erros acumulados.

PID (Proportional – Integral – Derivative)

O controlo PID, para além do controlo PI, possui um controlo derivativo que tem como função diminuir o *overshoot* causado pelo controlo proporcional e integrativo. O controlo derivativo adiciona ao controlo PI a diferença entre o erro actual e o erro anterior, amplificada por um ganho derivativo. A principal desvantagem da inserção deste controlo consiste nas variações bruscas (ruído) que provoca ganhos elevados criando oscilações indesejáveis.

Na Tabela VI estão comparadas todas as técnicas referidas em função da sua precisão, velocidade de resposta, resposta a baixa rotação e custo. A técnica de maior precisão é a PLL no entanto o seu custo é o mais elevado, sendo a técnica de controlo PID a que tem melhor relação desempenho/preço, tendo sido a técnica implementada.

As descrições das técnicas de controlo anteriormente referidas foram baseadas na compilação de ideias obtidas das seguintes referências, [58], [59] e [60].

		Controlo				
		On-Off	**P**	**PI**	**PID**	**PLL**
Precisão	**Alta**					
	Média					
	Baixa					
Velocidade	**Alta**					
	Baixa					
Bom em baixa rotação						
Custo	**Alta**					
	Média					
	Baixa					

Tabela VI – Comparação das várias técnicas de controlo quando aplicadas no controlo de velocidade [58]

5.3.3 Algoritmo PID e Software

A técnica de controlo PID, Fig. 101, é implementada no microcontrolador sendo o elo de ligação de todos os módulos implementados.

Para implementar o controlo PID no microcontrolador foi utilizado como base o artigo AN937, ”Implementing a PID Conttroller Using a PIC18 MCU” [59], juntamente com um exemplo de código em assembler [62], fornecido pela Microchip. Do código e artigo fornecidos utilizou-se o algoritmo PID tendo sido programado em linguagem C e adaptado ao projecto pretendido.

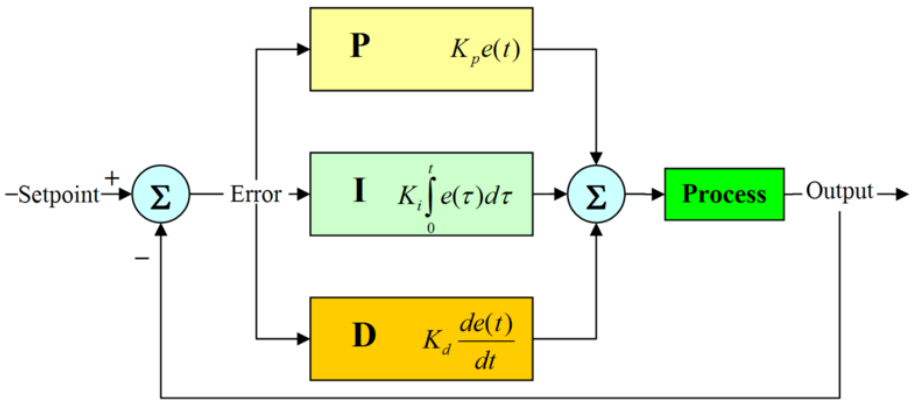


Fig. 101 – Diagrama de blocos de um controlador PID [61]

Não existindo nenhum método específico para implementar um algoritmo PID num microcontrolador, a solução utilizada é apenas uma solução que se adequa ao projecto pretendido. A rotina PID implementada permite a escolha do controlo P, PI ou PID, bastando configurar os ganhos correspondentes com o valor 0 caso não se pretenda utilizar o controlo em causa.

O resultado da rotina PID é composto pela soma dos termos proporcional integral e derivativo como mostra a equação (20), sendo as variáveis as usadas no projecto e descritas na Tabela VII.

$$pid\_out = prop\_value + \text{int}\ eg\_value + deriv\_value \quad \textbf{(8)}$$

O controlo proporcional origina um valor de correcção proporcional ao erro sendo a constante de proporcionalidade o ganho (Kp) do controlador, equação (26).

$$prop\_value = Kp \times new\_error \quad \textbf{(21)}$$

O controlo integral acumula todos os erros passados em intervalos de tempo fixos, originando um valor de correcção contínuo, eliminando o erro residual do controlo proporcional, sendo calculado através da equação (22).

$$\text{int}\ eg\_value = Ki \times \text{int}\ eg\_error \quad \textbf{(221)}$$

O controlo derivativo, complementa o controlo proporcional e integral permitindo um ajuste dos problemas causados pelos mesmos (*overshoot* tempo de estabilização), este controlo é calculado através da diferença do erro actual e passado num intervalo de tempo resultando numa previsão da resposta do sistema.

Este é calculado através da equação (23).

$$deriv\_value = Kd \times delta\_error \quad \textbf{(23)}$$

A Tabela VII, descreve a lista de variáveis utilizadas na implementação do algoritmo PID.

Variável/Constante	Nº bits	Definição
dados[INTEG_MAX]	8	Erro acumulado máximo
dados[DIV_INTV]	8	Intervalo entre cálculos derivativos
new_vel	16	Velocidade real
vel_des	16	Velocidade desejada
pid_out	24	Variável de saída do algoritmo PID
new_error	16	Erro actual
old_error	16	Erro anterior
integ_error	16	Erro acumulado
delta_error	16	Diferença entre erro actual e o erro anterior
prop_value	24	Erro Proporcional
integ_value	24	Erro Integrativo
deriv_value	24	Erro Derivativo
dados[AMSTR_INTV]	16	Intervalo de amostragem
dados[KP]	8	Ganho proporcional
dados[KI]	8	Ganho integral
dados[KD]	8	Ganho derivativo
new_err_z	1	Flag que indica que o erro actual é 0
integ_err_z	1	Flag que indica que o erro acumulado é 0
delta_err_z	1	Flag que indica que o erro acumulado é 0
new_err_sign	1	Flag que indica que o sinal do erro actual
integ_err_sign	1	Flag que indica que o sinal da variável integ_error
old_err_sign	1	Flag que indica que o sinal da variável old_error
delta_err_sign	1	Flag que indica que o sinal da variável delta_error
pid_out_sign	1	Flag que indica que o sinal da variável pid_out
prop_value_sign	1	Flag que indica que o sinal da variável prop_value
integ_value_sign	1	Flag que indica que o sinal da variável integ_value
deriv_value_sign	1	Flag que indica que o sinal da variável deriv_value

Tabela VII – Lista de variáveis e constantes usadas na implementação do algoritmo PID

As *flags* de sinal são utilizadas para evitar o uso de 24/32 bits quando são utilizados 16/24 bits de dados e 1 de sinal, desperdiçando 7 bits. Da forma utilizada são utilizados 16/24 bits de dados e 1 bit para sinal, guardado numa estrutura, correspondendo o valor 1 ao sinal positivo e 0 ao sinal negativo.

As flags de zero são utilizadas para evitar várias comparações de 8/16/24bits, se a variável em causa na primeira comparação efectuada é 0, a flag correspondente fica com o valor 1, evitando posteriormente novas comparações de 8/16/24bits.

O primeiro passo do algoritmo PID implementado no microcontrolador consiste em inicializar variáveis e configurar o timer_1 de modo a gerar uma interrupção periódica, onde será calculado o algoritmo PID, Fig. 102.

O intervalo de tempo da interrupção pode ser configurado através do barramento I2C, dependendo o intervalo de tempo da resposta do processo a controlar.

As flags de sinal são todas inicializadas com o valor positivo por defeito e as flags de zero são inicializadas com o valor zero excepto a flag do erro actual, pois caso fosse zero o primeiro cálculo do algoritmo PID não seria executado.

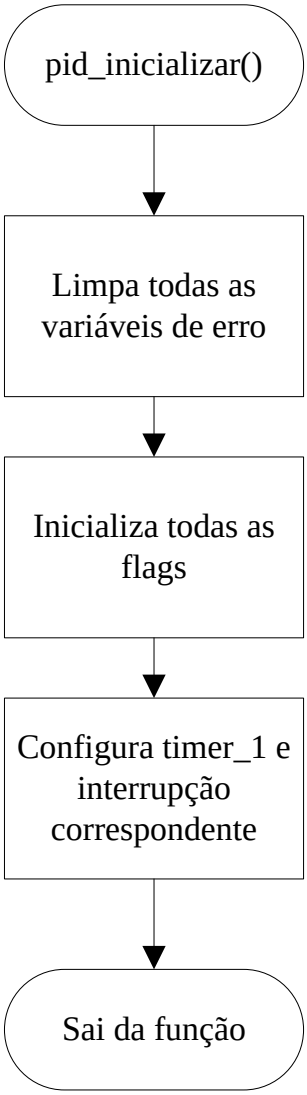


Fig. 102 – Algoritmo da função pid_inicializar()

Na função de interrupção é executado todo o algoritmo PID desde a leitura de velocidades até ao controlo do PWM. O algoritmo da Fig. 103, representa as funções chamadas na interrupção do *timer_1* inclusive o algoritmo PID.

Quando ocorre a interrupção do timer_1 é actualizado o valor da velocidade desejada, após a actualização é verificado se a velocidade desejada não é 0 ou se a travagem não é forçada e se não ocorreu nenhuma actuação dos sensores. Se estas condições se verificarem o algoritmo PID é executado, caso contrário a ponte H é desligada, procedendo a travagem suave.

O primeiro passo dentro do algoritmo PID é ler a velocidade real e calcular o erro entre a velocidade real e a desejada. Após o cálculo do erro actual é calculado o erro acumulado (soma de todos os erros) e o erro diferencial (diferença entre o erro actual e o anterior) através da função pid_interrupt().

As equações (9), (10) e (11) representam os cálculos dos erros descritos, a função pid_main() calcula o erro proporcional, integral e derivativo com os respectivos ganhos, somando-os de forma a obter o valor de saída do algoritmo PID.

$$new\_error = (\text{vel\_des} - \text{new\_vel}) \quad \textbf{(9)}$$

$$\text{int } eg\_error = (\text{int } eg\_error + new\_error) \quad \textbf{(10)}$$

$$delta\_error = (new\_error - old\_error) \quad \textbf{(11)}$$

O penúltimo passo consiste em adicionar ao valor do *duty_cycle* do PWM o valor de correcção resultante do cálculo PID, actualizando no gerador de PWM o valor do *duty_cycle* e direcção.

A ultima função chamada, ajustar_escala(), ajusta caso necessário, a escala de leitura de tempos dos níveis lógicos do encoder, como anteriormente referido na descrição do encoder.

Após a execução de todas estas funções é carregado o valor do timer_1 para gerar uma nova interrupção onde todo o processo descrito será repetido.

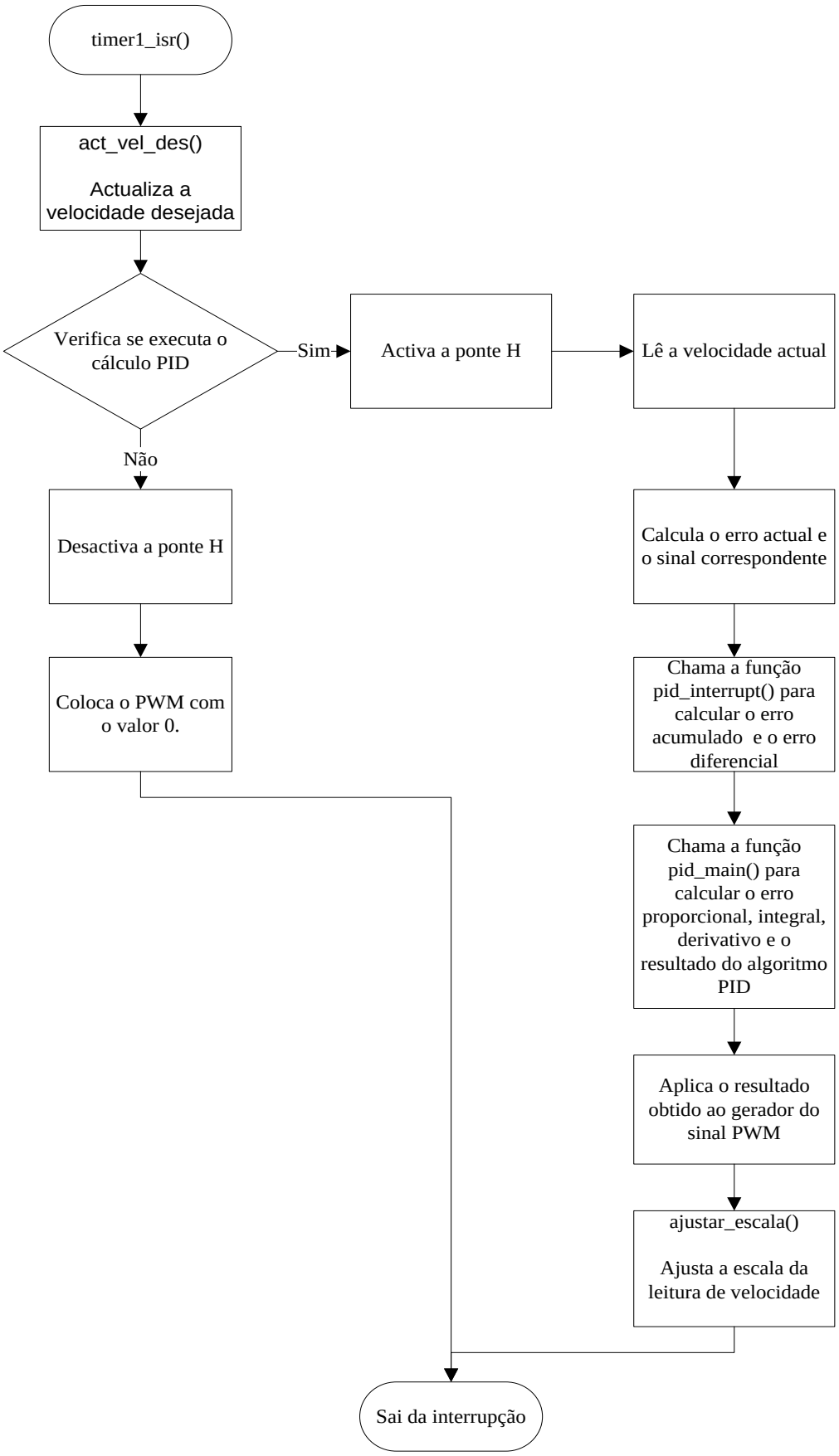


Fig. 103 – Algoritmo da interrupção do timer_1

A função pid_interrupt() é responsável pelo cálculo do erro acumulado e do erro diferencial, sendo necessário estes erros para o cálculo do termo integral e derivativo.

Foi anteriormente descrito que o controlo integrativo causa um aumento do *overshoot*, devido aos sucessivos erros acumulados, de forma a diminuir o *overshoot* é utilizado um limite máximo para o erro acumulado.

O controlo derivativo também causa oscilações bruscas quando existe ruído (variações bruscas de velocidade) no sistema, a forma encontrada para resolver este problema é a adição de um filtro digital. Este filtro consiste em calcular o erro diferencial de n em n interrupções do timer_1, de forma que o controlo proporcional e integral corrige a parte mais significativa do erro, contribuindo o controlo derivativo na diminuição do *overshoot* e tempo de estabilização quando o erro é menor.

O algoritmo seguinte é o implementado na função pid_interrupt(), na qual estão implementadas as soluções referidas, salienta-se o facto de não se executar o cálculo dos erros, caso o erro actual seja nulo.

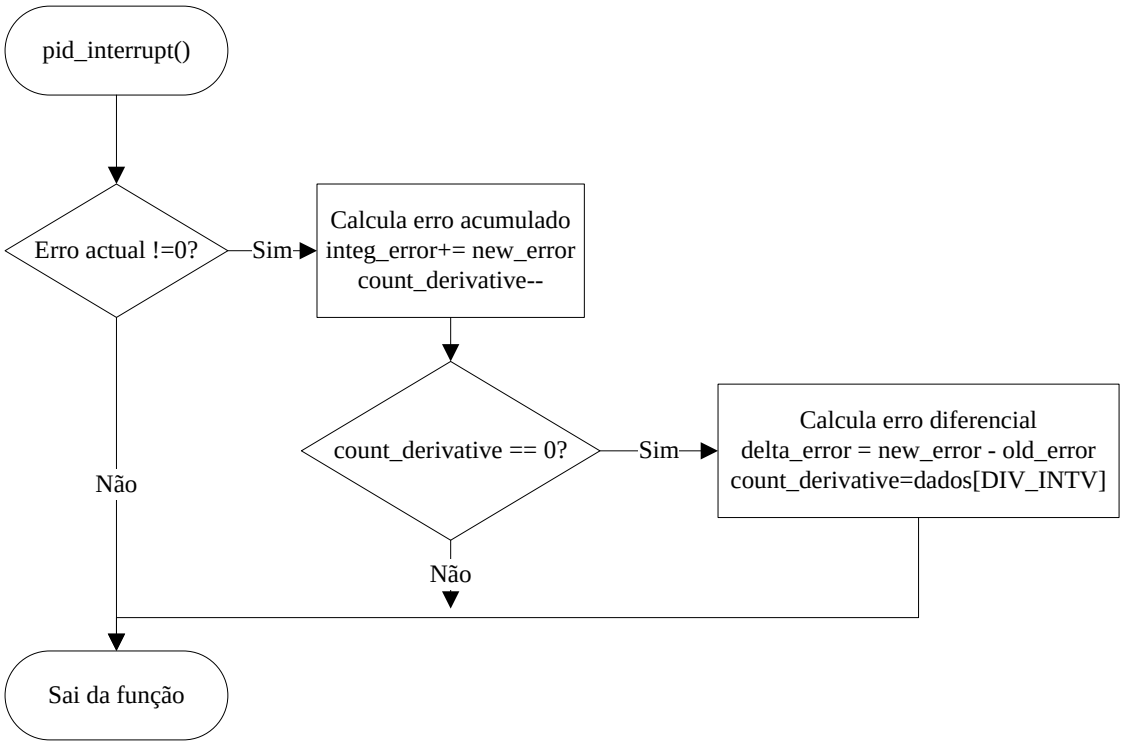


Fig. 104 – Algoritmo da função pid_interrupt()

Na função pid_main(), Fig. 105, são calculadas as partes proporcional, integral e derivativa, sendo somadas no final, resultando no valor de saída do algoritmo PID. O resultado final é reduzido numa escala de 128:1, evitando assim o uso de variáveis

flutuantes e diminuindo o efeito de correcção. Quando o intervalo entre cálculos do algoritmo PID é muito reduzido, o tempo de resposta do sistema não acompanha as correcções aumentando o erro rapidamente, a solução passa por aumentar o intervalo entre cálculos ou diminuir o efeito da correcção do PID.

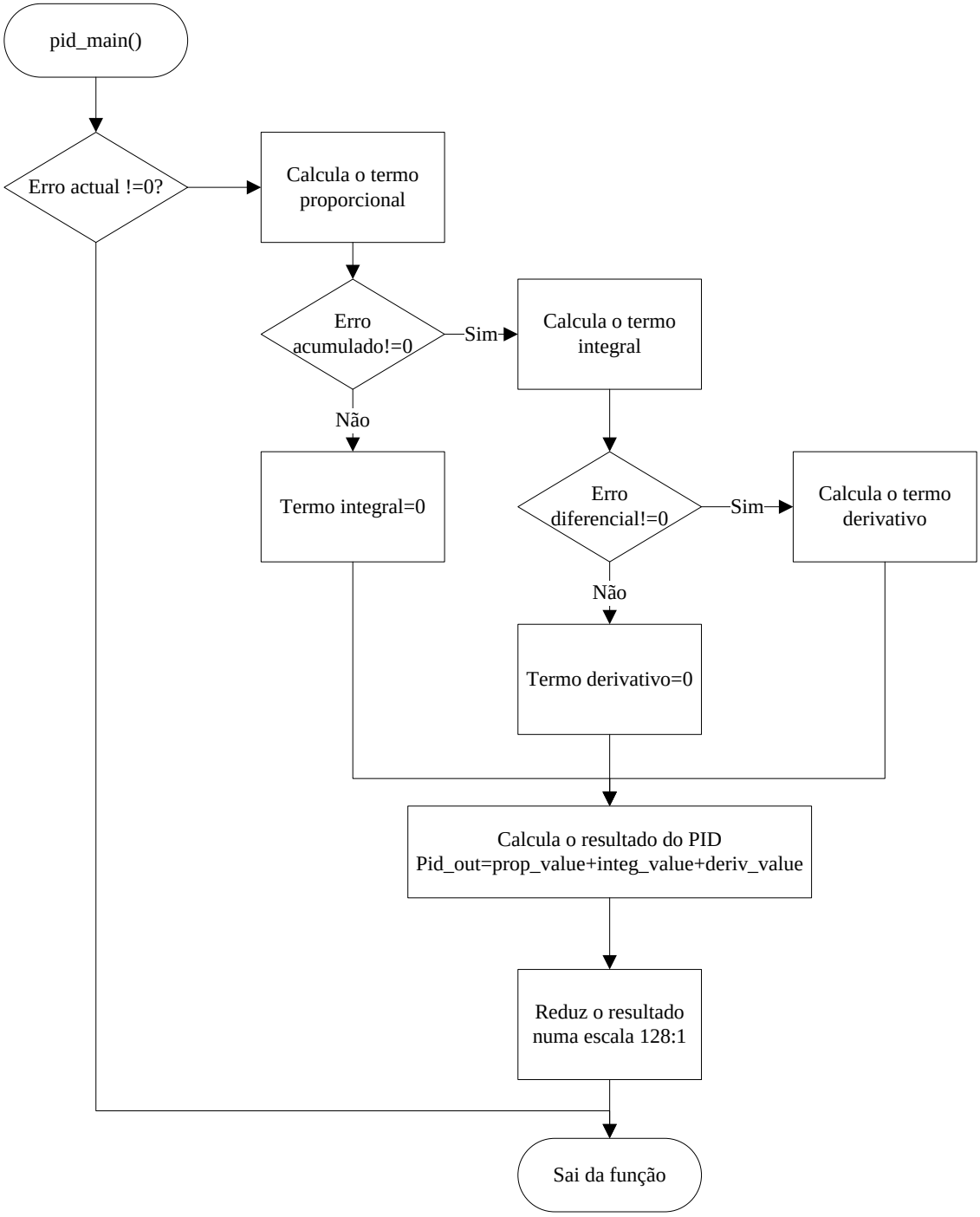


Fig. 105 – Algoritmo da função pid_main()

Para além das funções referidas foram implementadas funções de adição e subtracção para 16 e 24 bits. Estas funções são necessárias porque o sinal das variáveis intervenientes no cálculo encontra-se numa estrutura separada.

Todo o código das funções descritas relativamente ao algoritmo PID, encontra-se devidamente comentado no anexo II nos ficheiros **pid.c** e **pid.h**. estando a interrupção do *timer_1* no ficheiro **main.c**

5.3.4 Resultados

Após a implementação do PID, foi ajustado o intervalo de tempo em que este é executado para 3ms de forma a responder rapidamente, o valor limite do erro acumulado para 10 unidades, o número de interrupções executadas entre cálculos do erro diferencial para 5, e os ganhos Kp, Ki, Kd, para 8, 4 e 0 respectivamente. Estes valores foram ajustados manualmente, da seguinte forma:

1. Definido um intervalo de tempo próximo da resposta do motor de forma que o motor obtenha uma resposta rápida.
2. Definir uma velocidade baixa (100RPM) e ajustar o valor de Kp de modo a obter o tempo de resposta pretendido.
3. Ajustar o valor de Ki de modo a eliminar o erro em regime permanente.
4. Ajustar Kd caso utilizado, de forma a diminuir o *overshoot*.
5. Aumentar o valor da velocidade para próximo do limite.
6. Proceder a um ajuste fino dos ganhos de modo a obter uma resposta mais rápida.
7. Dependendo dos efeitos causados pelo erro acumulado e diferencial, ajustar o valor limite e o intervalo.

Com o método utilizado obtém-se uma resposta satisfatória, no entanto existem métodos experimentais que permitem calcular os ganhos do PID, o método de *Ziegler-Nichols, Cohen-Coon, Chien-Hrones-Reswick* são exemplos de alguns métodos experimentais.

O manual do anexo I contém todos os cuidados necessários no ajuste destes parâmetros.

Com as configurações utilizadas foram feitos alguns testes em vazio para as velocidades 30 RPM 100RPM, 1000RPM, 5000RPM e 8000RPM.

As figuras abaixo representam a velocidade em função do tempo decorrido.

Na Fig. 106, está representado a resposta do sistema para uma velocidade desejada de 30 RPM, o sistema responde de forma rápida, apresentando oscilações em regime permanente, este facto deve-se aos maiores intervalos de tempo na leitura de velocidades proporcionando erros maiores e aos ganhos configurados para rotações mais elevadas. O controlo derivativo quando implementado atenua as oscilações no entanto aumenta o tempo de resposta para as rotações mais elevadas.

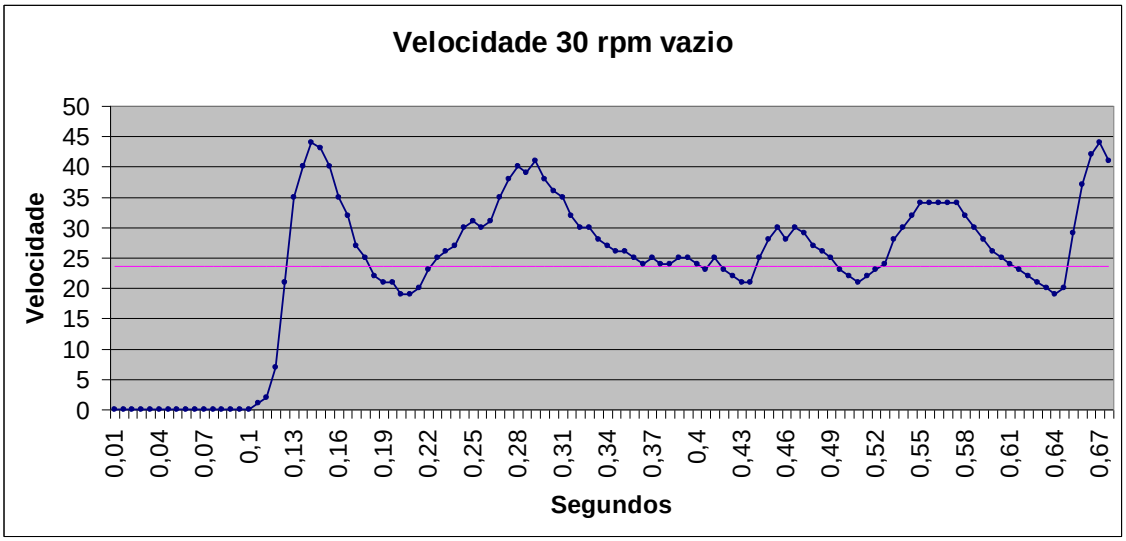


Fig. 106 – Resposta do sistema para uma velocidade de 30 RPM

Nas próximas figuras os tempos de resposta diminuem, existindo um pequeno *overshoot* que diminui com o aumento da velocidade desejada. Note-se que o tempo de resposta é tão mais rápido quanto a velocidade desejada. Este facto, deve-se à resposta do motor e à diminuição do tempo da leitura de velocidade.

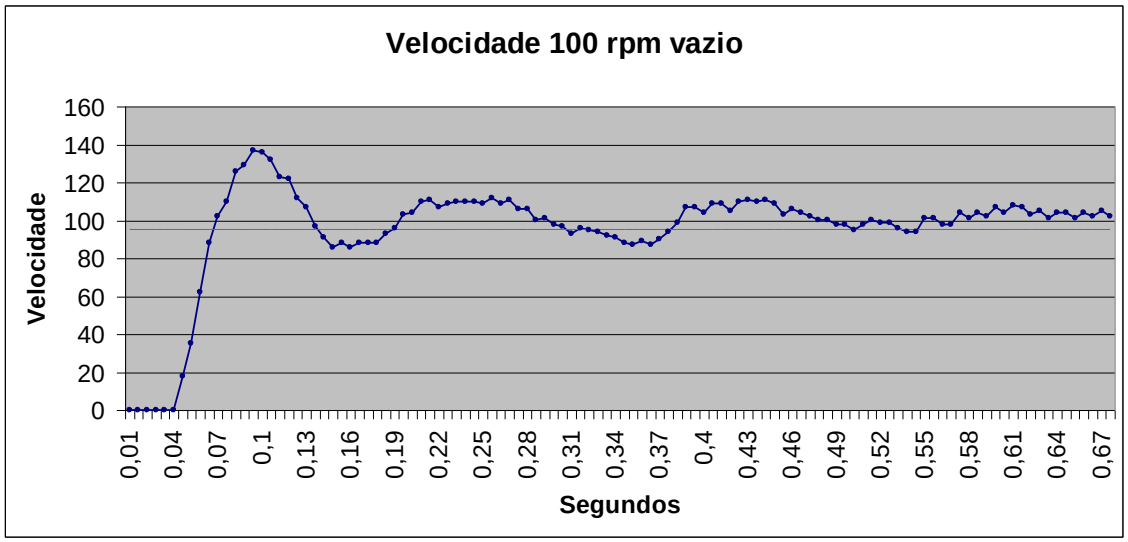


Fig. 107 – Resposta do sistema para uma velocidade de 100 RPM

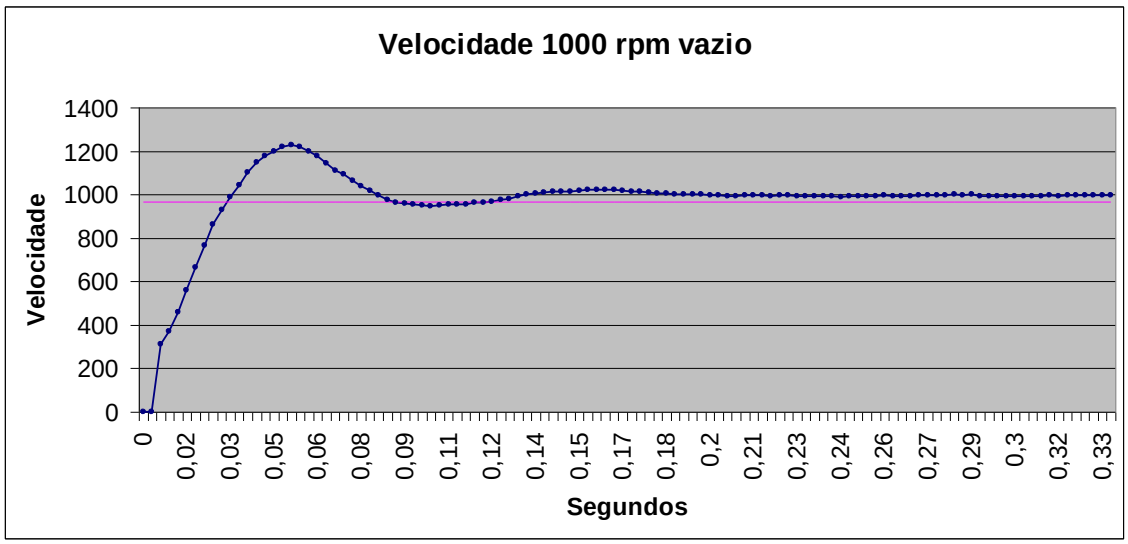


Fig. 108 – Resposta do sistema para uma velocidade de 1000 RPM

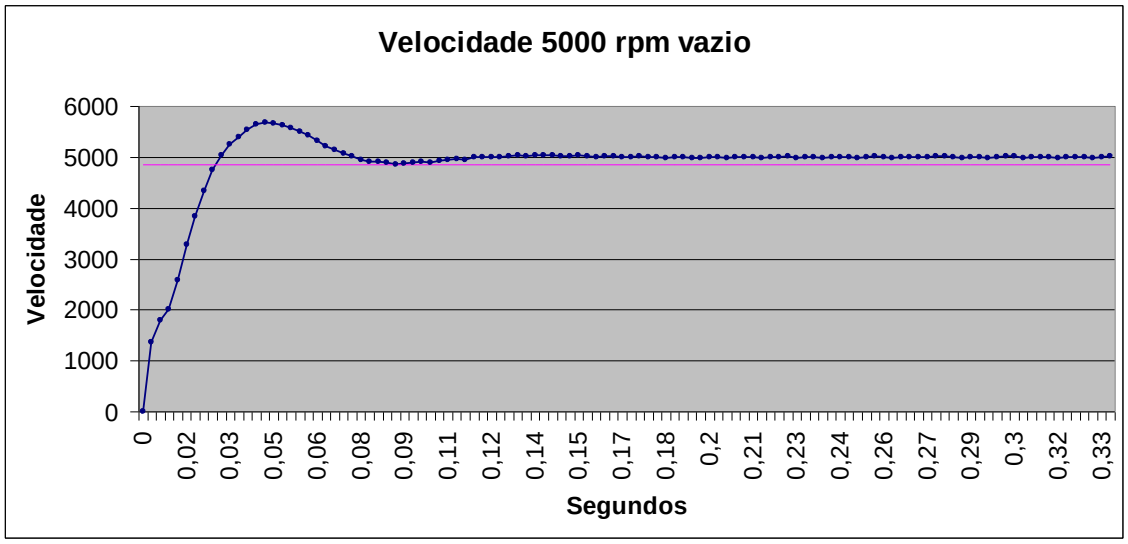


Fig. 109 – Resposta do sistema para uma velocidade de 5000 RPM

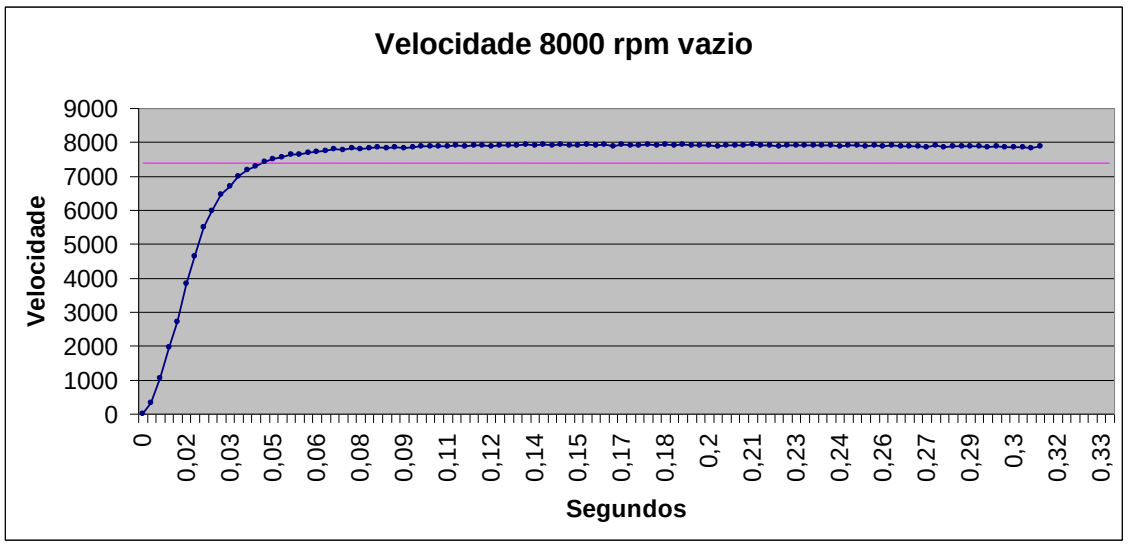


Fig. 110 – Resposta do sistema para uma velocidade de 8000 RPM

5.3.5 Conclusões PID

O controlo PID é o cérebro deste projecto, funcionando como um supervisor. Este lê, calcula e corrige a velocidade, de forma rápida, configurável e precisa.

Da implementação do controlo PID resultaram as seguintes conclusões:

- Em relação ao resultados obtidos conclui-se que quanto mais baixa é a velocidade desejada maiores são as oscilações em regime permanente, como a rapidez de leitura da velocidade não pode ser melhorada, resta a diminuição dos ganhos Kp e Ki e o aumento Kd.
- A solução ao problema anterior vai afectar as velocidades mais elevadas, sendo proposto como trabalho futuro o desenvolvimento de funções que calculem os ganhos Kp, Ki e Kd de forma dinâmica.
- As oscilações tendem a diminuir quando se aumenta a carga, a inércia do motor faz o efeito do ganho Kd, amortizando as oscilações.
- A utilização do algoritmo fornecido pela Microchip, como base de trabalho, proporcionou uma visão esclarecedora da implementação de um PID num microcontrolador, assim como as alterações necessárias para o projecto em causa, facilitando a aprendizagem e diminuindo o tempo de implementação.
- A separação dos bits de dados dos bits de sinal é baseada no algoritmo fornecido, proporcionado cálculos com 16 bits unicamente de dados, ficando os bits de sinal numa estrutura evitando o desaproveitamento de 7 bits em cada variável de 16 e 24 bits.
- O método utilizado para sintonização dos parâmetros não é o mais aconselhado, existindo outros métodos já referidos, no entanto como são apenas testes de funcionamento o método utilizado proporcionou um ajuste rápido e eficiente, cabendo ao utilizador final o ajuste (caso necessite) para o motor utilizado.

5.4 Conclusões

Neste capítulo foram descritos os principais módulos de software deste projecto. Estes tornaram possível o controlo de velocidade em malha fechada, proporcionando uma correcção da velocidade de forma precisa e rápida, para uma gama alargada de velocidades. Com a implementação destes módulos obtiveram-se as conclusões seguintes:

- Após cada leitura de tempo do nível lógico não deve ser gerada nenhuma interrupção, porque esta provoca atraso na leitura dos próximos tempos originando uma leitura incorrecta (tendo sido esta a primeira versão implementada).
- Com o método utilizado a leitura de velocidade varia de 1RPM a 6249RPM com precisão de 1 RPM, de 6250RPM a 8994RPM com precisão de 2 RPM e de 8995RPM a 10000RPM com uma precisão de 3 RPM.
- O encoder incremental foi a escolha correcta, devendo-se escolher um encoder com PPR que proporcione leituras rápidas a baixa rotação e precisão nas rotações elevadas.
- A baixas rotações o sistema apresenta uma resposta oscilatória, devendo-se este facto, às leituras de velocidade lentas em relação ao intervalo de execução do cálculo PID e ao ajuste dos ganhos para respostas de velocidade mais rápidas, nas velocidades mais elevadas. Propõe-se como trabalho futuro a implementação de funções que sintonizem os ganhos do PID de forma dinâmica, ajustando-se às altas e baixas rotações.
- Aplicando uma carga no motor as oscilações diminuem, devido à inércia do motor.
- Quando em carga é aconselhado utilizar rampas de aceleração e desaceleração de forma a evitar picos de corrente que poderão danificar o motor e a ponte H.
- Conclui-se por fim que o algoritmo implementado satisfaz os requisitos deste sistema apresentando respostas bastante rápidas e precisas.

Capítulo 6 - Extras

6.1 Introdução

Para a implementação e teste da placa controladora, desenvolveu-se algum hardware e software auxiliar.

Para tal foi desenvolvido um conversor RS232-I2C e respectivo software para PC, de forma a poder comunicar com a placa controladora. Posteriormente o software foi melhorado passando a comunicar através de um conversor USB-I2C, permitindo ainda guardar valores de corrente, tensão e temperaturas num ficheiro.

Estes módulos auxiliares são descritos neste capítulo sendo feita uma abordagem do ponto de vista funcional.

6.2 Conversor RS232 - I2C

Quando se implementou o barramento I2C surgiu a necessidade de o testar e posteriormente de o utilizar para configurações e controlo da placa controladora de velocidade. Essa necessidade levou ao desenvolvimento de um conversor RS232-I2C, Fig. 111, que converte os dados provenientes do protocolo RS232 existente na maioria dos PCs para o protocolo IC2 utilizado na placa controladora.

Para a implementação deste conversor foi utilizado um microcontrolador PIC16F873A da Microchip, que possui entre outras características, comunicação RS232 e I2C, permitindo assim a conversão de dados. O código implementado neste microcontrolador foi desenvolvido na linguagem C sendo compilado com o CCS C compiler V.4.

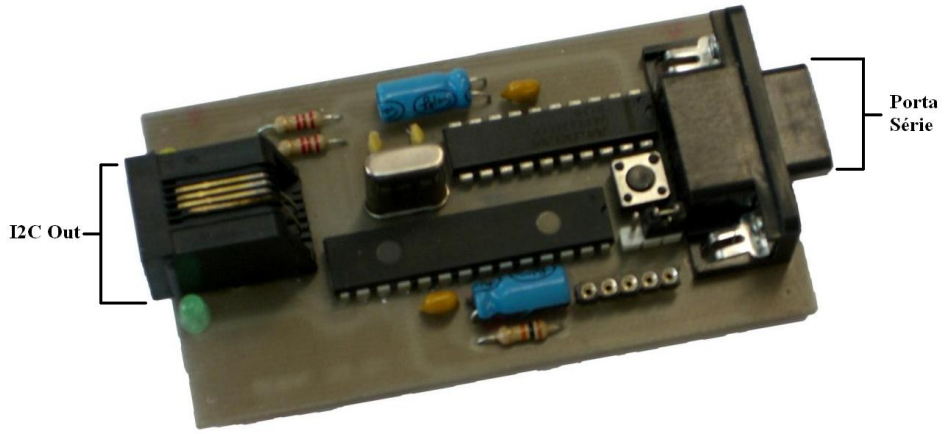


Fig. 111 – Imagem do conversor RS232-I2C

Para além do microcontrolador foi utilizado o circuito integrado MAX233, sendo este um conversor de nível que converte sinais TTL em RS232 e vice-versa.

O circuito do conversor está abaixo representado, estando no anexo V novamente representado juntamente com o PCB (*Printed Circuit Board*) desenvolvido na plataforma eagle versão 4.16r2.

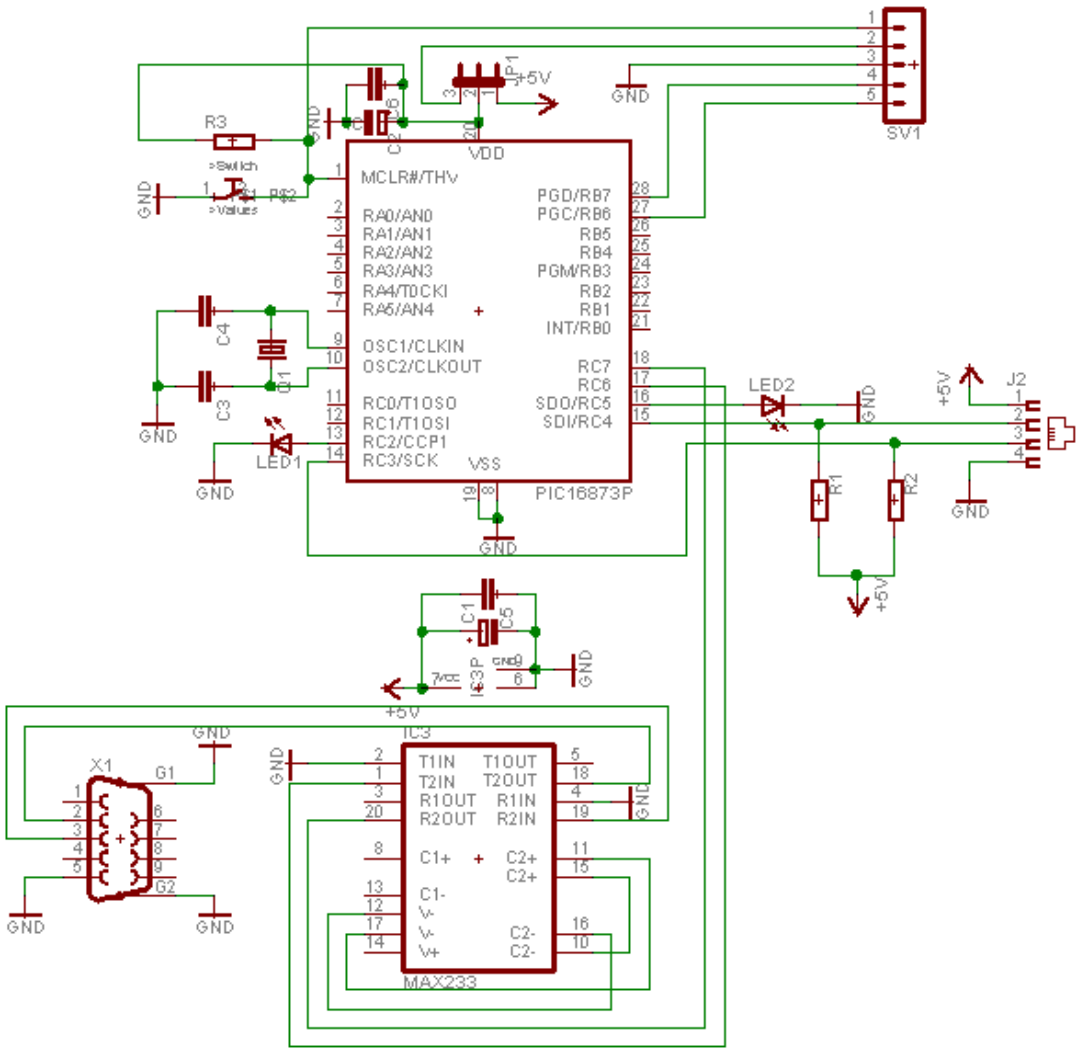


Fig. 112 – Circuito eléctrico do conversor RS232-I2C

Para comunicar com o conversor é necessário configurar a comunicação RS232 com os seguintes valores:

Baud rate → 19200 bps

Bits de dados → 8 bits

Paridade → sem paridade

Stop bits → 1 stop bit

Controlo de Fluxo → Sem controlo de Fluxo

De forma que o microcontrolador interprete as tramas enviadas através de RS-232 estas têm uma estrutura que permite o envio de 1 ou 2 bytes de dados e recepção de 2 bytes, estando representada abaixo a estrutura das mesmas.

Enviar 2 bytes:

Address Motor		R/$\overline{W}$		Address Register		Data MSB		Data LSB	
0 - 254	;	0	:	0 - 38	,	Byte 0	-	Byte 1	.

Enviar 1 byte:

Address Motor		R/$\overline{W}$		Address Register		Data	
0 - 254	;	0	:	0 - 38	,	Byte	+

Ler 2 bytes:

Envia

Address Motor		R/W		Address Register		Data	
0 - 254	;	1	:	0 - 38	,	Byte	.

Recebe

Data MSB		Data LSB	
Byte 0	;	Byte 1	.

Estas tramas são interpretadas no microcontrolador e enviadas através do barramento I2C para o controlador correspondente.

Para mais informações sobre pinos, tensões de alimentação, actualização ou modificação do software aconselha-se a consulta do manual do controlador, no anexo I.

No anexo III, está o código implementado no microcontrolador devidamente comentado.

6.3 Software Serial_to_I2C

Para enviar as tramas para o conversor foi criado um programa que envia a trama correcta em função do envio ou recepção pretendido. O programa Serial_to_I2C foi desenvolvido no compilador Microsoft Visual C++ 6, sendo utilizada a classe CSerialPort para efectuar a comunicação série, classe esta que foi desenvolvida pelo PJ Naughter [63].

A Fig. 113, representa o interface do programa sendo de seguida descritos os procedimentos de funcionamento.

Fig. 113 – Imagem do programa desenvolvido

Inicialmente deve-se configurar na parte superior do interface, a porta série com os valores descritos no conversor R232-I2C e a "Porta" correcta. Após a configuração é necessário abrir a ligação carregando em "Open", se não aparecer nenhuma mensagem a ligação foi aberta com sucesso, podendo-se agora configurar ou controlar a placa controladora de velocidade.

Na parte inferior do interface, existem várias *edit box* que permitem inserir e visualizar dados.

Para comunicar com a placa controladora de velocidade deve-se inserir o endereço da placa em "Dispositivo", a operação pretendida (escrita ou leitura) em "Acção", a posição do *array* de dados na qual se pretende escrever ou ler em "Endereço". Dependendo o "Valor 0" e "Valor 1" da acção pretendida, se for escrita de 1 byte, o "Valor 0" toma o valor do byte, devendo-se colocar no "Valor 1" o valor -1, caso se pretenda escrever 2 bytes o "Valor 1" toma o valor do segundo byte, caso se pretenda ler, os valores lidos são colocados nas *edit box* "Valor 0" e "Valor 1", para finalizar o processo de leitura ou escrita deve-se carrega no botão "Enviar".

Para facilitar o envio de velocidades desejadas e as leituras de velocidades reais, foram criadas duas *edit box* para o efeito, Assim para enviar uma velocidade desejada deve-se escolher a placa controladora, marcar a *check box* correspondente ao envio da velocidade desejada, inserir o valor da velocidade desejada (positivo para a direita, negativo para a esquerda) e carregar em "Enviar", para a leitura, o processo é igual mudando apenas a *check box* aparecendo a velocidade real na *edit box* "Velocidade Motor".

A velocidade a que a roda vai ou está a rodar, aparece na *edit box* "Velocidade Roda", esta *edit box* foi criada para poder comparar a velocidade da roda acoplada ao motor com a velocidade medida no tacómetro. A velocidade da roda é aproximadamente 15 vezes menor que a velocidade do motor devido à caixa redutora.

A última função do programa consiste em gerar rampas de aceleração e desaceleração testando a comunicação e resposta do motor, para tal usaram-se rampas pré-definidas sendo enviados dados a cada 20ms.

Todo o processo pode ser parado a qualquer momento carregando no botão "Parar".

6.4 Software Usb_to_I2C

A placa RS232-I2C limita o envio de dados a intervalos de 20ms, bloqueando em intervalos de tempo menores. Dependendo do número de placas controladoras utilizadas, a actualização das mesmas pode-se tornar lenta (3 placas, 60ms entre actualizações), a forma encontrada para resolver este problema consiste em utilizar um conversor USB-I2C igual ao da Fig. 114, este conversor permite a escrita ou leitura até 60 bytes de forma rápida e eficaz, podendo-se consultar as suas características na referência [64].

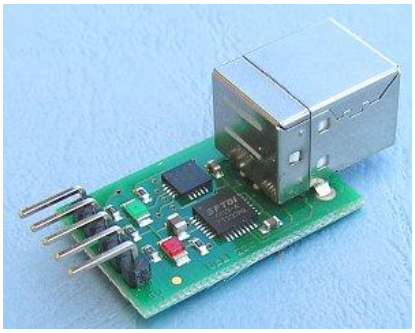


Fig. 114 – Conversor USB-I2C [64]

Para utilizar este conversor é necessário alterar o programa anterior, alterando as tramas que são enviadas de acordo com o manual do conversor [64]. O funcionamento do novo programa é o mesmo que o anterior, tendo-se acrescentado uma nova funcionalidade que permite guardar os valores da corrente, tensão e temperatura da placa controladora num ficheiro em intervalos de 1s.

Para testar a comunicação foram enviados comandos a cada 1ms tendo-se revelado muito eficaz.

As funções de envio e recepção deste programa encontram-se no anexo IV.

6.5 Conclusões

O conversor USB-I2C é mais compacto e eficaz que o conversor desenvolvido, permitindo comunicações mais rápidas.

O desenvolvimento do conversor RS232-I2C permitiu alargar o conhecimento na utilização de várias ferramentas e de um novo microcontrolador 16f873A, de família diferente à utilizada no projecto.

Como trabalho futuro é proposto melhorar a estrutura das tramas enviadas e organização do código de modo a aumentar a velocidade no conversor desenvolvido.

Capítulo 7 - Resultados

7.1 Introdução

A placa controladora de velocidade foi projectada de forma a cumprir, quer os requisitos mínimos quer os objectivos propostos. Este capítulo visa mostrar o resultado final obtido, circuitos finais, PCBs, resultados nos testes de potência e temperatura e outros resultados obtidos durante a implementação.

7.2 Circuito Final

A Fig. 115, representa uma das três placas controladoras de velocidade desenvolvidas. Cada placa é composta por um módulo de controlo e um módulo de potência estando unidos por um conector, facilitando assim a substituição de forma rápida de uma das partes.

O circuito final dos módulos de controlo e potência estão abaixo representados, assim como a lista de material utilizado na sua implementação.

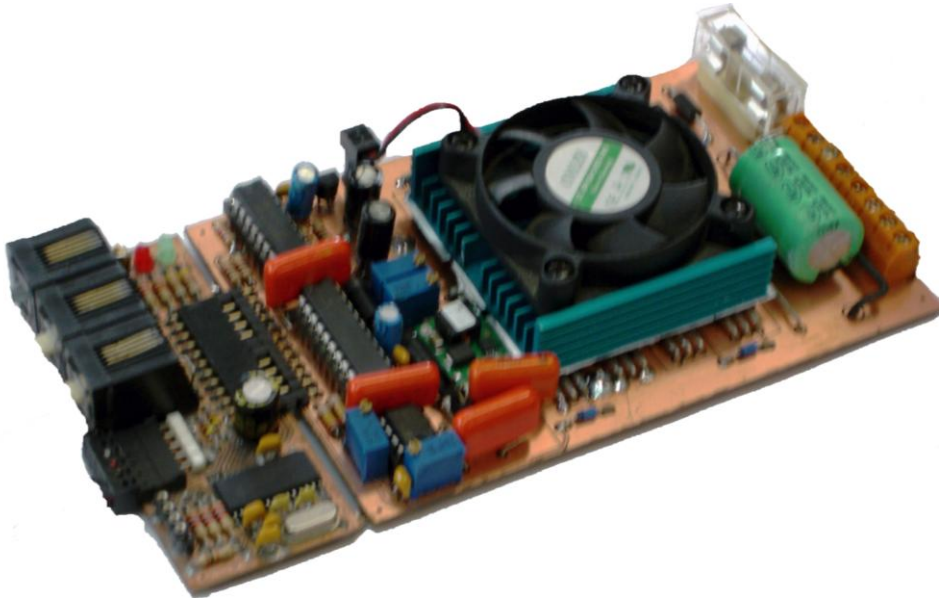


Fig. 115 – Placa controladora de velocidade desenvolvida

Circuito final do módulo de potência

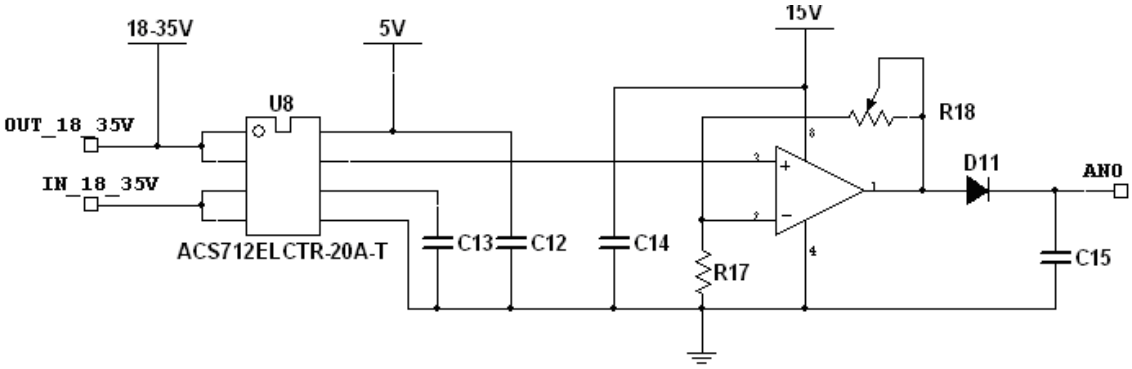


Fig. 116 – Circuito sensor de corrente

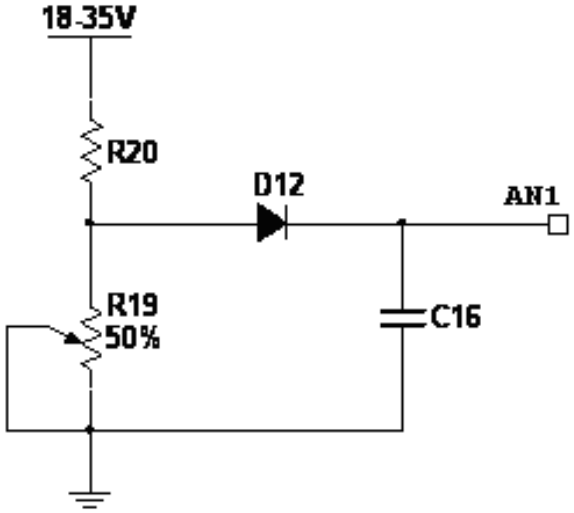


Fig. 117 – Circuito sensor de tensão

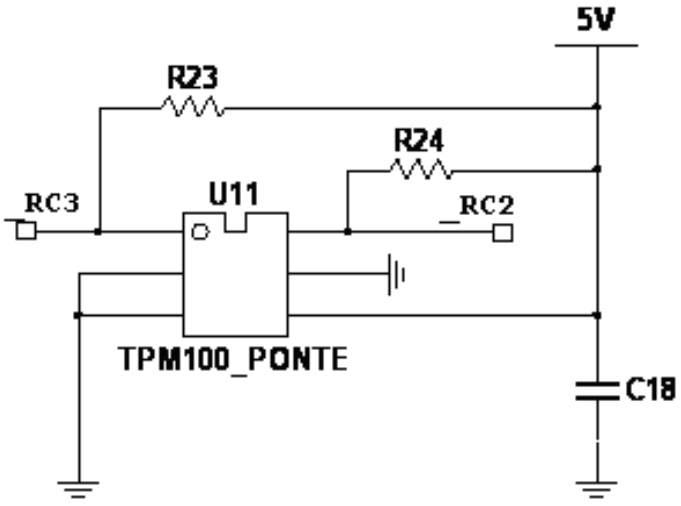


Fig. 118 – Circuito sensor de temperatura da ponte H

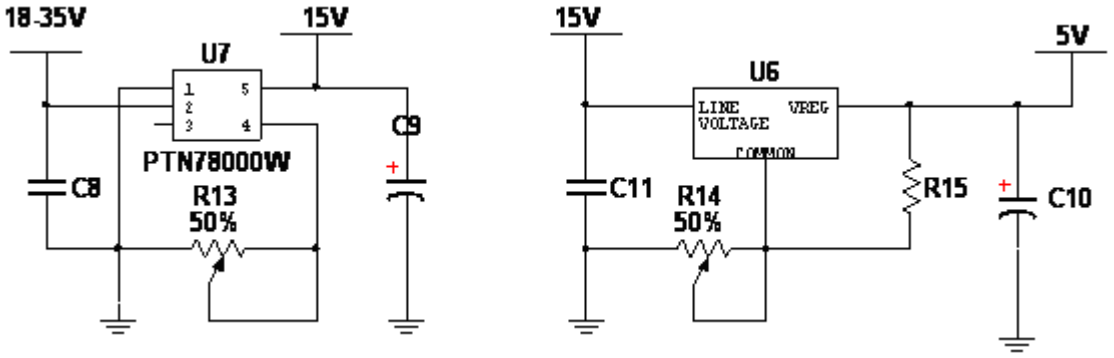


Fig. 119 – Circuitos reguladores de tensão para 15V e 5V

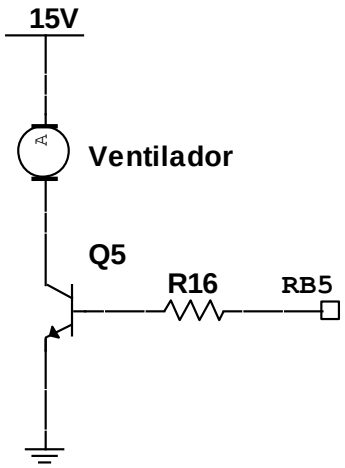


Fig. 120 – Circuito do ventilador do dissipador da ponte H

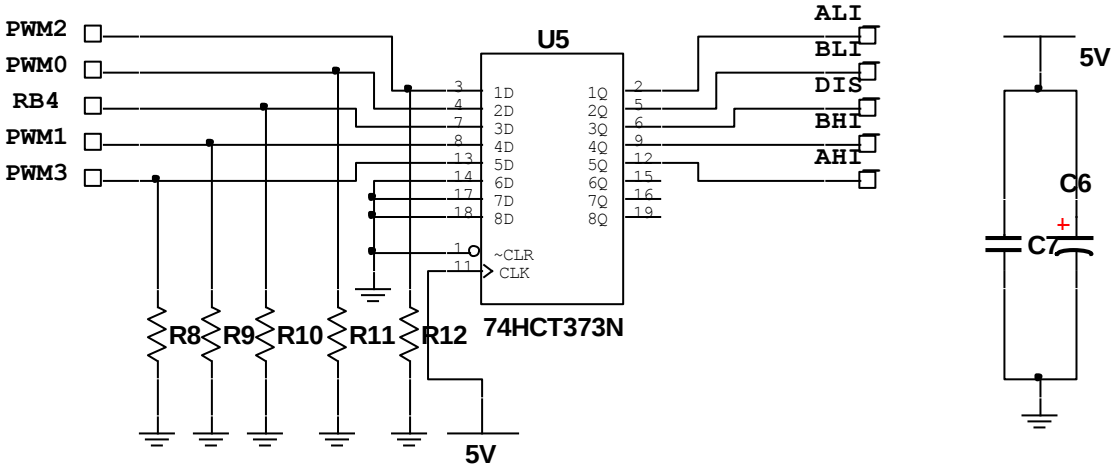


Fig. 121 – Circuito do buffer colocado entre o microcontrolador e o controlador HIP4081A

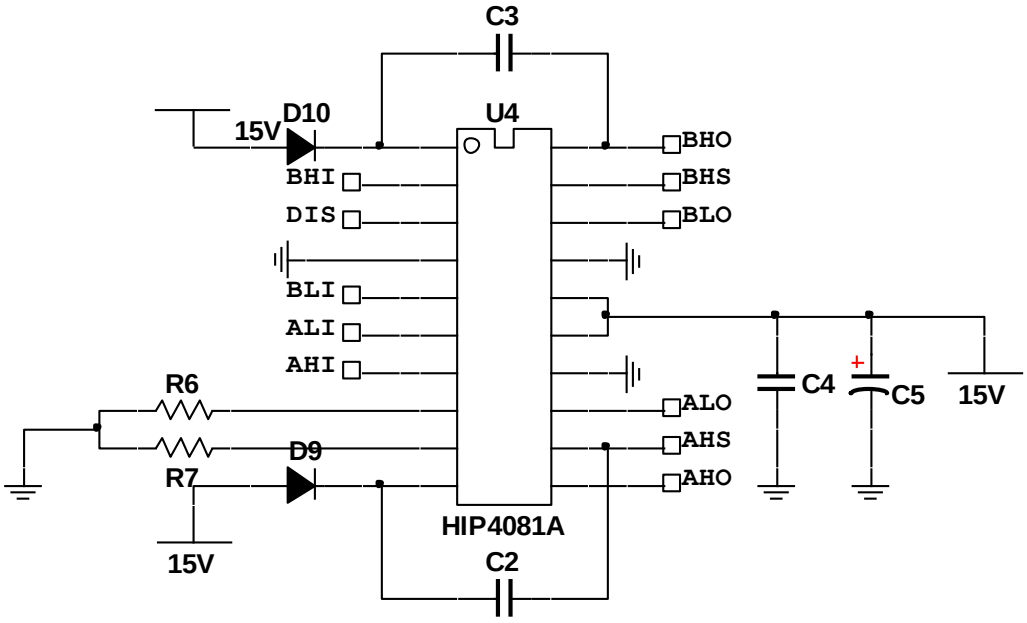


Fig. 122 – Circuito do controlador dos MOSFETs, HIP4081A

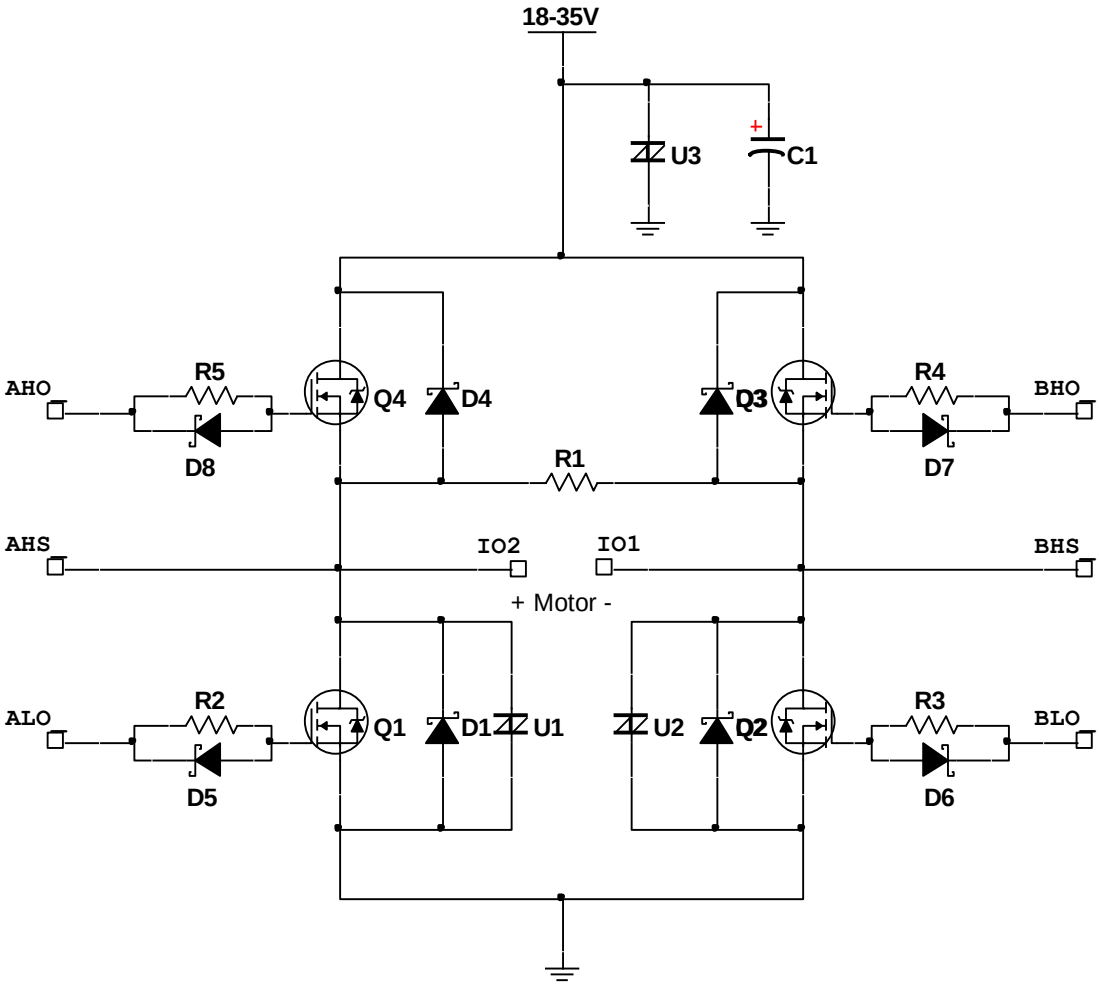


Fig. 123 – Circuito da ponte H

Circuito final do módulo de controlo

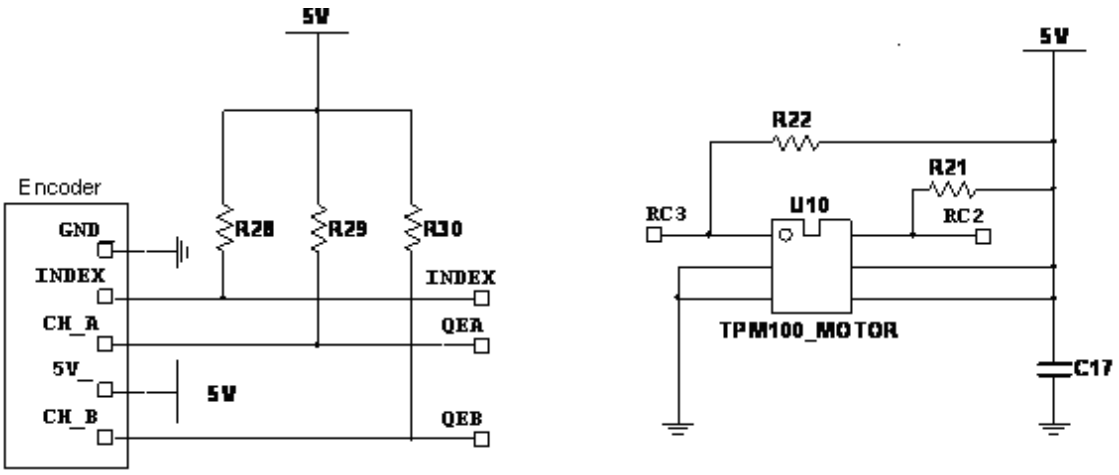


Fig. 124 – (a) Circuito do encoder óptico, (b) circuito do sensor de temperatura do motor

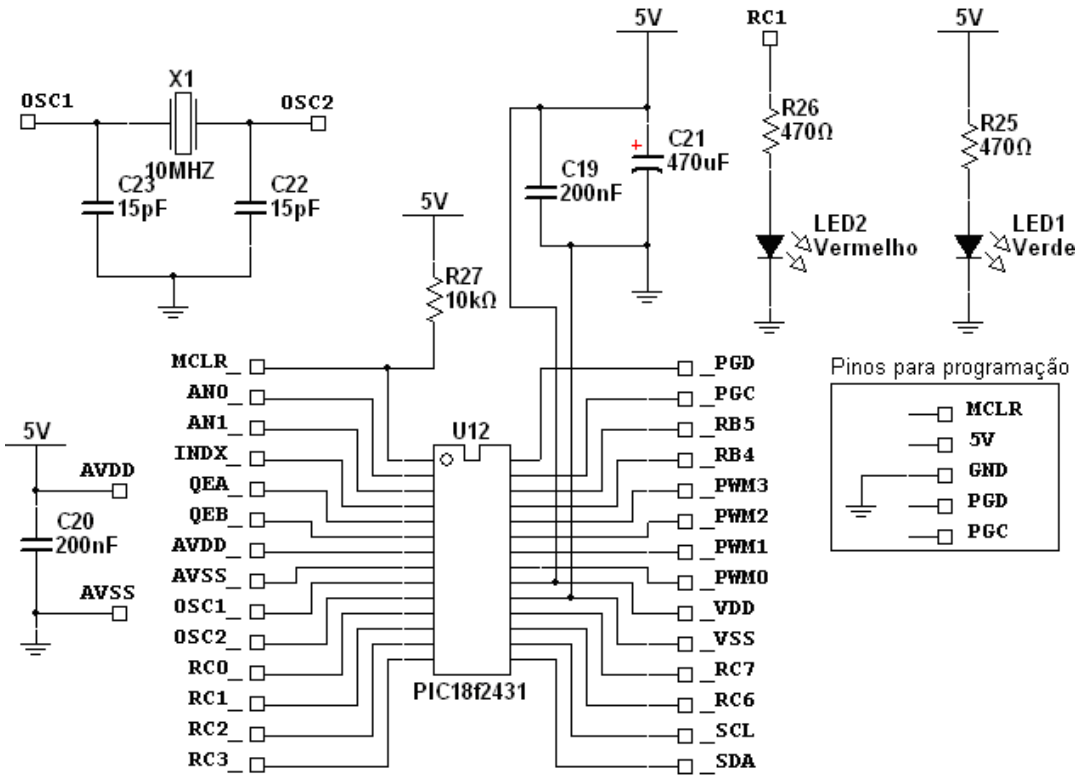


Fig. 125 – Circuito do microcontrolador PIC18F2431

Lista de Componentes

Quantity	Description	RefDes
4	SCHOTTKY_DIODE, STTH3002CT	D4, D3, D2, D1
4	SCHOTTKY_DIODE, 1N5711	D8, D7, D6, D5
2	DIODES UF4002	D10, D9
2	DIODE, 1N4148	D11, D12
1	CONTROLLER HIP4081A	U4
1	OPAMP, LM358A	U9
1	ACS712ELCTR-20A-T	U8
1	74HCT373N	U5
2	TMP100	U11, U10
1	PIC18f2431	U12
3	VOLTAGE_SUPPRESSOR, P6KE47CA	U3, U2, U1
4	POWER_MOS_N, stb140nf55	Q4, Q3, Q2, Q1
1	BJT_NPN, 2N2222A	Q5
1	Cristal 10MHz	X1
1	LED_green	LED1
1	LED_red	LED2
1	VOLTAGE_REGULATOR, LM117H	U6
1	STEP_DOWN, ptn78000w	U7
1	CAP_ELECTROLIT, 1500uF	C1
1	CAP_ELECTROLIT, 470uF	C21
2	CAP_ELECTROLIT, 100uF	C10, C9
2	CAP_ELECTROLIT, 10uF	C5, C6
4	CAPACITOR, 1uF	C13, C8, C3, C2
2	CAPACITOR, 200nF	C19, C20
9	CAPACITOR, 100nF	C14, C12, C15, C16, C18, C11, C7, C4, C17
2	CAPACITOR, 15pF	C23, C22
2	RESISTOR, 240kΩ	R7, R6
1	RESISTOR, 100kΩ	R20
3	RESISTOR, 10kΩ	R17, R27, R1
3	RESISTOR, 4.7kΩ	R30, R29, R28
4	RESISTOR, 2.2kΩ	R24, R23, R22, R21
6	RESISTOR, 1kΩ	R12, R11, R10, R9, R8, R16
2	RESISTOR, 470Ω	R25, R26
1	RESISTOR, 240Ω	R15
4	RESISTOR, 47Ω	R5, R4, R3, R2
2	POTENTIOMETER, 10kΩ	R18, R14
2	POTENTIOMETER, 50kΩ	R19, R13

Tabela VIII – Lista de Componentes

7.3 PCBs

Os PCBs foram desenvolvidos na plataforma Eagle 4.16r2, tendo-se desenhado 4 PCBs, dos quais 2 são de dupla face. Na projecção dos PCBs foi tida em atenção a dimensão das pistas condutoras para a potência em causa e a inserção de circuitos de protecção. Foi inserido um fusível de forma a proteger o sistema de curto-circuitos ou sobrecargas e um circuito que protege o sistema de possíveis inversões na tensão de alimentação.

PCB da placa de controlo

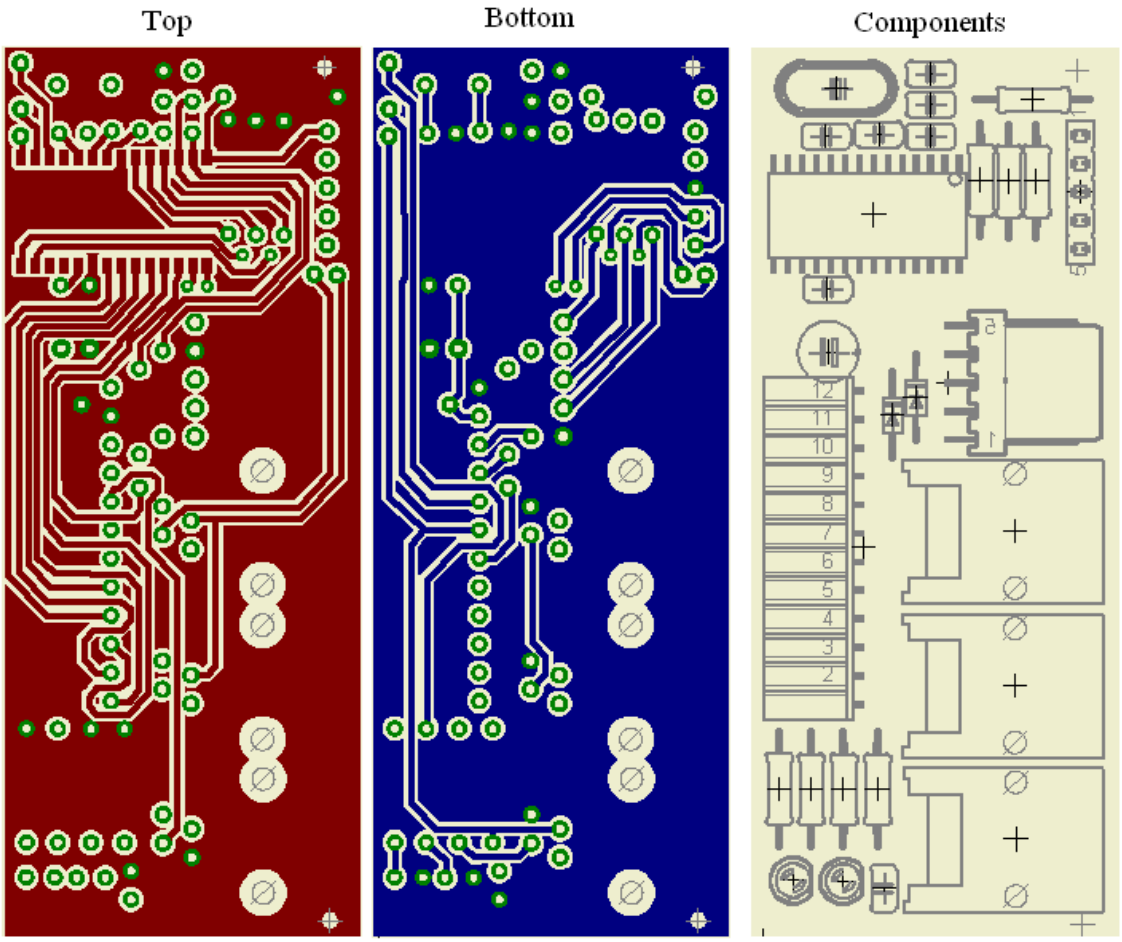


Fig. 126 – PCB da placa de controlo

PCB da placa de Potência

Top

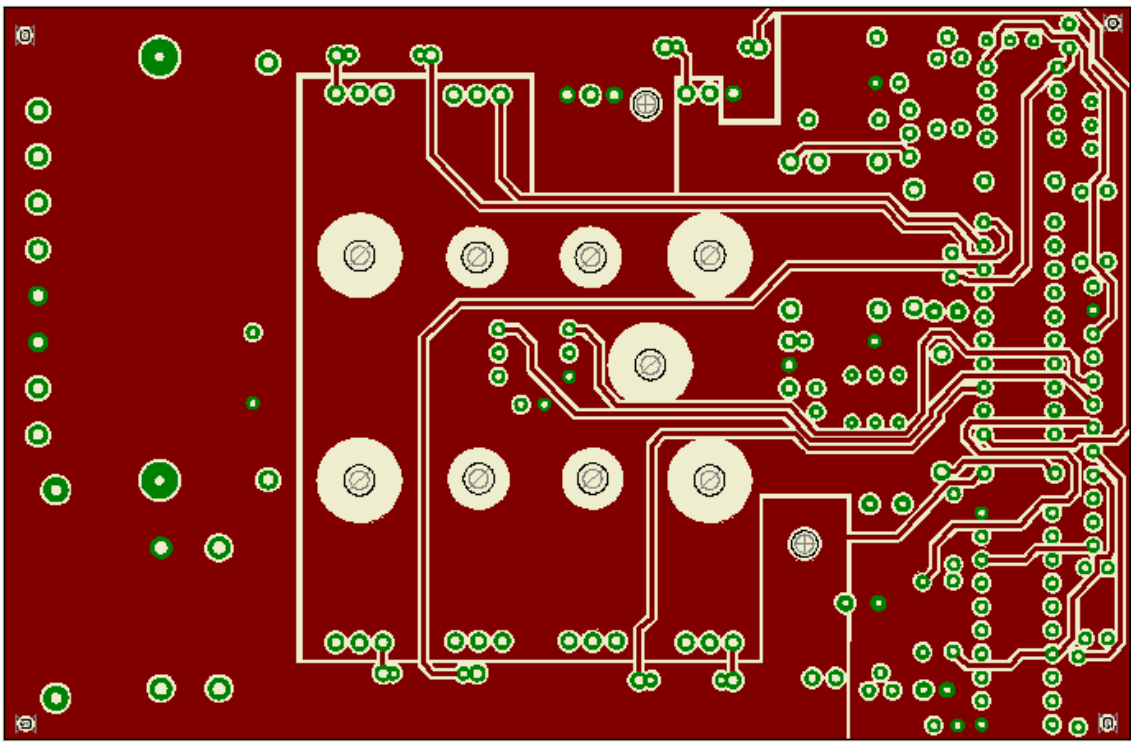


Fig. 127 – PCB da placa de potência, lado dos componentes (top)

Bottom

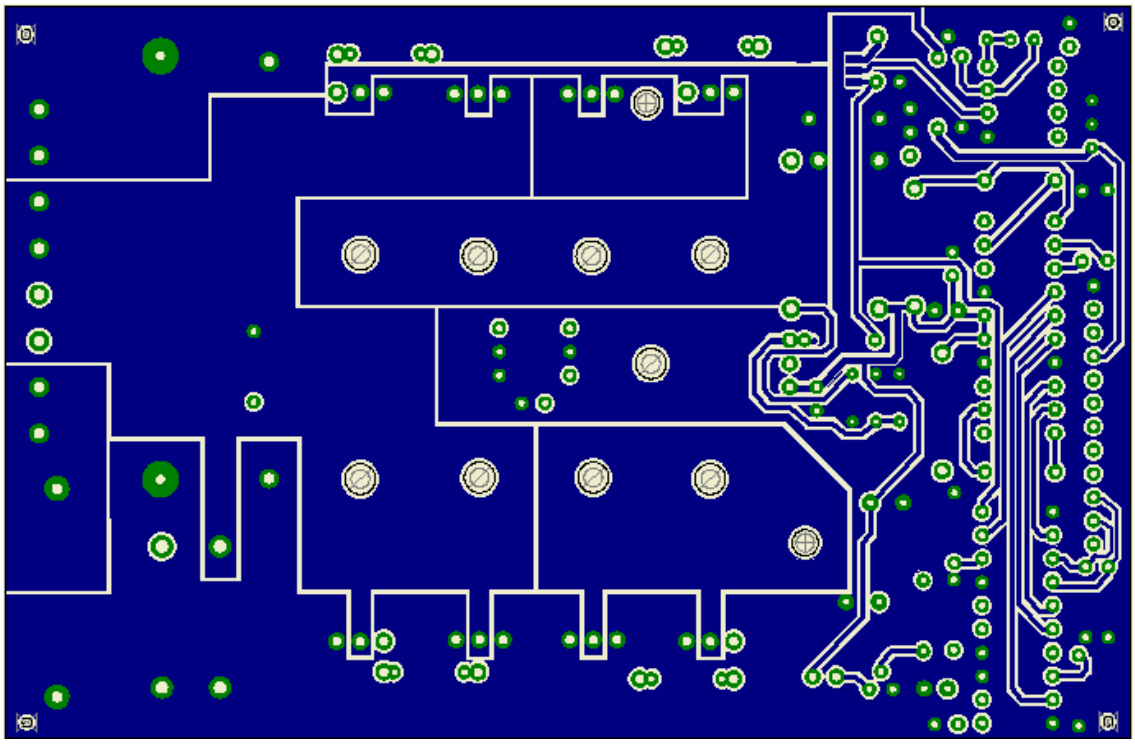


Fig. 128 - PCB da placa de potência, lado das soldas (botton)

Components

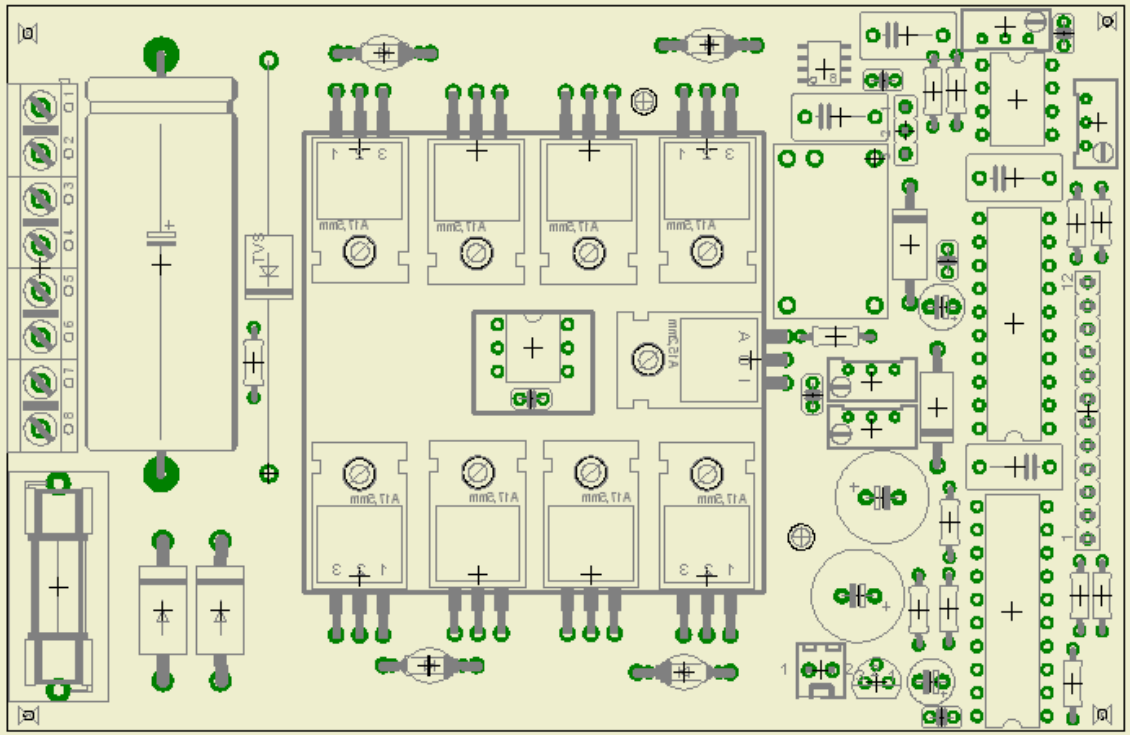


Fig. 129 – Componentes da placa de potência

PCB da placa do Sensor Temperatura TMP100 Ponte H

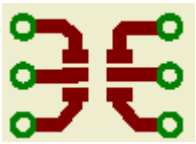


Fig. 130 – PCB da placa do sensor de temperatura da ponte H

PCB da placa do Sensor Temperatura TMP100 do Motor

Fig. 131 – PCB da placa do sensor de temperatura do motor

7.4 Testes

Todos os testes de funcionalidade foram executados ao longo da implementação: testes de resposta de velocidade em vazio, testes de comunicação, testes de configuração/controlo/telemetria. Neste subcapítulo são descritos alguns testes efectuados que complementam e comprovam os objectivos propostos. Estes testes são de potência, resposta em carga e temperatura.

Para efectuar os testes em carga foram utilizados 4 motores, Fig. 132. Um motor a qual está acoplado o encoder e se pretende controlar a velocidade, dois motores de 120W cada, que operam como geradores e funcionam de carga para um terceiro motor de 350W que por sua vez é a carga da ponte H, todo este sistema em conjunto proporciona uma carga superior a 500W.

Os testes efectuados consistem em medir a corrente, tensão e temperaturas em intervalos de 1 segundo para aumentos graduais de velocidade até à carga máxima.

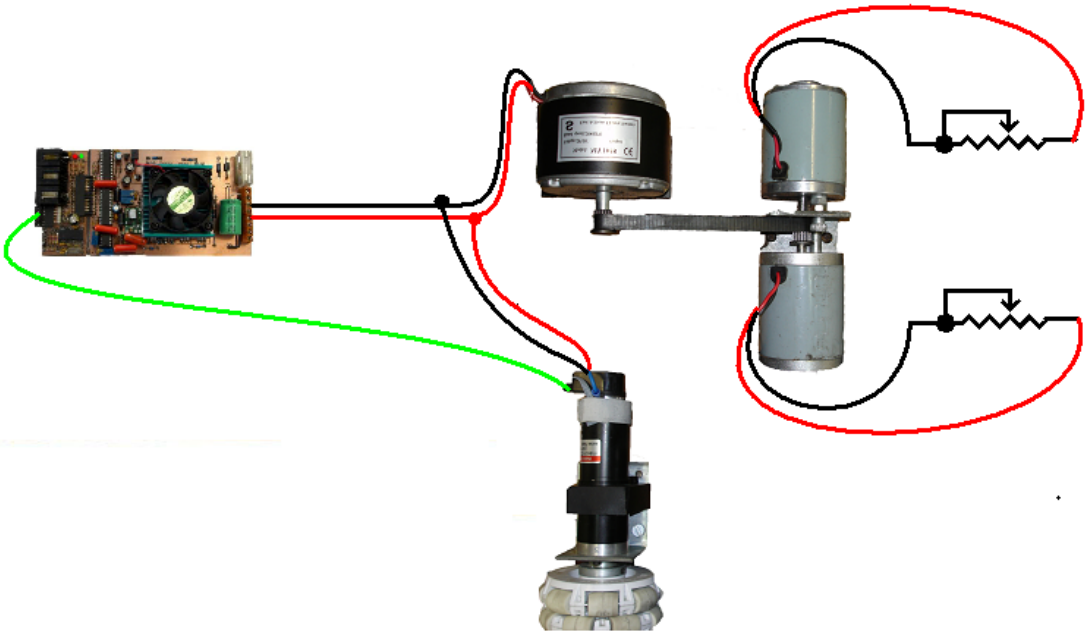


Fig. 132 – Motores utilizados para testar a ponte H

A Fig. 133, representa a resposta do sistema em carga, à variação da velocidade desejada. Note-se que este acompanha a velocidade desejada para os vários níveis de carga a que está sujeito até à potência nominal.

Quando a carga é menor existe um pequeno *overshoot*, que tende a desaparecer com o aumento da potência consumida, velocidade e inércia do motor.

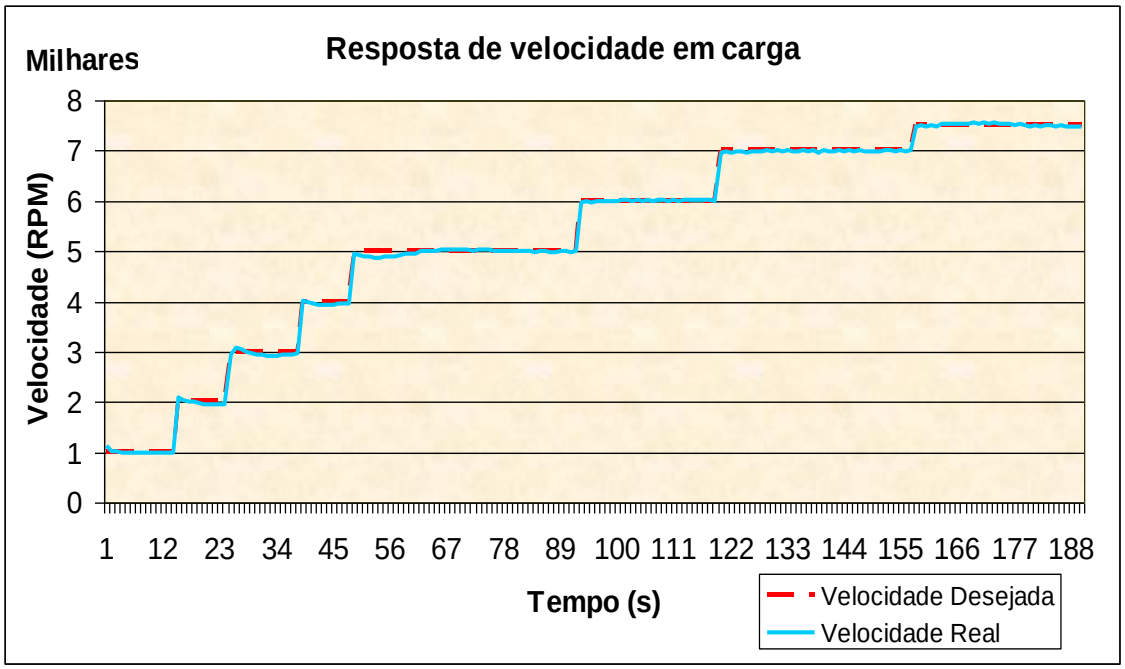


Fig. 133 – Gráfico que representa a resposta a uma velocidade desejada em intervalos de 1 segundo

As figuras seguintes demonstram a variação da corrente, tensão e potência fornecidas pela ponte H à carga, através das quais se pode verificar que a potência proposta é alcançada (500W), tendo ainda uma margem de segurança de 150W.

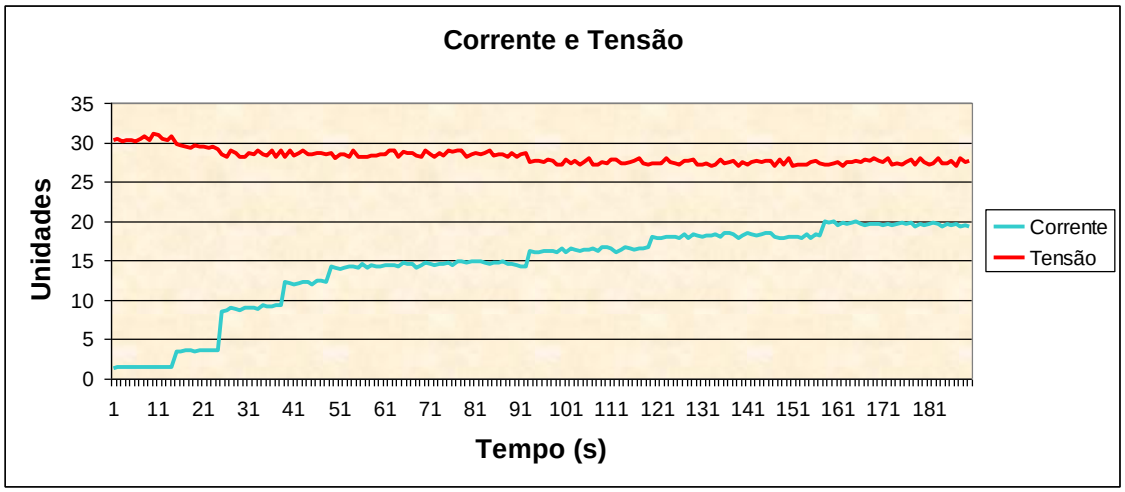


Fig. 134 – Gráfico da variação da corrente e tensão de alimentação, com o aumento da velocidade

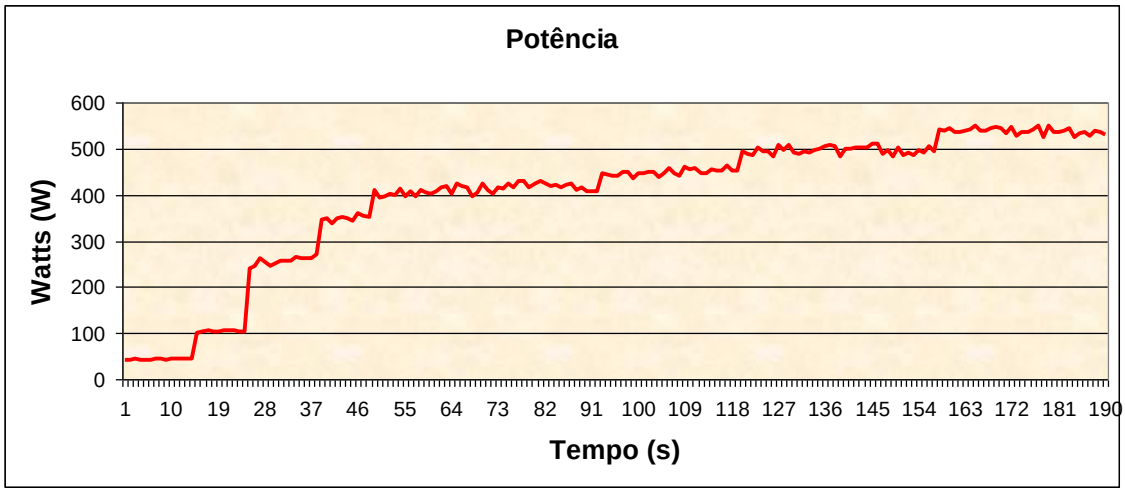


Fig. 135 – Gráfico da potência fornecida pelo controlador de velocidade, com o aumento da velocidade

As figuras seguintes representam os pulsos injectados nas *gates* dos MOSFETs e tensão aos terminais do motor, para as situações de 0, 25, 50, 75, 100% *duty cycle* com o motor em carga a rodar para a esquerda.

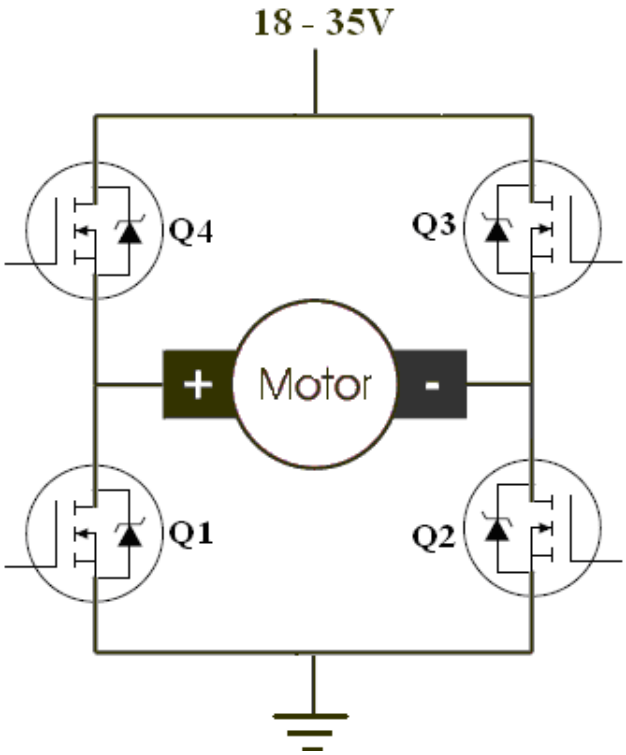
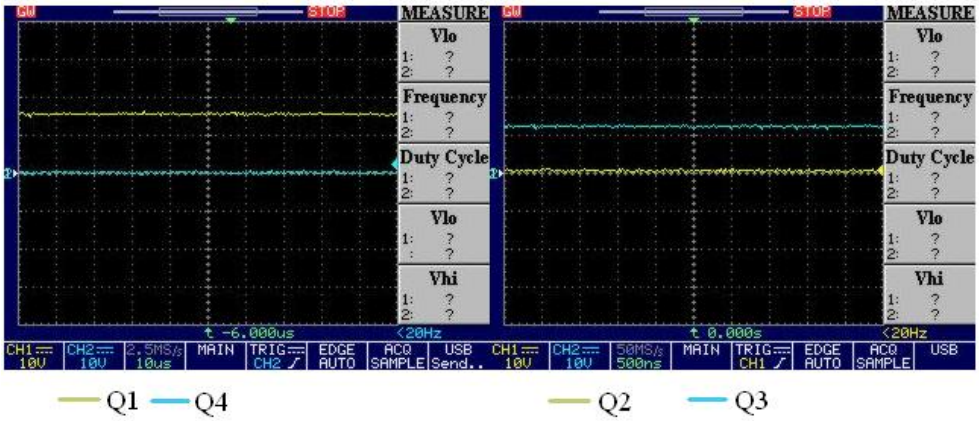
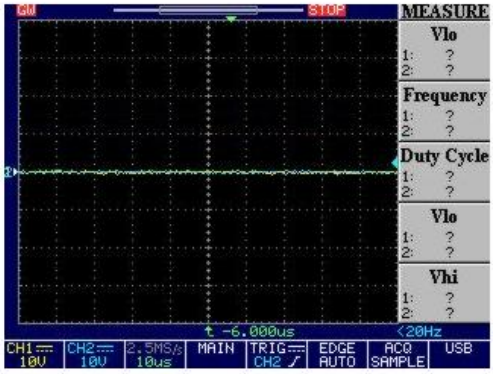


Fig. 136 – Ponte H com MOSFETs de canal N

Dutty Cycle = 0%

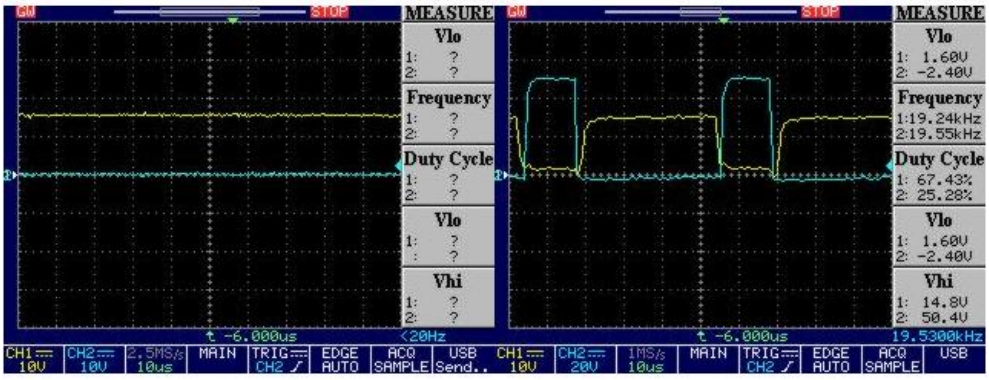


Q1 Q4 Q2 Q3

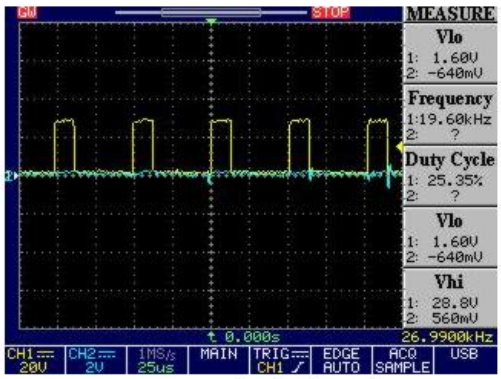


Tensão entre os terminais do motor e massa

Dutty Cycle = 25%



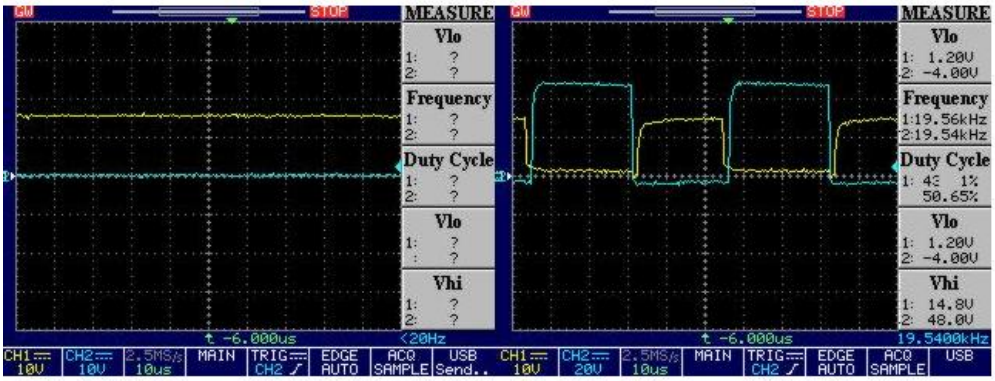
Q1 Q4 Q2 Q3

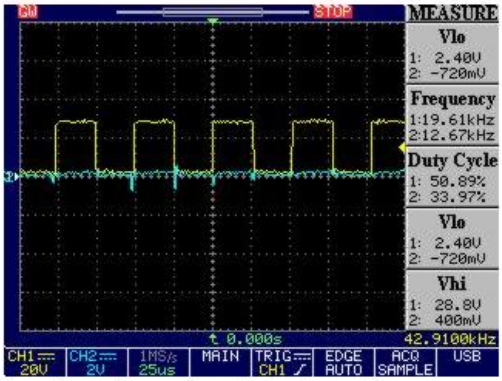


Tensão entre os terminais do motor e massa

Fig. 137 – Pulsos na ponte H e tensão no motor para 0 e 25% do *duty cycle*

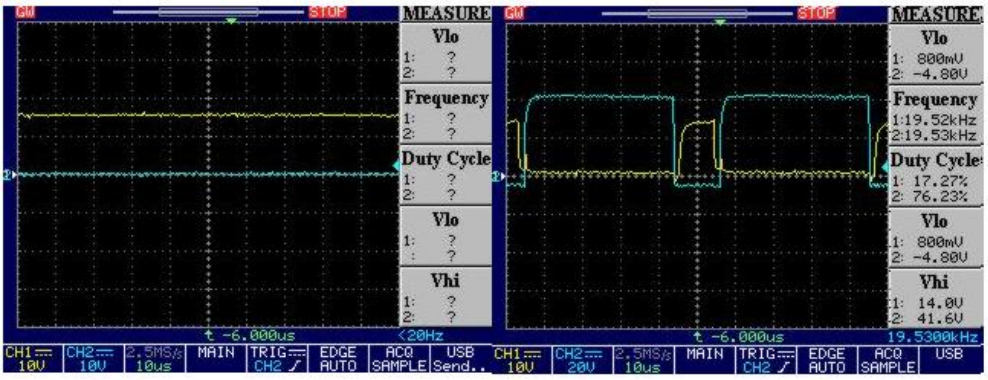
Dutty Cycle = 50%

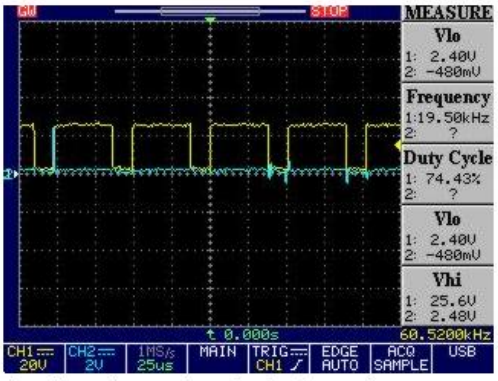




Tensão entre os terminais do motor e massa

Dutty Cycle = 75%





Tensão entre os terminais do motor e massa

Fig. 138 – Pulsos na ponte H e tensão no motor para 50 e 75% do *duty cycle*

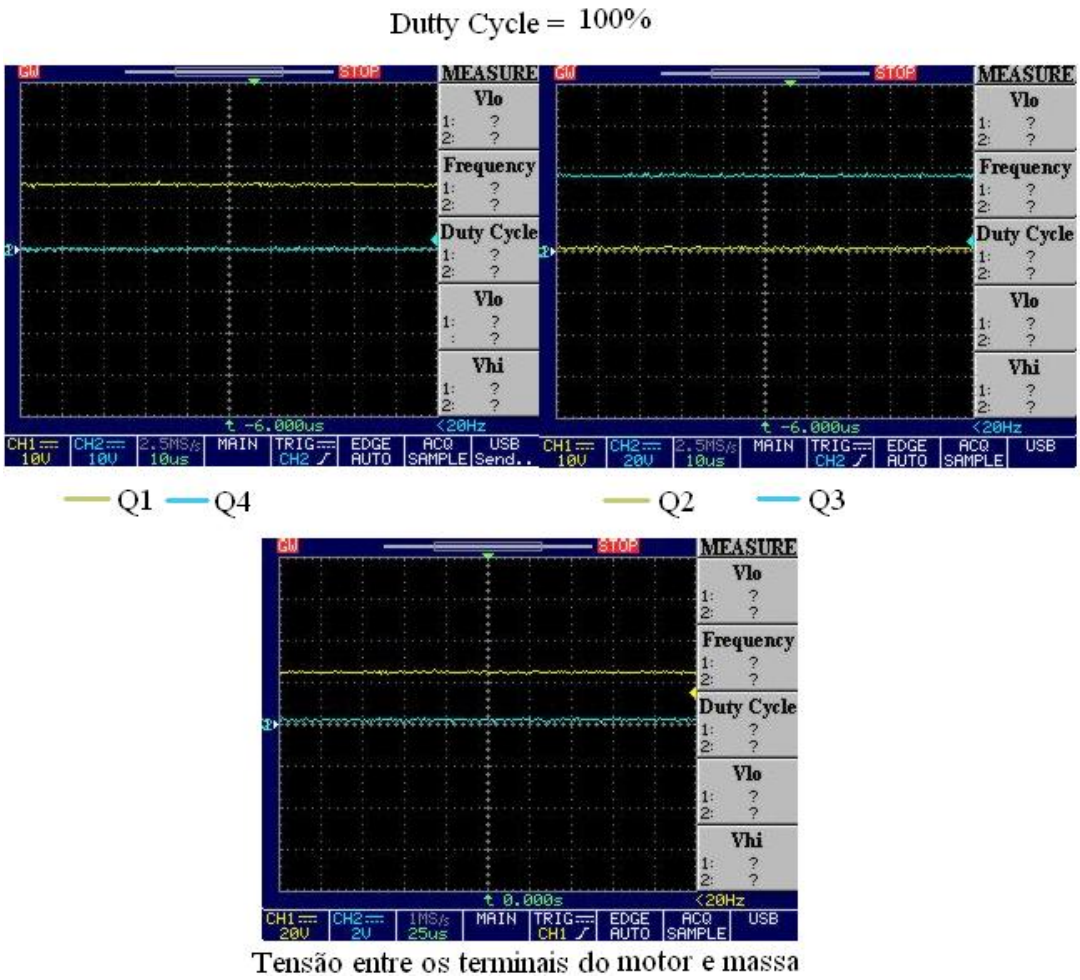


Fig. 139 – Pulsos na ponte H e tensão no motor para 100% do *duty cycle*

Através das figuras anteriores conclui-se que não existe a necessidade de implementar circuitos auxiliares de comutação (snubber), uma vez que o ruído existente devido à comutação é mínimo. Verifica-se também que a tensão aplicada no motor é uma onda quadrada quase perfeita não existindo distorção devido às indutâncias.

7.5 Conclusões

O sistema desenvolvido cumpre todos os requisitos mínimos e propostos.

Este possui uma alimentação que pode variar de 18 a 35V DC (requisito 24V), sendo capaz de accionar até 128 motores DC de ímanes permanentes (requisito 3 motores) com uma potência máxima de 500W (requisito 200W) nos dois sentidos.

Possui comunicação I2C que permite configurar 31 parâmetros, controlar e ler a velocidade, corrente, tensão, temperatura na ponte H e no motor reais. Faz a leitura do encoder óptico de modo rápido e preciso com descriminação do sentido de rotação, o seu PWM opera a 19,5 kHz não sendo audível.

Os seus sistemas de protecção configuráveis, colocam em standby a placa controladora de modo a proteger os seus componentes, o PCB também possui protecções caso as anteriores falhem.

A lei de controlo aplicada foi o PID, que permite que todos os seus parâmetros sejam ajustados através do barramento I2C. O controlo PID faz com que a velocidade desejada seja atingida de forma muito rápida independentemente das variações de carga. A placa controladora possui dois sistemas de travagem configuráveis, um livre em que o motor pára livremente e uma travagem forçada em que o motor é obrigado a parar através do controlo PID impedindo que este rode mesmo após a sua paragem.

Os componentes de maior custo são amostras, o que permitiu uma redução significativa no preço do protótipo, no entanto caso não o fosse o custo de fabrico também não seria elevado.

Em suma, todo o sistema se comporta de maneira eficaz cumprindo todos os requisitos, ficando como trabalho futuro, o fabrico e montagem dos PCBs de forma mais profissional, evitando problemas após montagem causados pela deficiência na PCB ou na montagem.

Capítulo 8 - Conclusões e Trabalho Futuro

8.1 Conclusões

O projecto do Controlador e Accionador de 3 motores DC em Malha Fechada foca o seu objectivo no controlo de velocidade de 3 motores DC (até 200W) através de um algoritmo de controlo em malha fechada que lhe permita rodar a uma velocidade pretendida com o mínimo erro possível. A solução proposta e desenvolvida consiste no controlo de um único motor (até 500W) por placa controladora, permitindo a agregação de até 128 placas por barramento I2c, alargando assim as aplicações possíveis e diminuindo o risco de perca de tracção total em caso de avaria. Os objectivos secundários passam pela comunicação I2C, telemetria de correntes, tensões e temperaturas sendo um sistema silencioso e de baixo custo. Na análise do mercado não foi encontrado nenhum sistema que cumprisse todos os requisitos pretendidos dando espaço para o desenvolvimento deste controlador de velocidade.

Do estudo ao motor DC salienta-se a variação da velocidade com a tensão de alimentação e sentido de rotação com o sentido da corrente nos enrolamentos, não esquecendo que o binário nominal deve ser aplicado à velocidade nominal, caso contrário a corrente é superior, podendo danificar o motor.

O uso da ponte H em conjunto com o gerador PWM traz vantagens relativamente a outros sistemas, permitindo o controlo de velocidade e direcção de forma estática e eficiência energética acima dos 90%.

O microcontrolador escolhido adequa-se perfeitamente aos módulos implementados, assim como a utilização da linguagem de programação C que diminui o tempo de programação, permitindo implementar as mesmas funções que linguagem assembler.

O interface dos dispositivos através do barramento I2C é muito acessível, tendo-se desenvolvido tramas de leitura e escrita que permitem configurar, controlar e telemetrizar o controlador de velocidade.

O uso de sensores de corrente, tensão e temperatura permite verificar constantemente o estado do controlador, permitindo que este seja colocado em standby caso haja algum problema, um LED indicará o problema ocorrido, voltando o sistema a funcionar quando a anomalia deixar de existir.

O método de leitura do encoder óptico incremental utilizado é muito eficaz e rápido, proporcionando precisões de 1 RPM entre 1RPM e 6249RPM, de 2 RPM de 6250RPM a 8994RPM e de 3 RPM entre 8995RPM E 10000RPM.

O algoritmo de controlo utilizado é o PID, proporcionando uma resposta rápida em situações de carga e vazio. Para muito baixas rotações, o sistema apresenta uma resposta oscilatória, devendo-se este facto, às leituras de velocidade lentas em relação ao intervalo de execução do cálculo PID e ao ajuste dos ganhos para respostas de velocidade mais rápidas nas velocidades mais elevadas, não influenciando este facto no desempenho real do sistema. Quando em carga é aconselhado utilizar rampas de aceleração e desaceleração de forma a evitar picos de corrente, que poderão danificar o motor e a ponte H. No controlo PID implementado é possível configurar ganhos, intervalos entre cálculos e limites através do barramento I2C.

O conversor RS232-I2C em conjunto com o software desenvolvido permite comunicar através do barramento I2C, com o controlador, no entanto, o conversor necessita de um intervalo de 20ms entre comunicações inviabilizando o uso deste para actualizações rápidas de várias placas controladoras. O conversor USB–I2C permite comunicações rápidas superando o conversor construído, tendo sido utilizado no teste de comunicação das várias placas, tendo-se modificado o software desenvolvido para PC para comunicar através do conversor USB–I2C.

O PCB final está dividido em duas partes, controlo e potência, estando ligadas através de um conector, facilitando a rápida substituição de uma das partes em caso de avaria. A projecção deste, teve em vista as correntes que nele circulam projectando-se as pistas de acordo com as mesmas, sendo necessário um cuidado especial na sua montagem, os componentes e pistas encontram-se muito próximos.

Em síntese, este controlador de velocidade com todas as suas características permite a aplicação nos casos em que é necessário o controlo de velocidade com precisão podendo-se utilizar de 1 até 128 motores DC por barramento I2C com potências até 500W.

Do ponto de vista do autor, para além da satisfação de concluir o projecto com todos os requisitos cumpridos, salienta-se o conhecimento adquirido nas várias áreas de electrónica desde um projecto proposto à implementação e teste do mesmo.

8.2 Trabalho Futuro

O projecto actual encontra-se a funcionar cumprindo os requisitos propostos, no entanto este pode ser melhorado tornando-o mais robusto sendo proposto como trabalho futuro os seguintes pontos:

- Execução de testes mais rigorosos, utilizar uma carga (de preferência motor) e uma fonte de alimentação que permitam a operação do controlador de velocidade com uma potência de 500W durante longos períodos de tempo.
- Ainda numa fase de testes, manusear a placa de forma descuidada de forma a encontrar pontos fracos (conectores, componentes) corrigindo-os tornando a placa mais resistente.
- Diminuir o espaço ocupado pelos componentes de potência (ponte H), substituindo por componentes SMD.
- Colocar a placa num invólucro de forma a protegê-la de meios adversos (ex. pó, humidade, condutores soltos).
- Construir o PCB de forma mais profissional, de modo que a soldadura não cause problemas.
- Elaborar funções que ajuste os ganhos do PID de forma dinâmica, proporcionando respostas quase ideais para baixas e altas rotações.
- Desenvolver funções de odometria sobre as funções básicas já criadas para o efeito.
- Optimizar o código do conversor RS232-I2C tornando-o uma solução viável.

Referências

[1] Antenando - Construir motor eléctrico em 30 segundos
http://www.antenando.com.br/tecnologia/arquivo/faca-um-motor-eletrico-em-30-segundos

[2] Escola de mecânica – Motor eléctrico
http://escolademecanica.wordpress.com/2007/09/08/motor-eletrico/

[3] Siemens – Motores de corrente contínua.
http://www.coep.ufrj.br/~toni/files/coe481/siemens_motores_cc.pdf

[4] Fraunhofer – Manual do controlador de velocidade TMC200
http://www.ais.fraunhofer.de/BE/volksbot/tmc-download/Firmware/ver1.16/doc/v1_17_TMC200HandbuchEnglish_.pdf

[5] VolksBot® Motor Controller VMC
http://www.volksbot.de/vmc.php

[6] Maxon Motor – Manual do controlador ADS 50/10
http://test.maxonmotor.com/docsx/Download/Product/Pdf/ads50-10_en.pdf

[7] Robot Electronics – Site do controlador MD03
http://www.robot-electronics.co.uk/htm/md03tech.htm

[8] Acroname - Manual da placa controladora S24-15A-30V
http://www.acroname.com/robotics/parts/S24-15A-30V-HBRIDGE.pdf

[9] Modular circuits – Site do controlador de velocidade µM-H-Bridge
http://www.modularcircuits.com/h-bridge.htm

[10] Robot shop – Site de venda do controlador de velocidade RoboteQ AX1500
http://www.robotshop.ca/home/suppliers/RoboteQ-en/roboteq-ax1500-robot-controller.html

[11] Acroname - Site do controlador MD22
http://www.acroname.com/robotics/parts/R220-MD22.html

[12] Robot Electronics - Site do controlador MD23
http://www.robot-electronics.co.uk/htm/md23tech.htm

[13] Robot Power – Site de informação do controlador simple-h
http://www.robotpower.com/products/simple-h_info.html

[14] Artigo sobre Motores de corrente contínua
Electrônica Básica Para Mecatrônica , Newton C. Braga, Editora Saber, Capitulo 2 "Motores de Corrente Contínua".

[15] Wikipedia – Motor de corrente contínua
http://pt.wikipedia.org/wiki/Motor_de_corrente_cont%C3%ADnua

[16] Magnet Lab – Modelo de funcionamento de um Moto DC
http://www.magnet.fsu.edu/education/tutorials/java/dcmotor/index.html

[17] Mechatronics Wiki - Brushed DC Motor Theory
http://hades.mech.northwestern.edu/wiki/index.php/Brushed_DC_Motor_Theory

[18] Maxon Motor – Maxon DC Motor
http://hades.mech.northwestern.edu/wiki/images/e/ee/Maxon_Motor_Guide.pdf

[19] Riobotz – Tuturial Rio Botz robots de combate
http://www.riobotz.com.br/Tutorial%20RioBotz.pdf

[20] Artigo sobre controlo linear de potência
Electrônica Básica Para Mecatrônica , Newton C. Braga, Editora Saber, Capítulo 5 "Controlo Lineares de Potência - Movimento".

[21] Artigo sobre controlo PWM de potência
Electrônica Básica Para Mecatrônica , Newton C. Braga, Editora Saber, Capítulo 6 "Controlo PWM de Potência".

[22] Rutronik – Circuito BTS 7960B que constitui meia ponte H
http://www.rutronik.com/index.php?id=59&L=0&tx_ttnews[tt_news]=45&tx_ttnews[backPid]=120&no_cache=1

[23] Infineon - Data sheet do circuito BTS 7960B que constitui meia ponte H
http://www.infineon.com/dgdl/BTS7960_Datasheet.pdf?folderId=db3a304412b407950112b408e8c90004&fileId=db3a304412b407950112b43945006d5d

[24] Imagem da ponte H, VNH2SP30
http://shop.embedit.de/gen_image.php?img=VNH2SP30.png&type=fv

[25] ST microelectronic's – Data sheet da ponte H VNH2SP30-E
http://www.st.com/stonline/products/literature/ds/10832.pdf

[26] ST microelectronic's – Data sheet da ponte H VNH3SP30-E
http://www.st.com/stonline/products/literature/ds/12688/vnh3sp30-e.pdf

[27] Sparkfun – Imagem da ponte H LMD 18201
http://www.sparkfun.com/commerce/product_info.php?products_id=747#

[28] National Semiconductor –Data sheet da ponte H LMD18201
http://cache.national.com/ds/LM/LMD18201.pdf

[29] ST microelectronic's – Data sheet da ponte H L298
http://www.st.com/stonline/products/literature/ds/1773.pdf

[30] M.S.kennedy - Data sheet da ponte H 4225
http://www.mskennedy.com/client_images/catalog19680/pages/files/4225rd.pdf

[31] Eletrônica de Potência - Cap. 1, J. A. Pomilio, Limites de operação de semicondutores de potência
http://www.dsce.fee.unicamp.br/~antenor/pdffiles/eltpot/cap1.pdf

[32] Modularcircuits – H-bridge secrets parts
http://www.modularcircuits.com/h-bridge_secrets1.htm

[33] Livro MOSFETs e Amplificadores Operacionais
MOSFETs e Amplificadores Operacionais: Teoria e Aplicações, José Gerardo Vieira da Rocha, Netmove Comunicação Global, Lda, Capítulo 2 "Transístores de efeito de Campo".

[34] ST microelectronic's – Data sheet do MOSFET stb140nf55
http://www.st.com/stonline/products/literature/ds/11028/stb140nf55.pdf

[35] Davi_Poyastro – Dimensionamento térmico
http://www.agarrados.com/w2box/data/MIEEC_EIND_TP2_Davi_Poyastro.pdf

[36] Aavidthermalloy – Imagem de um dissipador térmico
http://www.aavidthermalloy.com/products/microp/embed.shtml

[37] Electronica-pt – Teoria dos Díodos
http://www.electronica-pt.com/index.php/content/view/33/37/

[38] MSCP – Teoria dos Díodos
http://www.mspc.eng.br/eletrn/semic_210.shtml

[39] Electronica-pt –Teoria dos Condensadores
http://www.electronica-pt.com/index.php/content/view/31/37/

[40] Fransco – Imagem de um condensador electrolítico
http://www.fransco.com.br/imagem/capacitor_eletrolitico_amp.jpg

[41] Intersil – Data sheet do controlador da ponte H HIP4081A
http://association.arobas.free.fr/doc/HIP4081-an9325.pdf

[42] Philips – Data sheet do buffer 74HCT373
http://www.datasheetcatalog.org/datasheet/philips/74HCT373.pdf

[43] Texas instruments – Data sheet da fonte comutada ptn78000w
http://focus.ti.com/lit/ds/symlink/ptn78000w.pdf

[44] National – Data sheet do regulador de tensão LM117
http://cache.national.com/ds/LM/LM117.pdf

[45] Microchip – Família dos microcontroladores PIC
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2551

[46] Microchip – Data Sheet do microcontrolador PIC18f2431
http://ww1.microchip.com/downloads/en/DeviceDoc/39616C.pdf

[47] Artigo – "Os segredos do I2C"
Revista Elektor, edição portuguesa de Maio 2008, artigo "Os segredos do I2C", autor Etienne Boyer

[48] NPX - THE I 2C-BUS SPECIFICATION
http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

[49] Microchip - Using the PICmicro® SSP for Slave I²C™ Communication
http://ww1.microchip.com/downloads/en/AppNotes/00734a.pdf

[50] Farnell – Data sheet do sensor de corrente da Allegro ACS712ELCTR-20A-T
http://www.farnell.com/datasheets/94750.pdf

[51] Texas Instruments – Data sheet do sensor de temperatura TMP100
http://www.datasheetcatalog.org/datasheet/texasinstruments/tmp100.pdf

[52] Avago – Data sheet do encoder utilizado HEDS 5700-A11
http://www.avagotech.com/assets/downloadDocument.do?id=954

[53] kalipedia – Imagem ilustrativa dos componentes de um encoder incremental
http://www.kalipedia.com/kalipediamedia/ingenieria/media/200708/21/informatica/20070821klpinginf_48.Ees.SCO.png

[54] Mecatrônica Actual – Circuito para determinar sentido de rotação
http://www.mecatronicaatual.com.br/artigos/cnc/encoder01.htm

[55] Farnell –Imagem do encoder óptico utilizado
http://pt.farnell.com/productimages/farnell/standard/42420173.jpg

[56] Mecatrônica Actual – Funcionamento dos encoders ópticos
http://www.mecatronicaatual.com.br/artigos/cnc/encoder06.htm

[57] Hibrid Control System – Imagen do controlador Watt flyball
http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/ahak/report.html

[58] Artigo, "Controlo para automação Industrial"
Revista Saber Electrônica, edição 3, artigo "Controlo para automação industrial", autor Alexandre Capelli.

[59] Artigo Microchip – "Implementing a PID Conttroller Using a PIC18 MCU"
http://ww1.microchip.com/downloads/en/AppNotes/00937a.pdf

[60] Apontamentos – Controlo automático II
Apontamentos da disciplina Controlo Automático II, Dep. Electrónica Industrial, Universidade do Minho , acetatos "Leis de Controlo", autora Estela Bicho Erlhagen.

[61] Wikipedia – Imagem do diagrama de blocos de um controlador PID
http://en.wikipedia.org/wiki/PID_controller

[62] Microchip – PID source code
http://ww1.microchip.com/downloads/en/AppNotes/PID_source%20_code.zip

[63] Naughter – Biblioteca de comunicação série para Visual C++
http://www.naughter.com

[64] Robot-italy – Imagem do conversor USB- I2C utilizado
http://www.robot-italy.com/product_info.php?cPath=13_43&products_id=672

Bibliografia

Livro
Electrônica Básica Para Mecatrônica , Newton C. Braga, Editora Saber

Livro
MOSFETs e Amplificadores Operacionais: Teoria e Aplicações, José Gerardo Rocha, Netmove Comunicação Global, Lda

Livro
Princípios da Electrónica 2 , 6ª Edição, Albert Paul Malvino, McGraw-Hill

Livro
Linguagem C, Luís Damas, FCA

Livro
Tecnologias, António Pinto, Vítor Alves, Porto Editora

Livro
Power Electronics: Converters, Applications, and Design, Second Edition; Ned Mohan, Tore M. Undeland, William P. Robbins; John Wiley & Sons, Inc

Livro
Engenharia de Controle Moderno 2, 3ª Edition; Katsuhiko Ogata;Pearson Educación

Artigo
Revista Elektor, edição portuguesa de Maio 2008, artigo "Os segredos do I2C", autor Etienne Boyer

Artigo
Revista Saber Electrônica, Fevereiro de 2004, artigo "Controlo para automação industrial", autor Alexandre Capelli.

Artigo
Revista Saber Electrónica; Janeiro de 2008; Reference Design "Controle de motor DC com Escovas"

Artigo
Futebol robótico; António Fernando Ribeiro; Departamento Autónomo de Arquitectura da Universidade do Minho
http://repositorium.sdum.uminho.pt/bitstream/1822/3284/1/44%20artigo%20para%20arquitectura.pdf

Artigo
Microchip - AN899; "Brushless DC Motor Control Using PIC18FXX31 MCUs"
http://ww1.microchip.com/downloads/en/AppNotes/00899a.pdf

Artigo
Microchip – AN937; "Implementing a PID Controller Using a PIC18 MCU"
http://ww1.microchip.com/downloads/en/AppNotes/00937a.pdf

Artigo
Microchip - AN734; "Using the PICmicro® SSP for Slave I2CTM Communication"
http://ww1.microchip.com/downloads/en/AppNotes/00734b.pdf

Artigo

"DC Motor With Inertia Disk Model Development, Proportional Controller, and State Feedback Controller With Full State Estimator" Erick L.Oberstar

Manual
Maxon Motor – "maxon DC motor and maxon EC motor"
http://test.maxonmotor.com/docsx/Download/catalog_2008/Pdf/08_wichgtiges_dc_ec_motoren_e.pdf

Manual
Philips Semiconductors – "THE I 2C-BUS SPECIFICATION"
http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

Manual
Microchip – PIC16f87XA
http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf

Manual
Microchip – PIC18f2331/ 2431/4331/4431 Data Sheet
http://ww1.microchip.com/downloads/en/DeviceDoc/39616C.pdf

Manual
Intersil - HIP4081, 80V High Frequency H-Bridge Driver; Data Sheet
http://association.arobas.free.fr/doc/HIP4081-an9325.pdf

Manual
Allegro - Sensor de Corrente ACS712ELCTR-20A-T-Data Sheet
http://www.farnell.com/datasheets/94750.pdf

Manual
Texas Instruments – Fonte Comutada PTN78000W – Data Sheet
http://focus.ti.com/lit/ds/symlink/ptn78000w.pdf

Manual
Microchip -PICkit. 2 Programmer/Debugger User.s Guide
http://ww1.microchip.com/downloads/en/DeviceDoc/51553E.pdf

Manual
Philips – Buffer 74HCT373 Data Sheet
http://www.datasheetcatalog.org/datasheet/philips/74HCT373.pdf

Manual
Microchip - MPLAB® C18 C COMPILER GETTING STARTED
http://kevin.org/frc/C18_3.0_getting_started.pdf

Manual
Microchip - MPLAB® C18 C COMPILER USER'S GUIDE
http://ww1.microchip.com/downloads/en/DeviceDoc/C18_User_Guide_51288j.pdf

Manual
Microchip - MPLAB® C18 C COMPILER LIBRARIES
http://elth.ucv.ro/ccrc/doc/MPLAB_C18_Libraries_51297f.pdf

Apontamentos
Máquinas Eléctricas – Máquinas CC- Faculdade de Tecnologia Álvares de Azevedo
http://www.faatesp.edu.br/publicacoes/maquina_CC.pdf

Apontamentos
Apontamentos da disciplina de Máquinas Eléctricas – Universidade do Minho ; João Luiz Afonso

Apontamentos

Apontamentos da disciplina de Controlo Digital – Universidade do Minho; Estela Bicho

Site
Cuidados a ter na construção de uma ponte H; "H-bridge secrets"
http://www.modularcircuits.com/h-bridge_secrets1.htm

Site
Controladores de Velocidade "Speed Controllers"
http://homepages.which.net/~paul.hills/SpeedControl/SpeedControllersBody.html

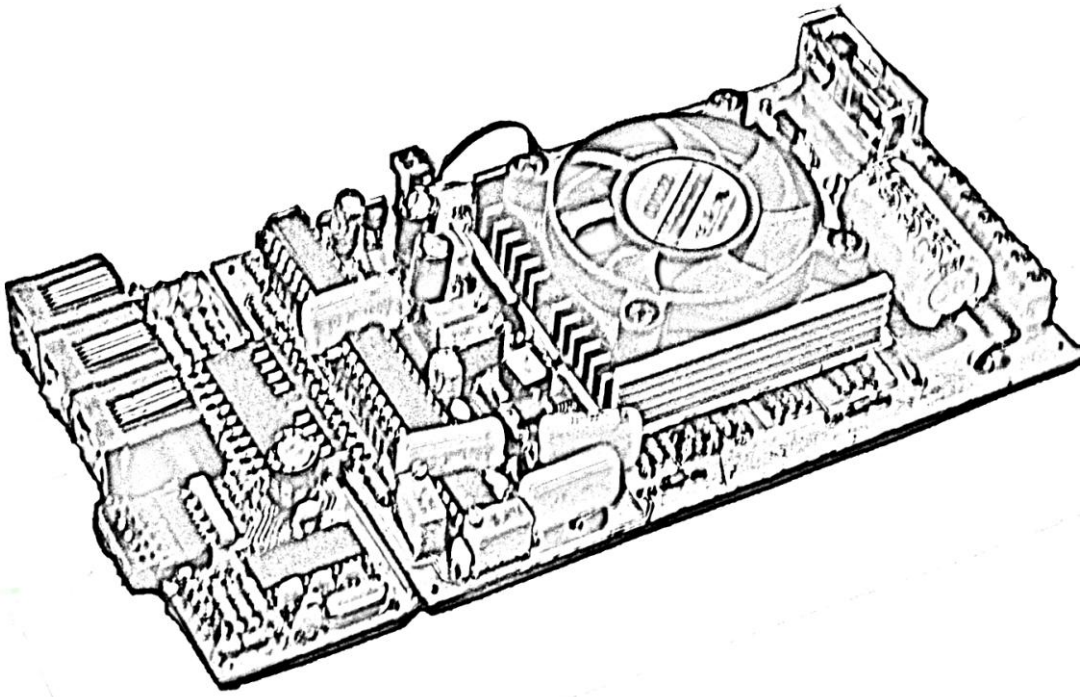
Site
Aplicação de um controlo PID; "Basic DC Motor Speed PID Control With The Infineon C167 Family"
http://www.hitex.co.uk/c166/pidex.html

Site
Ponte H; "Voltage Spikes in FET based H-bridges"; Autor: Chuck McManis
http://www.mcmanis.com/chuck/robotics/projects/esc2/hbridge_spiking.html

PCM-500

Universidade do Minho
Escola de Engenharia
Departamento de Electrónica Industrial

Placa controladora para motor DC



Manual de Utilização

1 Especificações Gerais

A estrutura de comunicação em barramento faz do PCM-500 um controlador de velocidade com inúmeras aplicações, permitindo a agregação desde 1 até 128 controladores por barramento. Cada PCM-500 controla 1 motor DC (*direct current*) de potência até 500W, tornando-se extensiva a aplicação em variadíssimas áreas, desde a robótica móvel a máquinas industriais. Sensores de temperatura, corrente e tensão garantem que o controlador e motor operem em condições nominais, caso contrário o motor é desligado e um sinal visual indica qual a condição que foi excedida. Todo o processo digital é feito com o uso de um microcontrolador de 8 bits que garante fiabilidade e desempenho.

1.1 Características

Com o controlador PCM-500 podem-se realizar, entre outras, as seguintes funções:

- Controlar a velocidade desde baixas a altas rotações nos dois sentidos, garantindo o binário necessário à carga.
- Reposta rápida a variações de carga mantendo a velocidade desejada.
- Monitorizar e limitar as correntes de forma que motor não exceda a corrente nominal.
- Monitorizar e limitar a tensão de alimentação.
- Monitorizar e limitar a temperatura do controlador e do motor.
- Configurar 31 parâmetros de funcionamento de forma digital.

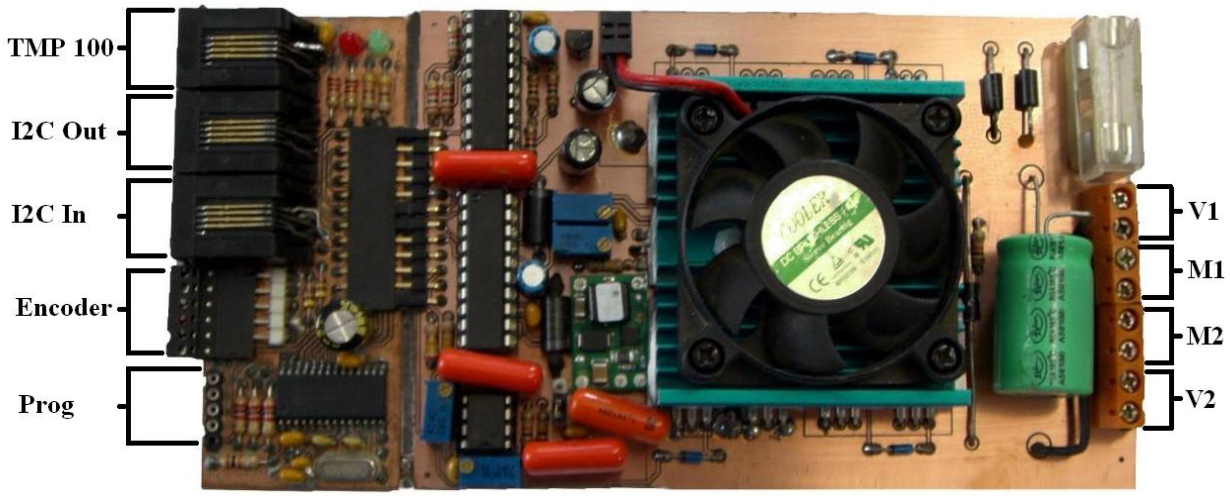
1.2 Conexões

O controlador PCM-500 possui 2 interfaces de comunicação.

- I2C
- RS232 (Opcional usando o conversor RS232 para I2C)

2 Instalação de Hardware

2.1 Placa PCM-500



2.1.1 Descrição de Pinos

Pinos de Alimentação:

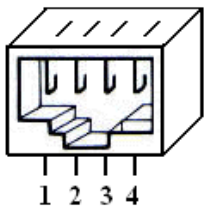
Símbolo	Designação
V1	GND
V2	+17…35V

Motor:

Símbolo	Designação
M1	Motor -
M2	Motor +

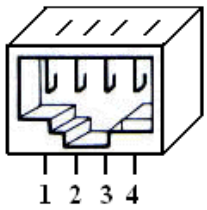
I2C In (Conector Rj-11 fêmea):

Pino	Designação
1	+5V
2	SDA
3	SDL
4	GND



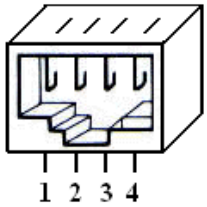
I2C Out (Conector Rj-11 fêmea):

Pino	Designação
1	NC
2	SDA
3	SDL
4	GND



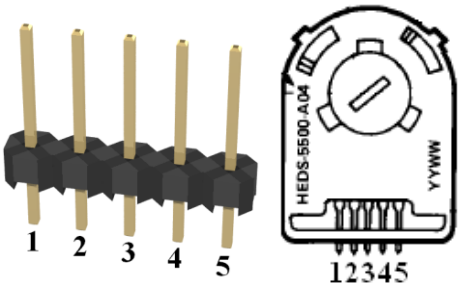
Sensor Temperatura I2C (Conector Rj-11 fêmea):

Pino	Designação
1	+5V
2	SDA
3	SDL
4	GND



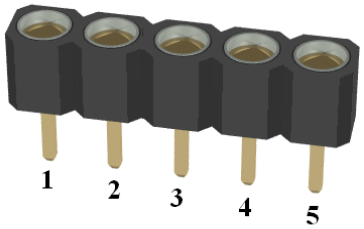
Encoder:

Pino	Designação
1	GND
2	INDEX
3	CH.A
4	+5V
5	CH.B

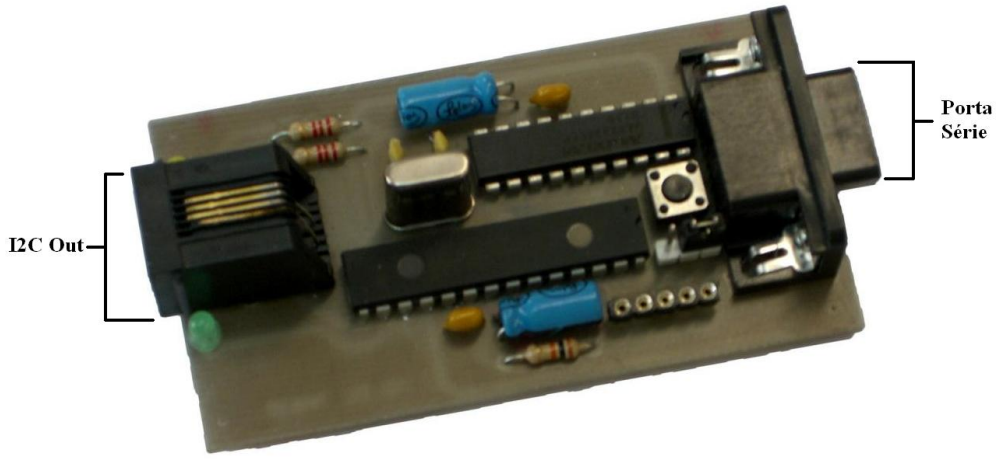


Conector para programação:

Pino	Designação
1	PGC
2	PGD
3	GND
4	+5V
5	MCLR



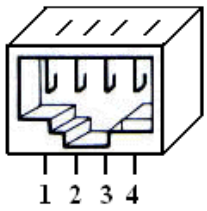
2.2 Conversor Série – I2C



2.2.1 Descrição dos pinos

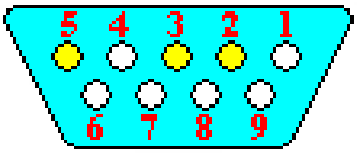
I2C Out:

Pino	Designação
1	+5V
2	SDA
3	SDL
4	GND



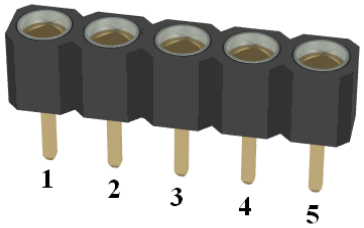
Porta Série:

Pino	Designação
2	RX
3	TX
5	GND



Conector para programação:

Pino	Designação
1	MCLR
2	+5V
3	GND
4	PGD
5	PGC



3 Actualização de Software

O controlador PCM-500 possui 31 parâmetros que podem ser ajustados através de registos sendo o suficiente para a maioria das aplicações. No entanto podem ser, substituídos microcontroladores, acrescentadas novas funções ao *software*, sendo necessário programar o microcontrolador.

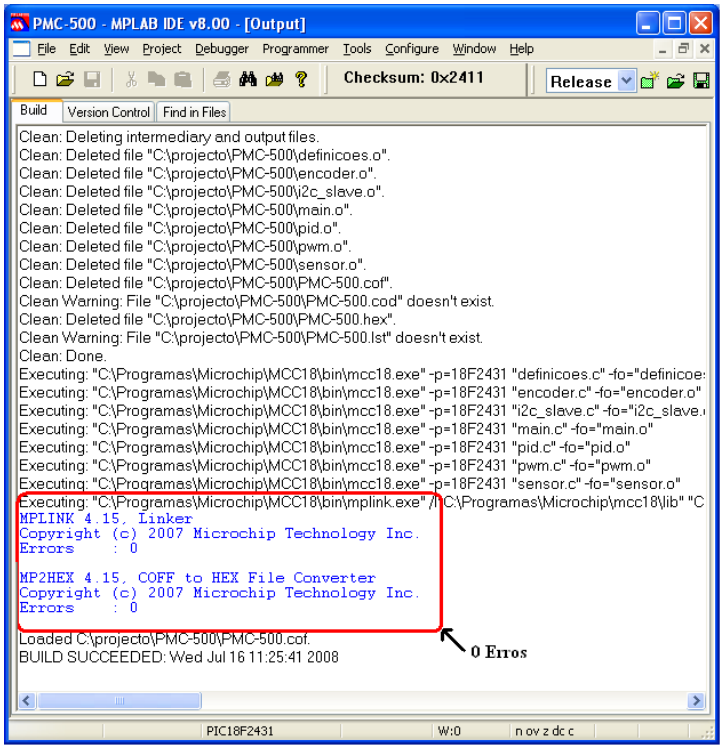
A programação do microcontrolador foi feita na linguagem de programação C com enxertos de código em *assembler*. O compilador utilizado foi o MPLAB C18 V3.15 instalado sobre o compilador base MPLAB IDE V8.00, O conversor série para i2c, também foi programado na linguagem C sendo o compilador utilizado o CCS C Compiler V4.0 instalado sobre o compilador base MPLAB IDE V8.00.

Os passos necessários para programar o controlador PCM-500 e o conversor série para i2c são parecidos no entanto para evitar possíveis equívocos serão explicados em separado.

3.1 Actualizar software do controlador PCM-500

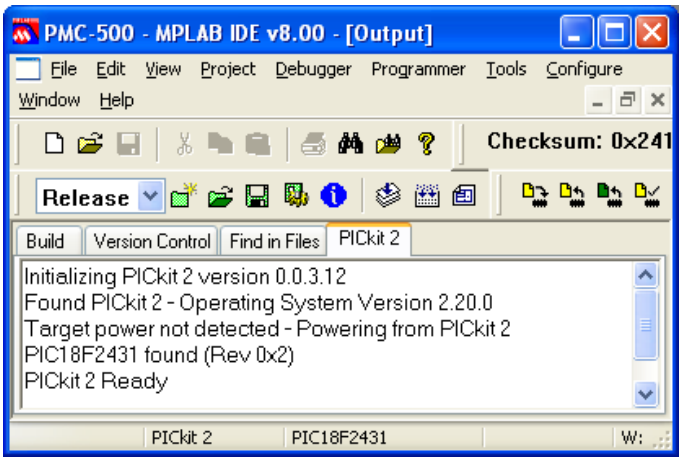
Os passos para instalar ou actualizar o software através do MPLAB IDE V8.00 são os seguintes:

1) Colocar o projecto numa directoria próxima da raiz ex. C:\projecto\PMC-500.
2) Abrir o projecto clicando no ficheiro "PMC-500.mcw"
3) Após ter o projecto aberto é necessário importar a EEPROM, este passo é muito importante, se não se programar a EEPROM do microcontrolador este não funcionará danificando o controlador. Para que tal não aconteça deve-se carregar os valores da EEPROM que estão num ficheiro "eeprom.hex", fazendo **File→ Import…→ Seleccionar** o ficheiro "eeprom.hex"**→ Abrir**.
4) Após estes passos o projecto está pronto para ser descarregado, caso haja alguma alteração no projecto é necessário voltar a compilar o projecto, fazendo os seguintes passos.
    a. Para compilar o projecto deve-se clicar em **Project → Build All**.
    b. A figura seguinte mostra a imagem que se deverá obter, caso contrário existe algum erro que deverá ser corrigido. Caso a compilação tenha sido bem sucedida pode-se proceder à programação do microcontrolador.

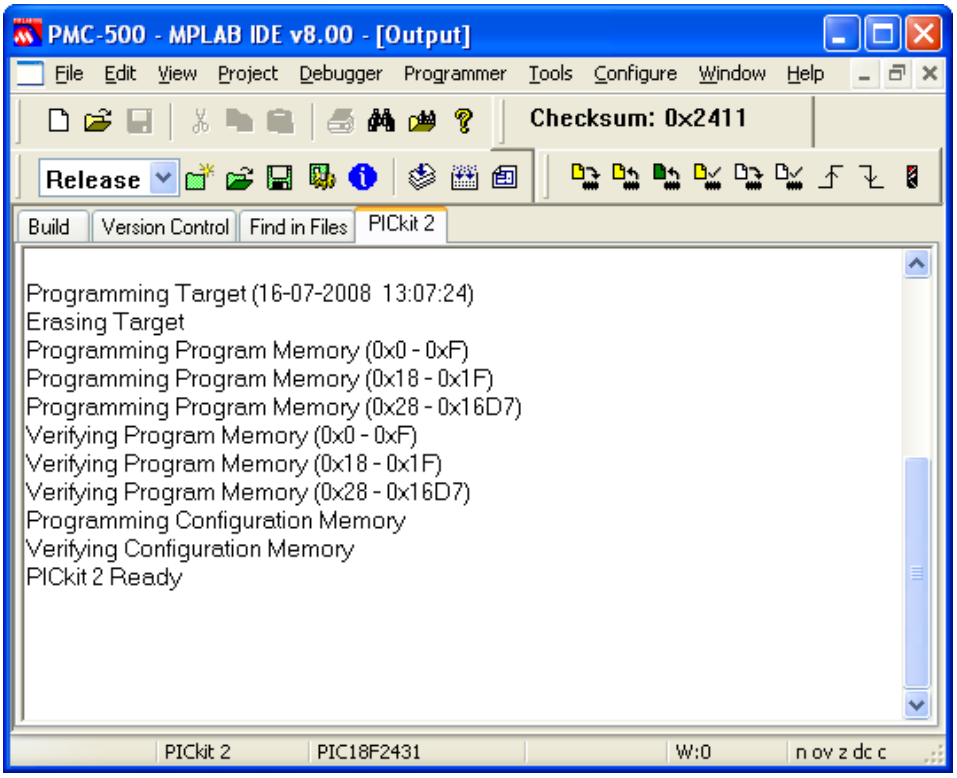


5) Dependendo do programador utilizado os passos seguintes podem ser diferentes. O programador utilizado neste caso foi um PICKit2 da Microchip, como mostra a figura seguinte. Os pinos de saída do PICKit2 devem ser ligados aos pinos de programação do controlador PCM-500 pela ordem correspondente.

6) Após a conexão física do programador à placa a programar e ao PC, é necessário escolher o programador no MPLAB IDE V8.00, clicando em **Programmer → Select Programmer → 4 PICKit2** devendo o MPLAB IDE V8.00 conectar-se automaticamente ao PICKit2, originando o resultado mostrado na figura seguinte.



7) O próximo e último passo consiste em programar o microcontrolador, para tal deve-se clicar em **Programer → Program**, devendo aparecer como resultado a imagem seguinte.

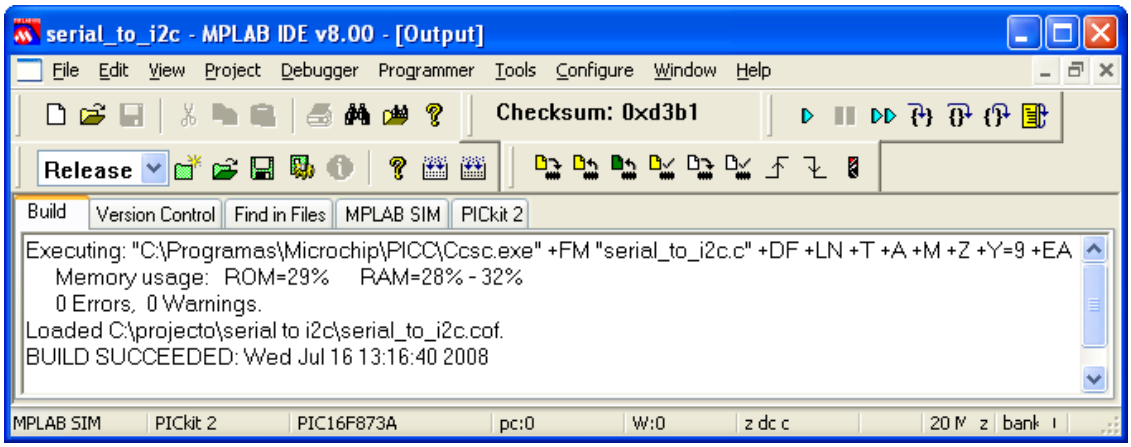


8) Aparecendo a imagem anterior, a programação foi bem sucedida sendo o próximo passo desligar o controlador, retirar o cabo de programação e voltar a ligar o controlador, estando pronto a funcionar.

3.2 Actualizar software do conversor Série I2C

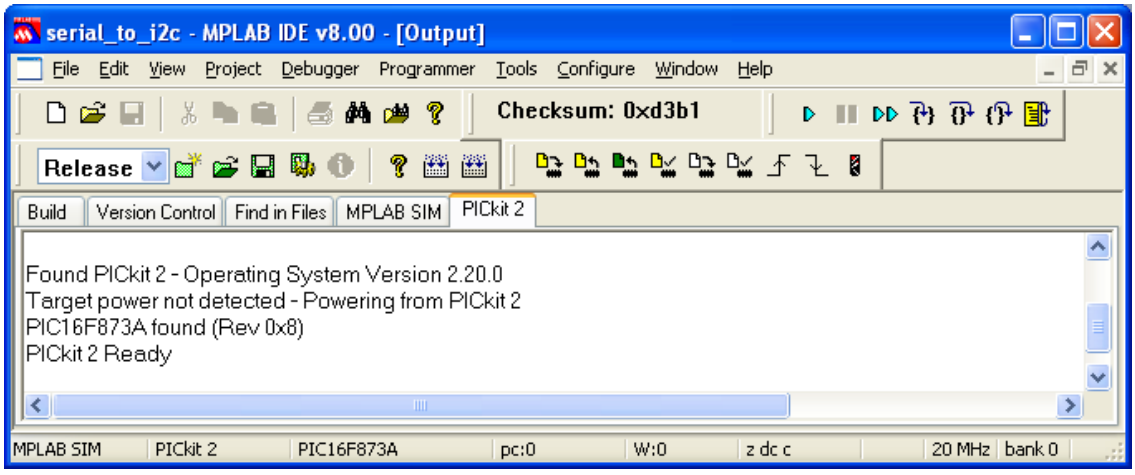
Os passos para instalar ou actualizar o software através do MPLAB IDE V8.00 são os seguintes:

1) Colocar o projecto numa directoria próxima da raiz ex. C:\projecto\serial to i2c.
2) Abrir o projecto carregando no ficheiro "serial_to_i2c.mcw"
3) Após estes passos o projecto está pronto para ser descarregado, caso haja alguma alteração no projecto é necessário voltar a compilar o projecto, fazendo os seguintes passos.
    a. Para compilar o projecto deve-se clicar em **Project → Build All**.
    b. A figura seguinte mostra a imagem que se deverá obter, caso contrário existe algum erro que deverá ser corrigido. Caso a compilação tenha sido bem sucedida pode-se proceder à programação do microcontrolador.

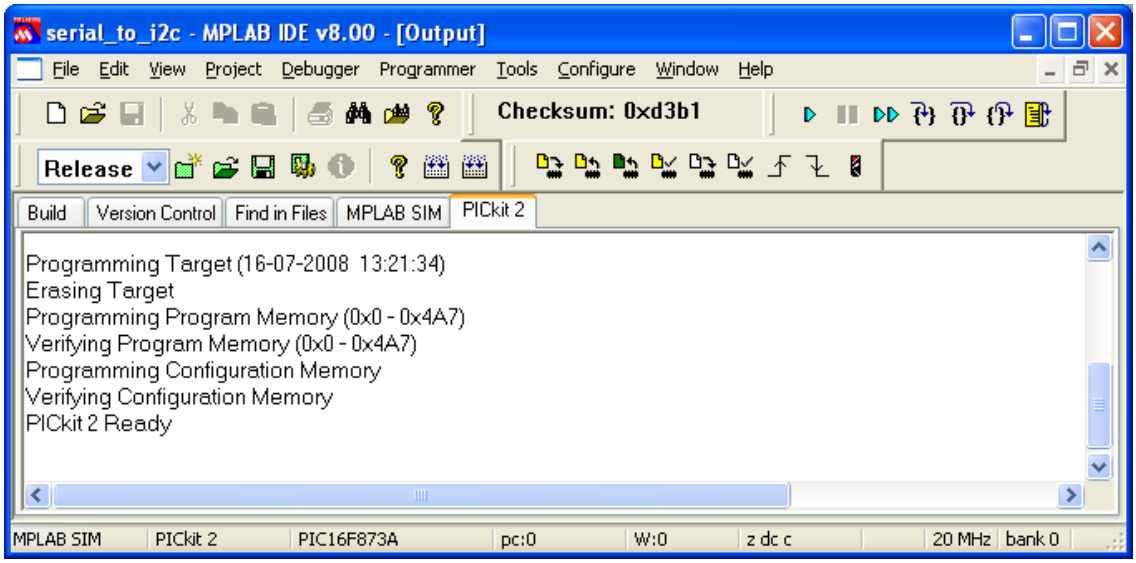


4) Dependendo do programador utilizado os passos seguintes podem ser diferentes. O programador utilizado neste caso foi um PICKit2 da Microchip, como mostra a figura seguinte. Os pinos de saída do PICKit2 devem ser ligados aos pinos de programação do conversor Série I2C.

5) Após a conexão física do programador à placa a programar e ao PC é necessário escolher o programador no MPLAB IDE V8.00, clicando em **Programmer → Select Programmer → 4 PICKit2** devendo o MPLAB IDE V8.00 conectar-se ao PICKit2, originado o resultado mostrado na figura seguinte.



6) O próximo e último passo consiste em programar o microcontrolador, para tal deve-se clicar em **Programer → Program**, devendo aparecer como resultado a imagem seguinte.



7) Aparecendo a imagem anterior a programação foi bem sucedida sendo o próximo passo desligar o conversor, retirar o cabo de programação e voltar a ligar o conversor.

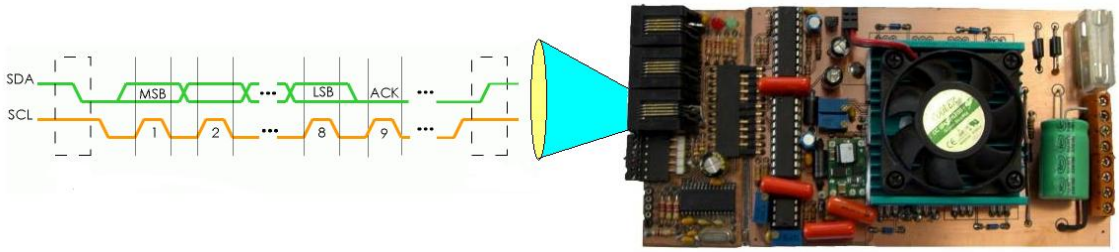
4 Dados Técnicos

4.1 Especificações gerais

PCM-500		
Descrição	**Valor**	**Unidade**
Tensão de Alimentação	17…35	V
Frequência PWM	19500	Hz
Corrente nominal	18	A
Corrente standby	110-160	mA
Nível lógico alto	2…5	V
Nível lógico baixo	0…0,8	V
Velocidade I2C sem conversor	100	kbps
Temperatura de funcionamento	0…70	°C
Temperatura máxima	-20…80	°C
Dimensões	80x160	mm
Peso	178	g
Conversor Série I2C		
Tensão nominal	5	V
Corrente nominal	6	mA
Velocidade I2C com conversor	2350	bps
Velocidade RS232	19200	bps
Dimensões	42x85	mm
Peso	23	g

5 Tramas de comunicação I2C e RS232

5.1 Interface I2C



5.1.1 Estrutura das tramas

Enviar n bytes:

	Address Motor R/$\overline{W}$			Address Register		Nº Bytes		Data MSB		Data LSB		
S	0 - 127	0	A	0 - 38	A	1-255	A	Byte 1	A	Byte n	A	P

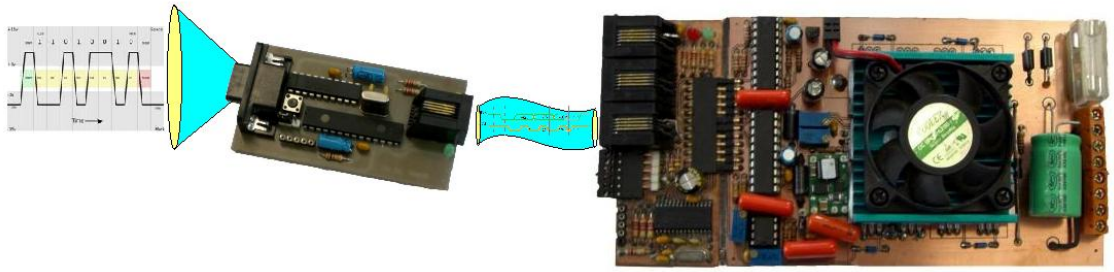
S → Start Bit
P → Stop Bit
R/$\overline{W}$ → Read/Write
A → Acnowledge

Ler n bytes:

	Address Motor R/$\overline{W}$			Address Register to Read			Address Motor R/$\overline{W}$			Data MSB Read		Data Read		Data LSB Read			
S	0 - 127	0	A	0 - 38	A	S	0 - 254	1	A	Byte 1	$\underline{A}$	Byte n	$\underline{A}$	PByte n	$\overline{A}$	P	

S → Start Bit
P → Stop Bit
R/$\overline{W}$ → Read/Write
A → Acnowledge by slave
$\underline{A}$ → Acnowledge by master
$\overline{A}$ → Not Acnowledge

5.2 Interface RS232



5.2.1 Configuração RS-232:

Baud rate → 19200 bps
Bits de dados → 8 bits
Paridade → sem paridade
Stop bits → 1 stop bit
Controlo de Fluxo → Sem controlo de Fluxo

5.2.2 Limitações de comunicação

A comunicação através do conversor RS-232 para I2C está limitada ao envio de tramas a cada 20 ms. Com o uso deste conversor só é possível enviar 1 ou 2 bytes de dados e receber sempre 2 bytes.

5.2.3 Estrutura das tramas

Enviar 2 bytes:

Address Motor		R/$\overline{W}$		Address Register		Data MSB		Data LSB	
0 - 254	;	0	:	0 - 38	,	Byte 0	-	Byte 1	.

Enviar 1 byte:

Address Motor		R/$\overline{W}$		Address Register		Data	
0 - 254	;	0	:	0 - 38	,	Byte	+

Ler 2 bytes:

Enviar

Address Motor		R/W		Address Register		Data	
0 - 254	;	1	:	0 - 38	,	Byte	.

Receber

Data MSB		Data LSB	
Byte 0	;	Byte 1	.

6 Registos de Configuração

0 -	MTR_ADRS	Endereço da placa controladora de velocidade.
1 -	MTR_FLAG	Sinal indicador da presença de motor.
2 -	KP	Ganho proporcional do controlo PID.
3 -	KI	Ganho integrativo do controlo PID.
4 -	KD	Ganho derivativo do controlo PID.
5 -	DIV_INTV	Intervalo de tempo entre cálculo derivativo.
6 -	INTEG_MAX	Limite máximo para o erro integrativo.
7 -	AMSTR_INTV_H	Intervalo de tempo entre cálculo do algoritmo PID.
8 -	AMSTR_INTV_L	Intervalo de tempo entre cálculo do algoritmo PID.
9 -	AMP_LIM	Corrente máxima no motor.
10 -	VOLT_LIM	Tensão mínima para a qual a ponte H deixa de funcionar.
11 -	TEMP_P_LIM	Temperatura máxima na ponte.
12 -	TEMP_M_LIM	Temperatura máxima no motor.
13 -	PPR_ENC	Número de pulsos por rotação do encoder.
14 -	LIM1_INF_M	Limites inferiores e superiores para mudança de escala na
15 -	LIM1_INF_L	leitura de velocidades.
16 -	LIM2_INF_M	
17 -	LIM2_INF_L	
18 -	LIM3_INF_M	
19 -	LIM3_INF_L	

20 - LIM1_SUP_M

21 - LIM1_SUP_L

22 - LIM2_SUP_M

23 - LIM2_SUP_L

24 - LIM3_SUP_M

25 - LIM3_SUP_L

25 - PWM_FREQ_M Frequência de oscilação de PWM.

27 - PWM_FREQ_L

28 - CRISTAL_H Frequência do cristal do microcontrolador.

29 - CRISTAL_L

30 - BREAK Sinal que indica se a travagem é forçada ou livre.

31 - VEL_DES_M Velocidade desejada para o motor.

32 - VEL_DES_L

33 - VEL_ACT_M Velocidade actual no motor.

34 - VEL_ACT_L

35 - AMP_ACT Corrente actual no motor.

36 - VOLT_ACT Tensão actual na ponte H.

37 - TEMP_P_ACT Temperatura actual da ponte H.

38 - TEMP_M_ACT Temperatura actual do motor.

6.1 Valores por defeito

Endereço	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Valor	0x00	0x01	0x08	0x04	0x00	0x05	0x0A	0x8C	0x2D	0x96
Endereço	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
Valor	0x6A	0x32	0x32	0x32	0x00	0x75	0x02	0xF6	0x05	0xFF
Endereço	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**
Valor	0x00	0xA7	0x03	0x5A	0x06	0xC7	0x4C	0x2C	0x27	0x10
Endereço	**30**	**31**	**31**	**33**	**34**	**35**	**36**	**37**	**38**	
Valor	00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	

6.2 Manipulação de registos

6.2.1 Alterar endereço da placa controladora

O controlador vem por defeito com o endereço 0, devendo ser alterado para evitar conflitos de software com a inserção de uma nova placa controladora.

Cada placa controladora contem 2 endereços, um para escrita e outro para leitura, que podem variar de 0…255.

O endereço que deve ser optado como endereço principal é o de escrita (endereços pares 0, 2, 4, 6, 8… 254).

Exemplo de trama a enviar através:

RS-232

Address Motor		R/$\overline{W}$		Address Register		Data	
Old Address	;	0	:	0	,	New Address	+

I2C

	Address Motor	R/$\overline{W}$		Address Register		Nº Bytes		Data		
S	Old Address	0	A	0	A	1	A	New Address	A	P

S → Start Bit
P → Stop Bit
R/$\overline{W}$ → Read/Write
A → Acnowledge

Para a operação de alteração do endereço ficar concluída deve-se reiniciar o microcontrolador, desligando e voltando a ligar a placa controladora. Após esta operação o endereço da placa controladora será o novo endereço inserido.

6.2.2 Manipulação do registo MTR_FLAG

O registo MTR_FLAG serve para indicar ao utilizador que o motor tem algum problema. Este registo tem por defeito o valor 1, quando existir alguma anomalia

relacionada com o motor todo o processo pára, o registo MTR_FLAG fica com o valor 0, e um sinal visual (LED vermelho) dá a indicação de erro. (ver sinais visuais de erro).

Para retomar o processo, após a correcção da anomalia, é necessário colocar o registo MTR_FLAG com o valor 1.

6.2.3 Manipulação dos registos Kp, Ki, Kd

Os registos Kp, Ki, Kd são correspondentes aos ganhos proporcional, integral e derivativo do algoritmo PID, estes ganhos são calculados pelo utilizador sendo diferentes para cada motor. Na maioria das aplicações, a função de transferência do sistema a controlar (motor neste caso) não é conhecida. Para sintonizar os parâmetros usam-se métodos experimentais, o método de *Ziegler-Nichols*, *Cohen-Coon*, *Chien-Hrones-Reswick* são exemplos de métodos experimentais que permitem determinar os ganhos do PID.

Os valores dos ganhos Kp, Ki e Kd podem variar entre 0 e 255, quando o valor é 0 significa que o ganho em causa não é calculado. De modo a que o microcontrolador não opere com valores fraccionários os ganhos inseridos estão à escala de 128 para 1, isto significa que um ganho de 1,5 corresponde a um valor de 192.

Atenção! Deve-se ter em atenção o valor dos ganhos colocados no controlador, ganhos elevados provocam oscilações bruscas de velocidade, podendo danificar o controlador e/ou o motor. Em caso de dúvida colocar valores baixos para os ganhos, ajustando-os até o valor pretendido.

6.2.4 Manipulação do registo INTEG_MAX

A forma como o cálculo integral é feito (soma de todos os erros) faz com que este possa ter valores elevados, a resposta do sistema e os intervalos de amostragem pequenos fazem com que o erro acumulado seja elevado, criando oscilações na resposta do sistema.

De forma a evitar erros acumulados (erro integral) elevados é colocado um limite máximo, de forma que o erro integral nunca ultrapasse esse limite.

Este limite pode variar entre 0 e 255. Valores elevados mais oscilações na velocidade, valores menores menos oscilações, no entanto pode não atingir a velocidade desejada.

6.2.5 Manipulação dos registos AMSTR_INTV_H e AMSTR_INTV_L

Estes registos determinam o intervalo de tempo em que é executado o algoritmo PID, em aplicações de resposta lenta (controlo de temperatura), este pode tomar valores mais elevados, em aplicações em que a resposta deve ser rápida este deve tomar valores mais reduzidos.

Neste caso a frequência de amostragem deve ser elevada de modo a que o motor tenha respostas rápidas, quer a variações de carga, quer em velocidade desejada.

O valor dos registos AMSTR_INTV_H e AMSTR_INTV_L variam de 0…255 para calcular o valor destes registos devem ser executados os seguintes passos:

O *timer* utilizado para gerar este tempo é de 16 bits e é incrementado a ¼ da frequência do cristal.

1) Calcular quantos incrementos (Ni) efectua o *timer* para um determinado intervalo de tempo Ta.

$$Ni = Ta \times Fosc$$

2) Após calcular Ni é necessário subtrair 349, este valor corresponde ao tempo necessário para entrar na interrupção.

$$Ni = Ni - 349$$

3) Como o timer entra na interrupção quando atingir o valor 65535, deve ser carregado inicialmente com:

$$Valor\_Timer = 65536 - Ni$$

4) O valor a inserir no registo AMSTR_INTV_H será o valor inteiro de:

$$AMSTR\_INTV\_H = \frac{Valor\_timer}{256}$$

5) O valor a inserir no registo AMSTR_INTV_L será o valor inteiro de:

$$AMSTR\_INTV\_L = Valor\_timer - (AMSTR\_INTV\_H \times 256)$$

Atenção! A frequência de amostragem não deve ser excessivamente elevada, pois torna o ajuste dos ganhos para baixas rotações muito trabalhoso ou até impossível, esta deve ser ajustada para valores que satisfaçam os tempos de resposta a baixas rotações, sofrendo posteriormente um ajuste fino, para velocidades mais elevadas.

6.2.6 Manipulação do registo DIV_INTV

O erro derivativo é a diferença entre o erro anterior e o novo erro num intervalo de tempo, se o motor estiver parado e a velocidade desejada passar para um valor elevado o erro derivativo multiplicado pelo ganho origina um erro ainda maior, tendo como consequência oscilações bruscas na velocidade do motor.

Por esta razão o erro derivativo tem mais eficácia quando a velocidade real está próxima da desejada. Para evitar arranques bruscos este não é calculado ao mesmo tempo que o erro proporcional e integrativo, sendo executado em intervalos de (DIV_INTV x intervalo amostragem).

O registo DIV_INTV pode tomar valores de 0…255.

Atenção! Intervalos de tempo muito curtos entre o cálculo derivativo podem provocar arranques e travagens muito bruscas no motor, podendo originar problemas indesejáveis quer no motor quer no controlador.

6.2.7 Manipulação do registo AMP_LIM

O registo "AMP_LIM" limita a corrente no motor independentemente da corrente máxima do controlador de velocidade, este encontra-se protegido por um fusível. Como o controlador suporta motores de várias potências é necessário limitar a potência fornecida pelo controlador para não danificar o motor.

Quando o valor limite da corrente é alcançado o controlador deixa de funcionar de modo a não danificar o motor, um LED vermelho indicará o erro (ver sinais erro visuais). O controlador volta a funcionar após alguns segundos, caso a corrente volte a exceder o limite o processo repete-se.

Este registo toma valores de 0…255 sendo o valor 10 vezes superior à corrente pretendida (ex. corrente de 15,5 A o valor do registo fica com o valor 155).

6.2.8 Manipulação do registo VOLT_LIM

O valor deste registo limita o valor inferior da tensão de operação do controlador, tensões abaixo deste valor provocariam o mau funcionamento do controlador.

Um sinal visual indica quando a tensão se encontra abaixo deste limite, o controlador deixa de funcionar de modo a evitar danos (ver sinais visuais de erro).

Quando o valor da tensão de alimentação for superior ao valor guardado no registo "VOLT_LIM" o controlador fica operacional aguardando novas instruções.

O valor deste registo varia de 0…255 correspondendo cada unidade a 160mV (ex. 17V corresponde a um valor aproximado de 106).

Atenção! Valores de tensão abaixo dos nominais provocam um sobre aquecimento da ponte h, podendo danificar os seus componentes.

6.2.9 Manipulação do registo TEMP_M_LIM

Este registo é utilizado para definir uma temperatura máxima no motor. Quando a temperatura no motor atingir o valor "TEMP_M_LIM" o motor pára, só voltando a aceitar comandos de velocidade quando a temperatura for inferior à temperatura limite.

Quando o motor parar um sinal visual (LED vermelho) indica o erro de sobre-temperaturas no motor (ver sinais visuais de erro).

O valor deste registo varia de 0 até 255 graus Célsius.

Atenção! Os valores de temperatura devem ser respeitados, sobredimensionar o valor de temperatura nominal do motor pode danificar o mesmo.

6.2.10 Manipulação do registo TEMP_P_LIM

Este registo limita a temperatura na ponte H, de modo a proteger o controlador. Quando a temperatura atingir o valor do registo "TEMP_P_LIM" o controlador pára de funcionar até que esta desça abaixo do valor limite, quando entrar novamente em funcionamento o sinal visual correspondente ao erro deixa de existir (ver sinais visuais de erro). O controlador contínua parado até receber uma nova velocidade desejada.

A ponte H está equipada com um dissipador ventilado. A ventilação entra em funcionamento 10º abaixo da temperatura definida no registo "TEMP_P_LIM".

O valor deste registo varia de 0 até 255 graus Célsius.

Atenção! Os valores de temperatura devem ser respeitados, sobredimensionar o valor de temperatura nominal da ponte H pode danificar os seus componentes.

6.2.11 Manipulação do registo PPR_ENC

O registo PPR_ENC guarda o valor dos pulsos por rotação do *encoder*, servindo de base para o cálculo da velocidade.

Este registo varia de 0…255, correspondendo cada unidade a uma dezena (ex. encoder de 500 PPR o valor a inserir no registo será 50).

6.2.12 Manipulação dos registos LIM1_INF_M, LIM1_INF_L, LIM2_INF_M, LIM2_INF_L, LIM3_INF_M, LIM3_INF_L, LIM1_SUP_M, LIM1_SUP_L, LIM2_SUP_M, LIM2_SUP_L, LIM3_SUP_M e LIM3_SUP_L

A leitura de velocidades é feita através de um *timer* que mede o tempo dos pulsos do *encoder*. O tempo de um pulso do *encoder* para baixas rotações é muito maior que o tempo do mesmo pulso para altas rotações, consequentemente o *timer* não pode usar a mesma escala para baixas e altas rotações.

Estes registos servem para configurar a velocidade em que o *timer* muda de escala, sendo calculados com a ajuda de uma folha Excel (encoder.xls).

Como a percepção desta configuração não é das mais comuns é dado um exemplo de como a fazer. A tabela seguinte mostra a folha de Excel usada para determinar os parâmetros de configuração do *encoder*.

	A	F	G	H	J	K	L	M	N	O	P
1	rpm	timer	64x	timer 16x	timer 4X	timer 1X oscilador\8	Encoder P	Timer*8	Cristal Periodo		
2	1	38400000	9600000	2400000	75000	500	8,00E-07	1,00E-07			
3	2	19200000	4800000	1200000	37500			1,00E+07			
4	3	12800000	3200000	800000	25000						
5	4	9600000	2400000	600000	18750	timer 1X os	timer 4X	timer 16x	timer 64x		
6	5	7680000	1920000	480000	15000	2	37	147	586	min	
7	6	6400000	1600000	400000	12500	281	1579	3123	6343	max	
8	7	5485714	1371428	342857	10714	279	1542	2976	5757	diferença	
9	8	4800000	1200000	300000	9375	142	808	1635	3465	ponto médio	
10	9	4266666	1066666	266666	8333						
11	10	3840000	960000	240000	7500	Lim_1_sup		Lim_2_sup		Lim_3_sup	
12	11	3490909	872727	218181	6818	167		858		1735	
13	12	3200000	800000	200000	6250	MSB	LSB	MSB	LSB	MSB	LSB
14	13	2953846	738461	184615	5769	0	167	3	90	6	199
15	14	2742857	685714	171428	5357						
16	15	2560000	640000	160000	5000	Lim_1_inf		Lim_2_inf		Lim_3_inf	
17	16	2400000	600000	150000	4687	117		758		1535	
18	17	2258823	564705	141176	4411	MSB	LSB	MSB	LSB	MSB	LSB
19	18	2133333	533333	133333	4166	0	117	2	246	5	255

O utilizador deve configurar todos os campos a amarelo (K2, M3, K6,K7, L6,L7, M6,M7, N6,N7) sendo os campos de saída a violeta (K14, L14, K19, L19, M14, N14, M19, N19, O14, P14, O19, P19).

Os pontos de configuração são os seguintes:

1) Inserir os pulsos por rotação do *encoder* (PPR) no campo K2 e a frequência do cristal no campo M3.
2) Percorrer a coluna J até encontrar o primeiro campo a verde, colocar o valor da rotação correspondente à mesma linha da coluna A no campo K6.
3) Continuar a percorrer a coluna J até encontrar um campo a vermelho, colocar o valor da linha anterior correspondente à coluna A no campo K7.
4) Repetir o ponto 2 e 3 para as colunas H, G, F, colocando os valores correspondentes nos campos L6, L7, M6, M7, N6 e N7 respectivamente.
5) A tabela seguinte mostra a correspondência entre as variáveis e os campos da folha Excel.

Registo	Campo
LIM1_SUP_M	K14
LIM1_SUP_L	L14
LIM1_INF_M	K19
LIM1_INF_L	L19
LIM2_SUP_M	M14
LIM2_SUP_L	N14
LIM2_INF_M	M19
LIM2_INF_L	N19
LIM3_SUP_M	O14
LIM3_SUP_L	P14
LIM3_INF_M	O19
LIM3_INF_L	P19

Todos os registos têm valores que variam de 0…255.

6.2.13 Manipulação dos registos PWM_FREQ_M e PWM_FREQ_L

Este registo define a frequência a que oscilam os sinais PWM.

A tabela seguinte retirada do *datasheet* do microcontrolador PIC 18f2431 (http://ww1.microchip.com/downloads/en/DeviceDoc/39616C.pdf), demonstra como varia a resolução do sinal PWM em função da frequência pretendida e da frequência de oscilação. Devendo-se encontrar uma relação que satisfaça a frequência pretendida de modo a ter a maior resolução.

PWM Frequency = 1/TPWM		
FOSC	PWM Resolution	PWM Frequency
40 MHz	14 bits	2.4 kHz
40 MHz	13 bits	4.9 kHz
40 MHz	12 bits	9.8 kHz
40 MHz	11 bits	19.5 kHz
40 MHz	10 bits	39.0 kHz
40 MHz	9 bits	78.1 kHz
40 MHz	8 bits	156.2 kHz
40 MHz	7 bits	312.5 kHz
40 MHz	6 bits	625 kHz
25 MHz	14 bits	1.5 kHz
25 MHz	12 bits	6.1 kHz
25 MHz	10 bits	24.4 kHz
10 MHz	14 bits	610 Hz
10 MHz	12 bits	2.4 kHz
10 MHz	10 bits	9.8 kHz
5 MHz	14 bits	305 Hz
5 MHz	12 bits	1.2 kHz
5 MHz	10 bits	4.9 kHz
4 MHz	14 bits	244 Hz
4 MHz	12 bits	976 Hz
4 MHz	10 bits	3.9 kHz

Cada registo varia de 0…255 sendo constituídos por números inteiros. Se for pretendida uma frequência de 19500Hz os registos tomam os seguintes valores.

$$PWM\_FREQ\_M = (int)\ \frac{19500}{256}$$

$$PWM\_FREQ\_L = (int)\ \left[19500 - (PWM\_FREQ\_M \times 256)\right]$$

6.2.14 Manipulação dos registos CRISTAL_H e CRISTAL_L

O valor do cristal do microcontrolador é guardado nos registos CRISTAL_H e CRISTAL_L, este valor serve para cálculos internos para funções de PWM e leituras de velocidade, varia de 0…255 cada registo sendo o valor inserido do cristal 1000 vezes inferior ao valor real. Se o cristal for de 10.000.000 Hz o valor a inserir será 10.000 ficando cada registo com o seguinte valor.

$$\text{CRISTAL\_H} = (\text{int})\ \frac{10000}{256}$$

$$\text{CRISTAL\_L} = (\text{int})\ \left| 10000 - (\text{CRISTAL\_L} \times 256) \right|$$

6.2.15 Manipulação do resisto BREAK

Este registo define o modo como a travagem é feita, quando tem o valor 0 a travagem é feita de forma livre o motor perde a inércia de forma natural ficando livre após a paragem, quando o registo *break* tem o valor 1 a travagem é forçada, após a travagem o motor fica preso não sendo possível movimentá-lo.

Considera-se que a travagem é livre ou forçada quando a velocidade desejada tem o valor 0, caso contrário o registo *break* não tem influência.

6.2.16 Manipulação dos registos VEL_DES_M e VEL_DES_L

Para enviar uma velocidade desejada ao motor são utilizados estes registos (VEL_DES_M e VEL_DES_L). Com estes registos é possível enviar velocidades desejadas de 0 até 65535 rpm, no entanto, o motor pode não atingir tais velocidades, de modo que cabe ao utilizador testar qual a velocidade máxima atingida pelo seu motor em vazio, definindo limites próprios de operação.

Os registos VEL_DES_M e VEL_DES_L para uma VEL_DES ficam com os seguintes valores.

$$\text{VEL\_DES\_M} = (\text{int})\ \frac{\text{VEL\_DES}}{256}$$

$$\text{VEL\_DES\_L} = (\text{int})\ \left| \text{VEL\_DES} - (\text{VEL\_DES\_M} \times 256) \right|$$

6.2.17 Manipulação dos registos VEL_ACT_M e VEL_ACT_L

Estes dois registos fornecem ao utilizador o valor actual de velocidade do motor, cada registo devolve um número inteiro que varia de 0…255. Para determinar o valor de velocidade actual (VEL_ACT) é necessário calcular a seguinte fórmula.

$$VEL\_ACT = (VEL\_ACT\_M * 256) + VEL\_ACT\_L$$

6.2.18 Manipulação do registo AMP_ACT

Este registo contém o valor actual da corrente que circula no controlador, o valor deste varia de 0…255, sendo o valor real 10 vezes inferior. Quando o valor apresentado for de 150, o valor real da corrente que circula no controlador é de 15A.

6.2.19 Manipulação do registo VOLT_ACT

Este registo contém o valor actual da tensão aplicada no controlador, o valor do registo varia de 0…255. Para obter o valor real da tensão é necessário multiplicar o valor contido no registo por 160×10^{-3}.

6.2.20 Manipulação do registo TEMP_P_ACT

Este registo contem o valor actual da temperatura na ponte H do controlador, o seu valor varia de 0…255, sendo o resultado apresentado em graus célsius.

6.2.21 Manipulação do registo TEMP_M_ACT

Este registo contem o valor actual da temperatura no motor, o seu valor varia de 0…255, sendo o resultado apresentado em graus célsius.

7 Sinais Visuais de Erro

A Placa controladora PCM-500 contém dois sinais visuais. O LED verde indica que a placa tem aplicada uma tensão de alimentação e um LED vermelho normalmente apagado indica, em caso de erro, qual o mesmo.

O modo com que o LED vermelho pisca ou fica ligado indica o erro ocorrido. A tabela seguinte indica o erro ocorrido em função do estado do LED.

Estado Led Vermelho	Erro
	Tensão inferior à tensão limite
	Corrente superior à corrente limite
	Temperatura do motor excedeu o limite
	Temperatura da Ponte H excedeu o limite
	Problema com o motor
	Problema de comunicação I2C

8 Contactos

Mail: **pcm.500.um@gmail.com**

Anexo II – Código C implementado no PIC18f2431

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                  MAIN.C                                                                    //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/* Inserção de bibliotecas e ficheiros */

#include <p18f2431.h>
#include <delays.h>
#include <i2c.h>
#include <stdio.h>
#include "pwm.h"
#include "encoder.h"
#include "pid.h"
#include "i2c_slave.h"
#include "sensor.h"

/* ************************** Configuration Bits ************************** */

#pragma config OSC = HSPLL          // Activa o PLL do Oscilador
#pragma config FCMEN = OFF
#pragma config IESO = OFF
#pragma config PWRTEN = OFF
#pragma config BOREN = OFF
#pragma config BORV = 20
#pragma config WDTEN = OFF
#pragma config WINEN = OFF
#pragma config WDPS = 128
#pragma config T1OSCMX = OFF
#pragma config HPOL = HIGH
#pragma config LPOL = HIGH
#pragma config PWMPIN = OFF
#pragma config MCLRE = ON
#pragma config STVREN = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config CP0 = OFF
#pragma config CP1 = OFF
#pragma config CPB = OFF
#pragma config CPD = OFF
#pragma config WRT0 = OFF
#pragma config WRT1 = OFF
#pragma config WRTB = OFF
#pragma config WRTC = OFF
#pragma config WRTD = OFF
#pragma config EBTR0 = OFF
#pragma config EBTR1 = OFF
#pragma config EBTRB = OFF

// inicialização das funções de interrupção
void timer5_isr(void);
void timer1_isr(void);
void timer0_isr(void);
```

```
void adc_isr(void);
void i2c_isr (void);

void high_interrupt(void);
void low_interrupt_aux(void);


        ////////////////////////// INTERRUPÇÕES DE BAIXA PRIORIDADE ////////////////////////////

#pragma code low_vector=0x18
        void low_interrupt(void)
        {
        low_interrupt_aux();            //Ocorreu uma interrupção de baixa prioridade
        }
#pragma code

                                //  Verifica qual a interrupção ocorrida
void low_interrupt_aux(void){
_asm
        BTFSC PIR1,0,0
        goto timer1_isr
                BTFSC PIR3,0,0
                goto timer5_isr
                        BTFSC INTCON,2,0
                        goto timer0_isr
                                BTFSC PIR1,6,0
                                goto adc_isr
                                nop
_endasm
}

          /***** Interrupção do timer_1 executa o algoritmo PID ****************/

#pragma interruptlow timer1_isr
void timer1_isr(void){
PIE1bits.TMR1IE=0;              // desabilita a interrupção do timer 1
act_vel_des();                  // actualiza a velocidade desejada

// Verifica se não existe nenhuma condição que impeça a execução do PID
if ((vel_des!=0 || dados[BREAK]!=0) && flag_alarm ==0){
                // Activa a ponte H pois poderia não o estar (velocidade anterior = 0)
        PONTE_H_EN = 0;                                                         //
        Actualiza a velocidade Real
        velocidade();
        // Calcula o erro entre a velocidade real e desejada
        PID_STATUSbits.new_err_sign=sub_16bits(dir_des,QEICONbits.UP,vel_des,new_vel
        ,&new_error);
        // Executa o cálculo do erro acumulado de diferençial
        pid_interrupt();
        // Executa o algoritmo PID
        pid_main();
        // Adiciona o resultado ao duty cycle do PWM e corrige a direcção
        pwm_direccao=sum_16bits(pwm_direccao,PID_STATUSbits.pid_out_sign,pwm_duty
_cycle,pid_out,&pwm_duty_cycle);
        // Actualiza o valor do duty cycle e direcção
```

```
        pwm_dir_e_duty_cycle();
         // Ajusta a escala para as próximas leituras de velocidade
        ajustar_escala();
        // Ajusta o intervalo entre execuções do algoritmo PID
        TMR1H=dados[AMSTR_INTV_H];
         TMR1L=dados[AMSTR_INTV_L];

}else{  // Caso a exista uma condição que impeça a execução do algoritmo PID
        PONTE_H_EN =1;              // Desactiva a ponte H
        pwm_duty_cycle=0;           // coloca o duty cycle com o valor 0
        pwm_dir_e_duty_cycle();     // actualiza o duty cyle
}

PIR1bits.TMR1IF=0;   // Apaga a flag de interrupção do timer 1
PIE1bits.TMR1IE=1;   // Activa a interrupção do timer 1
        }
           /***************** Fim da interrupção do timer_1 *****************/


 /********* Interrupção do timer_5 overflow no timer_5 a velocidade real é 0 rpm *********/

#pragma interruptlow timer5_isr
void timer5_isr(void){
        tmr5_overflow=1;        // Activa a flag que indica a existência de um overflow
        PIR3bits.TMR5IF = 0;  // Limpa a flag de overflow do timer_5
}
           /***************** Fim da interrupção do timer_5 *****************/


   /****** Interrupção do timer_0 responsável pelo intervalo de leitura dos sensores ******/

#pragma interruptlow timer0_isr
void timer0_isr(void){
        if(contador%5==0){              // A cada 5 interrupções faz a leitura de um sensor
                sensor_ler();           // Lê o sensor equivalente à variável sensor
                sensor_alarme();        // Verifica os valores lidos e actua caso necessário
                sensor++;               // Passa para o próximo sensor
                        if (contador==15){      // Se leu todos os sensores reinicia o processo
                                contador=0;
                                sensor=0;
                        }
        }
        contador++;                     // Incrementa do contador de interrupções ocorridas
        // Carrega o timer_0 com o valor correspondente ao intervalo de tempo
        TMR0L=TMR0_L;
        TMR0H=TMR0_H;
        INTCONbits.TMR0IF = 0;      // Limpa a flag de interrupção do timer_0
                                }
           /***************** Fim da interrupção do timer_0 *****************/


            /*************** Interrupção de leitura dos ADC *****************/
#pragma interruptlow adc_isr
void adc_isr(void){
adc[0]= ADRESH;      // Copia o últimos valores lidos pelos ADCs para o array adc[]
```

```
adc[1]= ADRESH;       // estes valores são respectivos à corrente e tensão
PIR1bits.ADIF=0;      // Limpa a flag de interrupção da leitura dos ADCs
                                    }

            /***************** Fim da interrupção do ADC ****************/


        ////////////////////////// INTERRUPÇÕES DE ALTA PRIORIDADE ////////////////////////////

#pragma code high_vector=0x08

void interrupt_at_high_vector(void){
_asm goto i2c_isr _endasm

}
#pragma code

            /***************** Interrupção da comunicação I2C ****************/

#pragma interrupt i2c_isr
void i2c_isr (void)
{
        PIR1bits.SSPIF = 0;     // Limpa a Flag de interrupção do I2C
        // Chama a função que descodifica a leitura ou escrita no barramento I2C
        i2c_slave_interrupt();
}

        /***************** Fim da interrupção da comunicação I2C *****************/

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
                                                main()
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


void main(void)
{
        // Tempo necessário para o PWM não ficar activo durante a conexão e programação
(255 ms)
        Delay10KTCYx(255);
        WTD=1;                                  // Activa WatchDog
        definicoes_ler_eeprom();                // Lê as definições anteriores da EEPROM
        // Calcula parâmetros iniciais evitando a repetição de cálculos
        definicoes_calcular_parametros();
        // Configura o porto C
        TRISC = 0b10110000;
        TRIS_PONTE_H_EN =0;       // Coloca o pino de activação da ponte H como saída

        i2c_slave_config();             // Configura a comunicação I2C
        sensor_config();                // Configura a leitura dos sensores
        encoder_config();               // Configura a leitura do encoder óptico
        pwm_config();                   // Configura o gerador de sinais PWM
        pid_inicializar();              // Inicializa o algoritmo PID
        ALL_INTERRUPT_ON();       // Activa todas as interrupções
```

```
        while(1){               // Espera a ocorrência de uma interrupção
                clrwtd();       // Faz reset ao watchdog
        }
}
```

//

// DEFINICOES.H //

//

```
#include <p18f2431.h>
                /* ------------------------------ Definições --------------------------- */

// Definições das posições do array dados[]

#define MTR_ADRS        0
#define MTR_FLAG        1
#define KP              2
#define KI              3
#define KD              4
#define DIV_INTV        5
#define INTEG_MAX       6
#define AMSTR_INTV_H    7
#define AMSTR_INTV_L    8
#define AMP_LIM         9
#define VOLT_LIM        10
#define TEMP_P_LIM      11
#define TEMP_M_LIM      12
#define PPR_ENC         13
#define LIM1_INF_M      14
#define LIM1_INF_L      15
#define LIM2_INF_M      16
#define LIM2_INF_L      17
#define LIM3_INF_M      18
#define LIM3_INF_L      19
#define LIM1_SUP_M      20
#define LIM1_SUP_L      21
#define LIM2_SUP_M      22
#define LIM2_SUP_L      23
#define LIM3_SUP_M      24
#define LIM3_SUP_L      25
#define PWM_FREQ_M      26
#define PWM_FREQ_L      27
#define CRISTAL_H       28
#define CRISTAL_L       29
#define BREAK           30
#define VEL_DES_M       31
#define VEL_DES_L       32
#define VEL_ACT_M       33
#define VEL_ACT_L       34
#define AMP_ACT         35
#define VOLT_ACT        36
#define TEMP_P_ACT      37
#define TEMP_M_ACT      38
```

```
// define que permite fazer reset ao watchdog
#define clrwtd() _asm CLRWDT _endasm
// define que permite transformar dois char de 8 bits num int de 16 bits
#define join(A,B) (unsigned int)(((unsigned int)A<<8)|(unsigned int)B)
// define que permite testar se um bit de um byte é 1
#define testbit(data,bitno) ((data>>bitno)&0x01);
//faz a distinção entre interrupções (alta/baixa prioridade)/ activar interrupções
#define ALL_INTERRUPT_ON() {RCONbits.IPEN = 1; INTCONbits.GIEH =
1;INTCONbits.PEIE=1;}

// definições de pinos
#define WTD WDTCONbits.SWDTEN
#define TRIS_PONTE_H_EN TRISBbits.TRISB4
#define PONTE_H_EN LATBbits.LATB4
#define TRIS_FLAG_PROBLEM TRISCbits.TRISC1
#define FLAG_PROBLEM LATCbits.LATC1
#define TRIS_FAN TRISBbits.TRISB5
#define FAN LATBbits.LATB5

// definições dos endereços dos sensores de temperatura TMP100
#define TMP_MOTOR 0b10010100
#define TMP_PONTE 0b10010000

// definição do intervalo de tempo entre leitura de sensores
#define TMR0_H 0xF0
#define TMR0_L 0xBD
                /* ---------------------------- Fim de Definições ---------------------- */

// Variáveis auxiliares a todos os ficheiros
extern unsigned char amp_aux;
extern unsigned char volt_aux;
extern unsigned char contador;
extern unsigned char flag_alarm;
extern unsigned char adc[2];
extern unsigned char sensor;
extern unsigned char flag_motor;
extern unsigned char dados[39];
extern unsigned long CRISTAL;
extern unsigned int FREQ_PWM;
extern unsigned int LIM1_INF;
extern unsigned int LIM2_INF;
extern unsigned int LIM3_INF;
extern unsigned int LIM1_SUP;
extern unsigned int LIM2_SUP;
extern unsigned int LIM3_SUP;
extern unsigned int PULSOS;
extern unsigned int vel_des;
extern unsigned char dir_des;

// definição da função que permite calcular os parâmetros iniciais
void definicoes_calcular_parametros(void);
void definicoes_ler_eeprom(void);               // definição da função de leitura da eeprom
// definição da função de escrita na eeprom
void definicoes_escrever_eeprom(unsigned char posicao);
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                          DEFINICOES.C                                                                      //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "definicoes.h"

// Declaração de variáveis globais de todos os ficheiros
unsigned char amp_aux;
unsigned char volt_aux;
unsigned char flag_alarm;
unsigned char contador;
unsigned char adc[2];
unsigned char sensor;
unsigned long CRISTAL ;
unsigned int FREQ_PWM ;
unsigned int LIM1_INF ;
unsigned int LIM2_INF ;
unsigned int LIM3_INF ;
unsigned int LIM1_SUP ;
unsigned int LIM2_SUP ;
unsigned int LIM3_SUP ;
unsigned int PULSOS ;
unsigned int vel_des;
unsigned char dir_des;
unsigned char flag_motor;
unsigned char dados[39]={
/*0*/                   0x00,   /*default 0x00  endereço motor */
/*1*/                   0x01,   /*default 0x01  1"        presença do motor */
/*2*/                   0x08,   /*default 0x08  8"        KP*/
/*3*/                   0x04,   /*default 0x04  4"        KI*/
/*4*/                   0x00,   /*default 0x00  0"        KD*/
/*5*/                   0x05,   /*default 0x05  intervalo da parte derivativa */
/*6*/                   0x0A,   /*default 0x0A  10"       erro integral máximo */
/*7*/                   0x8C,   /*default 0x8C  amostragem pid timer_1_hi */
/*8*/                   0x2D,   /*default 0x2D  3ms"      amostragem pid timer_1_lo*/
/*9*/                   0x96,   /*default 0x96  15A x 10          corrente limite  / 10*/
/*10*/                  0x6A,   /*default 0x6A  17 / 160mv        tensão limite */
/*11*/                  0x32,   /*default 0x32  50C       temperatura ponte limite */
/*12*/                  0x3C,   /*default 0x3C  60C       temperatura motor limite */
/*13*/                  0x32,   /*default 0x32  50 x 10 pulsos *10*/
/*14*/                  0x00,   /*default 0x00  lim1_inf MSB*/
/*15*/                  0x75,   /*default 0x75  (142-25)"         lim1_inf LSB*/
/*16*/                  0x02,   /*default 0x02  "lim2_inf MSB "*/
/*17*/                  0xF6,   /*default 0xF6  (808-50)"         lim2_inf LSB*/
/*18*/                  0x05,   /*default 0x05  lim3_inf MSB*/
/*19*/                  0xFF,   /*default 0xFF  (1635-100)"      lim3_inf LSB*/
/*20*/                  0x00,   /*default 0x00  "         lim1_sup MSB*/
/*21*/                  0xA7,   /*default 0xA7  (142+25)"         lim1_sup LSB*/
/*22*/                  0x03,   /*default 0x03  lim2_sup MSB*/
/*23*/                  0x5A,   /*default 0x5A  (808+50)"         lim2_sup LSB*/
/*24*/                  0x06,   /*default 0x06  lim3_sup MSB*/
/*25*/                  0xC7,   /*default 0xC7  (1635+100)"      lim3_sup LSB*/
/*26*/                  0x4C,   /*default 0x4C  freq_pwm MSB*/
/*27*/                  0x2C,   /*default 0x2C  19500Hz"         freq_pwm LSB*/
```

```
/*28*/                  0x27,   /*default 0x27  cristal * 1000 MSB*/
/*29*/                  0x10,   /*default 0x10  10000*1000"    cristal * 1000 LSB*/
/*30*/                  0x00,   /*default 0x00  break (0x01->bloqueado,0x00->livre)*/
/*31*/                  0x00,   /*default 0x00  velocidade desejada MSB*/
/*32*/                  0x00,   /*default 0x00  velocidade desejada LSB*/
/*33*/                  0x00,   /*default 0x00  velocidade actual MSB*/
/*34*/                  0x00,   /*default 0x00  velocidade actual MSB*/
/*35*/                  0x00,   /*default 0x00  corrente motor  / 10*/
/*36*/                  0x00,   /*default 0x00  tensão alimentação */
/*37*/                  0x00,   /*default 0x00  temperatura ponte H*/
/*38*/                  0x00,   /*default 0x00  temperatura motor */
                        };

// Função que calcula alguns parâmetros iniciais
void definicoes_calcular_parametros(void)
{

// Calcula a frequência do cristal
CRISTAL= ((unsigned long)join(dados[CRISTAL_H],dados[CRISTAL_L])*1000);

//Calcula a frequência do PWM
FREQ_PWM= join(dados[PWM_FREQ_M],dados[PWM_FREQ_L]);

//Calcula os limites para o ENCODER
LIM1_INF = join(dados[LIM1_INF_M],dados[LIM1_INF_L]);
LIM2_INF = join(dados[LIM2_INF_M],dados[LIM2_INF_L]);
LIM3_INF = join(dados[LIM3_INF_M],dados[LIM3_INF_L]);
LIM1_SUP = join(dados[LIM1_SUP_M],dados[LIM1_SUP_L]);
LIM2_SUP = join(dados[LIM2_SUP_M],dados[LIM2_SUP_L]);
LIM3_SUP = join(dados[LIM3_SUP_M],dados[LIM3_SUP_L]);

// Calcula o número de pulsos do encoder
PULSOS= ((unsigned int)dados[PPR_ENC]*10);
}

// Função de leitura da eeprom
void definicoes_ler_eeprom(void)
{
int z=0;

    for(z=0;z<31;z++)  // Copia as primeiras 31 posições da eeprom para o array dados[]
    {
    EEADR=z;
    EECON1bits.CFGS=0;
    EECON1bits.EEPGD=0;
    EECON1bits.RD=1;
    while(EECON1bits.RD==1);
    dados[z]=EEDATA;
    }
}

// Função que escreve numa posição especifica da eeprom o valor da mesma posição do array
dados[]
void definicoes_escrever_eeprom(unsigned char posicao)
{
```

```
    EEADR=posicao;
    EEDATA=dados[posicao];
    EECON1bits.CFGS=0;
    EECON1bits.EEPGD=0;
    INTCONbits.GIE=0;
    EECON1bits.WREN=1;
    EECON2=0x55;
    EECON2=0xAA;
    EECON1bits.WR=1;

        while(EECON1bits.WR==1);
        EECON1bits.WREN=0;
}
```

//

// ENCODER.h //

//

```
#include <p18f2431.h>
#include <math.h>
#include <delays.h>
#include "definicoes.h"


/* ---------------------------- Variáveis globais encoder ---------------------- */
extern unsigned char tmr5_overflow;
extern unsigned char vel_postscaler;
extern unsigned int new_vel;
extern unsigned long vel_calculo[4];
/* --------------- Fim de Declaração de Variáveis Globais ---------------- */


/* -------------------------- Declaração de Funções ---------------------- */
void calculoinicial(void);
void velocidade(void);
void encoder_config(void);
void activarleitura(void);
void my_delay(unsigned int valor);
void ajustar_escala(void);
unsigned int posicao(void);
int direccao(void);
void act_vel_des(void);
/* -------------------------- Fim de declaração de Funções --------------- */
```

//

// ENCODER.c //

//

```
#include "encoder.h"
// Calcula variável de 16 bits a partir de duas de 8bits
#define VELR (unsigned int)(((unsigned int)VELRH<<8)|(unsigned int)VELRL)
```

```
// Variáveis relativas à medição de velocidade
unsigned char tmr5_overflow;
unsigned char vel_postscaler;
unsigned int new_vel;
unsigned long vel_calculo[4];
unsigned int tempo;

void activarleitura(void);

//Função que gera um delay com "valor" ms
void my_delay(unsigned int valor)
{       int j;
        for (j=0;j<valor;j++)
        {Delay10KTCYx(1);}
}

//Função que calcula os parâmetros constantes utilizados no cálculo da velocidade em Rpm
void calculoinicial(void)
{       // calcula o tempo máximo que pode demorar a ler um pulso
        tempo= (unsigned int)((30000/PULSOS)+2) ;
        // calculo inicial para os vários prescalers e postscalers
        vel_calculo[0]=(unsigned long)((CRISTAL*15)/(PULSOS*4));
        vel_calculo[1]=(unsigned long)((CRISTAL/PULSOS)*120);
        vel_calculo[2]=(unsigned long)((CRISTAL/PULSOS)*480);
        vel_calculo[3]=(unsigned long)((CRISTAL/PULSOS)*1920);
}

//Função que calcula a velocidade real a partir dos tempo dos níveis lógicos medidos
void velocidade(void)
{
        unsigned int temp;
        // verifica se a velocidade é 0 Rpm (timer atinge o overflow)
        if (tmr5_overflow==0)           // se não for calcula a velocidade
                new_vel=(unsigned int)((vel_calculo[vel_postscaler])/(VELR+1));
        else{ // se o timer atingir o overflow então a velocidade real toma o valor 0
                tmr5_overflow=0;
                new_vel=0;
                // é aguardado o tempo necessário à ocorrência de uma medição à velocidade
                mínima de 1 Rpm
                my_delay(tempo);
        }
        temp=new_vel;
                if (QEICONbits.UP==0){ // insere a direcção no sinal da variável temp
                temp=((~new_vel)+1);
                }
                // insere a variável temp nas posições correspondentes à velocidade real no
                array dados[]
                dados[VEL_ACT_M]=(unsigned char)(temp>>8);
                dados[VEL_ACT_L]=(unsigned char)(temp);
}
// função de configuração da leitura de encoders

void encoder_config(void)
{
```

```
// chama a função que calcula os parâmetros constantes utilizados no cálculo da velocidade em
Rpm
calculoinicial();

// coloca o postscaler com o valor 0 pois o motor parte do repouso
// Postscaler responsável pela mudança da base de tempo da velocidade
vel_postscaler=0;
//inicializa  a velocidade real
new_vel=0;

// Coloca o pino A2 (INDX), A3 (QEA), A4 (QEB)  digital
ANSEL0 = ANSEL0 & 0b11100011;
// Coloca o pino A2 (INDX), A3 (QEA), A4 (QEB) como entrada
 TRISA = TRISA | 0b00011100;

//Configuração do módulo QEI

QEICONbits.VELM=1;          //manter a leitura de velocidade inactiva
QEICONbits.ERROR=0;         // apagar flag de overflow do contador de posição
QEICONbits.UP=1;            // colocar a flag de direcção para a direita
// colocar modo 2x update mode e reset quando (POSCNT=MAXCNT)
QEICONbits.QEIM2= 0;
// colocar modo 2x update mode e reset quando (POSCNT=MAXCNT)
QEICONbits.QEIM1= 1;
// colocar modo 2x update mode e reset quando (POSCNT=MAXCNT)
QEICONbits.QEIM0= 0;
QEICONbits.PDEC1=0;         // inicializar o postscaler de pulso a 1:1
QEICONbits.PDEC0=0;         // inicializar o postscaler de pulso a 1:1

//Input Capture configuration
//desactivar input1 capture e activar o reset do timer 5 após leitura de velocidade
CAP1CON = 0b01000000;
CAP2CON = 0b00000000;       //desactivar input2
CAP3CON = 0b00000000;       //desactivar input3

VELRL = 0x00;               //coloca o registo do tempo medido com o valor 0
VELRH = 0x00;

POSCNTL =0x00;              // coloca o registo de posicionamento do encoder com o valor 0
POSCNTH =0x00;

// registo que contem o número de pulsos do encoder
MAXCNTH = (int)((PULSOS)/128);
MAXCNTL = (2*PULSOS)-(MAXCNTH*256);

//configurar filtros
DFLTCON = 0b00000000;       // desactivar todos os filtros de entrada do encoder

//configurar timer5
//timer 5 T5SEN/RESEN/T5MOD/T5PS1/T5PS0/T5SYNC/TMR5CS/TMR5ON
T5CON = 0b01011000;
PR5L =0xFF;                 // prescaler do timer 5
PR5H =  0xFF;               // prescaler do timer 5
TMR5L = 0x00;               // registos do de contagem do timer 5
TMR5H = 0x00;               // registos do de contagem do timer 5
```

```
//configurar interrupção do timer 5
PIR3bits.TMR5IF = 0;            // apagar flag timer 5
PIE3bits.TMR5IE = 1;            // activar a interrupção do timer 5
IPR3bits.TMR5IP = 0;            // definir a interrupção do timer 5 com de alta prioridade
activarleitura();               // activar a leitura de velocidade
}

// função que activa a leitura da velocidade
void activarleitura(void)
{    T5CONbits.TMR5ON=1;   //activa timer5
     QEICONbits.VELM=0;    // activar leitura de velocidade
     my_delay(tempo);  // espera o tempo necessário à ocorrência de pelo menos uma leitura
}

// função responsável pelo ajuste do prescaler e postscaler na leitura de velocidades
void ajustar_escala(void)
{
          if (new_vel<=LIM1_INF && vel_postscaler!=0)
          {vel_postscaler=0;QEICONbits.PDEC1=0;QEICONbits.PDEC0=0;T5CONbits.T5PS1
=1;T5CONbits.T5PS0=1;}
          if (new_vel<=LIM2_INF && new_vel>LIM1_SUP && vel_postscaler!=1)
          {vel_postscaler=1;QEICONbits.PDEC1=0;QEICONbits.PDEC0=1;T5CONbits.T5PS1
=0;T5CONbits.T5PS0=0;}
          if (new_vel<=LIM3_INF && new_vel>LIM2_SUP && vel_postscaler!=2)
          {vel_postscaler=2;QEICONbits.PDEC1=1;QEICONbits.PDEC0=0;T5CONbits.T5PS1
=0;T5CONbits.T5PS0=0;}
          if (new_vel>LIM3_SUP && vel_postscaler!=3)
          {vel_postscaler=3;QEICONbits.PDEC1=1;QEICONbits.PDEC0=1;T5CONbits.T5PS1
=0;T5CONbits.T5PS0=0;}
}

// função que devolve a posição em que se encontra o encoder
unsigned int posicao(void)
{return (POSCNTL+(POSCNTH*256));
}

// função que devolve a direcção do encoder
int direccao(void)
{return  ((QEICON & 0b00100000)>>5);
}
// função que actualiza a velocidade desejada a partir do array dados[]
void act_vel_des(void)
{unsigned char bitt;
bitt= testbit(dados[VEL_DES_M],7);
          // verifica se a velocidade desejada é para esquerda ou direita actualizando as variaveis
          dir_des e vel_des
          if(bitt){
          dir_des=0;
          vel_des=~join(dados[VEL_DES_M],(dados[VEL_DES_L]-1));
          }else{
          dir_des=1;
          vel_des=join(dados[VEL_DES_M],dados[VEL_DES_L]);
          }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                  I2C.h                                                                       //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <p18f2431.h>
#include "definicoes.h"

/* ---------------------------- Variáveis Globais I2C---------------------- */
extern unsigned char i2cstat;
extern unsigned char end_dados;
extern unsigned char flag_end;
extern unsigned char n_bytes;
/* --------------- Fim de Declaração de Variáveis Globais ---------------- */

/* ------------------------- Declaração de Funções ---------------------- */
void i2c_slave_write(void);
void i2c_slave_interrupt(void);
void i2c_slave_config(void);
char i2c_slave_read(void);
/* ------------------------- Fim de declaração de Funções --------------- */

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                  I2C.c                                                                       //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "i2c_slave.h"

// Variaveis relativas à comunicação I2C

unsigned char i2cstat;
unsigned char end_dados;
unsigned char flag_end;
unsigned char n_bytes;
unsigned char flag_err;

// função para escrita no barramento I2C
void i2c_slave_write(void)
{
        while (SSPSTATbits.BF==1);
        do
        {
                SSPCONbits.WCOL=0;
                SSPBUF=dados[end_dados];
        }while(SSPCONbits.WCOL==1);

        SSPCONbits.CKP=1;

}
// função de descodificação do estado de leitura ou escrita
void i2c_slave_interrupt(void)
{

int i;
```

```
i2cstat= SSPSTAT;
i2cstat &= 0b00101101; // aplicação de uma mascara para retirar as flags  úteis

switch(i2cstat){
   //STATE 1
   // Modo de escrita, último byte escrito foi um endereço e o buffer está cheio
   case 0b00001001:{
                        // unicamente limpa o buffer sspbuf os dados são lixo
                        end_dados=SSPBUF;
                        flag_err = 1;
                     };break;
//STATE 2
// Modo de escrita, último byte escrito foi um dado e o buffer está cheio

case 0b00101001: {
                     flag_err = 1;
                     switch(flag_end){

                          case 0:{
                                   //Recebe endereço de escrita ou leitura no array
                                   end_dados=SSPBUF;
                                    flag_end++;
                                   };break;

                          case 1:{
                                   // Recebe o numero de bytes que vai escrever
                                   n_bytes=SSPBUF;
                                   FLAG_PROBLEM=n_bytes;
                                   flag_end++;
                                   };break;

                          default:{
                                   // guarda o byte recebido no array
                                   dados[end_dados]=SSPBUF;
                                   flag_end++;
                                   end_dados++;
                                   // verifica se leu todos os bytes
                                     if (flag_end>=(n_bytes+2)){
                                     // se sim coloca novamente o endereço no
                                     último byte escrito
                                     end_dados--;

                                     // se os parâmetros são de configuração escreve na eeprom
                                       if (end_dados<31){
                                            for (i=0;i<n_bytes;i++){
                                            definicoes_escrever_eeprom(end_dados);
                                            end_dados--;
                                            }
                                       }
                                       flag_end=0;
                                       n_bytes=0;
                                     }
                                   };break;
                        }
                     };break;
```

```
        //STATE 3
        //Modo de leitura, último byte escrito foi um endereço e o buffer está vazio
        case 0b00001100: {
                        flag_end=0;
                        i2c_slave_write(); // escreve no barramento
                        flag_err = 0;
                        };break;

        //STATE 4
        //Modo de leitura, último byte lido foi um dado e o buffer está vazio
        case 0b00101100:{
                        end_dados++;
                        i2c_slave_write(); // escreve no barramento
                        flag_err = 0;
                        };break;
        //STATE 5
        //NACK foi recebido quando estavam a ser enviados dados para o master
        case 0b00101000:{flag_err = 1;};break;

        default: {      //Erro na comunicação
                        if (flag_err==1) {
                        FLAG_PROBLEM=1;
                        while(1)
                        {clrwtd();}
                        }
                };break;
        }
}

// função de configuração da comunicação I2C
void i2c_slave_config(void)
{
    TRIS_FLAG_PROBLEM=0;
    FLAG_PROBLEM=0;
    flag_end=0;
    n_bytes=0;
    SSPCON=0x36;
    SSPADD=dados[MTR_ADRS]; //Endereço da placa controladora
    SSPSTAT=0x00;
    TRISCbits.TRISC5=1;
    TRISCbits.TRISC4=1;

    PIR1bits.SSPIF = 0; // apagar flag de interrupção i2c slave
    PIE1bits.SSPIE = 1; // activar a interrupção do i2c slave
    IPR1bits.SSPIP = 1; // definir a interrupção do i2c slave como de alta prioridade
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                  PID.h                                                                     //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <p18f2431.h>
#include "definicoes.h"
```

```
/* ---------------------------- Variaveis Globais PID ---------------------- */

extern unsigned char count_derivative;  //contador de cálculos acumulados já feitos
extern unsigned short long pid_out;     //16 bits saída PID
extern unsigned int new_error;          //16 bits erro
extern unsigned int integ_error;        //16 bits erro integral
extern unsigned int old_error;          //16 bits erro anterior
extern unsigned int delta_error;        //16 bits old_error - new error
extern unsigned short long prop_value;  //24 bits valor erro proporcional
extern unsigned short long integ_value; //24 bits valor erro integral
extern unsigned short long deriv_value; //24 bits valor erro derivativo


//11 bits de status do PID
extern  struct {
            unsigned new_err_z:1;
            unsigned integ_err_z:1;
            unsigned delta_err_z:1;
            unsigned new_err_sign:1;
            unsigned integ_err_sign:1;
            unsigned old_err_sign:1;
            unsigned delta_err_sign:1;
            unsigned pid_out_sign:1;
            unsigned prop_value_sign:1;
            unsigned integ_value_sign:1;
            unsigned deriv_value_sign:1;
} PID_STATUSbits;

/* --------------- Fim de Declaração de Variáveis Globais ---------------- */


/* ------------------------- Declaração de Funções ---------------------- */
void pid_inicializar(void);
void pid_main(void);
void proporcional(void);
void integral(void);
void derivativo(void);
void pid_resultado(void);
void get_integ_error(void);
void get_delta_error(void);
void pid_interrupt(void);
unsigned char sum_24bits(unsigned char sign_1,unsigned  char sign_2,unsigned short long
arg_1,unsigned short long arg_2,unsigned short long *result);
unsigned char sub_24bits(unsigned char sign_1,unsigned char sign_2,unsigned short long
arg_1,unsigned short long arg_2,unsigned short long *result);
unsigned char sum_16bits(unsigned char sign_1,unsigned  char sign_2,unsigned int
arg_1,unsigned int arg_2,unsigned int *result);
unsigned char sub_16bits(unsigned char sign_1,unsigned char sign_2,unsigned int
arg_1,unsigned int arg_2,unsigned int *result);
/* ------------------------- Fim de declaração de Funções --------------- */
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                  PID.c                                                                       //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "pid.h"

//declaração de variáveis do PID

unsigned char count_derivative;         //contador de cálculos derivativos já feitos
unsigned short long pid_out;            //24 bits saída PID
unsigned int new_error;                 //16 bits erro
unsigned int integ_error;               //16 bits erro acumulado
unsigned int old_error;                 //16 bits erro anterior
unsigned int delta_error;               //16 bits erro diferencial

unsigned short long prop_value;         //24 bits valor erro proporcional
unsigned short long integ_value;        //24 bits valor erro integral
unsigned short long deriv_value;        //24 bits valor erro derivativo

//11 bits status do PID
  struct {
  unsigned new_err_z:1;
  unsigned integ_err_z:1;
  unsigned delta_err_z:1;
  unsigned new_err_sign:1;
  unsigned integ_err_sign:1;
  unsigned old_err_sign:1;
  unsigned delta_err_sign:1;
  unsigned pid_out_sign:1;
  unsigned prop_value_sign:1;
  unsigned integ_value_sign:1;
  unsigned deriv_value_sign:1;
} PID_STATUSbits;

// função de configuração do PID
void pid_inicializar(void)
{
// inicializa erros com o valor 0
new_error=0;
integ_error=0;
old_error=0;
delta_error=0;
prop_value=0;
integ_value=0;
//coloca a zero a saída do PID
pid_out=0;

// Inicializa o contador entre cálculos derivativos
count_derivative = dados[DIV_INTV];

// Inicializa as flags de zero e sinal
PID_STATUSbits.new_err_z=0;         //Se zero = 1
PID_STATUSbits.integ_err_z=1;
PID_STATUSbits.delta_err_z=1;
```

```
PID_STATUSbits.new_err_sign=1;      //Positivo = 1
PID_STATUSbits.old_err_sign=1;
PID_STATUSbits.delta_err_sign=1;
PID_STATUSbits.integ_err_sign=1;
PID_STATUSbits.prop_value_sign=1;
PID_STATUSbits.integ_value_sign=1;
PID_STATUSbits.deriv_value_sign=1;

// Configura as interrupções do timer 1
PIR1bits.TMR1IF=0;
PIE1bits.TMR1IE=1;
IPR1bits.TMR1IP=0;

// Configura o timer 1 e inicializa o intervalo de tempo
T1CON=0b00000001;
TMR1H=dados[AMSTR_INTV_H];
TMR1L=dados[AMSTR_INTV_L];
}

//Função que calcula as partes proporcional, integral, derivativa e o resultado final do PID
void pid_main(void)
{
// se o erro actual for zero, o algoritmo PID não é executado
  if(new_error==0) {
    PID_STATUSbits.new_err_z=1;    // flag de zero do erro actual é activa
    }
    // erro actual não é zero
    else{
        //actualiza a flag de zero do erro actual
        PID_STATUSbits.new_err_z=0;
        proporcional();          // calcula parte proporcional
        integral();              // calcula parte integral
        derivativo();            // calcula parte derivativa
        pid_resultado();         // calcula resultado do algoritmo PID
        }
}

// calculo da parte proporcional
void proporcional(void)
{
//parte proporcional = Kp*erro actual
prop_value= new_error*dados[KP];
//actualiza a flag de sinal da parte proporcional
PID_STATUSbits.prop_value_sign=PID_STATUSbits.new_err_sign;
}

// calculo da parte integral
void integral(void)
{//verifica se o erro acumulado = 0
  if (PID_STATUSbits.integ_err_z==1){
    // se for 0 a parte integral toma o valor 0
    integ_value=0;
    PID_STATUSbits.integ_value_sign=1;
    }
    // se não for 0, calcula parte integral
```

```
    else{
                //parte integral = Ki* erro acumulado
                integ_value= dados[KI]*integ_error;
                //actualiza a flag de sinal da parte integral
                PID_STATUSbits.integ_value_sign=PID_STATUSbits.integ_err_sign;
        }
}
// calculo da parte derivativa
void derivativo(void)
{       //verifica se o erro diferencial = 0
        if (PID_STATUSbits.delta_err_z==1){
        // se for 0 a parte derivativa toma o valor 0
        deriv_value=0;
        PID_STATUSbits.deriv_value_sign=1;
        }
        // se não for 0 calcula parte derivativa
        else{
                //parte derivativa = Kd* erro diferencial
                deriv_value=dados[KD]*delta_error;
                //actualiza a flag de sinal da parte derivativa
                PID_STATUSbits.deriv_value_sign=PID_STATUSbits.delta_err_sign;
        }
}

// função que calcula o resultado do algoritmo PID
void pid_resultado(void)
{
//pid_out= prop_value+integ_value
PID_STATUSbits.pid_out_sign=sum_24bits(PID_STATUSbits.prop_value_sign,PID_STATU
Sbits.integ_value_sign,prop_value,integ_value,&pid_out);
//pid_out+=deriv_value
PID_STATUSbits.pid_out_sign=sum_24bits(PID_STATUSbits.pid_out_sign,
PID_STATUSbits.deriv_value_sign,pid_out,deriv_value,&pid_out);
// aplica a escala 128 para 1 de forma a diminuir a acção do resultado, evitando o uso de
variáveis float
pid_out>>=7; //128->1
}

// Função que calcula o erro acumulado
void get_integ_error(void)
{
//integ_error+=new_error
PID_STATUSbits.integ_err_sign=sum_16bits(PID_STATUSbits.integ_err_sign,PID_STATUS
bits.new_err_sign,integ_error,new_error,&integ_error);
// caso o erro acumulado seja zero activa a flag de zero do erro acumulado
        if (integ_error==0)
        PID_STATUSbits.integ_err_z=1;
        //Caso contrário, verifica se o erro acumulado não ultrapassou o máximo, se ultrapassou
        este toma o valor máximo
        else{
        integ_error=integ_error>dados[INTEG_MAX]?dados[INTEG_MAX]:integ_error;
        PID_STATUSbits.integ_err_z=0;
        }

}
```

```
// Função que calcula o erro diferencial
void get_delta_error(void)
{
//delta_error=new_error-old_error
PID_STATUSbits.delta_err_sign=sub_16bits(PID_STATUSbits.new_err_sign,PID_STATUSb
its.old_err_sign,new_error,old_error,&delta_error);
PID_STATUSbits.delta_err_z=delta_error==0?1:0;
}

// função que chama as funções que calculam o erro acumulado e diferencial
void pid_interrupt(void)
{       // se o erro actual não for 0
        if(PID_STATUSbits.new_err_z==0){
        get_integ_error(); // calcula erro acumulado
        count_derivative--; //decrementa o contador do erro diferencial
                if (count_derivative==0){       // caso contador igual a 0
                get_delta_error();              // calcula erro diferencial
                count_derivative=dados[DIV_INTV]; //inicializa novamente o contador
                }
        }
}

// função que soma duas variáveis de 24 bits com sinal separado
unsigned char sum_24bits(unsigned char sign_1,unsigned char sign_2,unsigned short long
arg_1 ,unsigned short long arg_2,unsigned short long *result)
{
        if (sign_1==sign_2){
                *result=arg_1+arg_2;
                return sign_1;
        }else{
                if(arg_1>=arg_2)
                {*result=arg_1-arg_2;
                 return sign_1;
                }else{
                 *result=arg_2-arg_1;
                 return sign_2;
                }
        }
}

// função que subtrai duas variáveis de 24 bits com sinal separado
unsigned char sub_24bits(unsigned char sign_1,unsigned char sign_2,unsigned short long
arg_1,unsigned short long arg_2,unsigned short long *result)
{

        if (sign_1!=sign_2){
                *result=arg_1+arg_2;
                return sign_1;
        }else{
                if(arg_1>=arg_2)
                {*result=arg_1-arg_2;
                 return sign_1;
                }else{
                 *result=arg_2-arg_1;
                 return ~sign_2;
```

```
                }
        }
}

// função que soma duas variáveis de 16 bits com sinal separado
unsigned char sum_16bits(unsigned char sign_1,unsigned  char sign_2,unsigned int
arg_1,unsigned int arg_2,unsigned int *result)
{
        if (sign_1==sign_2){
                *result=arg_1+arg_2;
                return sign_1;
        }else{
                if(arg_1>=arg_2)
                {*result=arg_1-arg_2;
                 return sign_1;
                }else{
                 *result=arg_2-arg_1;
                 return sign_2;
                }
        }
}

// função que subtrai duas variáveis de 24 bits com sinal separado
unsigned char sub_16bits(unsigned char sign_1,unsigned char sign_2,unsigned int
arg_1,unsigned int arg_2,unsigned int *result)
{
        if (sign_1!=sign_2){
                *result=arg_1+arg_2;
                return sign_1;
        }else{
                if(arg_1>=arg_2)
                {*result=arg_1-arg_2;
                 return sign_1;
                }else{
                 *result=arg_2-arg_1;
                 return ~sign_2;
                }
        }
}
```

//

// PWM.h //

//

```
#include <p18f2431.h>
#include "definicoes.h"

/* ---------------------------- Variaveis Globais PWM ---------------------- */
extern unsigned char pwm_direccao;
extern unsigned int pwm_duty_cycle;
extern unsigned int pwm_max_value;
/* --------------- Fim de Declaração de Variáveis Globais ---------------- */
```

```
/* ------------------------- Declaração de Funções ---------------------- */
void pwm_calculo_inicial(void);
void pwm_dir_e_duty_cycle(void);
void pwm_base_de_tempo(void);
void pwm_config(void);
/* ------------------------- Fim de declaração de Funções --------------- */

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                PWM.c                                                                   //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "pwm.h"

// variáveis utilizadas no módulo PWM
unsigned char pwm_direccao;
unsigned int pwm_duty_cycle;
unsigned int pwm_max_value;


// Função que calcula o máximo valor do duty cycle
void pwm_calculo_inicial(void)
{
pwm_max_value *=4;
pwm_max_value -=10;
}

// função que actualiza o valor do duty cycle e direcção desejada
void pwm_dir_e_duty_cycle(void)
{       OVDCONS= 0x00;
        OVDCOND= 0xff;

// caso o duty cycle seja maior que 100% ou menor que zero fica com os valor  de 100% e 0
respectivamente.
pwm_duty_cycle=pwm_duty_cycle>pwm_max_value?(pwm_max_value+14):pwm_duty_cycle;
// ajusta os sinais gerados em função da direcção e duty cycle
    if (pwm_direccao==0)
    {
        PDC1L = 0x00;
        PDC1H = 0x00;
        PDC0L = (unsigned char)pwm_duty_cycle;
        PDC0H = (unsigned char)(pwm_duty_cycle>>8);

    }else{
                PDC0L = 0x00;
                PDC0H = 0x00;
                PDC1L = (unsigned char)pwm_duty_cycle;
                PDC1H = (unsigned char)(pwm_duty_cycle>>8);
         }
}
// função que calcula a base de tempo dependendo da frequência de oscilação do pwm
void pwm_base_de_tempo(void)
{       pwm_max_value =(unsigned int)((CRISTAL/FREQ_PWM)-1);
        PTPERL = (unsigned char)pwm_max_value;
        PTPERH = (unsigned char)(pwm_max_value>>8);
```

```
}

// função de configuração do modulo PWM
void pwm_config(void)
{
pwm_base_de_tempo();          // calcula a base de tempo
pwm_calculo_inicial();        // calcula o valor max para o duty cycle
PTCON0=0x00;                        //
PTCON1=0x80;                         //
PTMRL=0x00;                          //
PTMRH=0x00;                          ///____  Configurações do pwm
PWMCON0=0b00110000;                  ///
PWMCON1=0x00;                        //
DTCON=0b00011110;                    //
FLTCONFIG=0x00;                     //
// inicializa uma direcção e  coloca o duty cycle a zero
pwm_direccao = 1;
pwm_duty_cycle=0;
pwm_dir_e_duty_cycle();       // actualiza os valores do duty cycle e direcção
}
```

//

// SENSORES.h //

//

```
#include <p18f2431.h>
#include "definicoes.h"
#include "encoder.h"
#include <sw_i2c.h>
#include <delays.h>

/* ------------------------- Declaração de Funções ---------------------- */
void sensor_ler(void);
void sensor_config(void);
void sensor_alarme(void);
/* ------------------------- Fim de declaração de Funções --------------- */
```

//

// SENSORES.c //

//

```
#include "sensor.h"
#include <stdio.h>

// função de leitura de sensores
void sensor_ler(void)
{unsigned char temp[2];

        switch (sensor)
        {
```

```
        // Lê o valor de corrente e tensão dos ADCs
        case 0:{
                dados[AMP_ACT]=255-(unsigned char)(adc[0]);
                dados[VOLT_ACT]=(unsigned char)adc[1];
                };break;

        // Lê o sensor de temperatura do motor
        case 1:{
                SWStartI2C();
                SWPutcI2C(TMP_MOTOR); // control byte
                SWAckI2C();
                SWPutcI2C(0x00); // word address
                SWAckI2C();
                SWRestartI2C();
                SWPutcI2C(TMP_MOTOR|0b00000001); // control byte
                SWAckI2C();
                SWGetsI2C(&temp[0],2);
                SWStopI2C();
                dados[TEMP_M_ACT]=temp[0];
                };break;

        // Lê o sensor de temperatura da ponte H
        case 2:{
                SWStartI2C();
                SWPutcI2C(TMP_PONTE); // control byte
                SWAckI2C();
                SWPutcI2C(0x00); // word address
                SWAckI2C();
                SWRestartI2C();
                SWPutcI2C(TMP_PONTE|0b00000001); // control byte
                SWAckI2C();
                SWGetsI2C(&temp[0],2);
                SWStopI2C();
                dados[TEMP_P_ACT]=temp[0];
                };break;

        // verifica caso a velocidade seja acima de 100Rpm se o motor roda
        case 3:{
                if(vel_des>100 && new_vel==0)
                flag_motor++;
                else
                flag_motor=0;
                };break;
        default:;break;
      }
}

// função de configuração dos sensores
void sensor_config(void)
{
        // configura o sensor de temperatura da ponte H
        SWStartI2C();
        SWPutcI2C(TMP_PONTE); // control byte
        SWAckI2C();
        SWPutcI2C(0x01); // word address
```

```
        SWAckI2C();
        SWPutcI2C(0x00); // data
        SWAckI2C();
        SWStopI2C();

        // configura o sensor de temperatura do motor
        SWStartI2C();
        SWPutcI2C(TMP_MOTOR); // control byte
        SWAckI2C();
        SWPutcI2C(0x01); // word address
        SWAckI2C();
        SWPutcI2C(0x00); // data
        SWAckI2C();
        SWStopI2C();

        // inicializa variáveis do módulo sensores
        amp_aux=0;
        volt_aux=0;
        flag_alarm=0;
        contador=0;
        adc[0]=0;
        adc[1]=0;
        sensor=0;
        TRIS_FAN =0;
        FAN =0;

        // configurar o timer 0 que gera o intervalo de tempo entre leituras dos sensores
        T0CON=0b10000111;
        TMR0L=TMR0_L;
        TMR0H=TMR0_H;
        INTCON2bits.TMR0IP=0;
        INTCONbits.TMR0IF=0;
        INTCONbits.TMR0IE=1;

        // configura interrupções do ADCs
        PIE1bits.ADIE=1;
        IPR1bits.ADIP=0;
        PIR1bits.ADIF=0;

        // configura ADCs
        ADCON0=0B00111001;
        ADCON1=0B00010000;
        ADCON2=0B01111110;
        ADCON3=0B10000000;
        ADCHS=0B00000000;
        ANSEL0=0B00000011;
        Delay10TCYx( 50 );          //Delay for 50TCY
        // activa a leitura dos ADC
        ADCON0bits.GO=1;            //começa a conversão
}

// Função que verifica os dados lidos pelos sensores e actua caso necessário
void sensor_alarme(void)
{int i=0;
        switch (sensor) {
```

```
// sensor de corrente e tensão
 case 0:{
// verifica 3 vezes se a corrente está acima da corrente limite
if (dados[AMP_ACT]>dados[AMP_LIM])
amp_aux++;
else
amp_aux=0;
//se a corrente ultrapassar a corrente limite
      if (amp_aux>=3) {
         INTCONbits.TMR0IE=0; // a leitura dos sensores pára
         dados[VEL_DES_M]=0;  // a velocidade desejada fica com o valor 0
         dados[VEL_DES_L]=0;
         PONTE_H_EN=1;          // a ponte H é inactiva
         //acciona a flag de alarme de modo que o algoritmo PID não seja executado
         flag_alarm=1;
         WTD=0;                      // desactiva o wacthdog
         // é enviado um sinal visual durante 10 segundos
           for (i=0;i<5;i++)
           {
            FLAG_PROBLEM=1;            //LED pisca 3 vezes
            Delay10KTCYx(200);
            FLAG_PROBLEM=0;
            Delay10KTCYx(200);
            FLAG_PROBLEM=1;
            Delay10KTCYx(200);
            FLAG_PROBLEM=0;
            Delay10KTCYx(200);
            FLAG_PROBLEM=1;
            Delay10KTCYx(200);
            FLAG_PROBLEM=0;
            Delay10KTCYx(255);
            Delay10KTCYx(255);
            Delay10KTCYx(255);
            Delay10KTCYx(255);
            }
      }

 // verifica 5 vezes se a tensão está abaixo da tensão limite
 if (dados[VOLT_ACT]<dados[VOLT_LIM])
volt_aux++;
else
volt_aux=0;
// se a tensão estiver abaixo da tensão limite
   if (volt_aux>=5)
   { volt_aux=0;
    INTCONbits.TMR0IE=0; // a leitura dos sensores pára
   dados[VEL_DES_M]=0;   // a velocidade desejada fica com o valor 0
   dados[VEL_DES_L]=0;
   PONTE_H_EN=1;           // a ponte H é inactiva
   //acciona a flag de alarme de modo que o algoritmo PID não seja executado
   flag_alarm=1;
   WTD=0;                      // desactiva o wacthdog
   // é enviado um sinal visual durante 10 segundos
         for (i=0;i<5;i++)
         {
```

```
                    FLAG_PROBLEM=1;                 //pisca 4 vezes
                    Delay10KTCYx(140);
                    FLAG_PROBLEM=0;
                    Delay10KTCYx(140);
                    FLAG_PROBLEM=1;
                    Delay10KTCYx(140);
                    FLAG_PROBLEM=0;
                    Delay10KTCYx(140);
                    FLAG_PROBLEM=1;
                    Delay10KTCYx(140);
                    FLAG_PROBLEM=0;
                    Delay10KTCYx(140);
                    FLAG_PROBLEM=1;
                    Delay10KTCYx(140);
                    FLAG_PROBLEM=0;
                    Delay10KTCYx(255);
                    Delay10KTCYx(255);
                    Delay10KTCYx(255);
                    Delay10KTCYx(255);
                    }
                }
            // após os 10 segundos
            WTD=1;              // é activo o wacthdog
            flag_alarm=0;       // é limpa a flag de problema
            FLAG_PROBLEM=0;     // apaga o sinal visual
            INTCONbits.TMR0IE=1;    // é activa a leitura dos sensores
};break;

//sensor de temperatura do motor
case 1:{
// enquanto que a temperatura do motor excede a temperatura limite:

    while(dados[TEMP_M_ACT]>dados[TEMP_M_LIM])
    {INTCONbits.TMR0IE=0; // a leitura dos sensores pára
    dados[VEL_DES_M]=0;    // a velocidade desejada fica com o valor 0
    dados[VEL_DES_L]=0;
    PONTE_H_EN=1;                 // a ponte H é inactiva
    //acciona a flag de alarme de modo que o algoritmo PID não seja executado
     flag_alarm=1;
    WTD=0;                        // desactiva o wacthdog
    // é enviado um sinal visual
    FLAG_PROBLEM=1;       //pisca 2 vezes
    Delay10KTCYx(170);
    FLAG_PROBLEM=0;
    Delay10KTCYx(170);
    FLAG_PROBLEM=1;
    Delay10KTCYx(170);
    FLAG_PROBLEM=0;
    Delay10KTCYx(255);
    Delay10KTCYx(255);
    sensor=1;           // lê o sensor de temperatura do motor
    sensor_ler();
    }
//quando a temperatura for menor ou igual a temperatura limite
WTD=1;                  // é activo o wacthdog
```

```
flag_alarm=0;                   // é limpa a flag de problema
FLAG_PROBLEM=0;                 // apaga o sinal visual
INTCONbits.TMR0IE=1;            // é activa a leitura dos sensores
};break;

// sensor de temperatura da ponte H
case 2:{
// enquanto que a temperatura da ponte H excede a temperatura limite:
    while(dados[TEMP_P_ACT]>dados[TEMP_P_LIM])
    {INTCONbits.TMR0IE=0;           // a leitura dos sensores pára
    dados[VEL_DES_M]=0;             // a velocidade desejada fica com o valor 0
    dados[VEL_DES_L]=0;
    PONTE_H_EN=1;                   // a ponte H é inactiva
    //acciona a flag de alarme de modo que o algoritmo PID não seja executado
    flag_alarm=1;
    WTD=0;              // desactiva o wacthdog
     FAN=1;             //Liga a ventilação
    // é enviado um sinal visual
     FLAG_PROBLEM=1;    // pisca 1 vez lento
     Delay10KTCYx(255);
     Delay10KTCYx(255);
     FLAG_PROBLEM=0;
     Delay10KTCYx(255);
     Delay10KTCYx(255);
    sensor=2;
    sensor_ler(); // lê o sensor de temperatura da ponte H
    }
        // se faltar menos de 20 graus para temperatura limite é accionada a ventilação
        if (dados[TEMP_P_ACT]>(dados[TEMP_P_LIM]-20))
        FAN=1;
        else
        FAN=0;
        //quando a temperatura for menor ou igual a temperatura limite
        WTD=1;                  // é activo o wacthdog
        flag_alarm=0;           // é limpa a flag de problema
        FLAG_PROBLEM=0;         // apaga o sinal visual
        INTCONbits.TMR0IE=1;    // é activa a leitura dos sensores
};break;

// presença do motor
case 3:{
// se a velocidade desejada for superior a 100Rpm e a velocidade actual for 0
if (flag_motor==2){
dados[MTR_FLAG]=0;// é activa a flag a avisar um problema com o motor no array dados[]
WTD=0;                  // é desactivo o watchdog
// enquanto a o problema não for reparado e a flag colocada a 0 no areio dados[]
    while(dados[MTR_FLAG]==0)
    {INTCONbits.TMR0IE=0;           // a leitura de sensores pára
    dados[VEL_DES_M]=0;             // a velocidade desejada é colocada a 0
    dados[VEL_DES_L]=0;
    PONTE_H_EN=1;                           // a ponte H é inactiva
    // acciona a flag de alarme de modo que o algoritmo PID não seja executado
    flag_alarm=1;
    //sinal visual indica o problema ocorrido
    FLAG_PROBLEM=1;                 // pisca 1 vez flash
```

```
      Delay10KTCYx(30);
      FLAG_PROBLEM=0;
      Delay10KTCYx(255);
      Delay10KTCYx(225);
      }
      flag_motor=0;
}
// quando o problema for reparado volta a funcionar
WTD=1;                  // é activo o wacthdog
flag_alarm=0;           // é limpa a flag de problema
FLAG_PROBLEM=0;         // apaga o sinal visual
INTCONbits.TMR0IE=1;    // é activa a leitura dos sensores
};break;

default:;break;

}
}
```

Anexo III – Código C do conversor RS232- I2C

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                    SERIAL_TO_I2C.c                                                                         //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#if defined(__PCM__)
#include <16f873A.h>
#include <stdlib.h>
#include <string.h>
#fuses HS,WDT,NOPROTECT,NOBROWNOUT,NOPUT       // bits de configuração
#use delay(clock=4000000)                      // configurar delay com cristal de 4 Mhz
#pragma use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)   // configurar porta série
#endif

#define I2C_SDA  PIN_C4                                  // configurar pinos I2C
#define I2C_SCL  PIN_C3


#use i2c(master, sda=I2C_SDA, scl=I2C_SCL,slow)      // configurar comunicação i2c

#define LED_SERIE_RX PIN_C2                          // configurar pinos de leds tx e rx
#define LED_SERIE_TX PIN_C5

// declaração da função de envio e recepção para o barramento i2c
void i2c_tx(unsigned char x);

// variaveis usadas na conversão rs232-i2c
unsigned char flag_serie_rx;
unsigned char flag_serie_tx;
unsigned char trama[5]="";
unsigned char trama_aux[5]="";
unsigned char aux;
```

```
unsigned char endereco_disp;
unsigned char funcao;
unsigned char endereco;
unsigned char valor_0;
unsigned char valor_1;

// interrupção da comunicação rs232
#INT_RDA
void rx_isr()
{
flag_serie_tx=1;
aux=getc();
// descodificação da trama recebida
        switch(aux) {
                case ';' :
                {endereco_disp= (unsigned char)atoi(trama);
                strcpy(trama,"");
                } break;

                case ':' :
                {funcao= (unsigned char)atoi(trama);
                 strcpy(trama,"");
                } break;

                case ',' :
                {endereco= (unsigned char)atoi(trama);
                strcpy(trama,"");
                } break;

                case '-' :
                {valor_0= (unsigned char)atoi(trama);
                strcpy(trama,"");
                } break;

                case '.' :
                {valor_1= (unsigned char)atoi(trama);
                strcpy(trama,"");
                i2c_tx(0);
                } break;

                case '+' :
                {valor_0= (unsigned char)atoi(trama);
                strcpy(trama,"");
                i2c_tx(1);
                } break;

                default: {sprintf(trama_aux,"%c",aux);
                 strcat(trama,trama_aux);
                 }; break;
        }

}
// função de envio e recepção para o barramento i2c
void i2c_tx(unsigned char x)
```

```
{
// enviar dados
if (funcao==0)
{
        i2c_start();
        i2c_write(endereco_disp);
        i2c_write(endereco);
        // envio de 2 byte
                if(x==0){
                i2c_write(2);
                i2c_write(valor_0);
                i2c_write(valor_1);
                // envio de 1 byte
                }else{
                i2c_write(1);
                i2c_write(valor_0);
                }
                i2c_stop();
        // receber dados
        }else{
                flag_serie_rx=1;
                i2c_start();
                i2c_write(endereco_disp);
                i2c_write(valor_1);
                i2c_start();
                i2c_write(endereco_disp | 0x01);
                valor_0 = i2c_read(1);
                valor_1 = i2c_read(0);
                i2c_stop();
                printf("%u;%u.",valor_0,valor_1);
        }
}

// função que gera um delay
void my_delay(unsigned char time)
{
#asm
DECF time,1
DECF time,1
nop
nop
loop:
        nop
        nop
        DECFSZ time,1
        goto loop
#endasm
}

//inicializar comunicação I2C
void init_I2C()
{
  output_float(I2C_SCL);
  output_float(I2C_SDA);
}
```

```
// função main()
void main()
{
// declaração de variaveis
unsigned char array[10];

//iniciarlização de variaveis
array[0]=10;
flag_serie_rx=0;
flag_serie_tx=0;

output_bit(LED_SERIE_TX, 1);
my_delay(array[0]);

// configurar watchdog
setup_wdt (WDT_144MS);

setup_adc_ports (NO_ANALOGS );
delay_ms(50);
// inicializa comunicação I2c
init_I2C();
// configura intterrupções
enable_interrupts(GLOBAL);
enable_interrupts(INT_RDA);

output_bit(LED_SERIE_TX, 0);

          while(TRUE)
          { // faz reset watchdog
          restart_wdt();
          // liga o led tx ou rx em função de leitura ou escrita ocorrida
                    if (flag_serie_rx==1){
                    output_bit(LED_SERIE_RX, 1);
                    delay_ms (5);
                    output_bit(LED_SERIE_RX, 0);
                    flag_serie_rx=0;
                    }
                    if (flag_serie_tx==1){
                    output_bit(LED_SERIE_TX, 1);
                    delay_ms (5);
                    output_bit(LED_SERIE_TX, 0);
                    flag_serie_tx=0;
                    }
          }
}
```

Anexo IV – Código C das funções de envio e recepção para o conversor USB-I2C

Interface gráfico


```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                   I2C_Read                                   //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int C_I2C_Converter::comando_I2C_R(unsigned char endereco, unsigned char registo, int
nrBytes, unsigned char *valor)
{
int status;
unsigned char aenviar[4];
CString erro;
aenviar[0]= 0x55;
aenviar[1]=( endereco | 0x01 );
aenviar[2]=registo;
aenviar[3]= (unsigned char) nrBytes;

        status=port.Write(&aenviar,4);          // envia array aenviar  pela porta série
        port.Flush();

        if(status<0)            // erro na escrita da porta serie
        {
        erro.Format("%d",status);
        AfxMessageBox("comando_I2C_W:erro ao escrever na porta serie | status="+erro);
        return 0;
        }
        // buffer da porta série
        BYTE* pBuf = new BYTE[1000];
        int L=0;
        CString str,aux;
        try
        {
                port.Set0ReadTimeout();
                L=port.Read(pBuf,1000);
```

```
                pBuf[L]='\0';
                // leitura do buffer da porta série
                for(int j=0;j<L;j++)
                {
                        aux.Format("%d",pBuf[j]);
                        valor[j]=atoi(aux);

                }

                if(L==0)
                AfxMessageBox("Timeout");
        }
        catch (CSerialException* pEx) // Erro na leitura da porta série
        {
                AfxMessageBox(pEx->GetErrorMessage());
                pEx->Delete();
        }
        // elimina o buffer da porta série
        delete [] pBuf;
        port.ClearReadBuffer();

        return 1;
}
```

//

// I2C_Write //

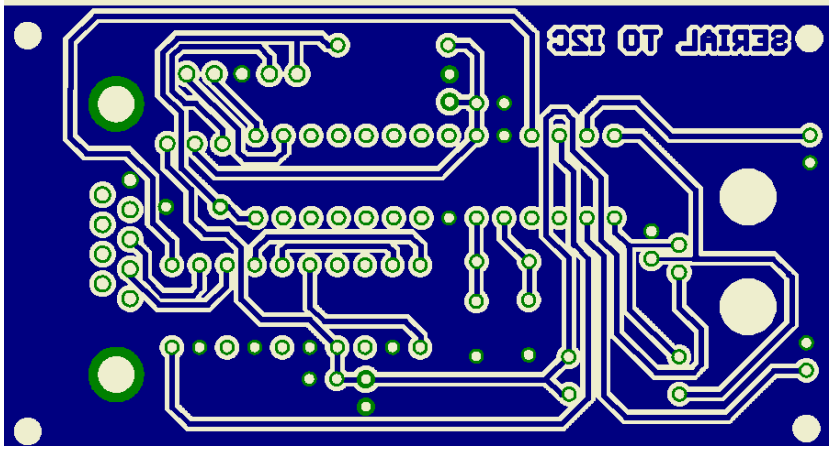
//

```
int C_I2C_Converter::comando_I2C_W(unsigned char endereco, unsigned char registo, int
nrBytes, unsigned char *comando)
{
int status;
CString erro;
unsigned char aenviar[60];

// só é possivel enviar 60 bytes de cada vez
if(nrBytes>=60 || nrBytes<=0 )
{
AfxMessageBox("comando_I2C_W:Nr de Bytes a enviar por i2c tem que ser 0<i2c<60");
return 0;
} else{ //constroi a trama a enviar
      aenviar[0]= 0x55;
      aenviar[1]=endereco;
      aenviar[2]=registo;
      aenviar[3]=(unsigned char)(nrBytes+1);
      aenviar[4]=(unsigned char)nrBytes;
      int i=0;
           for(i;i<nrBytes;i++)
           {
           aenviar[i+5]=comando[i];
           }
          status=port.Write(&aenviar,7); // envia a trama para porta série
          port.Flush();
```

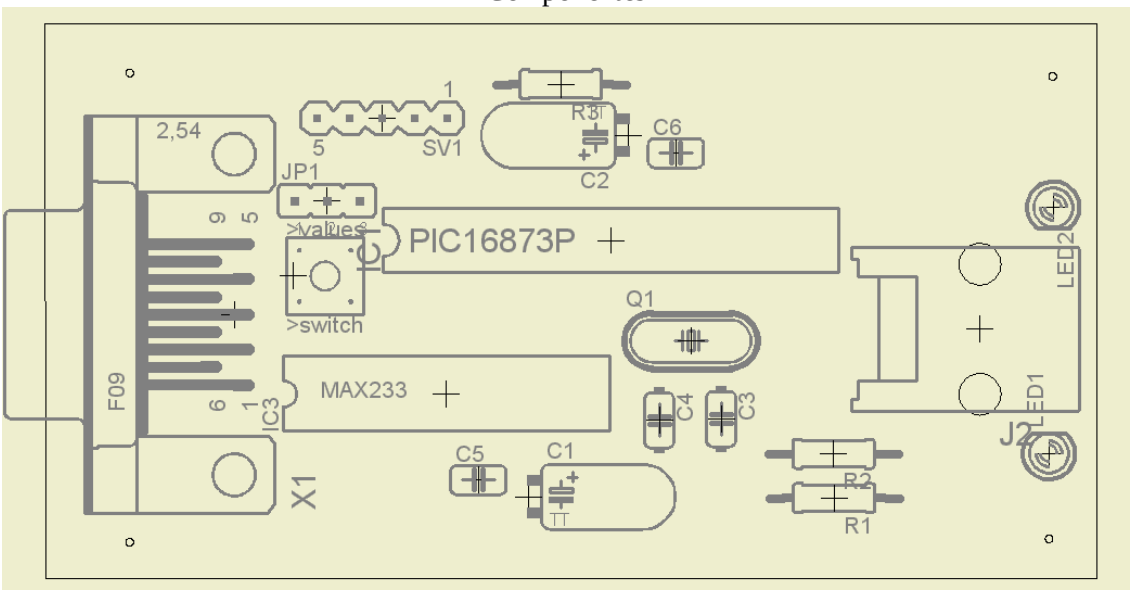
```
        // erro na escrita da porta série
        if(status<0)
        {
        erro.Format("%d",status);
        AfxMessageBox("comando_I2C_W:erro ao escrever na porta serie | status="+erro);
        return 0;
        }       // aguarda confirmação de envoi bem sucessido por parte do conversor usb I2C
                BYTE* pBuf = new BYTE[1000];
                int L=0;
                CString str,aux;
                try
                {
                        port.Set0ReadTimeout();
                        L=port.Read(pBuf,1000);
                        pBuf[L]='\0';

                        for(int j=0;j<L;j++)
                        {
                                aux.Format("%c",pBuf[j]);
                                str+=aux;
                        }

                        status=atoi(str);
                        if(L==0)
                        AfxMessageBox("Timeout");
                }
                catch (CSerialException* pEx)
                {
                        AfxMessageBox(pEx->GetErrorMessage());
                        pEx->Delete();
                }

                delete [] pBuf;
                port.ClearReadBuffer();
        return 1;
        }
}
```

Anexo V – PCB e circuito do conversor RS232-I2C

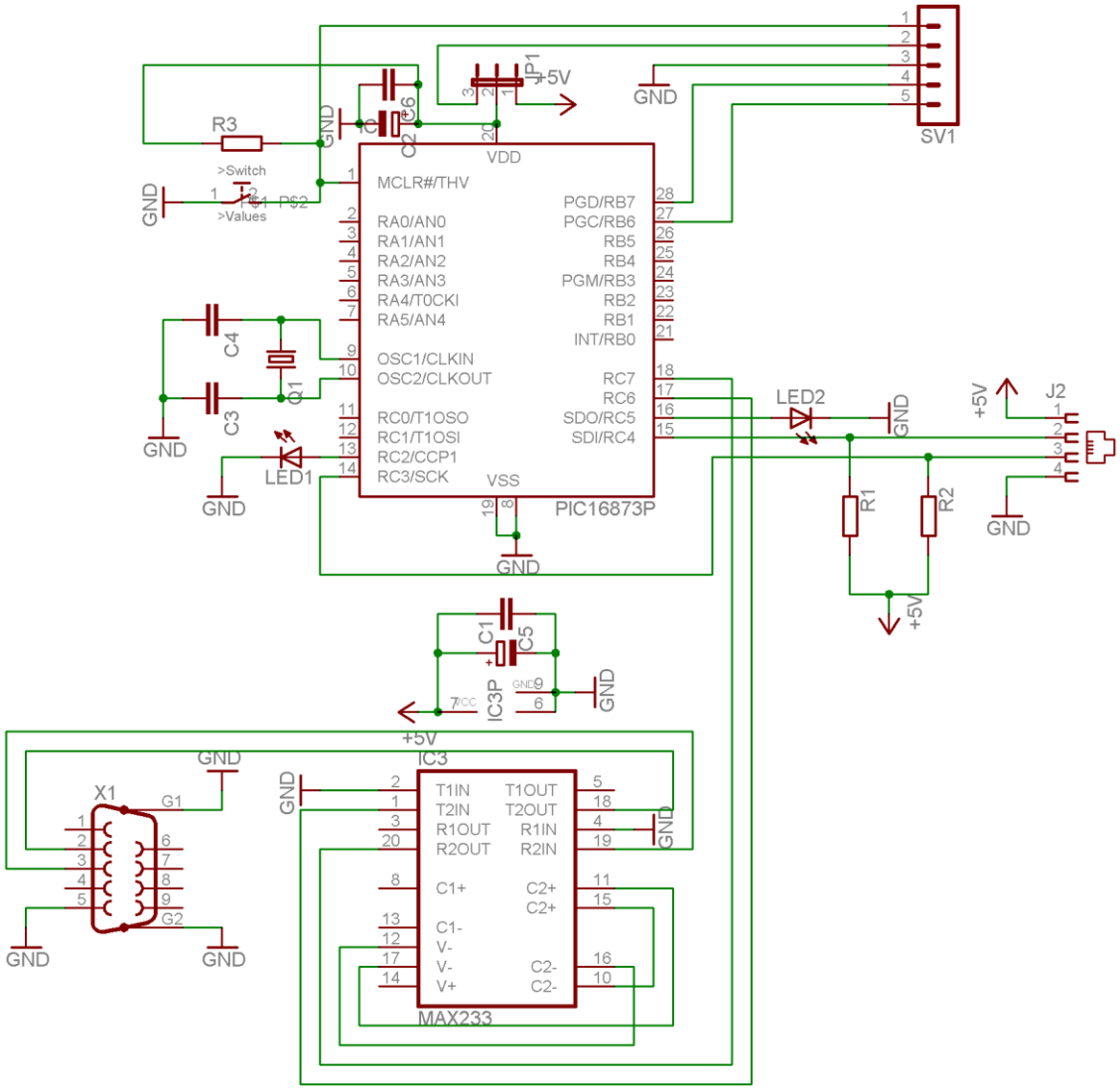
PCB



Componentes



Circuito


SV1
P1
+5V
GND
R3
C6
C2
VDD
>Switch
>Values
P$1
P$2
MCLR#/THV
RA0/AN0
RA1/AN1
RA2/AN2
RA3/AN3
RA4/T0CKI
RA5/AN4
OSC1/CLKIN
OSC2/CLKOUT
RC0/T1OSO
RC1/T1OSI
RC2/CCP1
RC3/SCK
PGD/RB7
PGC/RB6
RB5
RB4
PGM/RB3
RB2
RB1
INT/RB0
RC7
RC6
SDO/RC5
SDI/RC4
VSS
PIC16873P
C4
C3
Q1
LED1
LED2
R1
R2
J2
C1
C5
IC3P
IC3
X1
G1
G2
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
C1+
C2+
C1-
V-
V+
C2-
MAX233