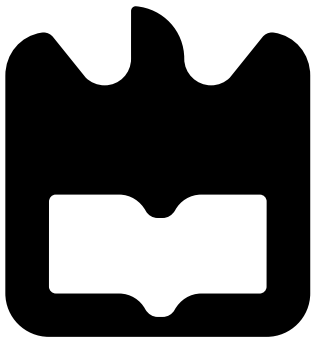
Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
2015

Hugo Fernandes Araújo

Demonstrador para rede tempo-real HaRTES



Hugo
Fernandes
Araújo

Demonstrador para rede tempo-real HaRTES

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Paulo Bacelar Reis Pedreiras, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Professor Doutor Valter Filipe Miranda Castelão da Silva, Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda.

Este trabalho é financiado por Fundos Nacionais através da FCT – Fundação para a Ciência e a Tecnologia no âmbito do projecto Serv-CPS -PTDC/EEA-AUT/122362/2010.

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA

o júri / the jury

Presidente / president	**Professor Doutor José Alberto Gouveia Fonseca** Professor Associado da Universidade de Aveiro
Arguente Principal / Main Examiner	**Professor Doutor Frederico Miguel do Céu Marques dos Santos** Professor Adjunto do Instituto Superior de Engenharia de Coimbra
Orientador / Advisor	**Professor Doutor Paulo Bacelar Reis Pedreiras** Professor Auxiliar da Universidade de Aveiro

agradecimentos

É com um enorme prazer que aproveito esta oportunidade para agradecer a todas as pessoas que estiveram envolvidas de uma forma direta ou indireta na realização desta dissertação. A todas elas expresso a minha gratidão, contudo agradeço particularmente:

Ao Professor Paulo Pedreiras, por me acompanhar de uma forma sempre atenta, pela disponibilidade, paciência e pelo incentivo. Agradeço também as sugestões e conhecimento técnico e científico transmitidos que foram essenciais para a realização desta dissertação.

Ao Professor Valter Silva pelo empenho, dedicação e pelas sugestões técnicas e científicas.

Aos meus pais, irmão e restante família por todo o apoio e preocupação que demonstraram durante todo o percurso académico e em toda a minha vida. Agradeço especialmente aos meus pais por me darem a oportunidade de estudar e pelo apoio incansável.

À minha namorada, Diana Capela, por todo o apoio, motivação e compreensão ao longo do meu percurso académico.

A todos os meus amigos que estiveram sempre presentes e pelo enorme espírito de camaradagem, com especial agradecimento ao Pedro Vilas-Boas, Rui Morais e Luís Videira por todo o apoio motivação e por tornarem o meu percurso académico mais harmonioso e enriquecedor.

Agradeço também ao Instituto de Telecomunicações (IT), pólo de Aveiro, por ter disponibilizado os recursos necessários para a realização desta dissertação.

Resumo

A utilização de sistemas embutidos distribuídos em diversas áreas como a robótica, automação industrial e aviónica tem vindo a generalizar-se no decorrer dos últimos anos. Este tipo de sistemas são compostos por vários nós, geralmente designados por sistemas embutidos. Estes nós encontram-se interligados através de uma infra-estrutura de comunicação de forma a possibilitar a troca de informação entre eles de maneira a concretizar um objetivo comum.

Por norma os sistemas embutidos distribuídos apresentam requisitos temporais bastante exigentes. A tecnologia *Ethernet* e os protocolos de comunicação, com propriedades de tempo real, desenvolvidos para esta não conseguem associar de uma forma eficaz os requisitos temporais das aplicações de tempo real aos requisitos Quality of Service (QoS) dos diferentes tipos de tráfego. O *switch* Hard Real-Time Ethernet Switching (HaRTES) foi desenvolvido e implementado com o objetivo de solucionar estes problemas devido às suas capacidades como a sincronização de fluxos diferentes e gestão de diferentes tipos de tráfego.

Esta dissertação apresenta a adaptação de um sistemas físico de modo a possibilitar a demonstração do correto funcionamento do sistema de comunicação, que será desenvolvido e implementado, utilizando um *switch* HaRTES como o elemento responsável pela troca de informação na rede entre os nós. O desempenho da arquitetura de rede desenvolvida será também testada e avaliada.

Abstract

The use of distributed embedded systems in several areas like robotics, industrial automation and avionics has become more common in current years. These type of system consists of several intelligent nodes, generally designated by embedded systems. This nodes are interconnected through a communication infrastructure in order to enable the exchange of information between them in order to achieve a common target.

In general the distributed embedded systems demonstrate highly demanding temporal constraints. Ethernet technology and its communication protocols, with real time properties, can't efficiently associate the temporal constraints of real time applications to the QoS demands of different types of traffic. The Hard Real-Time Ethernet Switching (HaRTES) was developed and implemented in order to solve these problems due to its capabilities as synchronization of different flows and management of different types of traffic.

This dissertation shows the adaptation of a physical system in order to enable the demonstration of the correct way of the communication system to work, which is going to be developed and implemented with a HaRTES switch as the responsible element of the exchange of information in the network between nodes.

The performance of the architecture of the developed network will also be tested and assessed.

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

PD	Proporcional derivativo
PID	Proporcional integrador e derivativo
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
LAN	Redes locais
MAC	Media access protocol
OSI	Open Systems Interconnection
EtherNet/IP	Ethernet Industrial Protocol
CIP	Common Industrial Protocol
CBA	Component Based Automation
M2M	Machine-to-machine
IO	Input Output
TTEthernet	Time-Triggered Ethernet
SW	Switched Ethernet
TT	Time-triggered
RC	Rate-constrained
BE	Best-effort
FTT	Flexible Time Triggered
FTT-SE	Flexible Time Triggered over Switched Ethernet
EC	Elementary Cycle
TM	Trigger Message

HaRTES	Hard Real-Time Ethernet Switching
FPGA	Field-programmable Gate Array
QoS	Quality of Service
RT	Tempo Real
IRT	Tempo Real Isócrono
RTOS	Real-Time Operating System
IP	Internet Protocol
PWM	Pulse-Width Modulation
UART	Universal Asynchronous Receiver Transmitter
ISR	Interrupt Service Routine
I2C	Inter-Integrated Circuit
COTS	Commercial Off-The-Shelf

Capítulo 1

Introdução

Nos dias de hoje, os sistemas ciber-físicos são utilizados nos mais diversos setores que envolvem o nosso quotidiano, tais como, automação industrial, medicina, energia e muitos outros. Estes sistemas caraterizados, por serem sistemas de tempo real e sistemas embutidos distribuídos, são constituídos por diversos nós inteligentes que comunicam entre si através de uma rede de modo a cumprir um objetivo global. Nesta dissertação é implementado um sistema ciber-físico, sendo utilizado e testado um *switch* HaRTES responsável pela troca de mensagens entre os nós na rede.

1.1 Motivação

No decorrer dos últimos anos, o uso de sistemas embutidos com o objetivo de gerir e controlar inúmeras aplicações e equipamentos tem vindo a generalizar-se [3]. Com o avanço da tecnologia o custo de desenvolvimento e produção destes sistemas é cada vez mais reduzido. Este avanço permite que os sistemas se tornem cada vez mais eficientes em termos de tamanho e consumo de energia, da mesma forma o aumento da capacidade computacional é notório assim como a capacidade de comunicação.

Nos sistemas de uso geral os equipamentos utilizados, como por exemplo, computadores pessoais, demonstram ser bastante versáteis e apresentam a capacidade e responsabilidade pela realização de diversas tarefas. Aos sistemas embutidos, por outro lado, é-lhes atribuída a responsabilidade de realizarem apenas uma tarefa ou um conjunto diminuto de tarefas. De uma maneira geral, os sistemas embutidos interagem de uma forma direta com atuadores e sensores. Têm como responsabilidade ler a informação proveniente dos sensores, processar e analisar essa mesma informação e por último acionar os atuadores conforme as especificações que lhe foram atribuídas.

A facilidade e custo reduzido da instalação, manutenção e uma tolerância mais flexível a falhas representam as vantagens da modularidade dos sistemas embutidos, que por sua vez contribuíram para o desenvolvimento

e implementação de sistemas embutidos distribuídos. Os diversos nós que constituem um sistema embutido distribuído encontram-se interligados através de uma infra-estrutura de comunicação de modo a possibilitar a partilha de informação como a coordenação de atividades. Cada nó tem as suas funções e responsabilidades distintas, mas ao mesmo tempo cooperam entre eles tendo como objetivo a concretização de uma finalidade comum.

A utilização de sistemas embutidos distribuídos, tem vindo a aumentar significativamente no decorrer dos últimos anos em diversas áreas como em automação industrial, indústria automóvel e aeroespacial [4]. Um dos principais motivos pela utilização de sistemas distribuídos nestas áreas é a capacidade de partilha de recursos na rede (*hardware* e *software*), dados e serviços. Desta forma é possível executar variadas tarefas em paralelo pelos diferentes nós traduzindo-se num melhor desempenho geral do sistema. Uma outra vantagem destes sistemas é a capacidade do sistema continuar a funcionar, por vezes com desempenho degradado, no caso de ocorrerem falhas ou erros nos seus subsistemas, visto que as falha ou erros ocorridos num determinado nó não implica que os restantes deixem de funcionar corretamente.

Geralmente os sistemas embutidos distribuídos apresentam requisitos temporais bastante exigentes. Na área de automação industrial, por exemplo, estes requisitos são estritamente necessários de modo a que o sistema apresente o funcionamento e desempenho desejado. As comunicações industriais representam comunicações de tempo real, pelo que a nível de qualidade de serviço são privilegiados os requisitos temporais. Para este tipo de comunicações os requisitos temporais caracterizam-se por *deadlines*, tempos de atraso e tempos de resposta.

A tecnologia *Ethernet*, juntamente com os protocolos de comunicação desenvolvidos para esta, é usada cada vez mais para interligar os sistemas embutidos pertencentes a sistemas distribuídos. Num sistema distribuído, os nós que o constituem interagem entre si constantemente, de modo que a comunicação feita entre eles deve ser garantida, segura e eficiente. O baixo custo de instalação e manutenção desta tecnologia é um dos motivos para a sua utilização ter vindo a crescer nos últimos tempos.

A *Ethernet* tradicional é na maior parte dos casos utilizada nas nossas casas e em escritórios. Para a implementação de aplicações de tempo real a *Ethernet standard* não garante a comunicação entre os nós assim como a eficiência e segurança dessa comunicação. Nos sistemas embutidos distribuídos existem três tipos de tráfego, designados de tráfego periódico, esporádico e aperiódico, que não são explicitamente suportados pela *Ethernet* tradicional visto que esta não consegue responder de uma forma adequada ao indeterminismo temporal. Mesmo com o desenvolvimento e implementação de diferentes protocolos de comunicação *Ethernet*, estes não conseguem associar de uma forma eficaz os requisitos temporais das aplicações de tempo real aos requisitos de QoS dos três tipos de tráfego. O *switch* HaRTES foi desenvolvido com o intuito de resolver estes problemas. Este *switch* é capaz

de solucionar os problemas mencionados anteriormente, visto que tem a capacidade de sincronização de diferentes fluxos, gestão dos diferentes tipos de tráfego e permite ainda o escalonamento do tráfego *online*.

Neste trabalho é utilizado um demonstrador que consiste num sistema físico de dois eixos. As capacidades do sistema de comunicação desenvolvido utilizando o *switch* HaRTES, para este demonstrador, serão testadas gerando tráfego periódico, esporádico e aperiódico, com requisitos de tempo real e sensível ao *jitter*. O tráfego gerado será usado para efetuar o controlo do sistema físico, sendo o *switch* HaRTES o elemento responsável pela gestão e sincronização do tráfego na rede.

Os testes que serão efetuados, para verificar o funcionamento do sistema de comunicação, serão realizados com um *switch Ethernet* comum da Cisco e um *switch* HaRTES. Desta forma será possível provar o devido funcionamento do sistema de comunicação e o desempenho do *switch* HaRTES em comparação com um *switch Ethernet* comum.

Além dos testes e validações do funcionamento do sistema de comunicação, com a utilização deste demonstrador, é possível verificar o funcionamento total do sistema de uma forma física e visual.

1.2 Objetivos

No âmbito desta dissertação, pretende-se testar o sistema de comunicação implementado, para o demonstrador mencionado anteriormente, e o desempenho do *switch* HaRTES comparativamente a um *switch* tradicional.

Para que o objetivo final seja atingido terão de ser percorridos os seguintes passos:

- Estudo dos conceitos fundamentais relativos a sistemas de controlo, sistemas ciber-físicos, protocolos de comunicação *Ethernet*.
- Estudo do *hardware* disponível para a realização do projeto.
- Desenvolvimento e implementação de *software* de controlo.
- Desenvolver e implementar aplicações para o sistema.
- Testes e validações.

1.3 Organização da dissertação

Esta dissertação é dividida em seis capítulos:

- **Capítulo 2**: são apresentados os conceitos fundamentais relativos a sistemas de controlo, *Ethernet* de tempo real, protocolos de comunicação e *switch* HaRTES.

- **Capítulo 3**: neste capítulo é apresentado o sistema físico a controlar assim como o *hardware* disponível para a realização desta tarefa.
- **Capítulo 4**: neste capítulo é apresentado o *software* desenvolvido e implementado para o controlo e a comunicação do sistema.
- **Capítulo 5**: apresenta o teste e validação do funcionamento do sistema. Os testes efetuados e a análise dos resultados são também apresentados neste capítulo.
- **Capítulo 6**: é efetuada uma análise ao trabalho efetuado e objetivos alcançados e são feitas algumas sugestões de trabalho futuro.

Capítulo 2

Conceitos Fundamentais

Neste capítulo serão introduzidos os conceitos fundamentais relativos a sistemas de controlo, sistemas ciber-físicos, *Ethernet* de tempo real e ao *switch* HaRTES.

2.1 Sistemas de Controlo

Um sistema de controlo define-se por ser um sistema que manipula um determinado elemento a fim de este obter o comportamento desejado [5].

Sistemas de controlo são frequentemente utilizados no nosso dia a dia. O próprio ser humano efetua ações que envolvem operações de controlo como, por exemplo, apanhar um objeto com a mão. É, neste caso, uma operação que envolve um movimento coordenado sendo este controlado pelo nosso cérebro em função das informações recebidas pelos nossos sentidos. Regular a temperatura de uma sala através de equipamento de ar condicionado é também um outro exemplo de um sistema de controlo que é usado no nosso quotidiano.

2.1.1 Modelo de um sistema de controlo

Existem duas formas de controlar um sistema, controlo em malha aberta e controlo em malha fechada.

Define-se um sistema de controlo em malha aberta como um sistema em que a saída não interfere com a entrada(figura 2.1[5]).

Este tipo de controlo é caraterizado por a sua saída não exercer nenhuma ação no sinal de controlo. O sinal de saída não é medido nem comparado com a referência, logo não afecta a malha de entrada. Este tipo de controlo só é utilizado em sistemas que se conhece a correspondência entre a saída e a entrada. São sistemas que geralmente não sofrem pertubações internas e externas significativas.

Figura 2.1: Sistema de controlo em malha aberta.

Em sistemas de controlo em malha fechada o sinal de saída é comparado com o sinal de entrada e consequentemente afeta o sinal de controlo, como se pode observar na figura 2.2. Um sistema de controlo em malha fechada define-se como um sistema em que o sinal de saída é adicionado ou subtraído algebricamente ao sinal de entrada, podendo ser considerada realimentação positiva ou negativa respectivamente[5]. O sinal resultante da operação algébrica entre o sinal de entrada e o sinal de saída é definido como o sinal de erro. Este tipo de controlo tem como objetivo reduzir o erro do sistema e manter na saída o valor pretendido pelo operador.

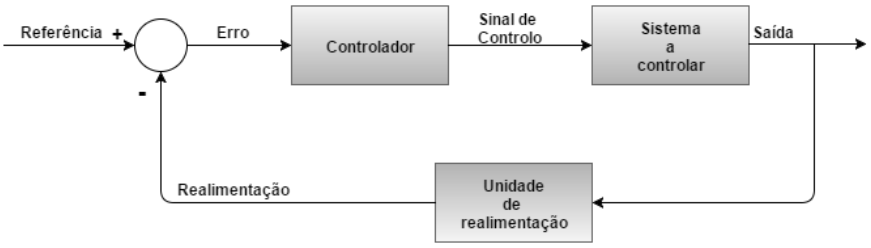


Figura 2.2: Modelo de blocos de um sistema de controlo realimentado.

Como se pode verificar na figura 2.2, sistema de controlo com realimentação negativa, existe um conjunto de elementos que são caraterísticos a um sistema de controlo, tais como:

- Referência: sinal que representa o objetivo a atingir.
- Erro: sinal resultante da diferença entre a referência e o sinal de realimentação.
- Sinal de controlo: sinal que atuará de uma forma direta no sistema a controlar.
- Realimentação: representa o sinal de saída usado para ser feito o cálculo do sinal de erro juntamente com o sinal de referência. A realimentação tem como objetivo tornar o sistema mais preciso e permite que

reaja a fatores externos. A unidade de realimentação nos casos mais simples é representada como um fio de ligação.

2.1.2 Controladores Elementares

Este sub-capítulo introduz um conceito geral de alguns controladores elementares realimentados.

Controlador ON-OFF

O controlador ON-OFF é considerado, entre os controladores realimentados, o mais fácil de implementar. Em contrapartida é também o controlador mais limitado. Neste tipo de controladores o sinal de erro é aplicado diretamente na entrada de um comparador cuja saída atua diretamente na entrada do sistema. O sinal de controlo toma o valor máximo ou o valor mínimo em função do sinal de erro. Se o erro for negativo significa que a saída é superior ao sinal de entrada, logo o sinal de controlo tomará o valor mínimo. Caso o sinal de erro for positivo, o sinal de saída é inferior ao sinal de entrada, o que leva o sinal de controlo a tomar o valor máximo.

Controlador Proporcional

Neste tipo de controladores o sinal de controlo é diretamente proporcional ao sinal de erro do sistema. Uma solução capaz de evitar as oscilações apresentadas na saída de um controlador ON-OFF, é a utilização de um controlador proporcional com um ganho Kp reduzido para baixos valores do erro.

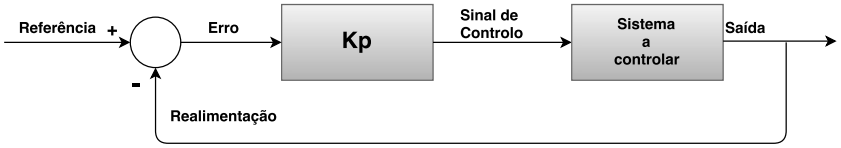


Figura 2.3: Controlador proporcional.

Controlador PD

O controlador PD destaca-se, em relação ao controlador proporcional, por possuir uma componente antecipativa correspondente à tendência do sinal de erro. Sendo capaz de antecipar estas ações, o controlador consegue reduzir significavelmente o *overshoot* e oscilações em geral do sistema.

Controlador PID

Nesta dissertação o controlo do sistema a implementar é feito em malha fechada usando controladores do tipo PID (figura 2.4[6]). Este tipo de controladores é o mais usado na indústria, sendo os principais motivos os seguintes [7]:

- Ter um desempenho robusto para uma grande variedade de condições de funcionamento.
- Simplicidade funcional, que permite operá-los diretamente de forma simples.
- É composto por três componentes ajustáveis (proporcional, integrador e derivativo).

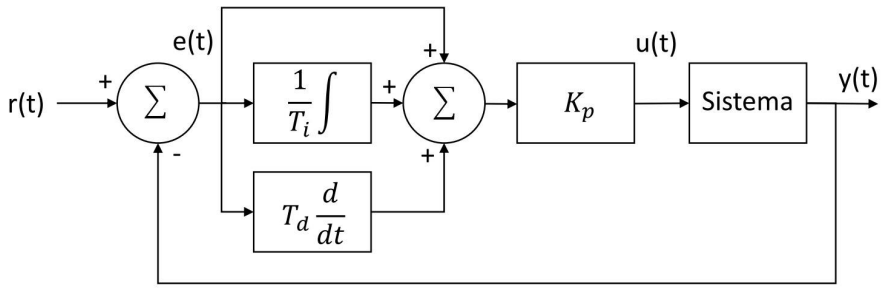


Figura 2.4: Contralador PID.

No domínio do tempo, a expressão do sinal de controlo, *u(t)*, é dada por:

$$u(t) = K_p\Big(e(t) + \frac{1}{T_i}\int e(t)\mathrm{d}t + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}\Big) \tag{2.1}$$

De modo a implementar o controlador, de uma forma direta, num computador digital é possível discretizar a equação 2.1[7] em:

$$u(k) = K_p\Big(e(k) + i(k-1) + \frac{h}{T_i}e(k) + T_d\frac{e(k)-e(k-1)}{h}\Big) \tag{2.2}$$

2.2 Sistemas ciber-físicos

Sistemas ciber-físicos caraterizam-se por ser sistemas computacionais que atuam e controlam sistemas físicos [8]. O sistema ciber-físico a implementar no âmbito desta dissertação é caraterizado por ser um:

- Sistema embutido e distribuído.

- Sistema de tempo real.

Assim serão introduzidos de seguida conceitos fundamentais relativos a sistemas de tempo real e a sistemas embutidos e distribuídos.

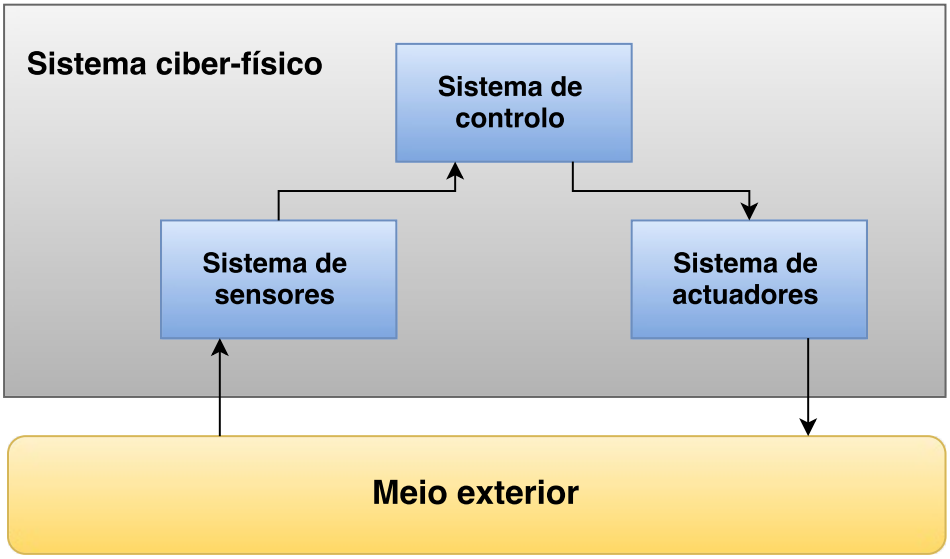


Figura 2.5: Sistema ciber-físico.

2.2.1 Sistemas embutidos e distribuídos

Um sistema embutido é caraterizado por ser um sistema programável, que tem como objetivo utilizar a sua capacidade computacional para a realização de uma determinada tarefa [9].

Nos dias de hoje com a boa performance e o baixo custo dos microcontroladores os sistemas de controlo mecânicos e electrónicos convencionais, que se encontram em vários produtos, estão-se a tornar obsoletos em comparação com sistemas embutidos. Geralmente um sistema embutido faz parte de um sistema maior bem especificado. Todas as aplicações embutidas devem ser implementadas usando a arquitetura adequada. Nesta dissertação foi usada uma arquitetura distribuída, ou seja, o sistema tem diferentes nós (dispositivos) interligados através de uma infra-estrutura de comunicação permitindo assim partilha de recursos e a coordenação de atividades(figura 2.6). Este tipo de sistemas apresenta caraterísticas vantajosas como:

- Comunicação por mensagens: não existem variáveis globais partilhadas.
- Falhas independentes: a falha de um dispositivo não impede necessariamente que os outros deixem de funcionar.

- Escalável: várias tarefas podem estar a ser executadas em paralelo nos diferentes dispositivos.
- Facilidade na adição e na remoção de nós.

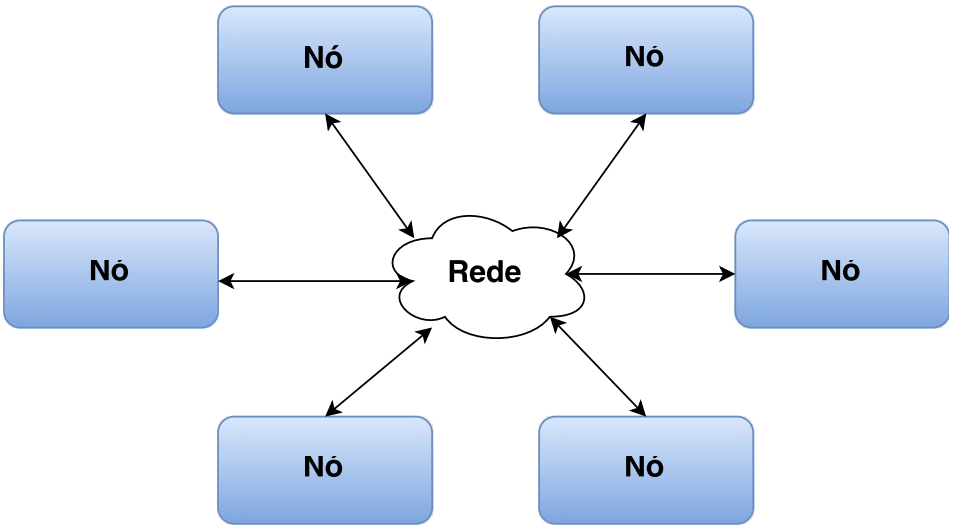


Figura 2.6: Sistema distribuído.

2.2.2 Sistema de tempo real

Sistema de tempo real é um sistema computacional em que o comportamento correto deste não depende apenas dos resultados gerados, mas também do instante de tempo em que são realizados [10].

Uma aplicação de tempo real é composta normalmente por várias tarefas, sendo estas classificadas como as unidades de processamento sequencial do sistema. As restrições temporais que definem estas aplicações têm de ser devidamente respeitadas para que o sistema se comporte da maneira pretendida.

As tarefas devem ser executadas dentro de um tempo limite que lhes é atribuído, designado de "*deadline*". Conforme as potenciais consequências em caso de incumprimento de *deadlines*, estas podem ser classificadas como:

- "*hard*": possibilidade de acontecerem falhas catastróficas no sistema, que pode levar a danos irreparáveis nos equipamentos ou em casos extremos perdas de vidas humanas.
- "*soft*": se não cumprir a *deadline* o resultado que lhe está associado ainda é útil.

- "*firm*": caso não cumpra a *deadline* o resultado que lhe está associado é considerado como inútil.

Outra caraterização, relativa às tarefas, pode ser feita com base nas suas frequências de ativação. Quando uma tarefa é ativada em intervalos regulares, chamados de períodos, são designadas por tarefas periódicas. Por outro lado, se a ativação de uma tarefa for feita em resposta a eventos internos ou externos podendo ocorrer aleatoriamente, as tarefas são definidas como tarefas aperiódicas.

A nível de segurança, este sistema pode ser classificado como:

- Sistemas não críticos de tempo real. Podem ocorrer falhas mas não desfavorecem os benefícios do sistema em operação normal.
- Sistemas críticos de tempo real. As falhas ocorrentes ultrapassam em larga escala os benefícios normais do sistema.

Os sistemas de tempo real apresentam os seguintes requisitos:

- temporais;
- funcionais;
- de dependabilidade.

Os requisitos temporais do sistema são designados como os intervalos de tempo máximo referentes aos atrasos de observação do estado do sistema e aos atrasos de computação dos novos valores de controlo [11]. A variação dos atrasos referidos anteriormente tem o nome de *jitter*.

Os principais requisitos funcionais que um sistema de tempo real deve obedecer são [12]:

- Controlo digital direto, o controlador tem acesso aos atuadores e sensores.
- Interação com o operador, fornecendo informações relativas ao estado do sistema e suporte à operação do sistema.
- Recolha de dados, capacidade de amostragem das variáveis do sistema.

Quanto aos requisitos de dependabilidade, estes implicam que o sistema de tempo real na presença de falhas ou erros se mantenha com um funcionamento correto.

2.3 *Ethernet* de tempo real

Ethernet é a tecnologia de comunicação mais usada na atualidade em redes locais(LAN) e muito usada em automação industrial como em muitas outras aplicações. Isto deve-se à robustez, fiabilidade, à extensa largura de banda, à quantidade de fabricantes, e aos preços reduzidos que esta tecnologia oferece. Esta tecnologia é constituída por *hardware* e *software* que operam em conjunto com a finalidade de haver troca de informação entre terminais(computadores)[13]. A tecnologia *Ethernet* é composta por quatro elementos básicos, quando combinados proporcionam um sistema funcional. Estes elementos são [14]:

- O pacote, que corresponde a uma sequência de bits normalizada, utilizado para enviar a informação ao destino pretendido.
- Protocolo MAC, que permite vários terminais a comunicarem num canal *Ethernet* de uma forma justa seguindo uma sequência de regras.
- Componentes de sinalização, ou seja, os dispositivos responsáveis pela troca de informação entre terminais numa rede *Ethernet*.
- Meio físico, ou seja, para a informação ser trocada via *Ethernet* entre os diferentes terminais na mesma rede é necessário um meio físico de "transporte"designadamente cabos e outro *hardware*.

Topologias de rede em anel ou em barramento foram as primeiras topologias usadas nos sistemas [13]. Estas topologias eram bastante limitadas e geravam muitos problemas. A simples adição ou remoção de um terminal à rede, cabos danificados, podiam comprometer todo o sistema. Por estas razões foram adotadas outras topologias como a topologia em estrela. Deste modo todos os terminais estariam conetados a um dispositivo central que se encarregaria pela troca de informação entre estes, facilitando significativamente, por exemplo, a adição ou remoção de terminais à rede. Inicialmente o dispositivo central era um repetidor(*hub*), ou seja, replicava o sinal de entrada em todas as suas saídas. A implementação do sistema com este dispositivo tinha uma grande desvantagem pois só era possível ter um único domínio de colisões pelo que quantos mais terminais estiverem ligados à rede maior seria a probabilidade de haver colisões. Por esta razão este dispositivo central(*hub*) foi substítuido por um novo elemento que pertence ao segundo nível do modelo OSI, designado como *switch*, de modo a ter vários domínios de colisões.

Na área de automação industrial existem certos requisitos que os processos industriais devem obdecer, como por exemplo, a existência de precedências temporais entre mensagens e a predictabilidade é favorecida em detrimento da taxa de transmissão média [15]. Existem vários protocolos *Ethernet* de tempo real que não só resolvem os problemas anteriormente

referidos mas também obedecem aos requisitos fundamentais mencionados para o bom funcionamento do sistema. De seguida serão introduzidos conceitos fundamentais sobre alguns desses protocolos.

2.3.1 EtherNet/IP

O protocolo EtherNet/IP é um dos protocolos de comunicação mais usados na automação industrial[16]. É designado por ser um Common Industrial Protocol (CIP) implementado sobre a *Ethernet* tradicional. A sua vantagem justifica-se pela compatibilidade e coexistência com várias aplicações e protocolos devido à sua implementação sobre a *Ethernet* tradicional e ao uso da tecnologia TCP/IP.

2.3.2 PROFINET

PROFINET, assim como o EtherNet/IP, é implementado sobre a *Ethernet* tradicional o que possibilita a sua utilização, por exemplo, em escritórios sem precisar de interfaces adicionais. Este protocolo permite também uma comunicação TCP/IP a decorrer em paralelo com a troca de dados feita habitualmente [17].

A troca de dados para elevadas exigências de velocidade pode ser feita de duas maneiras distintas pelo que existem duas versões de PROFINET, o PROFINET IO e o PROFINET CBA. Estas duas versões podem ser usadas separadamente ou em simultâneo se assim for pretendido.

O PROFINET IO tem como função conetar ao sistema todos os dispositivos IO (entrada e saída) que se encontram no ambiente envolvente, podendo serem conetados à *Ethernet*. A configuração, parametrização e a troca de dados entre os dispositivos é feita pelo PROFINET IO. PROFINET IO Tempo Real Isócrono (IRT) é usado para comunicações de tempo real do tipo *hard*. Neste tipo de comunicações em que o tempo de resposta é crítico é necessário *hardware* dedicado para que se consiga um tempo de resposta adequado [18].

O PROFINET CBA consiste na divisão do sistema em módulos individuais, sendo estes independentes entre si. Este protocolo é usado em comunicações Machine-to-machine (M2M) assim como em sistemas distribuídos de tempo real baseando-se na divisão do processos em subprocessos.

2.3.3 TTEthernet

Time-Triggered Ethernet(TTEthernet) define-se por ser um protocolo switched Ethernet(SW) de tempo real usado em automação industrial. Este protocolo é também utilizado em aplicações de segurança para transporte de dados. Isto deve-se ao elevado grau de segurança do método *time-triggered* [19]. Este método deteta falhas ou irregularidades no transporte de dados

na rede do sistema. A integração deste protocolo com os equipamentos de *Ethernert* tradicionais é outra caraterística vantajosa deste protocolo.

TTEthernet foi desenvolvido com o objetivo de permitir comunicações *time-triggered* em *Ethernet*, permitindo também que estas comunicações coexistam com o restante tráfego. Esta coexistência exige a presença de um controlador com a capacidade de sincronizar o tráfego da rede com os restantes elementos do sistema, *switches* e controladores.

O tráfego de informação que percorre a rede pode ser dividida em três classes: tráfego *rate-constrained*(RC), tráfego *time-triggered*(TT) e tráfego *best-effort*(BE) [20]. Destas três classes o tráfego TT representa a classe com a maior prioridade na rede visto ser a classe que exige uma elevada precisão temporal e um grande rigor no cumprimento de *deadlines* [21].

O tráfego RC, baseado no paradigma *event-triggered*, em comparação com o tráfego TT não exige o mesmo determinismo e rigor temporal que este, garante que as mensagens sejam enviadas dentro do seu *deadline* e apresenta atrasos limitados. Ao contrário do tráfego TT não é feita uma sincronização juntamente com o restante tráfego pelo que podem ser feitas comunicações em simultâneo para o mesmo ponto da rede.

Por último, o tráfego BE em comparação com o tráfego TT e RC é o que apresenta menor prioridade pelo que só são enviadas mensagens deste tipo quando existir largura de banda disponível. Esta classe não garante que a troca de informação seja feita com sucesso nem apresenta definições de atraso temporal [22].

2.3.4 FTT-SE

Flexible Time Triggered over Switched Ethernet (FTT-SE) é um protocolo *Ethernet* de tempo real baseado no paradigma Flexible Time Triggered (FTT) concebido para a execução de aplicações de tempo real reconfiguráveis e adaptáveis [14]. As vantagens deste protocolo apresentam-se como [23]:

- Gestão de diversos tipos de tráfego: periódicos, aperiódicos e esporádicos.
- Fluxos de informação atualizados automaticamente de uma forma rápida.
- Capacidade de coordenação da totalidade do tráfego na mesma linha temporal.
- Suporte de um número significativo de diferentes períodos dos fluxos.

A arquitetura do FTT-SE baseia-se na existência de dois tipos de nós, o nó *master* e os nós do tipo *slave*. O nó *master* é o nó encarregue da coordenação e do agendamento do tráfego dos *slaves*(figura 2.7 [1]).

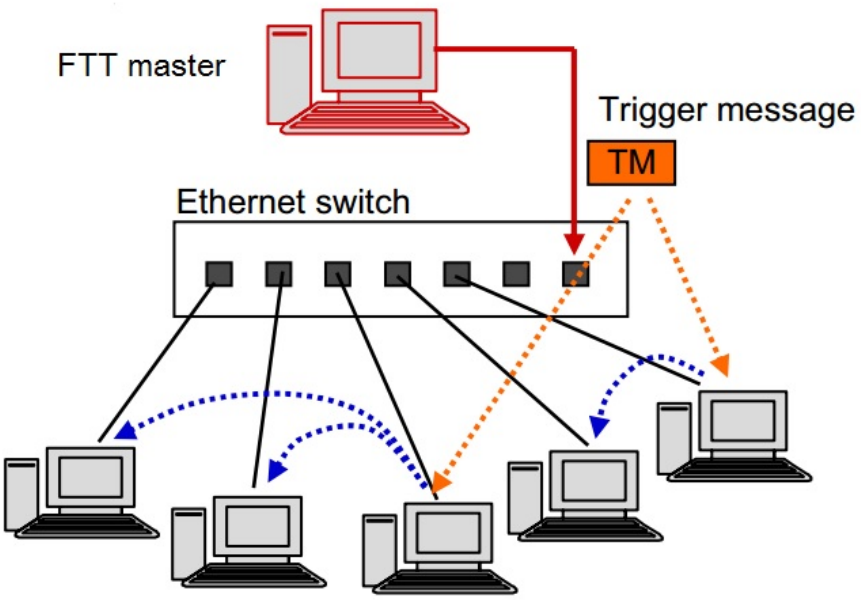


Figura 2.7: Arquitetura FTT-SE [1].

Esta arquitetura carateriza-se por ser uma arquitetura centralizada de agendamento, ou seja a previsibilidade do sistema é mantida devido ao controlo na totalidade do tráfego que percorre a rede evitando assim o sobrecarregamento dos *buffers* de entrada nos *switches*.

O funcionamento deste protocolo baseia-se no facto de as comunicações serem organizadas em tramas de duração temporal fixa, Elementary Cycle (EC). Estes ciclos são inicializados por uma mensagem Trigger Message (TM) enviada pelo *master*. Cada EC divide-se em dois campos, um dedicado ao tráfego síncrono e o outro relativo ao tráfego assíncrono(figura 2.8 [2]).

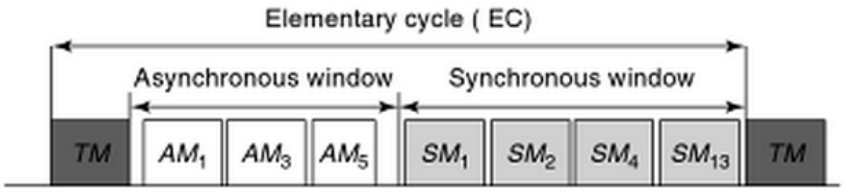


Figura 2.8: Elementary Cycle [2].

O *master* agenda o tráfego síncrono, aproveitando ao máximo o paralelismo de caminhos disjuntos de informação, e envia para os *slaves* essa informação através da TM(figura 2.9 [1]). Os *slaves* assim que recebem a TM, analisam-na e emitem logo de seguida as mensagens agendadas. Quanto ao tráfego assíncrono, para que não haja conflito com o tráfego síncrono, os

slaves possuem filas onde é guardado o tráfego, sendo no fim de cada EC enviado para o *master* o estado dessas filas.

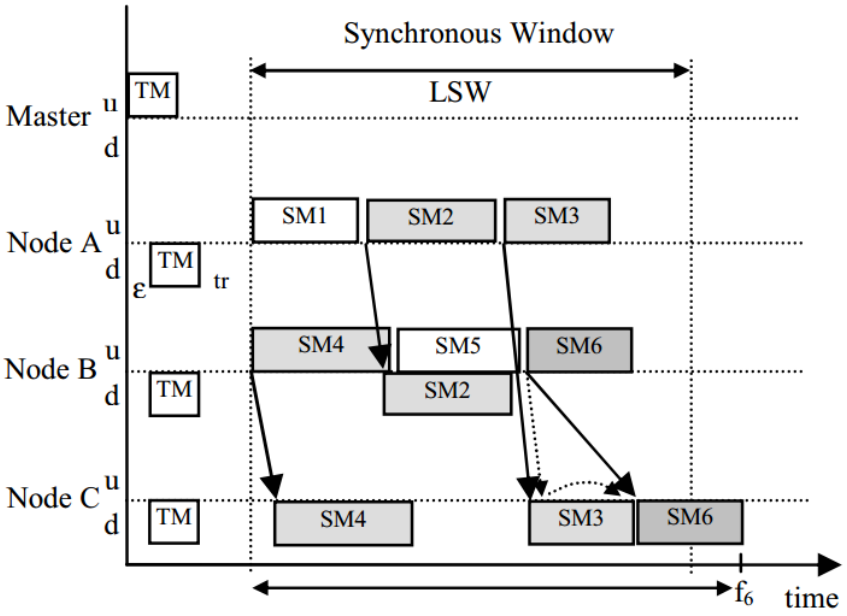


Figura 2.9: Tráfego mensagens.

2.4 HaRTES

HaRTES é um *switch Ethernet*, implementado com recurso à tecnologia Field-programmable Gate Array (FPGA), adaptado para comunicações *Ethernet* de tempo real. O *switch* HaRTES proporciona uma maior integridade do sistema de comunicação na presença de uma comunicação de tempo real do tipo *hard* garantindo por exemplo que comunicações que tenham um comportamento anómalo sejam bloqueadas nas portas do *switch*. Este tipo de comunicações são tratados de uma forma eficiente não comprometendo o sistema visto que este *switch* fornece serviços de comunicação tempo real do tipo *hard*, com uma elevada flexibilidade operacional, com gestão de recursos e apresenta também uma utilização eficiente e adequada da largura de banda [24].

O desenvolvimento deste *switch* é feito com base no protocolo de comunicação de tempo real FTT-SE. A junção deste *switch* com o FTT-SE permite a resolução de algumas limitações que este protocolo possui, como a obrigação de todos os nós respeitarem os EC tal como as suas restrições temporais.

Este *switch* utiliza o mesmo tipo de arquitetura que o protocolo FTT-SE

dispondo de um nó *master* e diversos nós *slaves* com a diferença de que o *master* da rede encontra-se inserido no próprio *switch*. A inserção do *master* no *switch* permite que as mensagens deste nó não passem pelas camadas do FTT-SE o que se traduz numa maior precisão na transmissão destas mensagens, sendo transmitidas de uma forma direta do *switch* para a rede. Desta forma consegue-se uma melhoria significativa na latência e no *jitter*, pelo que é bastante vantajoso para aplicações que apresentem uma necessidade elevada de sincronização da rede. Outra vantagem do *switch* HaRTES comparativamente ao protocolo FTT-SE em relação ao tráfego assíncrono designa-se pela capacidade do *switch* guardar este tráfego em memória e transmiti-lo quando for apropriado. Contudo o desenvolvimento deste *switch* foi feito com o objetivo de conseguir o seguinte [24]:

- Introdução de capacidades de controlo de transmissão nos *switches Ethernet* por forma a sincronizar de diversos fluxos em diferentes portos a decorrer em paralelo e ainda o disparar de transmissões com baixo *jitter*.

- Integrar serviços de gestão (QoS) e um escalonamento flexível de forma a permitir que aplicações de tempo real sejam removidas, adicionadas ou ainda atualizadas *online*, garantindo os requisitos temporais.

- Separação de diferentes tipos de tráfego nos portos de entrada através de implementação de gestão de tráfego e ainda a gestão do tráfego com isolamento. Desta forma pode ser feita a integração de nós *Ethernet* tradicionais, como por exemplo PC's, sem comprometer a integridade temporal do sistema.

A arquitetura interna do *switch* HaRTES é apresentada na figura 2.10[20]. Como se pode ver na figura 2.10 a área sombreada representante do módulo *master* corresponde ao FTT *master* responsável por um conjunto de operações como o gestor QoS, controlo de admissão e ainda o escalonador responsável pelo agendamento do tráfego.

A unidade *Switching and Control Logic* é responsável pela gestão da transmissão e recepção dos pacotes de dados atendendo aos sinais de controlo que são fornecidos pela unidade *MAC Interface*.

A unidade *MAC Interface* tem como função a configuração do módulo *MAC IP Core*. Esta unidade tem também como responsabilidade a recepção dos dados atendendo à classificação, validação e gestão destes. Esta unidade está também encarregue do armazenamento dos dados na unidade *Memory Pool*.

A *Memory Pool* tem como função o armazenamento dos dados. A memória desta unidade é segmentada em blocos o que permite armazenar cada pacote em cada bloco.

Para ser efetuada a leitura ou escrita na memória são utilizados os módulos *Tx Multiplexing Unit* e *Rx Multiplexing Unit*.

A unidade *MAC IP Core* através do PHY *Ethernet* fornece os relógios com a frequência necessária para transmissão e recepção de dados.

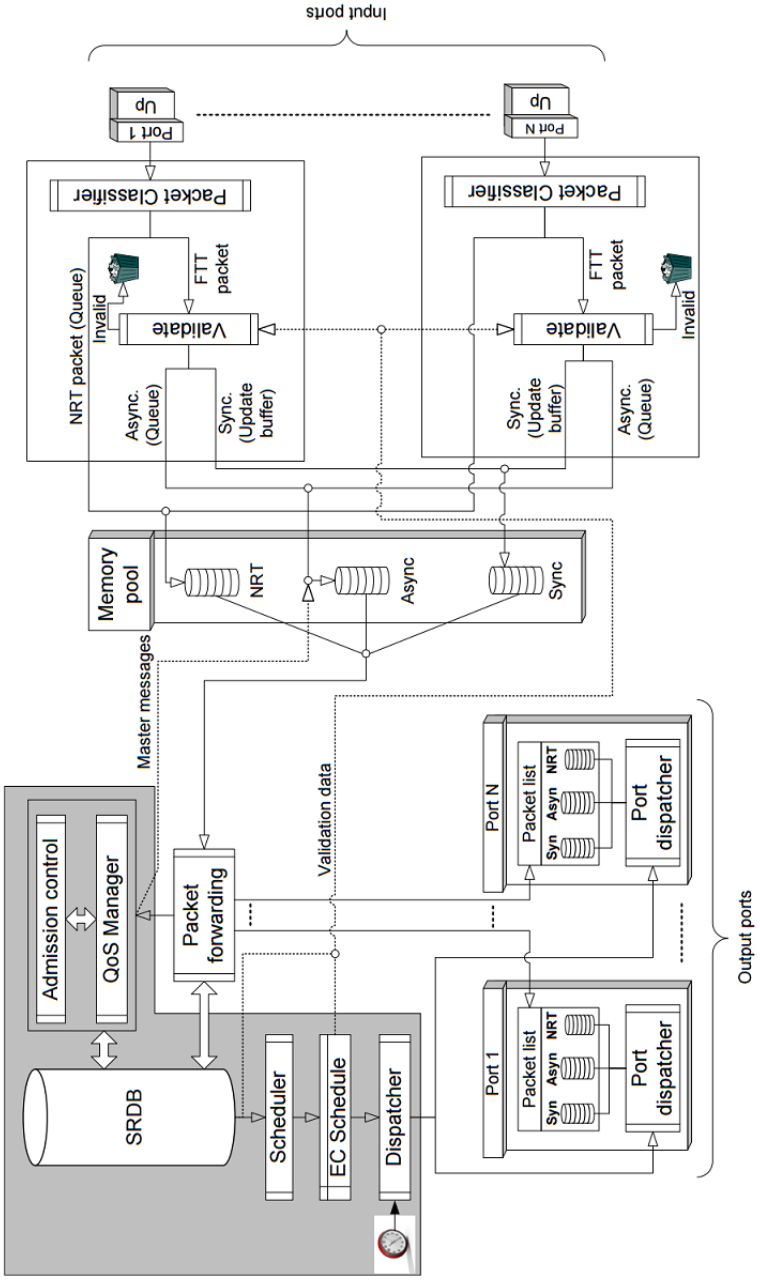


Figura 2.10: Arquitetura interna switch HaRTES.

Capítulo 3

Elementos do demonstrador

O trabalho desenvolvido nesta dissertação é definido como um sistema ciber-físico. O demonstrador utilizado, para a implementação deste sistema, é designado como uma *Plotter*, sendo esta caraterizada por ser um sistema físico de dois eixos. O sistema físico a controlar e o *hardware*, disponível para a realização do trabalho, serão apresentados neste capítulo.

3.1 Sistema físico a controlar

Como foi mencionado no início deste capítulo o sistema físico apresenta uma *Plotter* (figura 3.1).

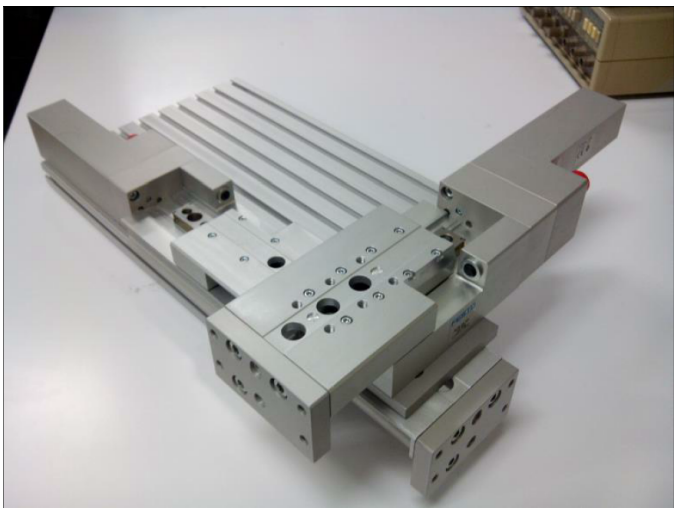


Figura 3.1: Sistema físico a controlar.

Cada eixo do sistema corresponde a um *"Mini slide SLTE, electric"* fabricado pela FESTO.

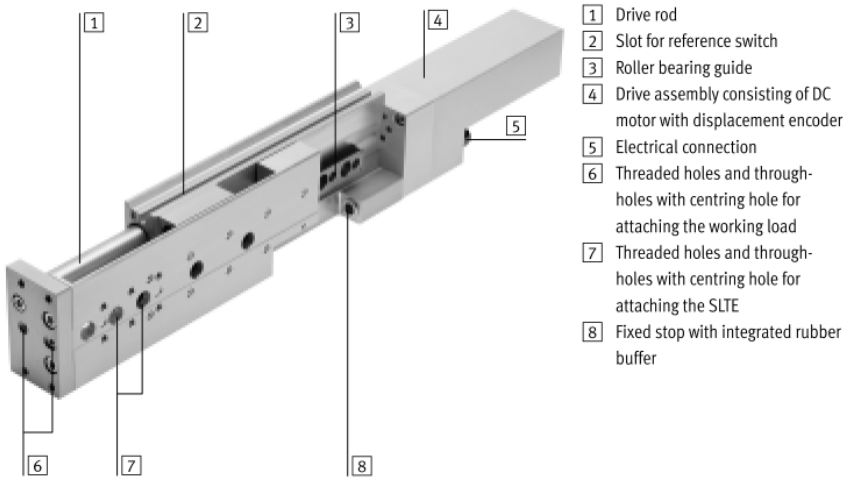


Figura 3.2: *Mini slide SLTE, electric.*[1]

Como se pode observar na figura 3.2, o espaço indicado pelo ponto *"4"* é onde se encontra o motor DC e os respetivos *encoders* que serão utilizados para o controlo do movimento da calha sobre o eixo. O acesso aos *encoders* assim como a alimentação destes e do motor DC é representada pela conexão do ponto *"5"*. Um esquema básico de conexão entre os motores DC e respetivos *encoders* é representado na figura 3.3.

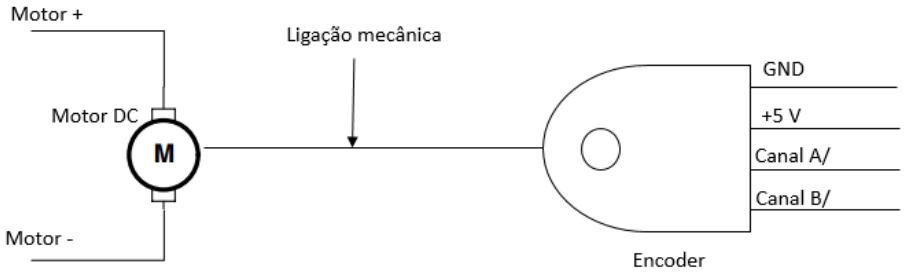


Figura 3.3: Conexão Motor DC e *encoder*.

Os dois eixos são perpendiculares entre si, estando o eixo vertical acoplado sobre o eixo horizontal como é possível ver na figura 3.1. Desta forma o movimento efetuado verticalmente é dependente da posição da calha que se encontra no eixo horizontal.

Um dos objetivos deste trabalho será conseguir o controlo e sincronização dos dois eixos. O comportamento desejado do sistema é semelhante ao de uma *Plotter* de duas dimensões.

[1] Source: `https://www.festo.com/net/SupportPortal/Files/26918/info_152_en.pdf`

3.1.1 Caraterísticas do equipamento

Os dois eixos da *Plotter* apresentam as mesmas caraterísticas diferenciando apenas no comprimento do eixo, como se pode verificar na tabela 3.1. Os motores DC e os *encoders* respetivos a cada eixo são iguais.

Eixos	***Eixo 1***	***Eixo 2***
Alimentação do motor [V DC]	24	
Comprimento dos eixos [mm]	80	150
Resolução do codificador (pulsos por rotação)	1000	
Alimentação *encoders* [V DC]	5	

Tabela 3.1: Caraterísticas dos dois eixos.

3.1.2 Interface

Como foi mencionado anteriormente, cada eixo tem um motor DC e *encoder* associado. A cada eixo está associada uma ficha com 12 pinos. A alocação dos pinos da ficha de conexão está representada na figura 3.4. Os pinos utilizados para a realização do trabalho foram os seguintes:

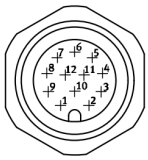
- Pino 1 e 2: correspondentes à alimentação dos motores DC.
- Pino 10: *ground*.
- Pino 9: alimentação dos *encoders*.
- Pino 4 e 6: pinos utilizados para a leitura dos sinais "negados"provenientes dos canais A e B gerados pelo *encoder*.

O canal I do *encoder* tem como função aumentar a resolução das medidas mas a sua utilização não foi necessária visto que os canais A e B apresentam uma elevada resolução.

3.1.3 Encoders

O modelo dos *encoders*, incorporados nos eixos do sistema físico, não é especificado no *datasheet* correspondente ao *Mini Slide SLTE, electric* fornecido pela FESTO. Contudo, no *datasheet* é mencionado que se trata de *encoders* ópticos rotativos. Estes *encoders* permitem obter uma posição relativa e o sentido do movimento da calha no eixo.

Pin allocation of connection plug



Plug M12		
Pin	Connection	Function
1	Motor +	Motor conductor
2	Motor –	Motor conductor
3	A	Encoder signal RS 485
4	A/	Encoder signal RS 485
5	B	Encoder signal RS 485
6	B/	Encoder signal RS 485
7	I	Encoder signal RS 485
8	I/	Encoder signal RS 485
9	+5 V DC	Signal supply
10	0 V	Signal ground
11	–	–
12	–	–

Figura 3.4: Alocação dos pinos da ficha de conexão.[2]

O princípio de funcionamento destes *encoders* consiste num disco óptico no qual são gravadas marcas a espaços regulares. A leitura destas marcas é feita por um par emissor/recetor óptico, gerando uma trama de impulsos à medida que o motor roda [25](figura 3.5 [25]).

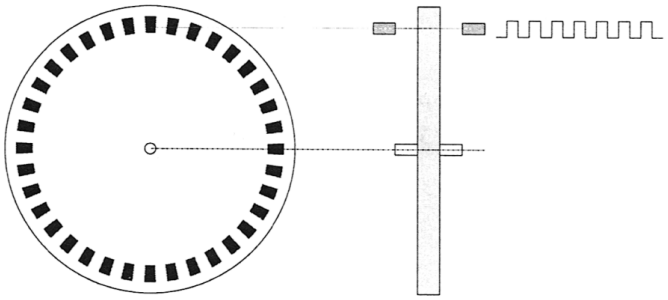


Figura 3.5: *Encoder* óptico rotativo.

Para discriminar qual o sentido do movimento são necessários dois sinais em quadratura (desfasados 90°) provenientes dos *encoders*, designadamente os sinais fornecidos pelos canais A e B referidos anteriormente. Quando a calha se desloca num sentido o canal A está 90° em atraso de fase relativamente ao canal B enquanto que no sentido inverso o canal A encontra-se 90° em avanço de fase face ao canal B (figura 3.6[25]).

Outra vantagem de os canais A e B estarem em quadratura é o fato de se conseguir quadruplicar a resolução do *encoder* no que respeita a impulsos por rotação.

[2]Source: `https://www.festo.com/net/SupportPortal/Files/26918/info_152_en.pdf`

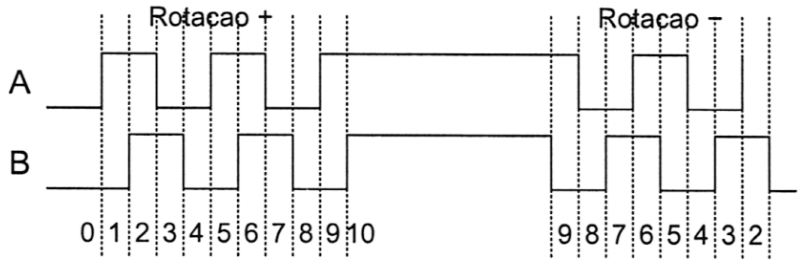


Figura 3.6: Sinais do *encoder* em quadratura.

3.1.4 Driver do Motor

Os integrados L293D, da Texas Instruments, foram utilizados como *drivers* dos motores DC. O L293D é basicamente uma ponte H capaz de controlar um motor DC fornecendo correntes bidirecionais até 600 mA para tensões entre 4.5 V até 36 V. O diagrama lógico dos *drivers* está representado na figura 3.7.

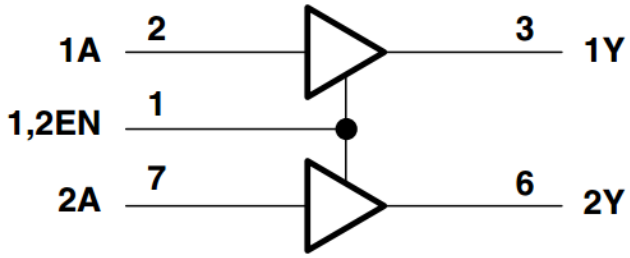


Figura 3.7: Diagrama lógico.[3]

O *enable* das duas entradas é feito pelo 1,2EN. Quando o *enable* está a *"1"* as saídas 1Y e 2Y estão ativas e em fase com as entradas 1A e 2A respetivamente. Por outro lado quando o *enable* está a *"0"* as entradas ficam desativadas e as respetivas saídas em alta impedância.

Na tabela 3.2 está representado o comportamento do motor em função das entradas da ponte H.

3.2 *Hardware*

O *hardware* que será utilizado neste trabalho será apresentado nesta subsecção. O motivo da utilização deste *hardware* deve-se ao fato de ser o material disponível para a realização do trabalho.

[3]Source: `http://www.ti.com/lit/ds/symlink/l293.pdf`

Inputs			Outputs
1,2EN	**1A**	**2A**	
1	1	1	Motor stopped
1	0	0	Motor stopped
1	1	0	Motor spins(direction 1)
1	0	1	Motor spins(direction 2)
0	X	X	Motor is in free running

Tabela 3.2: Comportamento do motor em função das entradas da ponte H.

3.2.1 PIC32-MAXI-WEB

Para a realização do trabalho, tem-se à disposição dois microcontroladores desenvolvidos da Olimex designados como PIC32-MAXI-WEB (figura 3.8).

Esta placa apresenta um processador PIC32(PIC32MX795F512L), da Microship, com um módulo de *Ethernet* de 100Mbit embutido, o que permite o desenvolvimento de aplicações *Ethernet* de tempo real. A placa apresenta ainda uma frequência máxima do processador de 80 MHz e um módulo Universal Asynchronous Receiver Transmitter (UART) RS-232.

Para esta placa a Olimex disponibiliza um *software demo* que demonstra a funcionalidade dos vários periféricos embutidos, assim como as entradas dedicadas ao utilizador, comunicação série, interface gráfica e conexão de rede.

A demo é desenvolvida segundo a biblioteca Microship Solutions.

Outras bibliotecas que a demo inclui são:

- Microship Graphics Library v.3.06, utilizada para a demonstração da interface gráfica no *display* LCD que o microcontrolador inclui;
- Microchip TCP/IP Stack Library v5.42 representa um conjunto de programas que fornece serviços para diversas aplicações TCP/IP, por exemplo, servidor HTTP, protocolo UDP e protocolo Transmission Control Protocol (TCP);
- Microchip MDD File System Library 1.4.0 (sistema de ficheiros FAT, FAT32), utilizada para escrita e leitura, de informação de um cartão SD, entre o microcontrolador e um PC;
- Microchip USB, usada para a criação de aplicações USB.

Para que as tarefas sejam executadas em tempo real a demo inclui um sistema operativo de tempo real(Real-Time Operating System (RTOS)) distribuído grátis designado como FreeRTOS.

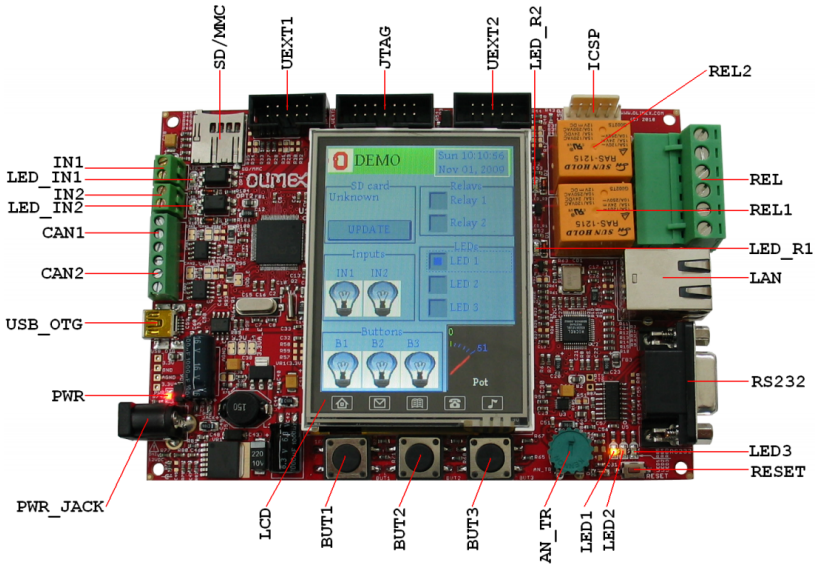


Figura 3.8: PIC32-MAXI-WEB.[4]

3.2.2 DETPIC32

As DETPIC32 são microcontroladores desenvolvidos no Departamento de Electrónica, Telecomunicações e Informática (DETI) da Universidade de Aveiro.

Estas placas, que contêm microcontroladores PIC32MX795F512H(TQFP) da Microship incorporados, apresentam as seguintes caraterísticas:

- Frequência máxima do processador de 80 MHz.
- Disponibilização de vários portos de I/O (PORTB, PORTC, PORTD, PORTE, PORTF e PORTG).
- Um módulo de conversão analógico-digital (A/D), de 10 bits, com um modelo de programação, que permite múltiplas possibilidades de configuração, é também disponibilizado nestas placas.
- Estão disponíveis 5 *timers*, T1 a T5, que podem ser usados para a geração periódica de eventos de interrupção ou para a geração de sinais de Pulse-Width Modulation (PWM).

[4]Source: `https://www.olimex.com/Products/PIC/Development/PIC32-MAXI-WEB/resources/PIC32-MAXI-WEB.pdf`

- Módulo UART que é um canal de comunicação série assíncrona *full--duplex*. Implementa o protocolo RS232 o que permite a ligação a um PC.

- Estão disponíveis 4 módulos Inter-Integrated Circuit (I2C) que correspondem a uma interface de comunicação série bidirecional *half-duplex* utilizada para interligar o microcontrolador e dispositivos periféricos.

As DETPIC32 ao contrário das PIC32-MAXI-WEB não possuem nenhum módulo *Ethernet*.

Capítulo 4

Projeto e implementação do demonstrador

Neste capítulo será apresentado o sistema ciber-físico implementado assim como o módulo de *software* desenvolvido para a sua implementação. Serão também apresentados todos os passos efetuados, durante o trabalho, no que diz respeito ao controlo do sistema físico e ao *software* desenvolvido para a implementação do sistema de comunicação.

4.1 Sistema implementado

O sistema implementado para controlar a *Plotter* é representado pelo diagrama de blocos da figura 4.1. O *Process Controller* representa o programa que estará a correr num computador sendo este o responsável pelo controlo de posição e velocidade dos eixos. Os nós 1 e 2 representam os dois microcontroladores PIC32-MAXI-WEB da Olimex(um para cada eixo), sendo estes os sistemas embutidos que servirão de interface com a rede de comunicação *Ethernet*. O *Process Controller* e os nós 1 e 2 comunicam entre si, via *Ethernet*, através do *switch Ethernet* a ser testado. O *switch* é encarregue da troca de informação entre estes 3 elementos assim como a sincronização dos fluxos de informação e o agendamento do tráfego de tempo real do sistema gerado por estes. Os dois microcontroladores recebem e analisam os sinais provenientes dos *encoders* de modo a calcular a posição da calha e a sua velocidade. São também responsáveis por gerirem os sinais de controlo dos eixos (sinal de PWM) para os motores DC.

4.1.1 Alimentação

A alimentação do sistema é feita por 3 fontes de tensão fixa independentes de 24 V, 12 V e 5 V. A fonte de tensão de 12 V alimenta as PIC32-MAXI-WEB, enquanto que os 24 V alimentam os motores DC e os 5 V os *encoders*.

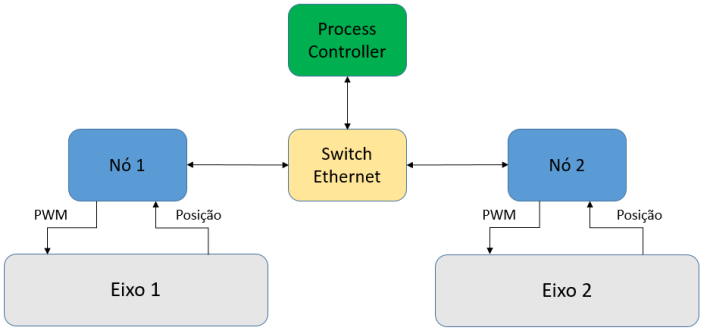


Figura 4.1: Diagrama de blocos do sistema

4.2 Módulo de *Software*

Após o *hardware* estar montado e ligado à *Plotter* foi feita uma calibração aos eixos que será apresentada no próximo capítulo. O módulo de *software* implementado e as respetivas fases de desenvolvimento serão apresentados com detalhe nesta secção. Os principais problemas que ocorreram durante a implementação do trabalho, e que influenciaram a solução final, serão também apresentados.

4.2.1 Controlo de um eixo

Para a realização do controlo individual dos eixos foi utilizado um microcontrolador DETPIC32 que, como foi mencionado no capítulo anterior, disponibiliza pinos I/O que permitem a geração do sinal PWM necessário, entradas e saídas digitais ligadas diretamente ao *hardware*, pinos I/O dedicados para interrupções externas e um módulo UART RS-232. A razão de ser utilizado este microcontrolador para os primeiros testes foi para evitar, no caso de haver algum erro ou falha, que fossem danificadas as PIC32-MAXI-WEB visto que o preço destas é elevado. Após o controlo estar efetuado as DETPIC32 goram substituídas pelas PIC32-MAXI-WEB.

Através da calibração dos eixos é conhecida a correspondência entre:

- Número de impulsos e a distância percorrida da calha da *Plotter*.
- Velocidade da calha e o *duty-cycle* do sinal de PWM que controla os motores DC dos eixos.

No processo de calibração, que será descrito no próximo capítulo, foi definida uma frequência de 15 KHz para o sinal PWM. Este valor é também justificado no capítulo seguinte. Tendo sido feita a calibração dos eixos, foi efetuado o controlo de posição e velocidade para cada um deles.

Na figura 4.2 está representado o esquemático da DETPIC32 com o *hardware* necessário para ser feito o controlo do eixo. As saídas digitais 1 e 2 controlam qual o sentido com que a calha se move. A ponte H tem como responsabilidade colocar nas suas saídas os sinais adequados para o movimento do motor em função dos sinais recebidos pela DETPIC32. O *duty-cycle* do sinal PWM, define o tempo que o *enable* da ponte H se encontra a *"ON"*. O canal B do *encoder* está ligado a um pino da DETPIC32 capaz de detetar interrupções externas. O canal A do *encoder* está ligado a pino da DETPIC32 configurado como entrada digital.

No código desenvolvido para a DETPIC32 a variável que representa número de impulsos é incrementada ou decrementada sempre que o sinal do canal B gera uma interrupção externa. A rotina de serviço a esta interrupção é chamada sempre que deteta uma transição de *"0"* para *"1"* no sinal do canal B. Dentro desta rotina é feita uma verificação ao sinal do canal A pela DETPIC32 (Entrada digital), se estiver a *"0"* significa que a calha se move com um determinado sentido e o número de impulsos é incrementado, caso contrário a calha move-se no sentido oposto e o número de impulsos é decrementado (figura 3.6).

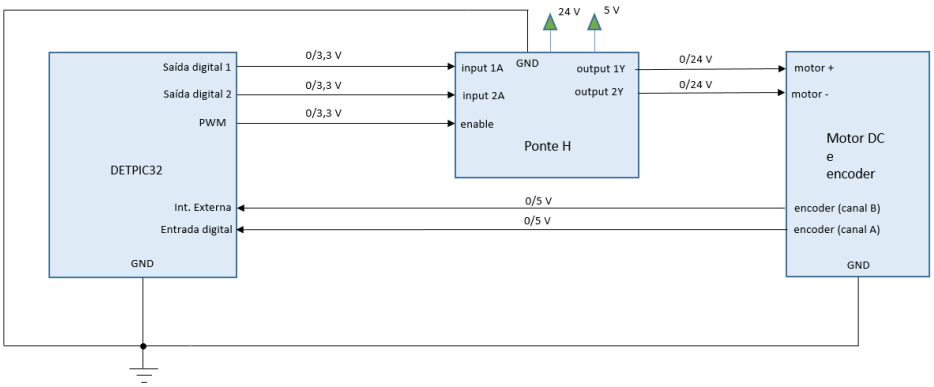


Figura 4.2: Esquemático do circuito da DETPIC32 com o *hardware*.

O cálculo da velocidade é feito com recurso a um *timer* do microcontrolador. Este *timer* foi configurado para gerar uma interrupção a uma frequência de 370 Hz. Este valor foi escolhido com base no conhecimento adquirido em algumas disciplinadas lecionadas pertencentes ao plano curricular do curso de Engenharia Electrónica, Telecomunicações e Informática, como Controlo I, Controlo II e Controlo Digital. Uma frequência de amostragem na ordem das centenas de Hz por norma será suficiente para se realizar o controlo, de velocidade da calha, de uma forma eficaz e precisa. A frequência de amostragem não deve ser demasiado baixa, pois o controlo não será preciso e poderá tornar o sistema instável, nem demasiado elevada pelas seguintes razões:

- A Interrupt Service Routine (ISR) externa tem maior prioridade de

atendimento relativamente à ISR utilizada para o cálculo de velocidade, de modo a não ocorrerem perdas de impulsos, e por isso pode interrompê-la (*Interrupt nesting*). Assim a frequência da ISR utilizada para o cálculo de velocidade deve ser baixa o suficiente para que o atraso causado pela interrupção da ISR externa não seja significativo. Caso a frequência seja muito elevada, o atraso causado pelo *"Interrupt nesting"* será bastante significativo, o que provocará uma degradação no desempenho do controlador pois não será aplicado nos instantes de tempo corretos.

- Para não "gastar" uma grande parte da capacidade de processamento do CPU do microcontrolador, pois será necessária para a troca de mensagens feita pelo sistema de comunicação desenvolvido que será apresentado mais à frente neste capítulo.

O controlo de velocidade é feito utilizando um controlador PID com uma frequência de amostragem definida pelo *timer* usado para o cálculo da velocidade. Inicialmente as variáveis correspondentes à posição inicial(posInit), atual(pos) e final(posEnd) são inicializadas. De seguida é verificado o sentido do movimento que a calha fará e são configuradas as saídas digitais da DETPIC32 de acordo com o sentido definido. Após este processo são configurados e inicializados os *timers* atribuídos para o movimento da calha(PWM) e para o cálculo de velocidade(timer_int). Assim que a calha inicia o movimento, a variável representante da posição atual(pos) é atualizada sempre que o sinal proveniente do canal B do *encoder* gera uma interrupção. Este conjunto de procedimentos é representado pelo fluxograma apresentado na figura 4.3.

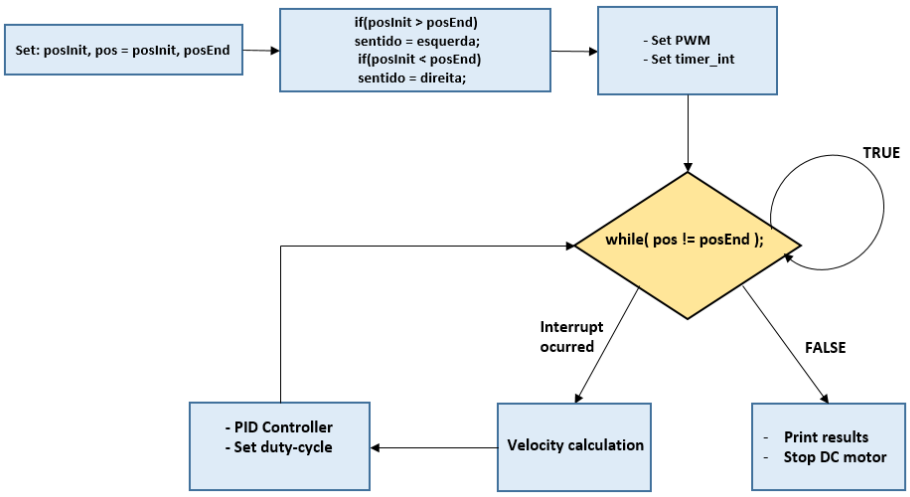


Figura 4.3: Fluxograma do controlo de velocidade.

O controlo de posição é implementado sobre o controlo de velocidade. Foram criados perfis de velocidade em que a velocidade da calha depende da sua posição atual e final. Como se pode observar na figura 4.4 foram designados três casos possíveis.

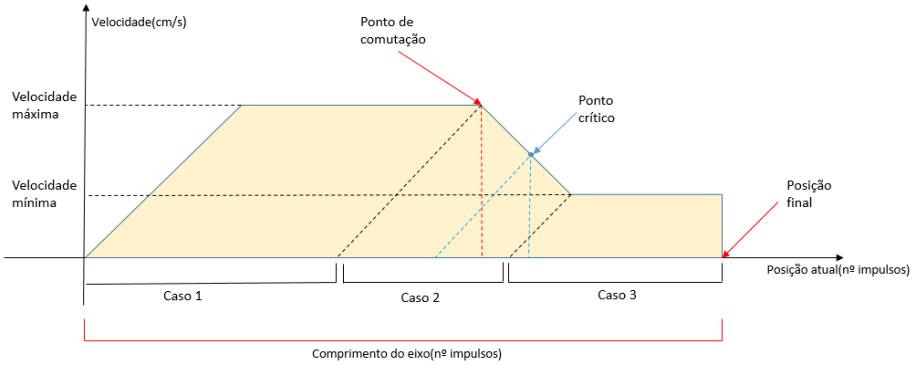


Figura 4.4: Perfis de velocidade.

A posição atual é igualada à posição inicial da calha antes do início do movimento. Quando a posição atual e final forem iguais é desligado o motor.

A escolha do caso para ser feito o controlo é feita em função da posição inicial.

Os três casos definem-se por:

- Caso 1: neste caso a posição inicial da calha encontra-se bastante distanciada da posição final. O sinal de referência do controlador PID, destacado para o controlo de velocidade, é definido com o valor de velocidade máxima. Quando a posição atual atinge o ponto de comutação (figura 4.4) a referência do controlador PID é alterada para a velocidade mínima de maneira a que haja tempo suficiente para a desaceleração da calha e estabilização à velocidade mínima.

- Caso 2: antes do início do movimento é calculado o ponto de interseção(ponto crítico mostrado na figura 4.4) entre a reta de aceleração e a reta de desaceleração. Após ser feito este cálculo, é iniciado o movimento da calha com a referência do PID igual à velocidade máxima. Assim que a posição atual atinge o ponto crítico a referência do PID passar a ser igual à velocidade mínima. A velocidade diminui até estabilizar no valor mínimo definido. O cálculo do ponto crítico é feito com o intuito de tornar o sistema mais reativo possível, pretendendo atingir esse ponto rápido e assim que o atingir diminuir a velocidade para o valor mínimo.

- Caso 3: a posição inicial encontra-se próxima da posição final, portanto não há necessidade da calha se deslocar com velocidade máxima. Por

esta razão a referência do PID é definida com a velocidade mínima e mantida desta forma até que a posição atual e final se igualem.

Na figura 4.5 está representado o fluxograma do controlo posição juntamente com o controlo de velocidade.

Inicialmente é definido um *array* que contém as posições finais desejadas. As variáveis correspondentes à posição inicial e atual são inicializadas e igualadas. De seguida é executada a função *"orientation()"* que é responsável por verificar o sentido do movimento e qual dos três casos, referentes à figura 4.4, será executado.

Após este processo é chamada a função *"move();"* que configura e inicializa os *timers* responsáveis por gerar o PWM e calcular a velocidade, iniciado assim o movimento da calha.

Logo após a execução destas funções entra-se num ciclo infinito onde é feito a chamada constante da função *"controloPos()"* em que a posição atual é constantemente verificada e comparada com outras variáveis.

Quando o *timer* responsável pelo cálculo de velocidade gera uma interrupção é feito o mesmo processo que foi descrito no fluxograma representado na imagem 4.3. Caso contrário é feito uma comparação entre a posição atual e final verificando se são iguais. Se esta comparação for falsa é verificado qual dos três casos possíveis dos perfis de velocidade foi definido pela função *"orientation()"*.

Após esta verificação se corresponder ao caso 1 a posição atual é comparada com a posição correspondente ao ponto de comutação. Se o caso verificado for o caso 2 a posição atual é comparada com a posição correspondente ao ponto crítico. No caso 1 assim como no caso 2, se as condições se verificarem, a referência do PID é alterada para o valor de velocidade mínima definida.

Quando a posição atual for igual à final é parado o motor. De seguida é feita uma verificação se a posição final que fora alcançada é a última do *array* definido inicialmente. Se não for o último elemento do *array* o elemento seguinte a este é definido como nova posição final e é repetido todo o processo. Caso corresponda ao último elemento do *array* a função *"controloPos()"* quebra o ciclo infinito e são mostrados os resultados.

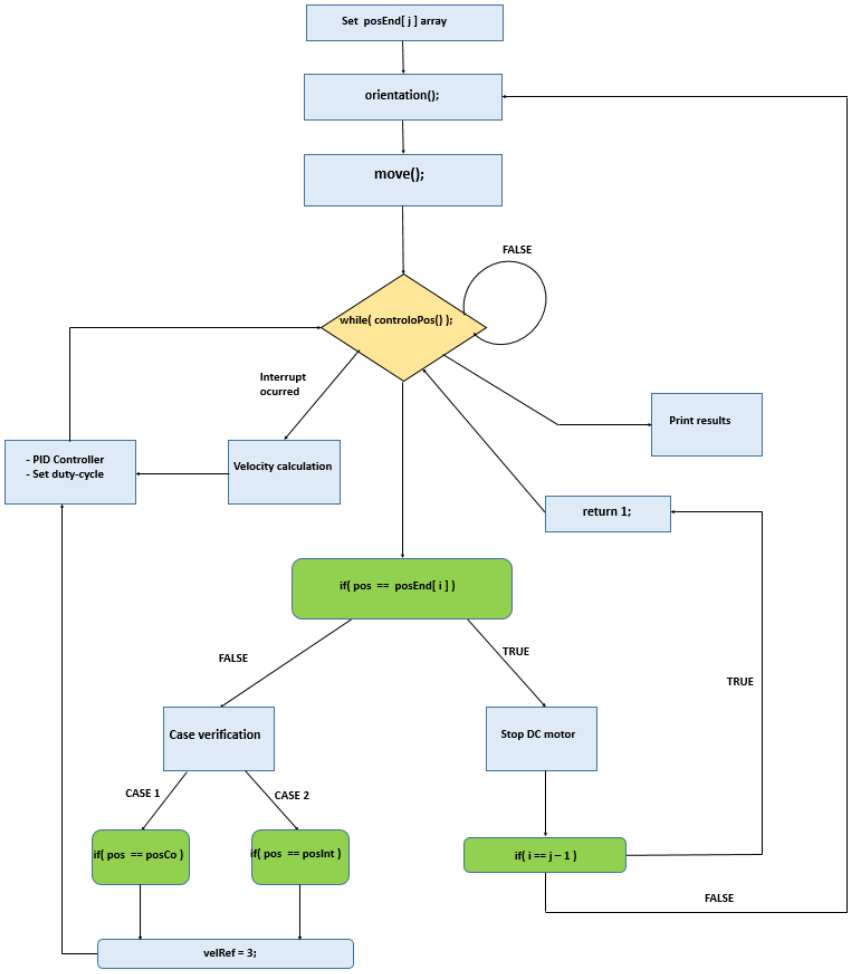


Figura 4.5: Controlo de posição e velocidade.

4.2.2 *Stack* TCP/IP

Para a implementação do projeto uma das funções atribuídas às PIC32-MAXI-WEB é a troca de mensagens UDP, via *Ethernet*, entre elas e o *Process Controller*.

Como foi referido anteriormente a demo disponibilizada pela Olimex inclui um conjunto de bibliotecas utilizadas para demonstrar a funcionalidade dos periféricos embutidos que este microcontrolador possui. Tendo à disposição esta demo foi feito um estudo sobre o seu funcionamento de modo a poder adaptá-la e usá-la na implementação do trabalho.

O estudo do funcionamento do sistema operativo de tempo real FreeRTOS foi realizado de maneira a entender como é que as diferentes tarefas das respetivas *stacks* são executadas simultaneamente.

A *stack* TCP/IP foi estudada com maior detalhe visto que esta é a responsável pelo funcionamento do módulo *Ethernet*. Esta *stack* desenvolvida pela Microship representa um conjunto de funções que fornecem serviços para aplicações TCP/IP. A estrutura desta *stack* é representada na figura 4.6.

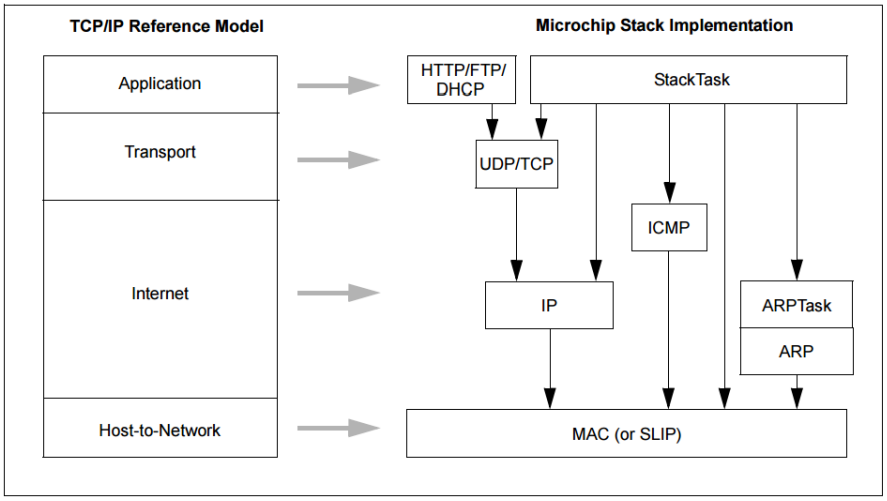


Figura 4.6: Estrutura *stack* TCP/IP.[1]

Por defeito, nas definições de configuração da *stack* TCP/IP da demo, o protocolo dinâmico de configuração automática de endereços Internet Protocol (IP) da camada *link* designado de *"STACK_ USE_ AUTO_ IP"* encontra-se ativo. Desta forma sempre que o microcontrolador era conetado a um computador através de um cabo *Ethernet* o seu endereço IP era alterado. Para que os dois microcontroladores tivessem endereços IP fixos, o protocolo *"STACK_ USE_ AUTO_ IP"* foi desativado e foi definido um IP para cada microcontrolador. Desta forma a conexão com o *Process Controller* torna-se mais simples pois identifica diretamente qual a placa com quem está a comunicar.

4.2.3 *Sockets* UDP

Os protocolos UDP e TCP pertencem à camada de transporte do modelo de referência TCP/IP representado na figura 4.6.

O protocolo UDP, ao contrário do TCP, não garante a entrega das mensagens ao destinatário pretendido. O TCP por forma a garantir que todas as mensagens sejam entregues o recetor confirma ao emissor o sucesso da receção das mensagens. No caso de ocorrerem falhas no envio de mensagens o destinatário solicita ao emissor a retransmissão destas [26].

[1] Source: `http://ww1.microchip.com/downloads/en/AppNotes/00833c.pdf`

O sistema a implementar neste trabalho é um sistema de tempo real com requisitos temporais exigentes. A troca de mensagens que ocorre entre o *Process Controller* e os dois microcontroladores deve ser feita no menor tempo possível de maneira a que o sistema cumpra os requisitos temporais. O uso do protocolo TCP poderia provocar atrasos significativos no sistema, prejudicando o seu desempenho. Por estas razões foi utilizado o protocolo UDP, visto que a perda de um pequeno número de mensagens terá um impacto mínimo no desempenho do sistema. A perda de um número mínimo de mensagens poderá provocar que a calha termine o seu movimento numa posição muito pouco distanciada da posição final ou, por exemplo, a calha alterar a sua velocidade quando se encontra a uma distância de poucos impulsos da posição onde esta devia ser alterada.

O protocolo TCP foi então desativado, visto que este não será utilizado na implementação do projeto.

Para ser feita a troca de mensagens entre um microcontrolador e o *Process Controller* é necessário a criação de *sockets* UDP. Os *sockets* UDP representam a conexão feita entre dois terminais, microcontrolador e *Process Controller* neste caso.

Em cada terminal é criado um *socket* com a seguinte informação:

- Endereço IP do destino.
- Porto de destino.
- Porto de origem.

Para a criação dos *sockets* nos microcontroladores foram usadas as funções disponibilizadas pelo ficheiro *"UDP.c"* pertencente à *stack* TCP/IP. Para iniciar o módulo UDP é chamada a função *"UDPInit()"*. Após o módulo ser iniciado é criado o *socket* através da função *"UDPOpenEx()"*.

No caso do *Process Controller*, a criação dos *sockets* é feita incluindo os *headers files*, no código desenvolvido, designados de *<sys/socket.h>* e *<arpa/inet.h>* que permitem a utilização de um conjunto de funções necessárias para a criação destes. A criação do *socket* é feita pela função *"socket()"*. Após o *socket* ser criado este é associado, através da função *"bind()"*, a um endereço IP assim como aos portos de envio e receção.

4.2.4 Mensagens UDP

Em cada mensagem UDP será enviado um conjunto de variáveis que caraterizam o estado do sistema. Por forma a que o envio e leitura destas variáveis sejam feitos de uma forma direta é criada uma estrutura que contenha essas variáveis. Para verificar se uma estrutura era enviada e lida corretamente foi desenvolvido um código simples para o *Process Controller*. O código consiste na criação um *socket* UDP efetuando depois leitura e escrita de uma

estrutura recebida pela mensagem UDP. A demo da PIC32-MAXI-WEB foi adaptada de modo realizar as mesmas operações que o *Process Controller*.

De maneira a que a leitura e escrita sejam bem efetuadas, a estrutura de dados criada na demo como no *Process Controller* tem de ser igual.

A estrutura criada para este propósito foi a seguinte:

```
typedef struct __attribute__((packed)){
char  eixo;
unsigned short impulsos;
} estrutura;
```

Uma das PIC32-MAXI-WEB foi ligada diretamente ao *Process Controller* através de um cabo *Ethernet*. O diagrama temporal correspondente ao processo descrito acima é representado na figura 4.7. Inicialmente a estrutura é criada no microcontrolador e no *Process Controller*. Após a criação da estrutura, os terminais efetuam o processo de criação de *sockets* UDP, com recurso às funções mencionadas na subsecção anterior, de maneira a que possa ser realizada a comunicação.

No *Process Controller* para ser enviada uma mensagem é utilizada a função *"sendto()"* enquanto que para receber é usada a função *"recvfrom()"* que são disponibilizadas pelo *header file* <*sys/socket.h*>. Quanto às PIC32-MAXI-WEB para enviar e receber uma mensagem UDP é necessário recorrer a um conjunto de funções, disponibilizadas pela *stack* TCP/IP, que se caraterizam por:

- *"UDPIsPutReady()"*: esta função tem de ser chamada antes de ser enviada uma mensagem, verificando se o *socket* está livre para ser preenchido com uma nova mensagem.
- *"UDPPutArray()"*: para enviar uma mensagem, com a estrutura pretendida, é utilizada esta função após ser chamada a *"UDPIsPutReady()"*.
- *"UDPIsGetReady()"*: de modo a verificar se existe alguma mensagem no *socket* pronta a ser lida é chamada esta função.
- *"UDPGetArray()"*: Após a chamada da *"UDPIsGetReady()"* a função *"UDPGetArray()"* permite ler do *socket* a mensagem UDP que nele está contida.

De modo a verificar se a conexão entre os dois terminais está ativa, o microcontrolador envia uma mensagem de conexão para o *Process Controller* enquanto este não responder com a mesma. Assim que a PIC recebe a mensagem de conexão é enviada a estrutura numa nova mensagem. Se não houver falhas na troca de mensagens as variáveis *"contador"* e *"impulsos"* são incrementadas no *Process Controller* e enviadas de volta para o

microcontrolador. Se o microcontrolador receber as mensagens a estrutura é atualizada com os valores recebidos e é enviada essa mesma estrutura. Caso haja falhas na receção a PIC volta a enviar a mesma estrutura da mensagem anterior enviada.

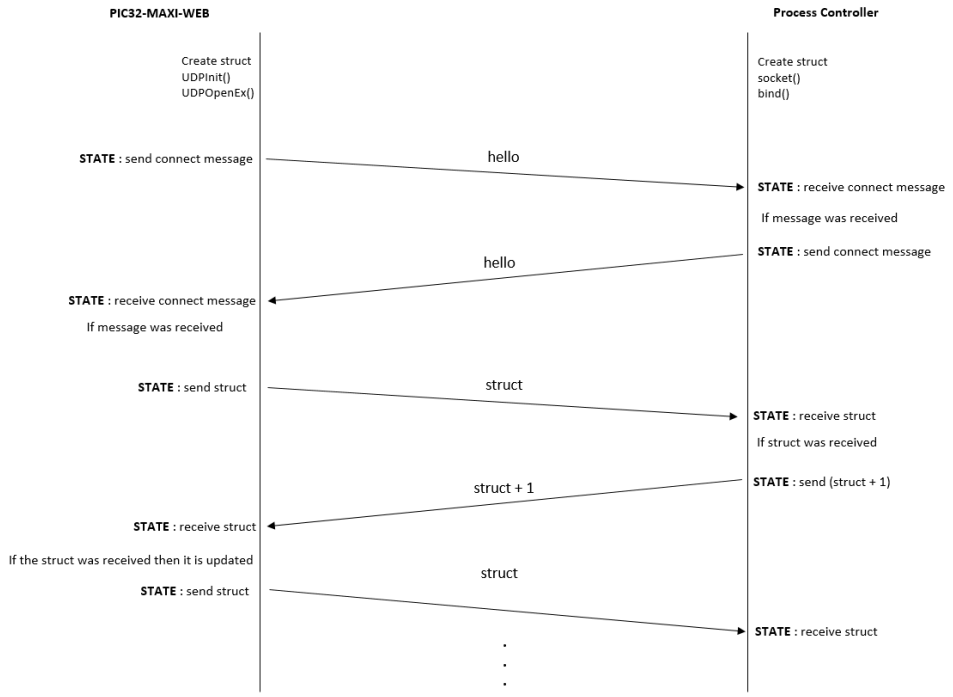


Figura 4.7: Diagrama temporal da troca de mensagens UDP.

4.2.5 Sistema de comunicação

Nos primeiros testes efetuados as PIC32-MAXI-WEB corriam uma versão adaptada da demo fornecida pela Olimex. Na realização destes testes verificou-se que havia uma perda significativa de contagem de impulsos nos microcontroladores. A razão desta perda deve-se ao fato de o FreeRTOS desativar as interrupções do microcontrolador quando chama a função *"vTaskScheduler()"*, para efetuar a gestão das diferentes tarefas, e só as voltar a ativar depois da execução dessa função.

De modo a resolver este problema, optou-se por desenvolver o código de raiz utilizando apenas a *stack* TCP/IP. Esta solução resolveu com sucesso o problema mencionado permitindo assim o controlo pretendido do sistema físico.

O sistema de comunicação desenvolvido neste trabalho consiste na troca de mensagens UDP *Unicast* entre o *Process Controller* e as PIC32-MAXI-WEB.

De forma a abrir uma conexão entre o *Process Controller* e cada um dos microcontroladores foram criados *sockets* UDP nestes terminais, usando o processo que foi descrito na subsecção 4.2.3.

Como foi mencionado anteriormente, o *Process Controller* é o elemento responsável pelo controlo, de posição e velocidade dos dois eixos, enquanto que as PIC32-MAXI-WEB são responsáveis pela contagem de impulsos e gerar o sinal PWM para o controlo dos motores DC.

Nos microcontroladores foi configurado um *timer* para gerar interrupções a uma frequência de 370 Hz correspondente à frequência de amostragem para o cálculo da velocidade instantânea. Sempre que a ISR é chamada é enviada uma mensagem para o *Process Controller* com a seguinte estrutura:

```
typedef struct __attribute__((packed)){
char  eixo;
unsigned short contador;
unsigned short impulsos;
} estrutura_sens;
```

A variável *eixo* identifica qual dos eixos está a comunicar com o *Process Controller* de modo a que seja aplicado o respetivo controlo. A variável *contador* permite ao *Process Controller* verificar se houve falha no envio de mensagens. Por último, a variável *impulsos* representa, como o próprio nome indica, o número de impulsos contabilizados, correspondente à posição atual da calha no eixo.

Para cada mensagem enviada pelo microcontrolador é esperada uma mensagem de resposta do *Process Controller* com a seguinte estrutura:

```
typedef struct __attribute__((packed)){
unsigned short sentido;
unsigned short duty_cycle;
} estrutura_atu;
```

O sentido com que a calha se desloca no eixo é definido pelos microcontroladores em função da variável *sentido* lida na estrutura que receberam do *Process Controller*. Consoante o valor desta variável o microcontrolador atribui às saídas digitais, que estão ligadas aos *encoders*, os valores correspondentes ao sentido pretendido. Para além do sentido, os microcontroladores recebem e atualizam o valor *duty-cycle*, calculado e enviado pelo *Process Controller*, de maneira a obter a velocidade pretendida da calha. Quando a calha atinge a posição final pretendida, o *Process Controller* envia para o microcontrolador uma nova mensagem em que a variável *duty-cycle* terá o valor 0, o que corresponde a que a calha pare o movimento.

As estruturas *"estrutura_atu"* e *"estrutura_sens"* foram definidas tanto no código desenvolvido para o *Process Controller* como para os microcontro-

ladores de forma a que a leitura e escrita destas nas mensagens UDP sejam feitas corretamente.

No código desenvolvido para as PIC32-MAXI-WEB, após as configurações iniciais, entra-se num ciclo infinito onde se verifica constantemente se existe alguma mensagem no *socket* para ser lida.

No código desenvolvido para o *Process Controller*, tal como nos microcontroladores, após as configurações iniciais entra-se num ciclo infinito que verifica constantemente se chegou alguma mensagem dos microcontroladores. No caso de haver uma mensagem no *socket* é feita a leitura da estrutura *estrutura_sens* contida na mensagem. Primeiramente é verificado a variável *eixo* de modo a saber qual dos microcontroladores enviou a mensagem. Sabendo qual o eixo de que recebeu a mensagem são lidas as variáveis *contador* e *impulsos*. As funções desenvolvidas para o controlo de posição e velocidade são chamadas e a estrutura *estrutura_atu* é preenchida com os novos valores e enviada para a PIC. À semelhança do processo descrito na subsecção 4.2.1, figura 4.5, no caso de a posição atual lida na mensagem, variável *impulsos*, ser igual à posição final é feita uma verificação se a posição final alcançada é a última do *array* (de posições finais). Caso não seja a última posição final, é calculado um novo sentido e a variável *sentido* da estrutura *estrutura_act* é preenchida com um novo valor.

Capítulo 5

Testes e resultados

Neste capítulo serão apresentados os testes efetuados e respetivos resultados correspondentes ao:

- Funcionamento do sistema físico.
- Desempenho do sistema de comunicação utilizando um *switch* HaRTES e um *switch Ethernet* comum.

As calibrações do sistema físico, efetuadas aos dois eixos, serão também apresentadas.

5.1 Calibração

Antes de ser aplicado o controlo de velocidade e posição a um eixo foi necessário efetuar a sua calibração. O esquemático do circuito utilizado para a calibração está representado na figura 4.2. De modo a verificar qual a frequência do sinal PWM, a aplicar na ponte H com o intuito de controlar o motor DC, foi desenvolvido um código para a DETPIC32 com o objetivo de iniciar o movimento da calha sobre o respetivo eixo num determinado sentido. A escolha da frequência foi feita por tentativa e erro, começando com uma frequência de 8 KHz. Foi verificado que para frequências inferiores a 15 KHz a calha deslocava-se sobre o eixo, mas o som produzido pelo funcionamento do motor e pela deslocação da calha, era demasiado elevado. A frequências de 15 KHz ou superiores observou-se que a calha se deslocava "normalmente" sobre o eixo com pouco ruído, tendo sido definido desta forma 15 KHz para a frequência do sinal PWM.

De seguida foram efetuadas calibrações de forma a determinar a correspondência entre:

- Número de impulsos e distância percorrida pela calha.
- Velocidade da calha e o *duty-cycle* do sinal de PWM.

Estas calibrações foram executadas primeiramente no eixo de maior comprimento, tendo desta forma uma margem de manobra maior, evitando que a calha embatesse nas extremidades do eixo durante estes processos.

Por forma a determinar a correspondência entre o número de impulsos e a distância percorrida pela calha foi desenvolvido um código para a DETPIC32, que desloca a calha segundo um sentido até a variável correspondente à posição atual atingir um determinado valor predefinido. Após a calha terminar o movimento foi medida a distância percorrida, obtendo desta forma a correspondência pretendida. Foi verificado que para um distância de 10 cm foram contabilizados 52500±2 impulsos. Logo a equação que descreve a correspondência pretendida é a seguinte:

$$dist\hat{a}ncia \pm 0.005\ [cm] = \frac{n^o\ impulsos\ atual \times 10}{52500 \pm 2\ impulsos} \tag{5.1}$$

O *duty-cycle*, para o sinal PWM, foi escolhido com o valor mais baixo possível de forma a que a calha iniciásse o movimento com uma velocidade relativamente baixa evitando assim o efeito de inércia que prejudicaria a calibração.

O código para determinar a correspondência entre o *duty-cycle* do sinal PWM e a velocidade da calha em malha aberta, foi desenvolvido de maneira a que esta efetuasse um trajeto com uma distância predefinida e fosse determinado o tempo da execução desse movimento. O cálculo da velocidade é feito de uma forma direta, dividindo a distância percorrida pelo tempo demorado pela calha a percorrer essa distância. O sinal PWM é configurado previamente com um determinado *duty-cycle* podendo assim ser feita a correspondência entre este e a velocidade calculada. Os resultados obtidos foram os seguintes:

d(cm)	**duty (%)**	**v1(cm/s)**	**v2(cm/s)**	**v3(cm/s)**	**v4(cm/s)**
10	30	5.416	5.381	5.395	5.404
10	35	9.207	9.226	9.201	9.231
10	40	14.245	14.161	14.207	14.203
10	45	17.663	17.641	17.622	17.587
10	50	20.056	20.057	20.153	20.126
10	55	21.962	21.943	22.015	22.013
10	60	22.981	23.158	23.123	22.987

Tabela 5.1: Correspondência entre *duty-cycle* e a velocidade da calha em malha aberta.

O processo para determinar a velocidade em função do *duty-cycle* foi efetuado quatro vezes como se pode observar na tabela 5.1.

5.2 Resultados do controlo de um eixo

Com o controlo em malha fechada de posição e velocidade aplicado aos dois eixos individualmente foram efetuados diversos testes, com a DETPIC32, de forma a verificar o seu funcionamento. Estes testes foram mais tarde efetuados com as PIC32-MAXI-WEB, a correrem o mesmo código que a DETPIC32, verificando-se os mesmos resultados.

Foram efetuados testes de repetibilidade descritos na figura 5.1. A posição inicial e final são definidas previamente e é executado o percurso representado pela seta castanha. Após a calha executar o percurso descrito, vinte e cinco vezes, foi medida a distância (Δd), correspondente à distância entre a posição onde a calha iniciou o movimento e onde parou.

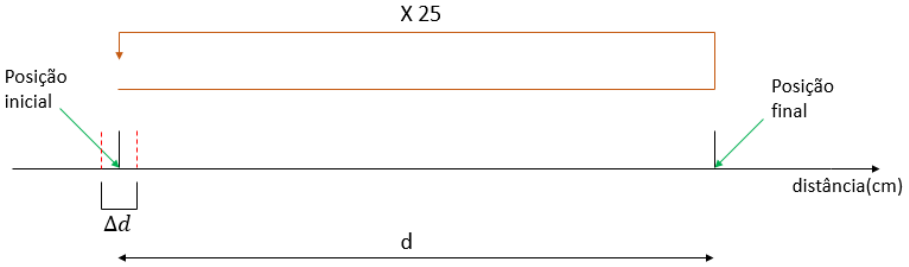


Figura 5.1: Teste de repetibilidade relativa à posição.

Os resultados obtidos foram os seguintes:

d(cm)	**Δd(mm)**
10	$\leq$1
8	$<$ 1
6	$<$ 1
4	$<$ 1
2	$<$ 1
1	$<$ 1

Tabela 5.2: Resultados dos testes de repetibilidade.

O erro percentual verificado nas medições relativas à posição é representado como um erro de $\leq$1%.

Para o cálculo da velocidade, como foi mencionado no capítulo anterior, é configurado um *timer* para gerar uma interrupção a uma frequência de 370 Hz. Quando a ISR é chamada é calculado uma nova amostra do valor de velocidade. Quando a calha estabilizada com velocidade em torno do valor pretendido, verificou-se que os valores calculados apresentavam a forma $v \pm \Delta v$, em que v representa a velocidade pretendida e Δv o erro apresentado relativamente ao valor pretendido. O valor máximo verificado para Δv foi de 0.2 cm/s, pelo que o erro percentual associado para este valor, comparativamente à velocidade mínima definida como 3 cm/s, é apresentado como $\leq$6.67%. Relativamente à velocidade máxima, definida como 10 cm/s, o erro máximo verificado foi de 0.2 cm/s, sendo apresentado o erro percentual como $\leq$2%.

5.3 Troca de mensagens UDP

Como foi mencionado na capítulo 4 o protocolo UDP não garante que as mensagens enviadas, de um emissor para o(s) receptor(es), sejam entregues. Por este motivo foi implementada troca de mensagens descrita na subsecção 4.2.4 com o objetivo de verificar se ocorrem perdas de mensagens UDP entre os terminais. Neste teste é também verificado se a estrutura que é enviada nas mensagens é lida e escrita corretamente. A troca de mensagens UDP foi testada primeiramente com a PIC32-MAXI-WEB a correr a uma versão da demo adaptada para a realização deste teste e com a PIC32-MAXI-WEB a correr um código desenvolvido utilizando apenas a *stack* TCP/IP. O código foi desenvolvido para que uma PIC32-MAXI-WEB enviasse 10000 mensagens UDP e esperar no melhor dos casos obter 10000 respostas do *Process Controller*. Em cada mensagem é enviada a seguinte estrutura (mencionada no capítulo anterior):

```
typedef struct __attribute__((packed)){
char  eixo;
short contador;
short impulsos;
} estrutura;
```

O microcontrolador envia uma mensagem, a uma frequência aproximadamente de 7.5 KHz, com a estrutura mencionada acima e é esperada a resposta do *Process Controller* com a mesma estrutura mas com as variáveis *"contador"* e *"impulsos"* incrementadas. Os resultados obtidos relativos ao envio e recepção das mensagens, utilizando a demo adaptada na PIC32-MAXI-WEB, estão indicados na tabela 5.3:

PIC32-MAXI-WEB			Process Controller		
Msg. enviadas	Msg. recebidas	Perdas na receção	Msg. recebidas	Msg. enviadas	Perdas na receção
10000	9982	18	10000	10000	0
10000	9977	23	10000	10000	0
10000	9989	10	9999	9999	1
10000	9991	6	9997	9997	3
10000	9986	14	10000	9986	0

Tabela 5.3: Perdas de envio e recepção de mensagens UDP usando a demo.

Como é possível observar na tabela 5.3, o pior caso dos testes efetuados foi a perda de 23 mensagens, o que representa um valor pouco significativo comparado com 19977 mensagens enviadas com sucesso. Verificou-se também que a leitura da estrutura nos dois terminais foi efetuada com sucesso.

Com o microcontrolador a correr com o código desenvolvido utilizando a *stack* TCP/IP foram obtidos os resultados apresentados na tabela 5.4.

PIC32-MAXI-WEB			Process Controller		
Msg. enviadas	Msg. recebidas	Perdas na receção	Msg. recebidas	Msg. enviadas	Perdas na receção
10000	10000	0	10000	10000	0
10000	10000	0	10000	10000	0
10000	10000	0	10000	10000	0
10000	10000	0	10000	10000	0
10000	10000	0	10000	10000	0

Tabela 5.4: Perdas de envio e recepção de mensagens UDP usando a *stack* TCP/IP.

Como se pode observar pelas tabelas apresentadas, o código desenvolvido para PIC32-MAXI-WEB utilizando apenas a *stack* TCP/IP apresenta melhores resultados em comparação à demo visto que não foram perdidos pacotes na troca de mensagens entre o *Process Controller* e o microcontrolador.

A diferença de comportamento entre os dois testes deve-se ao facto de a demonstração que serviu de base ao primeiro teste ser baseada no FreeRTOS.

Dadas as limitações de desempenho da plataforma computacional, a sobrecarga devida à presença do FreeRTOS causou diversos problemas, entre eles a perda de mensagens aqui reportada e a perda de impulso do *encoder*. Por esta razão optou-se por desenvolver o software de raiz, sem usar o FreeRTOS.

5.4 Sistemas de testes

Nesta secção apresentam-se os resultados de um conjunto de testes realizados tendo por objetivo avaliar o funcionamento global do sistema. Estes testes encontram-se organizados em dois grupos. Num deles é usado um *switch Ethernet* Commercial Off-The-Shelf (COTS) da Cisco, enquanto no outro usa-se um *switch Ethernet* tempo-real HaRTES. Para cada um destes grupos testou-se o desempenho com e sem tráfego de interferência. Este tráfego é produzido pelos nós denominados "Gerador de tráfego", tal como representados na figura 5.2.

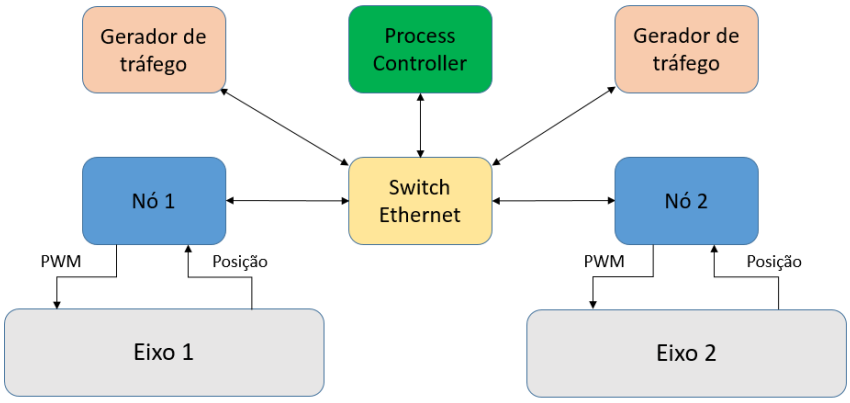


Figura 5.2: Diagrama de blocos do sistema de testes.

A figura 5.3 representa o percurso predefinido para a calha percorrer durante os testes efetuados. O percurso predefinido consiste em a calha descrever a trajetória descrita na figura 5.3 seis vezes sem pausas a uma velocidade constante de 4 cm/s. O controlo para os dois eixos do sistema físico é idêntico, com a excepção da distância que cada calha pode percorrer, tendo em conta que o comprimento dos dois eixos é diferente. Desta forma, serão apenas apresentados os resultados obtidos para o eixo de maior comprimento, visto que os resultados para o eixo de menor comprimento são idênticos, diferenciando-se apenas pela duração dos testes. A duração do teste para o eixo de maior comprimento é de aproximadamente 34 segundos. Para os testes realizados com tráfego de interferência será possível observar que o *jitter* afetará diretamente a velocidade e posição da calha. O controlador PID foi projetado para uma frequência de amostragem de 370

Hz. Se as mensagens chegarem atrasadas ou ocorrer perda de mensagens o controlador verá o seu desempenho degradado, com variações de velocidade excessivas. Relativamente à posição, se ocorrerem perdas de mensagens a calha poderá parar numa posição mais à frente do que seria esperado, ou até mesmo embater numa das extremidades do eixo, visto que o *Process Controller* não receberá as mensagens que contêm o número de impulsos atual e não conseguirá verificar se a calha chegou à posição final e informar o microcontrolador para terminar o movimento ou para trocar o sentido deste. Este teste permite também verificar visualmente a variação de velocidade da calha no caso de ocorrerem eventuais falhas no sistema ou até mesmo, no caso de perdas de mensagens, a calha embater numa das extremidades.

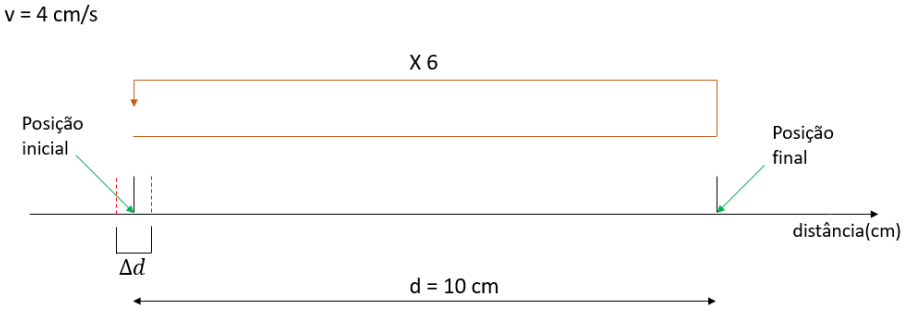


Figura 5.3: Percurso da calha para o sistema de testes.

Durante cada teste os resultados foram guardados num *array* no *Process Controller*. Após a execução de cada teste, os dados foram guardados num ficheiro de texto. Com recurso ao MATLAB, foram lidos os dados dos ficheiros sendo depois gerados os gráficos que representam os resultados obtidos.

5.5 Resultados dos testes com o *switch* tradicional

No teste que será apresentado de seguida foi utilizado um *switch* COTS da Cisco, modelo Ws-c2960-8tc-l. Como foi mencionado anteriormente, este teste foi realizado por duas vezes, uma vez sem qualquer tráfego de interferência e uma segunda vez em que foi introduzido no sistema tráfego de interferência. Os geradores de tráfego, representados no sistema de teste (5.2), correspondem a dois PC's que enviam mensagens UDP para o endereço IP do *Process Controller*. Cada gerador de tráfego envia mensagens que têm 141 *bytes* de tamanho e são enviadas a cada 31 μs. Este tráfego de interferência corresponde a uma carga total de 72.26% do sistema de comunicação, visto que este trabalha a $100 Mbits/s$. O *Process Controller* irá rejeitar estas mensagens de interferência e tentar responder atentadamente às mensagens que chegam provenientes do microcontrolador. Neste terminal

sempre que chegava uma mensagem do microcontrolador era guardado num *array* a seguinte informação:

- Diferença do número de impulsos lido na mensagem atual relativamente à mensagem anterior recebida.
- Valor da velocidade instantânea.
- Intervalo de tempo entre cada mensagem.
- Instante de tempo em que a mensagem chegou.

Os resultados obtidos, com e sem tráfego de interferência, foram os seguintes:

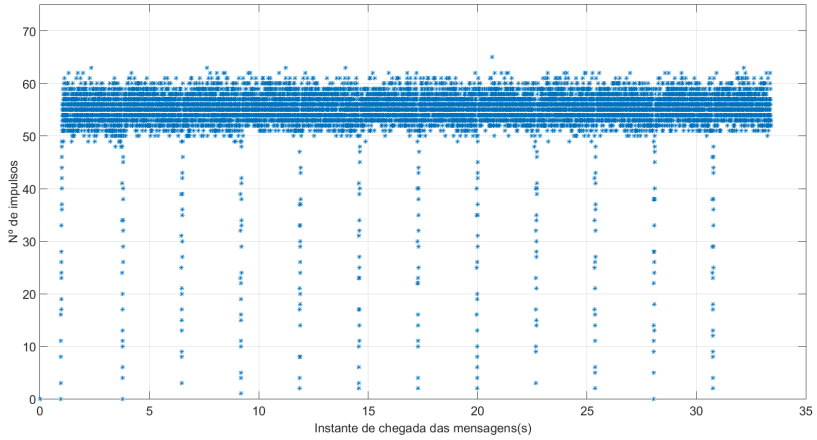


Figura 5.4: Número de impulsos entre mensagens sem interferência (Cisco).

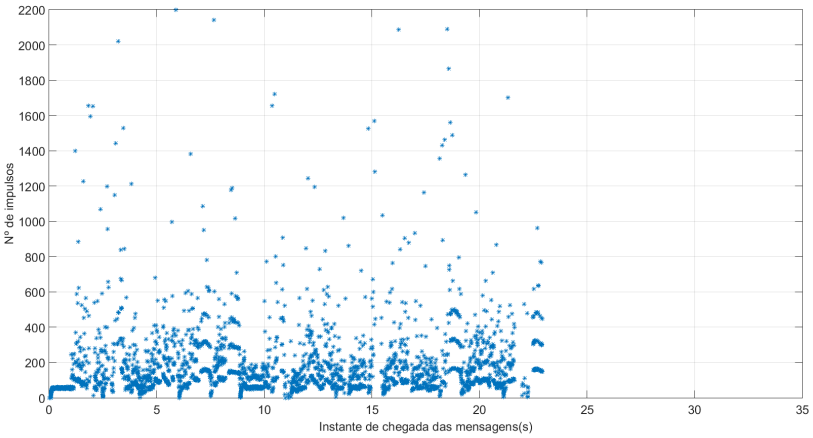


Figura 5.5: Número de impulsos entre mensagens com interferência (Cisco).

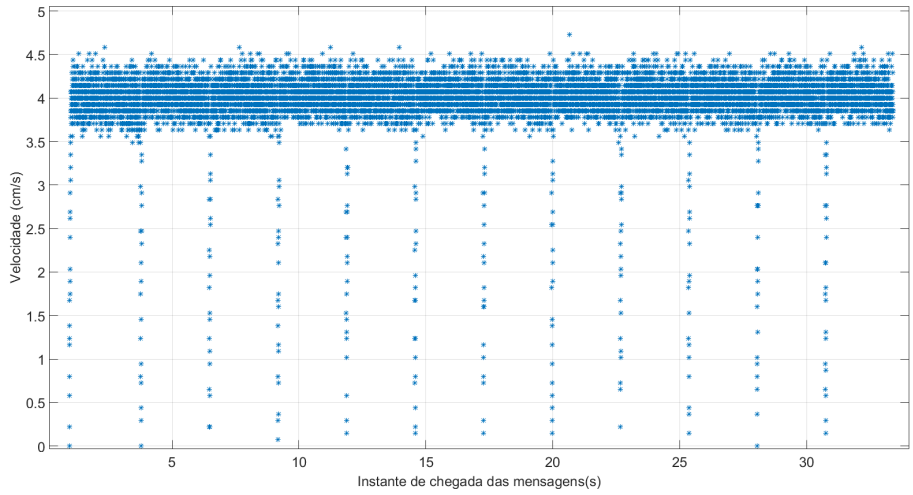


Figura 5.6: Velocidade calculada sem interferência (Cisco).

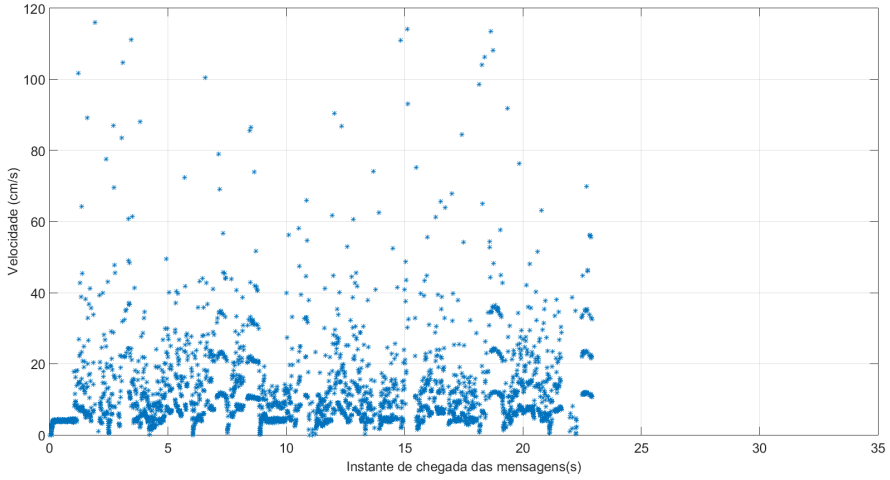


Figura 5.7: Velocidade calculada com interferência (Cisco).

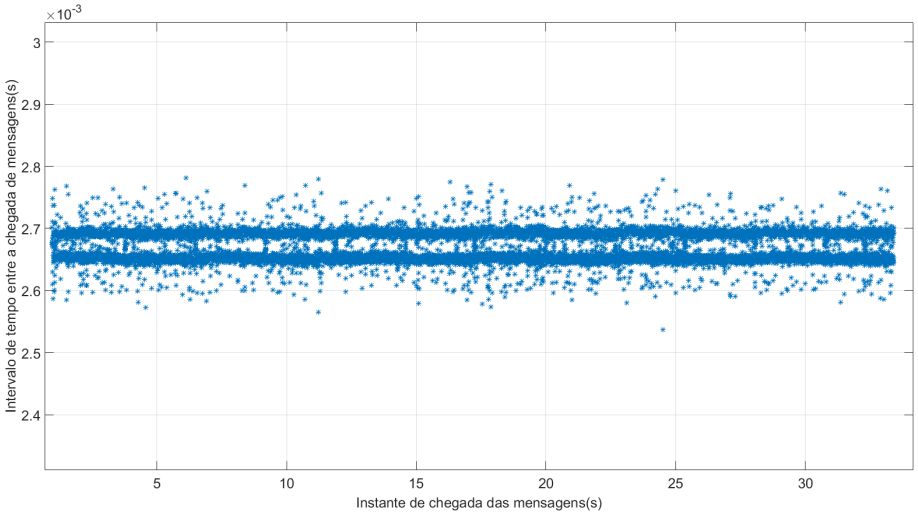


Figura 5.8: Intervalo de tempo entre mensagens sem interferência (Cisco).

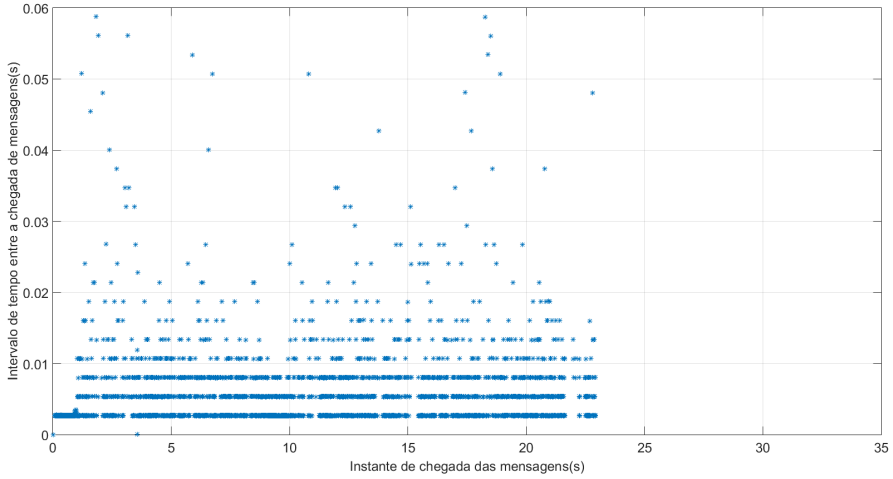


Figura 5.9: Intervalo de tempo entre mensagens com interferência (Cisco).

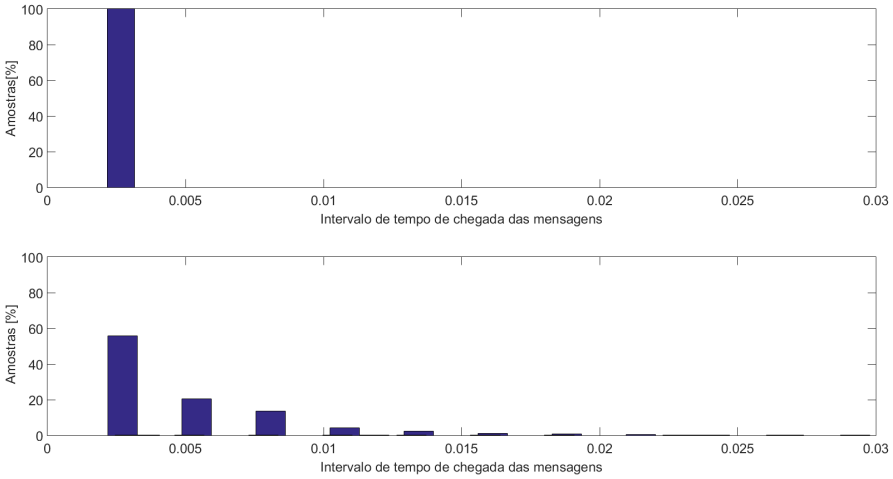


Figura 5.10: Histograma do nº de impulsos relativamente ao instante de tempo de chegada das mensagens sem e com interferência (cisco).

Nas figuras 5.4 e 5.5 é representada a diferença do número de impulsos lidos entre mensagens. No teste sem interferência é possível observar que a diferença entre o número de impulsos se encontra maioritariamente entre os 50 e os 60 impulsos (figura 5.4). As amostras apresentadas entre os 0 e 50 impulsos, na figura 5.4, representam a situação em que a calha se encontra parada e inicia o movimento até estabilizar em torno do valor pretendido e quando esta atinge a posição final pretendida. No caso em que existe tráfego de interferência apresentado na figura 5.5 verificou-se que o sistema se tornou bastante instável. O tráfego de interferência foi introduzido aproximadamente 1.2 segundos após a calha iniciar o movimento. Neste teste verificou-se que a amostras chegaram até um valor 40 vezes superior ao valor pretendido.

As amostras de velocidade são calculadas dependendo da diferença do número de impulsos lido entre mensagens. No teste sem interferência, figura 5.6, a velocidade estabiliza em torno do valor pretendido com um erro máximo de 19.2%. Tal como na figura 5.4 as amostras representadas entre 0 e 3.5 cm/s correspondem ao arranque da calha até estabilizar em torno do valor pretendido e ao instante em que esta chega à posição desejada e para o movimento. No teste com interferência, figura 5.7, verifica-se que as amostras da velocidade dispersam bastante em relação ao valor pretendido. Como foi mencionado anteriormente, o controlador PID foi dimensionado uma frequência de amostragem de 370 Hz sendo que no teste com interferência o atraso de chegada das mensagens e as perdas destas prejudicam significativamente o controlo da velocidade. Como se pode observar na figura 5.7 existem amostras de velocidade perto dos 120 cm/s que é um valor de cerca 30 vezes superior ao valor desejado.

Na figura 5.8 está representado o intervalo de tempo entre a chegada das mensagens em função do instante de tempo da sua chegada ao *Process Controller* sem interferência. O *jitter* médio é de 27.74 μs com um *jitter* máximo de 174 μs, correspondentes a 1.02% e 6.44%, respetivamente, em relação ao período de amostragem. No caso em que o sistema foi testado com tráfego de interferência, figura 5.9, verificou-se que houve uma perda 39.71% das mensagens enviadas o que corresponde a uma proporção de aproximadamente 4 mensagens perdidas a cada 10 mensagens enviadas. O *jitter* máximo verificado foi de 56.6 *ms* enquanto o *jitter* médio calculado foi de 3.6 *ms*, correspondentes a 2096.3% e 133.3%, respetivamente, em relação ao período de amostragem.

As figuras relativas ao número de impulsos lidos entre mensagens, amostras do cálculo de velocidade e intervalo de tempo de chegada das mensagens não foram representadas na mesma escala vertical visto que no teste efetuado com a presença de tráfego de interferência a perda de mensagens e o atraso de chegada destas causa a instabilidade do sistema e são calculadas amostras que se encontram numa escala bastante distante dos valores que eram pretendidos.

A figura 5.10 apresenta o histograma que representa o número de mensagens que chegaram, em percentagem, com os respetivos intervalos de tempo, no teste sem tráfego de interferência e com tráfego de interferência, permitindo obter uma visão mais clara dos atrasos de chegada das mensagens relativamente ao que era desejado. É possível observar, no teste em que existia tráfego de interferência na rede, que ocorreram atrasos significativos na chegada das mensagens. Neste teste, aproximadamente 60% das mensagens chegaram ao *Process Controller* com um intervalo de tempo semelhante ao valor verificado no teste efetuado sem interferência.

5.6 Resultados dos testes com o *switch* HaRTES

Neste segundo conjunto de testes foi usado um *switch* HaRTES. À semelhança do caso anterior, o sistema foi avaliado sem e com tráfego de interferência. À semelhança do teste descrito com o *switch* tradicional, o tráfego de interferência consiste em mensagens UDP com 141 *bytes*, de tamanho, que são enviadas, a cada 31 μs, para o endereço IP do *Process Controller* que as irá rejeitar. No *switch* HaRTES foram configurados servidores de modo a separar o tráfego de tempo real do tráfego de interferência. Desta forma o microcontrolador foi ligado ao servidor com maior prioridade e os geradores de tráfego de interferência foram ligados aos servidores configurados com menor prioridade. O servidor em que foi ligado o microcontrolador foi configurado com tamanho de 100 *bytes* e um com período de 2 *ms*. O EC foi definido como 1 *ms* e atribuídos 150 μs para a janela síncrona, 700 μs para janela assíncrona e o restante para *trigger-message*, sendo o EC representado

pela figura 2.8. Os *clocks* internos do *Process Controller*, do microcontrolador e do *switch* HaRTES são independentes. Desta forma o tempo atribuído à janela assíncrona é o tempo disponível para o envio das mensagens que o microcontrolador envia para o *Process Controller*. Na situação em que existam duas mensagens para ser enviadas no mesmo período do servidor, dedicado ao microcontrolador, é enviada a mensagem que está mais tempo à espera e a outra é enviada no próximo período. Os resultados obtidos foram os seguintes:

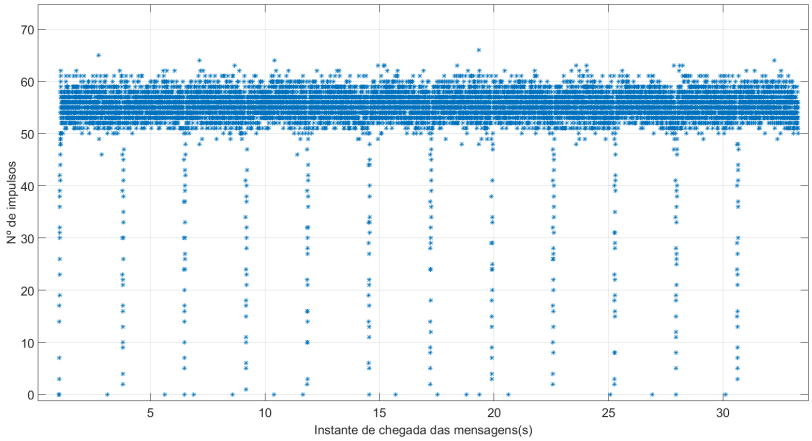


Figura 5.11: Número de impulsos entre mensagens sem interferência (HaRTES).

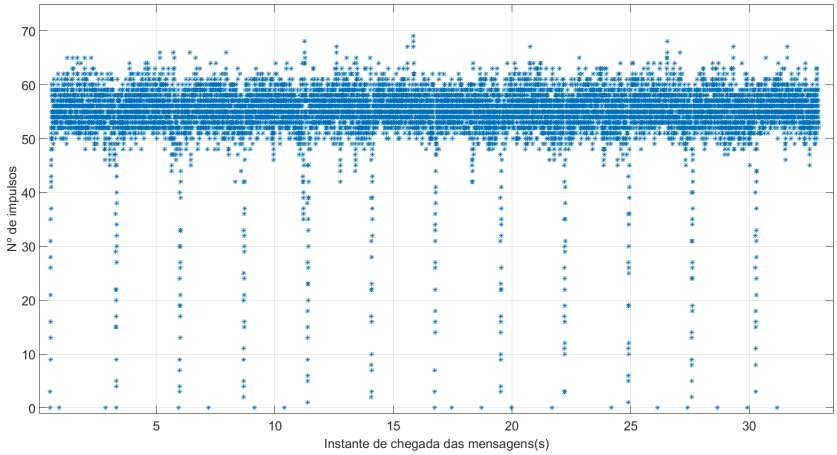


Figura 5.12: Número de impulsos entre mensagens com interferência (HaRTES).

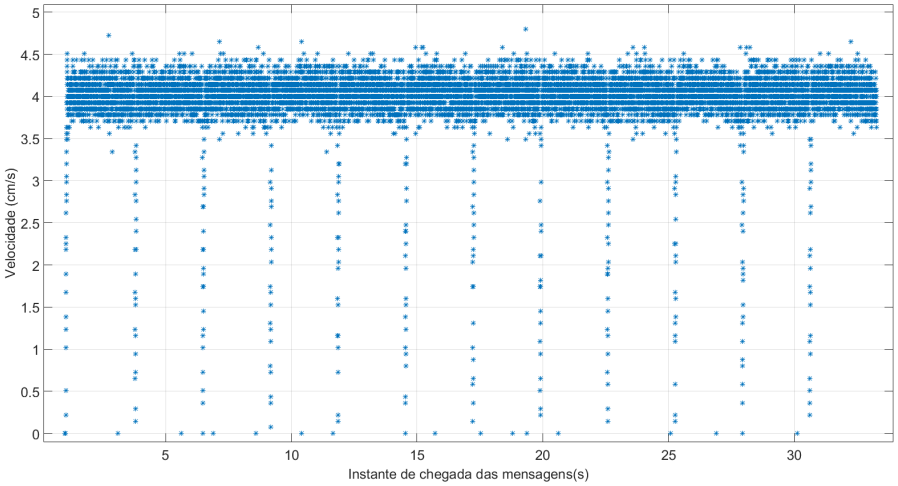


Figura 5.13: Velocidade calculada sem interferência (HaRTES).

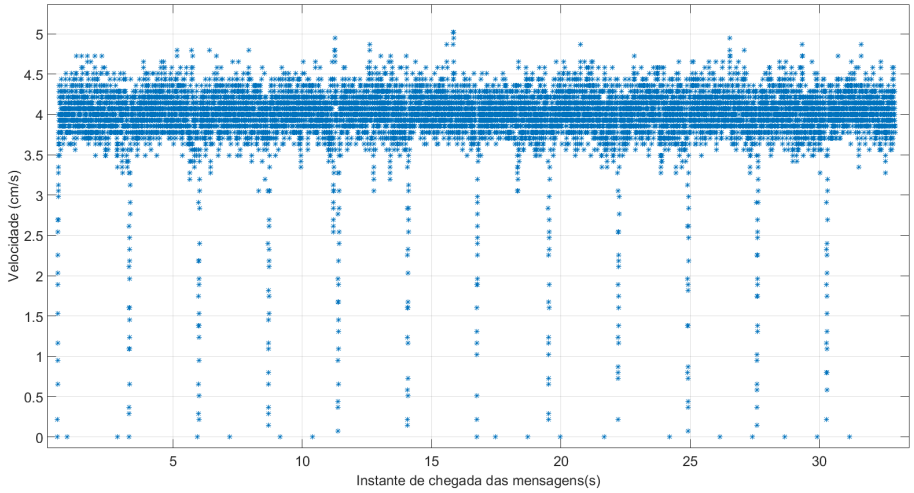


Figura 5.14: Velocidade calculada com interferência (HaRTES).

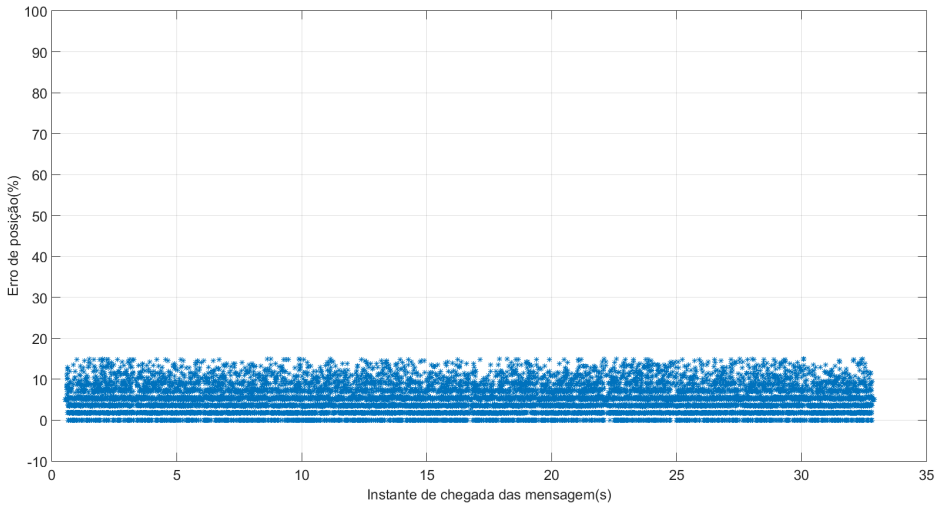


Figura 5.15: Erro da posição em percentagem (HaRTES).

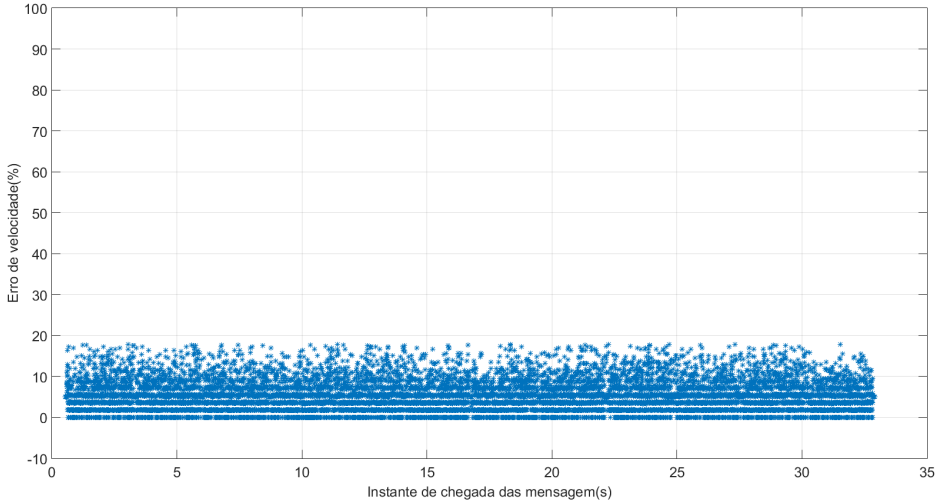


Figura 5.16: Erro da velocidade em percentagem (HaRTES).

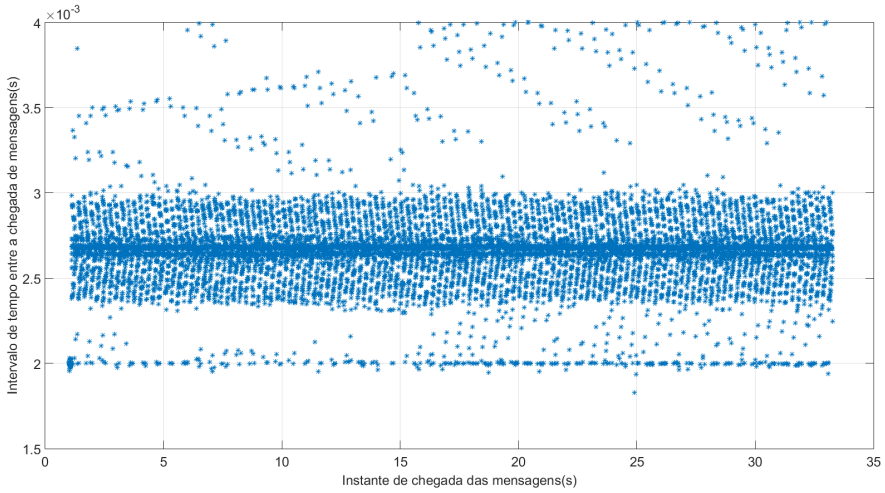


Figura 5.17: Intervalo de tempo entre mensagens sem interferência (HaRTES).

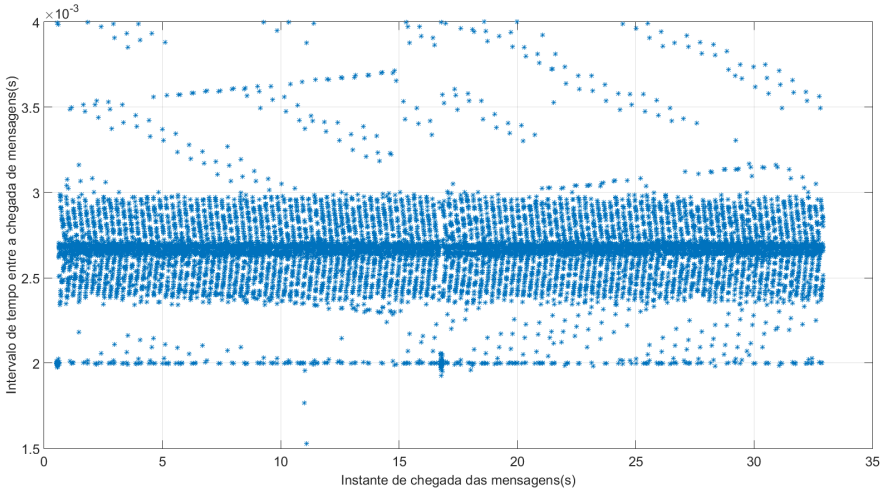


Figura 5.18: Intervalo de tempo entre mensagens com interferência (HaRTES).

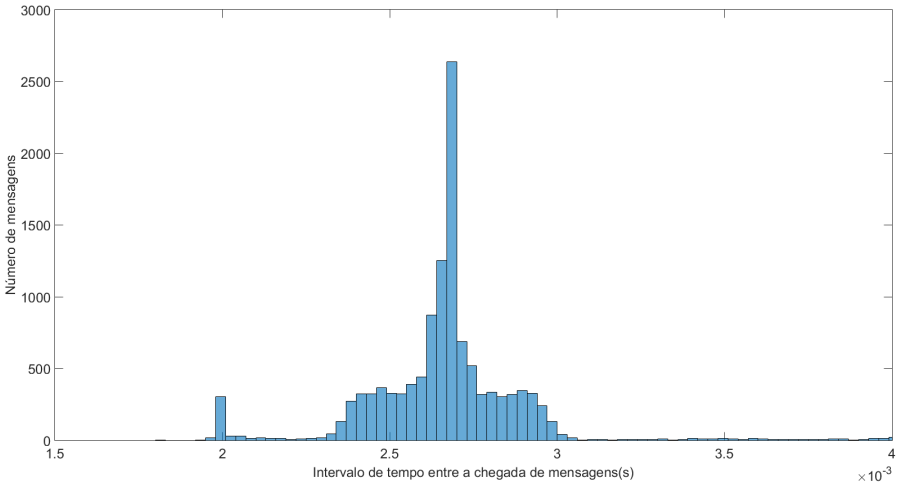


Figura 5.19: Histograma do nº de impulsos relativamente ao instante de tempo de chegada das mensagens sem interferência (HaRTES).

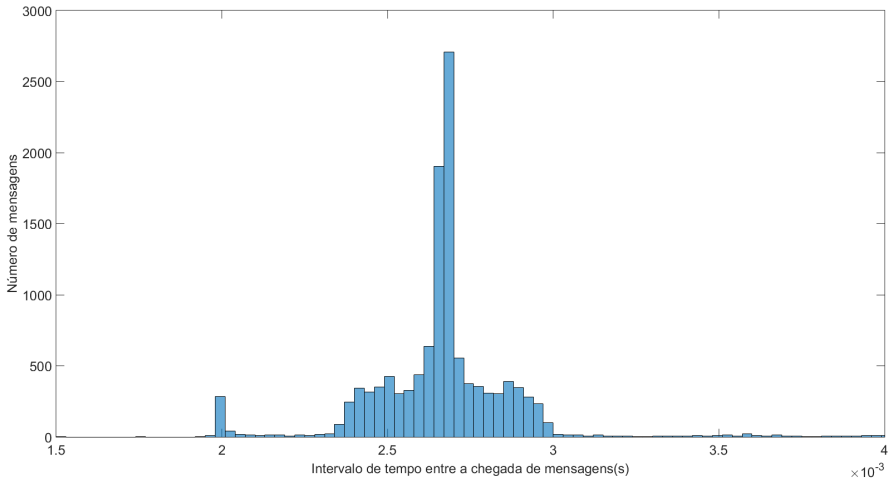


Figura 5.20: Histograma do nº de impulsos relativamente ao instante de tempo de chegada das mensagens com interferência (HaRTES).

Nas figuras 5.11 e 5.12 é representada a diferença do número de impulsos lidos entre mensagens com e sem tráfego de interferência. No teste efetuado sem tráfego de interferência o número de impulsos lido, figura 5.11, encontra-se maioritariamente entre os 50 e 60 impulsos tal como acontecera com o *switch* tradicional. No teste em que foi introduzido tráfego de interferência verificou-se que o impacto deste, no que diz respeito à determinação do número de impulsos, é pouco significativo como se pode observar na figura

5.12. Na figura 5.15 é possível visualizar a percentagem do erro do número de impulsos lidos entre mensagens com tráfego de interferência em relação ao número de impulsos lidos sem tráfego de interferência. O valor máximo do erro em percentagem verificado foi de 18.2% e a percentagem do erro média é apresentada como 7.1%.

Nas figuras 5.13 e 5.14 estão representadas as amostras de velocidade calculadas no teste sem interferência e com interferência respetivamente. É possível observar que o gráfico das amostras de velocidade calculadas sem a presença de tráfego de interferência no sistema é idêntico ao mesmo teste efetuado com o *switch* tradicional. No caso em que existe tráfego de interferência no sistema de comunicação é possível observar pela figura 5.14 que o gráfico das amostras de velocidade é bastante semelhante ao gráfico apresentado sem interferência. ao contrário do que foi verificado nos testes realizados com o *switch* da Cisco. Na figura 5.16 é representado o erro em percentagem das amostras de velocidade calculadas com tráfego de interferência, no sistema de comunicação, relativamente às amostras de velocidade sem a presença de tráfego. O erro percentual médio é 7.7% com um erro percentual máximo de 19.6%.

Na figura 5.17 está representado o intervalo de tempo de chegada das mensagens em função do instante de chegada destas na situação em que não há tráfego de interferência no sistema de comunicação. O *jitter* médio nesta situação é apresentado como 33.8 μs com um *jitter* máximo de 1.3 *ms*, correspondentes a 1.25% e 48.15%, respetivamente, em relação ao período de amostragem. Na situação em que existe tráfego de interferência na rede o intervalo de tempo entre a chegada de mensagens é bastante semelhante à situação em que não existe tráfego de interferência como se pode observar na figura 5.18. O *jitter* médio para a situação em que há tráfego de interferência é 39 μs com um *jitter* máximo de 1.3 *ms*, correspondentes a 1.44% e 48.15%, respetivamente, em relação ao período de amostragem.

Foram realizados dois histogramas, tal como para os testes com o *switch* tradicional, que representam o número de mensagens que chegaram com os respetivos intervalos de tempo no teste efetuado com a presença de tráfego de interferência e sem tráfego, figura 5.20 e 5.19 respetivamente. Como é possível observar, estas duas figuras são bastante semelhantes sendo o intervalo de tempo com o valor 2.7 *ms* o mais preenchido nas duas figuras, sendo este o valor esperado visto que corresponde ao intervalo de tempo com que o microcontrolador envia mensagens para o *Process Controller*. Os picos que se observam em 2ms correspondem a situações em que o microcontrolador envia mensagens durante a fase do EC reservada para a TM e tráfego periódico. Nestes casos as mensagens ficam temporariamente na fila do *switch*, sendo enviadas no início da janela assíncrona, a cada 2 EC's. Dado que os relógios não se encontram sincronizados, este cenário pode ocorrer durante parte da experiência.

A tabela 5.5 sumaria os resultados dos testes efetuados com e sem interfe-

rência, com o *switch* HaRTES e com *switch* tradicional, incluindo os valores do *jitter* máximo e médio e erro máximo e médio em percentagem. Os valores de erro apresentados foram calculados usando os valores de velocidade e número de impulsos pretendidos como referência.

Da análise da tabela 5.5 observa-se que o *switch* HaRTES é capaz de gerir o tráfego de tempo real e não ser afetado por tráfego que não é de tempo real.

	Switch COTS		Switch HaRTES	
	S. interf.	C. interf.	S. interf.	C. interf.
jitter médio (μs)	27.40	3600.00	33.80	39.00
jitter máximo (μs)	174.00	56600.00	1300.00	1300.00
erro médio impulsos(%)	2.37	238.10	0.42	0.67
erro máximo impulsos (%)	12.73	386.30	19.63	23.42
erro médio velocidade(%)	0.38	237.50	0.41	0.71
erro máximo velocidade(%)	19.20	2825.00	16.75	23.25

Tabela 5.5: Resultados obtidos com *switch* tradicional e *switch* HaRTES.

Capítulo 6

Conclusões e trabalho futuro

Neste capítulo é feita uma avaliação global do trabalho e o enquadramento dos resultados obtidos com os objetivos pretendidos. Serão também apresentadas propostas para trabalho futuro.

Os *device drivers* foram devidamente adaptados de modo a controlar os motores DC do demonstrador.

O demonstrador foi devidamente adaptado, tendo sido utilizado o *hardware* de interface disponível, juntamente com os *device drivers*, e criado o *software* de controlo necessário de modo a controlar os eixos do sistema físico.

O controlo individual dos dois eixos, pertencentes ao demonstrador, foi realizado com sucesso, tendo-se conseguido aplicar o controlo de posição com um erro inferior ou igual a 1% e o controlo de velocidade com um erro inferior ou igual a 6.67%. Estes valores de erro demonstram que o controlo aplicado ao sistema foi feito de forma eficaz e precisa.

Verificou-se que é possível efetuar o controlo do sistema com uma arquitetura distribuída, em que os nós comunicam via UDP, desde que a camada de rede ofereça uma qualidade de serviço adequada. No teste da troca de mensagens, utilizando o *software* demo, verificou-se que houve no pior dos casos uma perda de 23 mensagens num total de 10000, correspondendo a uma perda de 0.23%. No teste com o código desenvolvido de raiz, verificou-se que não houve perda de mensagens, sendo este o resultado desejado visto que a perda de impulsos degradaria o desempenho do sistema.

O sistema de comunicação baseado no *switch* HaRTES foi testado e foram comprovadas as suas mais valias em comparação com um *switch Ethernet* tradicional. Nos testes efetuados com o *switch* HaRTES observou-se que este é capaz de gerir o tráfego de tempo real e não ser afetado por tráfego que não é de tempo real, repercutindo-se num bom funcionamento do demonstrador na presença e na ausência de tráfego de interferência. No teste efetuado com o *switch* da Cisco, observou-se que quando existia tráfego de interferência na rede o demonstrador comportava-se de uma forma anómala, visto que este *switch* não é capaz de gerir o tráfego que é de tempo real e é afetado por

tráfego que não é de tempo real.

Como trabalho futuro, será necessário:

- Aplicar o controlo de posição e velocidade utilizando os perfis de velocidade criados para o demonstrador.
- Verificar se a velocidade é alterada no instante certo e na posição pretendida de acordo com os perfis predefinidos.
- Criar uma aplicação que permita ao demonstrador elaborar uma figura escrita num papel.

Uma sugestão como trabalho futuro será adicionar novos demonstradores ao sistema de comunicação e criar aplicações de teste de modo a testar o seu funcionamento.

Bibliografia

[1] R. Marau, L. Almeida, and P. Pedreiras, *Enhancing real-time communication over cots ethernet switches*, vol. 6. 2006.

[2] I. Lee, J. Y. Leung, and S. H. Son, *Handbook of real-time and embedded systems.* CRC Press, 2007.

[3] K. Hafner, *Honey, I programmed the blanket*, vol. 27. 1999.

[4] J. Axelson, *Embedded Ethernet and Internet Complete: Designing and Programming Small Devices for Networking.* lakeview research llc, 2003.

[5] A. P. de Melo, *Teoria dos sistemas de controlo lineares.* Universidade de Aveiro, 2010.

[6] A. Visioli, *Practical PID control.* Springer Science & Business Media, 2006.

[7] A. M. Mota, *Electrónica IV, Controladores Elementares.* Fevereiro de 2003.

[8] R. Baheti and H. Gill, *Cyber-physical systems*, vol. 12. IEEE Control Systems Society New York, NY, USA, 2011.

[9] G. B. Machado *et al.*, *Uma arquitetura baseada em web services com diferenciação de serviços para integração de sistemas embutidos a outros sistemas.* Florianópolis, SC, 2006.

[10] H. Kopetz, *Real-time systems: design principles for distributed embedded applications.* Springer Science & Business Media, 2011.

[11] *H. Kopetz, Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers, 1997.*

[12] L. Almeida and P. Pedreiras, *Sistemas de Tempo Real.* DETI, Universidade de Aveiro, 2003.

[13] C. Spurgeon, *Ethernet: the definitive guide.* "O'Reilly Media, Inc.", 2000.

[14] C. Gomes, *FTT-SE: Desenvolvimento de um dissector para um protocolo de tempo real.* 2010.

[15] J. M. ROSARIO, *Automação industrial.* Editora Baraúna, 2009.

[16] ODVA, *EtherNet/IP Quick Start for Vendors Handbook: A Guide for EtherNet/IP Developers.* 2008.

[17] G. Sestito, *Uso de Ethernet em Automação Industrial.* Universidade de São Paulo, 2011.

[18] A. Lugli, *Uma visão do protocolo industrial Profinet e suas aplicações.* Acesso em, 2013.

[19] D. Tamas-Selicean, P. Pop, and W. Steiner, *Synthesis of communication schedules for TTEthernet-based mixed-criticality systems.* 2012.

[20] R. G. V. d. Santos, *Enhanced ethernet switching technology for adaptive hard real-time applications: Tecnologia de comutação ethernet melhorada para aplicações adaptativas e críticas de tempo-real.* Universidade de Averio, 2011.

[21] F. M. N. Amado, *Implementação de pilha protocolar tempo-real para vídeo industrial.* Universidade de Aveiro, 2010.

[22] R. J. G. Ribeiro, *Switch ethernet distribuído para sistemas embutidos.* Universidade de Aveiro, 2012.

[23] R. R. D. Marau, *Real-time communications over switched Ethernet supporting dynamic QoS management.* Universidade de Aveiro, 2009.

[24] L. E. M. d. Silva, *Ligação de alto desempenho entre FPGAs para switch ethernet FTT.* Universidade de Aveiro, 2010.

[25] P. Fonseca, *Sistemas de Instrumentação Electrónica.* Universidade de Aveiro, Departamento de Electrónica e Telecomunicações, Edição 2011/12.

[26] D. Yon and G. Camarillo, *TCP-Based Media Transport in the Session Description Protocol (SDP).* 2005.