Desenvolvimento de Algoritmos de Controlo para Locomoção de um Robot Humanóide

Relatório Final de Projecto

Milton Ruas da Silva Nº21824

Orientação:

Prof. Dr. Filipe Silva (DETI-IEETA)
Prof. Dr. Vítor Santos (DEM-TEMA)

Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática, IEETA
Licenciatura em Engenharia Electrónica e Telecomunicações

Julho de 2006

Índice

INTRODUÇÃO

A concepção de um robot Humanóide constitui um dos maiores desafios na área da robótica: construir um ser artificial antropomórfico semelhante ao homem é um sonho inato do nosso engenho, e não é para menos, pois o ser humano é a forma de vida mais complexa existente à face da Terra. O século XX encheu a nossa imaginação com livros e filmes que demonstram esse sonho do ser artificial capaz de, além de ajudar a desempenhar tarefas, aprender coisas por si mesmo, interagir connosco, expressar emoções, possuir uma consciência própria... tudo características que por enquanto ainda consideramos como puramente humanas. Marcas como a Honda, a Sony ou a Fujitsu já deram os primeiros passos no desenvolvimento de máquinas que imitam os comportamentos físicos dos seres humanos, como caminhar, dançar ou pegar objectos. Outros passos também já foram dados no que respeita ao processamento de visão, de som, com o objectivo de realizar tarefas ou simplesmente de interagir com o ser humano.

Fig. 1: QRIO da Sony.

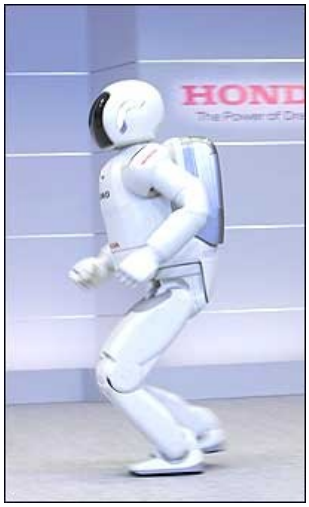


Fig. 2: Asimo da Honda.

Fig. 3: KHR 3 (Hubo Lab).

ENQUADRAMENTO DO PROJECTO

Muitos outros grupos de investigação iniciaram a construção de robôs de baixo custo no sentido de realizarem investigação em áreas tão diversas como o controlo, a percepção, a navegação, o comportamento ou a cooperação. Este foi, também, o móbil principal que levou um grupo do Departamento de Engenharia Mecânica da Universidade de Aveiro a encetar a tarefa de construção de uma tal plataforma. O estado actual de desenvolvimento perspectiva a abordagem de algoritmos eficientes ao nível do controlo, planeamento e percepção.

A motivação para os projectos propostos na área da Robótica Humanóide é encontrada em diversas vertentes, das quais se destacam as seguintes:

- ➔ A aposta nos robôs humanóides como a via mais promissora para chegar a sistemas de elevada mobilidade, versatilidade de operação e facilidade de interacção com os humanos.
- ➔ A criação de uma plataforma de investigação de grande valor pedagógico face aos enormes desafios científicos e técnicos, à diversidade de problemas, ferramentas e níveis de integração.
- ➔ A promoção do envolvimento de um grupo de estudantes da UA em competições robóticas internacionais. Por exemplo, o ROBOCUP e o FIRA são duas organizações internacionais que realizam anualmente competições na classe dos humanóides.

OBJECTIVOS DO TRABALHO

Neste trabalho pretende-se estudar, desenvolver e implementar um conjunto de estratégias e algoritmos simples de controlo para o robô humanóide. O projecto pode ser decomposto pelas seguintes fases:

1. Análise do estado actual de desenvolvimento do sistema e compreensão dos problemas tecnológicos envolvidos:
   - Avaliação do desempenho ao nível do controlo, planeamento e percepção tendo em conta os requisitos físicos e funcionais colocados pela participação no ROBOCUP.
   - Introduzir alguns melhoramentos possíveis ao sistema existente.
2. Melhoramento dos algoritmos de comunicação entre os diversos nós da arquitectura, de modo a optimizar a troca de informação relativa a actuação e leituras sensoriais;
3. Melhoramento dos algoritmos de actuação e leitura sensorial dos actuadores;
4. Estudo da melhor estratégia de controlo a adoptar nas juntas de modo a realizar comportamentos relativos à locomoção: o estudo do comportamento de um servomotor foi efectuado de modo a analisar qual a melhor estratégia de controlo;
5. Posteriormente testou-se a estratégia delineada efectuando movimentos com os membros inferiores da estrutura humanóide (pernas).
6. Contemplou-se comportamentos especiais relativos ao equilíbrio estático e dinâmico: como por exemplo equilibrar uma perna na vertical na variação do plano de suporte.

Por isso, o relatório está organizado em dois capítulos principais:

- Capítulo 1 – Implementação dos algoritmos de troca de informação entre os diversos módulos;
- Capítulo 2 – Implementação dos algoritmos de actuação directa e leitura sensorial dos servomotores e estudo da resposta dos actuadores introduzindo controladores externos para a compensação.

DESCRIÇÃO DA PLATAFORMA HUMANÓIDE

A plataforma humanóide possui um conjunto de 22 graus de liberdade, distribuídos da seguinte forma:

- 2 em cada pé (2x2);
- 1 em cada joelho (1x2);
- 3 em cada anca (3x2);
- 2 no tronco (2x1);
- 3 em cada braço (3x2);
- 2 no suporte da câmara (cabeça) (2x1).

A estrutura é constituída essencialmente por alumínio e aço nos eixos e outros pequenos componentes, pesando um total de 6Kg com as baterias incluídas, e medindo cerca de 60 cm. Estes valores foram estabelecidos de acordo com as regras impostas pelo RoboCup, baseando-se no pressuposto de que para valores superiores estes, o uso de servomotores de baixo custo poderá tornar-se inviável dada a impossibilidade de conciliar binários de motores e pesos dos equipamentos e acessórios como as baterias.
Por razões de estética e de acomodação de componentes, foi adoptada uma estrutura em forma de exoesqueleto (carapaça) dotando assim o sistema de módulos ocos onde são alojados os motores, sensores, clablagens, placas de controlo, etc (Fig. 4).

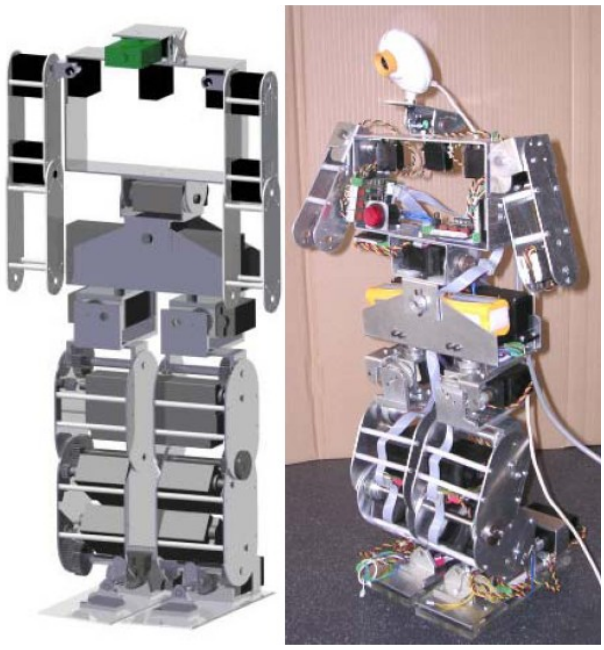


Fig. 4 - Modelo 3D do robot e implementação actual.

Preocupações com a autonomia energética do sistema, exigiram a escolha de baterias de elevada capacidade de corrente, até porque é necessário alimentar 22 motores de binário apreciável (embora uns mais do que outros). Tendo em conta esses aspectos utilizaram-se duas baterias ligadas de 7.4V ligadas em paralelo com uma capacidade máxima de 9600mAh.

No que respeita ao controlo, foram assumidas, desde logo, as vantagens de uma arquitectura distribuída e modular baseada num *bus* CAN responsável por permitir a troca de informação entre as diversas unidades de controlo a partir de uma unidade mestre que faz a gestão da rede e que está directamente ligada a uma unidade primária de decisão, baseada, neste momento, num computador vulgar. Posteriormente substituir-se-á o computador por uma *embedded motherboard* do tipo nano ITX com as mesmas funcionalidades que um PC, no que respeita ao necessário, com as vantagens de dimensões reduzidas e de baixo custo.

Neste momento, a plataforma é constituída por 9 unidades de controlo, 8 de controlo local dos actuadores e sensores, e um de controlo de tráfego na rede CAN. As unidades de controlo local estão distribuídas de forma a agrupar conjuntos de três actuadores relativos a um determinado membro, como é o caso das pernas

ou dos braços (Fig. 5). Com uma arquitectura deste género, pretende-se pretende-se que o *hardware* que constitui estes módulos seja idêntico com um software adaptado a adaptado a cada um deles. Implementando esta estratégia consegue-se um grau de fiabilidade superior, uma vez que os módulos são independentes permitindo que as anomalias sejam mais facilmente detectadas e corrigidas. Os módulos podem ser trocados facilmente e adaptados à tarefa necessária, pois bastará programar o algoritmo específico à tarefa.
Em resumo, apresentam-se as vantagens da arquitectura distribuída:

- Sistemas fiáveis (operação independente)
- Sistemas de controlo mais simples
- Mais fácil detecção de anomalias
- Actualização fácil de *firmware*

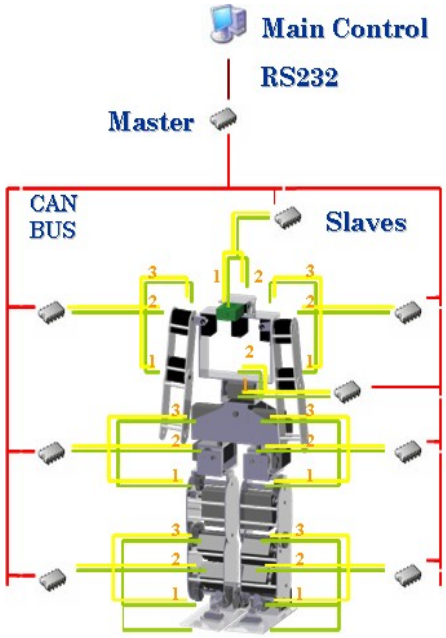


Fig. 5 - Arquitectura distribuída da plataforma.

Na percepção destacam-se os sensores proprioceptivos e alguns inerciais:

- Sensores de força para medição das forças de reacção dos pés;
- Potenciómetros de medição da posição das juntas.
- Inclinómetros e giroscópios para medição da aceleração gravítica e da velocidade angular respectivamente serão incorporados num futuro próximo.

Capítulo 1

1. ARQUITECTURA DAS COMUNICAÇÕES

Resumo:
Este capítulo descreve a forma como os diversos módulos estão organizados ao longo da estrutura humanóide e como é feita a comunicação entre eles de modo a que as ordens de actuação cheguem a cada junta com o mínimo de atraso temporal.

1.1 – ARQUITECTURA DO SISTEMA

O sistema de controlo implementado é, por isso, baseado numa configuração *master*/*slave*, e é constituído por três unidades diferentes ligadas em rede:

- A **unidade central** de controlo é responsável pela gestão global dos procedimentos, efectuando o cálculo das configurações que as juntas tem de adoptar em função dos valores dos sensores.
- A **unidade Master (mestre)** tem como principal tarefa servir de interface entre a unidade principal de controlo e as unidades slave;
- As **unidades Slave (escravo),** cujas principais funções são a geração da onda de pulso modulado (PWM) de controlo dos servomotores e a aquisição dos sinais dos diversos sensores da plataforma.

Entre os diversos nós são utilizados como meios de comunicação:

- Linha série ponto-a-ponto, baseada na norma **RS-232**, entre a unidade central e a unidade *Master*: acesso assíncrono byte a byte (8 bits) a um *baudrate* de 115200 bps.
- **CAN** (*Controller Area Network*) entre a unidade *master* e as unidades *slave*: é utilizada a versão *full*CAN 2.0A a uma taxa de transmissão/recepção de 833.3 Kbps.

A unidade central de controlo ainda não está completamente definida, permanecendo em aberto soluções baseadas num PDA, placas de controlo genéricas (como as baseadas no padrão PC104) ou placas de controlo dedicadas (Fig. 6). Por enquanto é utilizado um PC externo com recurso ao software *MatLab* para enviar e receber dados por uma linha série para o controlador *master*.

Para as unidades de controlo local (*master*/*slave*), a escolha recaiu sobre os microcontroladores PIC da série 18F da *Microchip* – PIC18F258 – por possuírem diversos periféricos e interfaces para redes de comunicações, incluindo o CAN (Fig. 8).

Fig. 6: Exemplo de uma board PC-104.

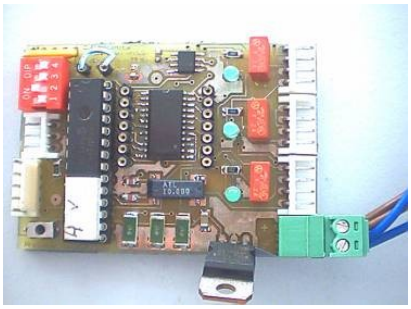


Fig. 7: Placa de controlo Master/Slave.

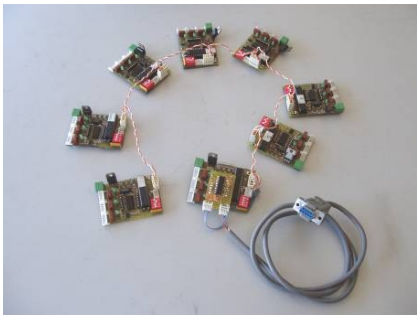


Fig. 8: Rede completa de Microcontroladores.

Até ao momento, a rede implementada é constituída por uma placa *master* (MCU – *Master Control Unit*) que efectua a interface entre a unidade principal e as unidades *slave*, e oito placas *slave* (SCU – *Slave Control Unit*) que efectuam o controlo local, permitindo o controlo de até três actuadores através da geração de uma onda de pulso modulado em largura (PWM), e a aquisição de até 16 sinais analógicos usando um *multiplexer* (Fig. 7).

Esta organização tem como objectivo agrupar as juntas que estão directamente relacionadas, como é o caso das juntas do tornozelo e do joelho que possuem um controlador dedicado, que, por aquisição dos sinais analógicos dos sensores de força instalados nos pés, pode controlar o equilíbrio por compensação em malha fechada. Numa evolução futura direccionada para a adaptação à irregularidade do solo, estas juntas poderão corrigir a sua posição de forma reactiva para que a projecção do centro de massa do robô não se situe fora da área de apoio dos pés. Obter-se-á assim um controlo localizado independente do resto do sistema sem que haja necessidade permanente de interagir com a unidade central de controlo.
Os sinais analógicos adquiridos actualmente, para além dos sensores de força, são os sinais provenientes dos potenciómetros de posição de cada actuador. Estes valores são convertidos e registados localmente e depois

enviados via CAN para o controlador *Master*. Os *Slaves* estão preparados também para receber mensagens via CAN. Estas mensagens consistem basicamente nas posições finais que os actuadores têm de tomar, a velocidade a que têm que se mover e os parâmetros de compensação PID para a realização do movimento.

O controlador *master* tem a tarefa de receber a informação enviada pelos *slaves* via CAN e registá-la para que esteja disponível para ser enviada para a unidade central de controlo quando solicitada. Este controlador mantém, por isso uma representação do estado actual das juntas (actuadores e sensores associados) que disponibilizará ao controlo central sempre que este o pedir. O processo é bidireccional e o controlador *master* também recebe as ordens da unidade central e despacha para o controlador *slave* respectivo.

Tabela 1: Características do Hardware.

Unidade Central de Controlo	**Computador com porta série RS-232** • Software de suporte: MatLab 7.01 (R14) • Device Drivers de comunicação série: cport v1.3
Unidades de controlo local *Master/Slave*	**PIC18F258 da Microchip** • Memória de programa: 2 MB • Memória de dados: 4 KB • Velocidade de processamento: 10 MIPS (f_{osc}=40MHz com a PLL activa) • Instruções de 16 bits e *datapath* de 8 bits • Definição de prioridades nas interrupções. • Diversos periféricos: *timers*, módulos CCP, interfaces para redes de comunicação, ADC, etc…

1.1.2. Protocolos de Comunicação

Desenvolveram-se dois protocolos de comunicação, nomeadamente para…
- a linha série RS-232 entre o PC e a unidade Master
- e para o CAN entre a unidade Master e os Slaves,

…de modo a poder trocar dados sensoriais e de actuação entre o PC e as unidades Slave.

Entre os dados sensoriais podem-se enumerar (para um SCU):
- Posição dos três servomotores (em graus);
- Velocidade correspondente (em graus/s);
- Corrente consumida por cada servomotor;
- Valores dos sensores de força de cada pé (quatro sensores por pé);
- Saída do Giroscópio (em graus/s)*;
- Saída do inclinómetro*.

E dos dados de actuação:
- Posição final a atingir por cada servomotor;
- Velocidade média para a realização do movimento;
- Parâmetros de controlo de posição do compensador PID (K_P, K_I e K_D).

Considera-se que o controlo de actuação deve ser feito de forma isolada a cada uma das três juntas, e por isso as diversas juntas podem realizar movimentos com diferentes velocidades e diferentes parâmetros de compensação PID. Tal é útil no último caso, pois cada junta pode estar sujeita a diferentes esforços, e por isso, é rentável aplicar diferentes parâmetros de controlo a cada uma.

* Pressupõe-se nesta versão que só existe um giroscópio e um inclinómetro ligados ao mesmo SCU.

1.1 – COMUNICAÇÃO RS-232

1.1.1. Protocolo

A comunicação RS-232 entre o PC e a unidade *Master* é efectuada assincronamente e é orientada ao byte (*start* bit+8 bits+*stop* bit), ou seja, é transmitido um byte de dados em cada transmissão/recepção. Pretende-se que numa única mensagem esteja contida toda a informação relativa a um parâmetro a ler/actuar das três juntas de um SCU, o que implica que cada mensagem seja constituída por vários bytes. Como por exemplo, para ordenar o SCU *X* a colocar as juntas nas posições *A*, *B* e *C* (três juntas), são necessários no mínimo quatro bytes: três para as três posições *A*, *B* e *C*; e um indicando a identificação do SCU.

Estrutura de um pacote USART (transmissão/recepção de um byte)

Comandos PC→Master:
As mensagens no sentido PC→*master* são constituídas por cinco bytes. O primeiro byte sinalizará a mensagem como sendo um comando de solicitação à unidade *master* (MESSAGE_REQ=*0x*FF); o segundo byte conterá um código (*opcode*) indicativo da operação a realizar, do SCU alvo e de parâmetros adicionais; e os últimos três bytes conterão parâmetros a atribuir às três juntas no caso de um comando de actuação, ou serão nulos se tratando-se de um comando de pedido de dados sensoriais.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*
MessageType=*0x*FF	OpCode	Joint 1	Joint 2	Joint 3

Mensagem USART PC→Master de actuação.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*
MessageType=*0x*FF	OpCode	0	0	0

Mensagem USART PC→Master de leitura sensorial.

Tabela 2: Campos das mensagens PC→Master via USART

Pacotes	Função:	Valores possíveis:
Message Type	Indica o tipo da mensagem	• *MESSAGE_REQ* (*0x*FF): Pedido de actuação/consulta • *MESSAGE_ERROR* (*0x*FE)*: Pedido de retransmissão da última mensagem enviada pelo *Master*. • *MESSAGE_TEST* (*0x*FA)*: Pedido de transmissão (por parte do *Master*) de uma mensagem de teste. A mensagem de resposta será da forma – FA F9 F8 F7 F6 F5 (hex).
OpCode	Código indicando o que é solicitado e a quem se destina (apenas útil para uma mensagem *MESSAGE_REQ*).	(Ver Tabela 3)
Joint 1/2/3	No caso de uma mensagem *MESSAGE_REQ* de actuação sobre as juntas, estes três bytes estabelecem valores a serem atingidos pelos actuadores.	• Mensagem de actuação: Posição final/velocidade média a atribuir a cada junta de um determinado SCU. • Mensagem de leitura sensorial: campos nulos.

(*) Nestas condições apenas é enviado o primeiro byte (*Message Type*), sem a necessidade dos restantes.

bit 7 6	5 4 3	2 1 *bit 0*
Operation	SCU *ID*	Parameter

Byte *OpCode*

Tabela 3: Campos do pacote *OpCode* nas mensagens PC→Master via USART.

SCU id	***Operation***	***Parameter***
SCU alvo relativo à operação (endereçável a 15 SCU's)	*OP_APPLY_JOINT* (*0b*00) Mensagem de actuação sobre as três juntas do SCU alvo (SCU id).	*PARAM_POSITION* (*0b*00) Posição (em graus) a ser atingida.
		PARAM_VELOCITY (*0b*01) Velocidade média do movimento a efectuar.
	OP_READ_JOINT (*0b*01) Mensagem de leitura dos dados sensoriais referentes às três juntas do SCU alvo.	*PARAM_POSITION* (*0b*00) Posição actual de cada junta.
		PARAM_VELOCITY (*0b*01) Velocidade média de cada junta.
		PARAM_CURRENT (*0b*10) Corrente drenada por cada servo.
		PARAM_CANSTATUS (*0b*11) Estado do barramento *CAN*.
	OP_READ_SENSORS (*0b*10) Mensagem de leitura dos sensores adicionais (sensores de força dos pés, giroscópio e inclinómetro).	SENSORS_REACTION – Left Foot (0b00) Leitura dos quatro sensores de força do pé esquerdo.
		SENSORS_REACTION – Right Foot (0b01) Leitura dos quatro sensores de força do pé direito.
		SENSORS_ORIENTATION (*0b*1x) Saída do Giroscópio e do Inclinómetro.
	OP_APPLY_CONTROL (*0b*11) Mensagem de actuação sobre as três juntas do SCU alvo com a definição dos parâmetros do controlador PID.	*PARAM_*K_I (*0b*00) Parâmetro de controlo integral.
		*PARAM_*K_P (*0b*01) Parâmetro de controlo proporcional.
		PARAM_KD (*0b*10) Parâmetro de controlo derivativo.
		PARAM_CONTROLON (*0b*11) Switch on/off do controlador.

De notar, que a actuação é feita directamente às três juntas numa única mensagem, mas com a desvantagem de apenas se poder actuar num parâmetro cada vez (posição final, velocidade média ou um parâmetro do controlador).

Respostas Master→PC:

Na resposta, o Master responde com uma mensagem de 6 bytes, cuja estrutura é a seguinte:

- *Message Type*: possui o valor *MESSAGE_SUCESS* (*0x*FB);
- *OpCode*: *opcode* utilizado pela mensagem original PC→Master;
- Data 1-4: dados sensoriais no caso de um pedido de consulta sensorial.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FB	OpCode	Data 1	Data 2	Data 3	Data 4

Formato geral de uma mensagem de resposta Master→PC.

Esta estrutura é geral e pode assumir várias formas de acordo com a operação envolvida:

- ➔ No caso de uma operação de actuação, os bytes 2-5 possuem o mesmo valor que a mensagem original com o último byte nulo;
- ➔ Numa leitura sensorial, o byte *OpCode* é igual à da mensagem original com os bytes sucessivos contendo a informação sensorial pedida. Se os dados sensoriais concernem aos servo motores, três bytes são utilizados para conter a informação relativa a cada um deles, e o último é utilizado para transmitir informação de *status* do SCU em causa. Se concernem aos sensores adicionais, são utilizados todos os quatro bytes ou para conter a informação de um dos conjuntos dos sensores de força (de um pé) ou para o conjunto giroscópio+inclinómetro.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FB	==	==	==	==	0

Mensagem de actuação aplicada com sucesso

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FB	==	Joint 1	Joint 2	Joint 3	SCU status

Mensagem de leitura sensorial das juntas

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FB	==	Force sensor 1	Force sensor 2	Force sensor 3	Force sensor 4

Mensagem de leitura dos sensores de força

byte 1	*Byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FB	==	Inclinometer 1	Inclinometer 2	Giroscope 1	Giroscope 2

Mensagem de leitura de inclinómetro+giroscópio

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FB	==	Error Code	# TX errors	# RX errors	0

Mensagem de leitura do estado do barramento CAN

De notar, que na leitura sensorial das juntas, na mesma mensagem é transportado os valores dos três servos. No entanto, apenas um parâmetro pode ser lido – ou posição, ou velocidade média ou corrente.

No entanto, podem ocorrer situações anómalas na recepção do comando por parte do Master, podendo-se discernir duas situações possíveis:

- ➔ Mensagem de pedido inválido: os parâmetros solicitados pelo PC não fazem sentido (ex.: SCU alvo não existente). Neste caso uma mensagem de Message Type de código *MESSAGE_INVREQ* (*0x*FC), com todos os restantes bytes iguais à da mensagem original, é retornada ao PC.
- ➔ Mensagem de erro: houve um erro na recepção da mensagem proveniente do PC. Uma mensagem com todos os 6 bytes de código *MESSAGE_ERROR* (*0x*FE) é retornada ao PC.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FC	==	==	==	==	0

Mensagem sinalizadora de um pedido inválido.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FE	*0x*FE	*0x*FE	*0x*FE	*0x*FE	*0x*FE

Mensagem sinalizadora de um erro na comunicação PC→Master.

1.1.2. Unidade Principal

1.1.2.1. Configuração da Unidade Principal

Como configurações gerais optou-se por efectuar as transacções à velocidade máxima permitida sem a aplicação de controlo de comunicações. Elas são:

- ➔ *Baudrate*: 115200 bps
- ➔ Tamanho da palavra de dados: 8 bits
- ➔ Número de *stop bits*: 1
- ➔ Bit de paridade: desactivado
- ➔ Controlo de comunicações por *handshaking*: desactivado

Utilizou-se um computador Pentium com porta série RS-232 embutida, com o sistema operativo Microsoft® Windows XP, como unidade principal, para comunicar com a unidade master. O software utilizado foi o MatLab 7.0 através da mini-toolbox *cport* v1.3 que permite trocar por RS-232 tanto caracteres como *strings* e valores numéricos inteiros.

Para inicializar as comunicações através do *cport* é necessário seguir as seguintes instruções:

1. Abrir as comunicações especificando a porta série a usar (COM?) guardando o handler da linha aberta.

   Porta: COM1

   ```
   handler=cportopen('com1')
   ```

   Se handler=0, a ligação falhou!

2. Configurar a linha especificando as seguintes características:

Campo	***Valor***
Baudrate	115200 bps
Tamanho do byte	8 bits
Descarte dos bytes nulos	desactivado
Controlo de fluxo pela linha DTR	desactivado
Controlo de fluxo pela linha RTS	desactivado
Término das operações de leitura/escrita na ocorrência de um erro	activado
Caracter a usar no caso de um erro de paridade	nenhum
Tempo de timeout entre dois caracteres consecutivos	1 ms

Tabela 4: Configurações gerais do *cport*.

```
state=cportconfig(handler,'BaudRate',115200,                    ...
                          'ByteSize',8,                         ...
                          'fNull','OFF',                        ...
                          'fDtrControl','OFF',                  ...
                          'fRtsControl','OFF',                  ...
                          'fAbortOnError','ON',                 ...
                          'ErrorChar',-1,                       ...
                          'ReadIntervalTimeout',1,              ...
                          'ReadTotalTimeoutMultiplier',5,       ...
                          'ReadTotalTimeoutConstant',100,       ...
                          'WriteTotalTimeoutMultiplier',1000,   ...
                          'WriteTotalTimeoutConstant',1000);
```

Para efeitos de *debugging* é útil usar um terminal RS-232. Recomenda-se a utilização do terminal R.E.Smith por permitir a troca de bytes de qualquer código (caracter ou não) e a visualização das saídas/entradas em formato hexadecimal.
Em caso de uso de um terminal as seguintes opções são suficientes:

Campo	***Valor***
Porta	COM1
Baudrate	115200
Bits de dados	8
Paridade	Nenhum
Bits de paragem	1
Controlo de fluxo	Nenhum

Tabela 5: Configurações de um terminal RS-232 (No caso do R.E.Smith, usar COM1, 115200, N-8-1).

No caso de haver algum problema durante o funcionamento torna-se necessário reinicializar as comunicações. Para isso basta fazer...

```
stat=cportreset(handler)
```

Para terminar as comunicações é só usar o comando *cportclose*:

```
stat=cportclose(handler)
```

Caso haja algum problema na configuração do *cport*, pode obter as definições correctas através do seguinte procedimento:

1. Fechar todos os programas e reiniciar o computador;
2. Ligar as comunicações através do terminal R.E.Smith seguindo as opções da Tabela 5;
3. Verificar a conectividade e de seguida desligar as comunicações;
4. Abrir o MatLab e ligar a porta série através dos comandos do *cport* definindo apenas o *baudrate*;
5. Verificar a conectividade, e anotar as configurações tomadas fazendo *stat=cportconfig*(H).
6. Redefinir as configurações do *cport* com estes dados.

Sempre que um programa tenha utilizado a porta série antes, as configurações a adoptar pelo *cport* serão as mesmas que desse programa. A partir daqui é só configurar a porta série com estes dados sempre que uma ligação é estabelecida.

1.1.2.2. Device Drivers da Unidade Principal

Para automatizar o processo de leitura/escrita dos dados sensoriais/actuadores, *device drivers* foram escritos na forma de funções em MatLab. Elas são:

Ficheiro	*Função*	*Descrição*
initcom.m	`[H,state]=initcom(gate,rate)`	Criação de uma nova ligação.
killcom.m	`stat=killcom(H)`	Término da ligação.
calibcom.m	`calibcom(H)`	Pedido de envio de uma sequência de teste.
readcanstat.m	`[array,...]=readcanstat(H)`	Consulta do estado do barramento CAN.
readjoint.m	`[servos,...]=readjoint(H,scu_id,param)`	Leitura das posições das juntas de um SCU.
readreaction.m	`[reaction,...]=readreaction(H,foot)`	Leitura da saída dos sensores de força de um pé.
readorient.m	`[array,...]=readorient(H)`	Leitura dos sensores de orientação (inclinómetro e giroscópio).
applyjoint.m	`[...]=applyjoint(H,scu_id,param,servos)`	Actuação nas juntas de um determinado SCU.
applycontrol.m	`[...]=applycontrol(H,scu_id,param,servos)`	Actualização dos parâmetros PID de uma determinado SCU.

Tabela 6: Lista de device drivers da unidade principal.

initcom

```
% Estabelecimento de uma nova ligação via RS-232
%
% [handler,state]=initcom(gate,rate)
%
% Entradas:
%  gate    -> Porta a utilizar (1,2,...)
%  rate    -> baudrate a definir
% Saídas:
%  handler -> ID da linha de comunicações
%  state   -> Configurações da linha

function [handler,state]=initcom(gate,rate)
```

killcom

```
% Término de uma ligação RS-232 existente.
%
% stat=killcom(handler)
%
% Entradas:
%  handler -> ID da linha série
% Saídas:
%  stat    -> retorna 1 em caso de sucesso

function stat=killcom(handler)
```

calibcom

```
% Pedido de envio de uma sequencia de teste por parte do master.
%
% calibcom(handler)
%
% Entradas:
%  handler -> ID da linha série.

function calibcom(handler)
```

readcanstat

```
% Leitura do estado do barramento CAN entre slaves.
%
% [array,state,rx,error,errorstr,tries]=readcanstat(H)
%
% Entradas:
%  H => Handler para comunicar com o Master
%
% Saídas:
%  array    => [estado de erro, #erros de transmissão, #erros de recepção]
%  state    => Bits de estado dos servos
%  rx       => Mensagem de baixo nível recebida
%  error    => Código de erro, se existente
%  errorstr => String descritiva do erro
%  tries    => Número de tentativas para efectuar a comunicação

function [array,state,rx,error,errorstr,tries]=readcanstat(H)
```

readjoint

```
% Leitura de um parametro sensorial dos servos de um SCU
%
% [servos,state,rx,error,errorstr,tries]=readjoint(H,scu_id,param)
%
% Entradas:
% H        => Handler para comunicar com o Master
% scu_id   => Identificador do SCU alvo
% param    => Parametro a ler (0:posição, 1:velocidade, 2:corrente)
%
% Saídas:
% servos   => Parametro de saída [servo1,servo2,servo3]
% state    => Bits de estado dos servos
% rx       => Mensagem de baixo nível recebida
% error    => Código de erro, se existente
% errorstr => String descritiva do erro
% tries    => Número de tentativas para efectuar a comunicação

function [servos,state,rx,error,errorstr,tries]=readjoint(H,scu_id,param)
```

readreaction

```
% Leitura dos sensores de Força de um pé.
%
% [reaction,rx,error,errorstr,tries]=readreaction(H,foot)
%
% Entradas:
% H    => Handler das comunicações com o Master
% foot => Pé alvo (0:direito, 1:esquerdo)
%
% Saídas:
% reaction => Valores dos sensores de força
% rx       => Mensagem de baixo nível recebida
% error    => Código de erro, se existente
% errorstr => String descritiva do erro
% tries    => Número de tentativas para efectuar a comunicação

function [reaction,rx,error,errorstr,tries]=readreaction(H,foot)
```

readorient

```
% Leitura dos sensores de orientação (inclinómetro e giroscópio)
%
% [array,rx,error,errorstr,tries]=readorient(H)
%
% Entradas:
% H    => Handler das comunicações com o Master
%
% Saídas:
% array    => [inclinómetro 1, inclinómetro 2, giroscópio 1, giroscópio 2]
% rx       => Mensagem de baixo nível recebida
% error    => Código de erro, se existente
% errorstr => String descritiva do erro
% tries    => Número de tentativas para efectuar a comunicação
%
function [array,rx,error,errorstr,tries]=readorient(H)
```

applyjoint

```
% Aplicação de uma ordem de posição ou velocidade a cada motor de uma
% junta.
%
% [rx,error,errorstr,tries]=applyjoint(H,scu_id,param,servos)
%
% Entradas:
% H        => Handler para comunicar com o Master
% scu_id   => Identificador do SCU alvo
% param    => Parametro a aplicar (0:posição, 1:velocidade)
% servos   => Dados a aplicar [servo1,servo2,servo3]
%
% Saídas:
% rx       => Mensagem de baixo nível recebida
% error    => Código de erro, se existente
% errorstr => String descritiva do erro
% tries    => Número de tentativas para efectuar a comunicação

function [rx,error,errorstr,tries]=applyjoint(H,scu_id,param,servos)
```

applycontrol

```
% Ajuste dos parametros do controlador PID para o posicionamento do servo
%
% [rx,error,errorstr,tries]=applycontrol(H,scu_id,param,servos)
%
% Entradas:
% H        => Handler para comunicar com o Master
% scu_id   => Identificador do SCU alvo
% param    => Parametro a modificar (0:K_I, 1:K_P, 2:K_D, 3:PID on)
% servos   => Dados a aplicar [servo1,servo2,servo3]
%
% Saídas:
% rx       => Mensagem de baixo nível recebida
% error    => Código de erro, se existente
% errorstr => String descritiva do erro
% tries    => Número de tentativas para efectuar a comunicação

function [rx,error,errorstr,tries]=applycontrol(H,scu_id,param,servos)
```

Todos os *device drivers*, tanto os de consulta sensorial como os de actuação, seguem um algoritmo semelhante ao enunciado na Fig. 9.

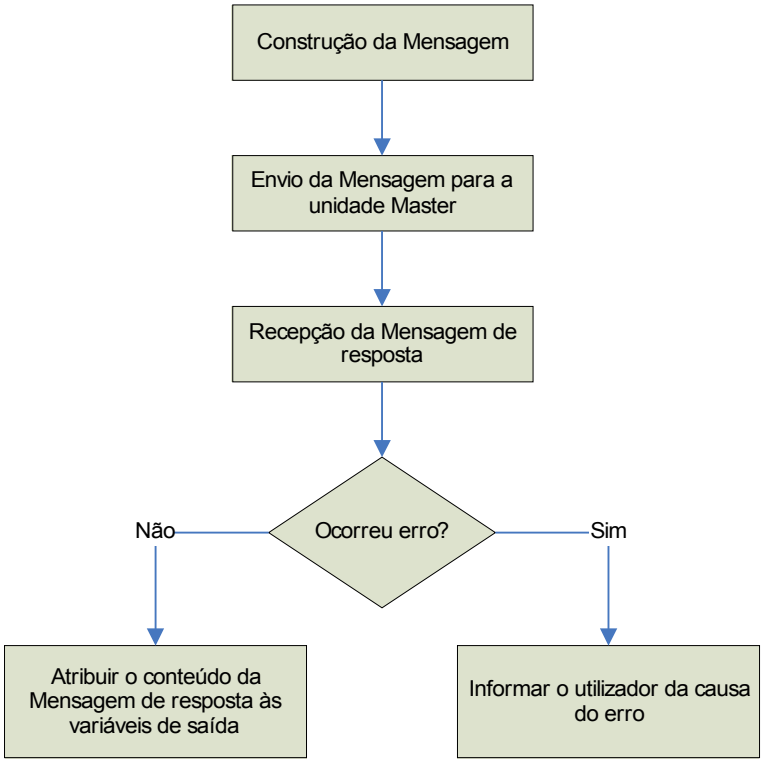


Fig. 9: Algoritmo geral dos *device drivers* da unidade principal.

A construção da mensagem baseia-se na definição de um array de bytes, cuja estrutura segue o formato de um comando PC→Master (secção 1.1.1).
O envio da mensagem, constituído pelos vários bytes, é feita segundo o algoritmo descrito na Fig. 10. É enviado um byte de cada vez para a porta série, e caso ocorra algum erro são reinicializadas as comunicações voltando de seguida a tentar reenviar o mesmo byte. O processo só termina quando todos os bytes forem enviados com sucesso.
A recepção da mensagem de resposta está descrita na Fig. 11 e é executada logo após o envio do comando. A mini-toolbox *cport* oferece-nos já uma função que permite ler imediatamente um array de 6 bytes da porta série indicando se a recepção teve sucesso em cada um deles. Caso tal não aconteça, as comunicações são reinicializadas e o reenvio do comando é repetido com a consequente leitura da resposta.

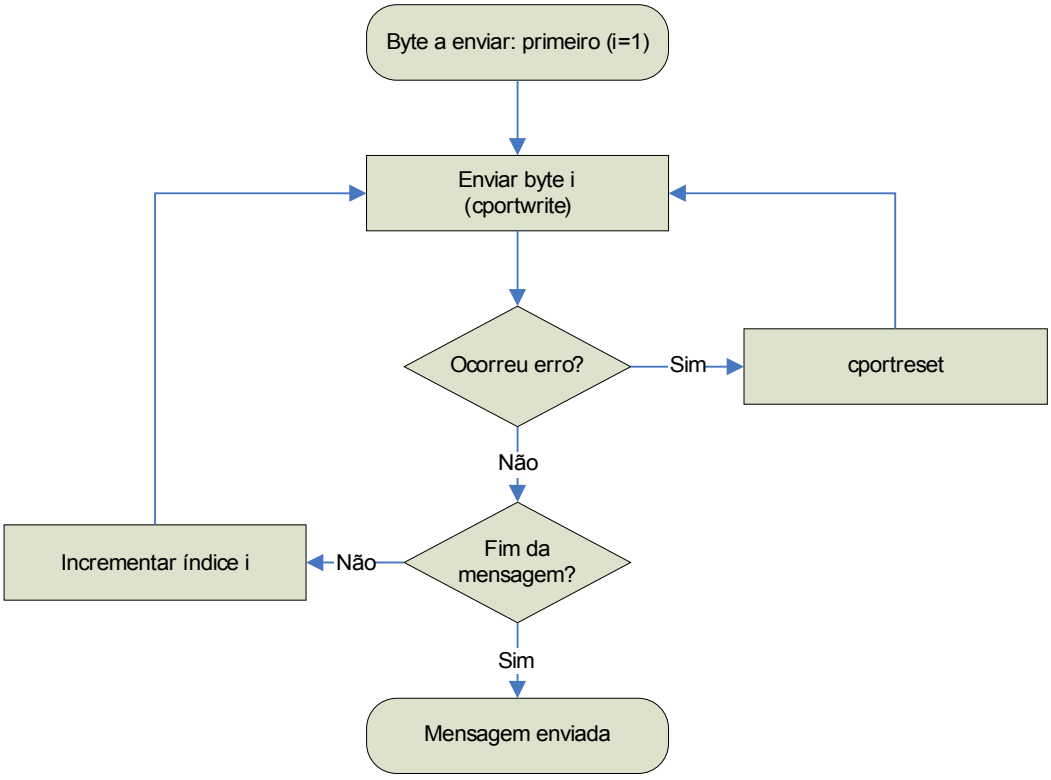


Fig. 10: Algoritmo de envio de uma mensagem para o Master.

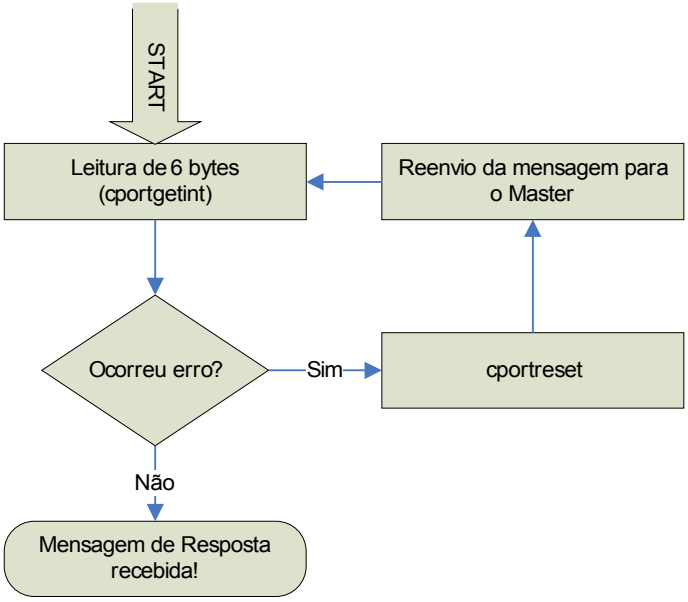


Fig. 11: Algoritmo para recepção da mensagem de resposta do Master.

1.1.3 – Unidade Master

1.1.3.1. Configuração do Master

Para configurar a unidade microcontroladora a operar com a linha RS-232 é necessário efectuar um conjunto de passos que serão descritos a seguir. Primeiramente é necessário configurar os pinos RX (recepção) e TX (transmissão) como entrada e saída respectivamente:

```
TRISC=(TRISC | 0x80) & 0xBF;
```

De seguida é necessário configurar o *baudrate* da comunicação através do registo SPBRG. Para a frequência de CPU F_{CPU}=10MHz e para um baudrate desejado de 115200, segundo a fórmula...

$$SPBRG = round\left(\frac{F_{CPU}}{4 * Baud}\right) - 1$$

... o registo SPBRG deve assumir o valor de 21.

Configurar o modo de funcionamento de transmissão através do registo TXSTA e de recepção pelo registo RCSTA.

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

Registo TXSTA

Campo	*Parâmetro*	*Valor*	*Descrição*
CSRC	Clock Source Select bit	X	——
TX9	9-bit Transmit Enable bit	0	Selects 8-bit transmission
TXEN	Transmit Enable bit	1	Transmit enabled
SYNC	USART Mode Select bit	0	Asynchronous mode
BRGH	High Baud Rate Select bit	1	High speed
TRMT	Transmit Shift Register Status bit	—	(Read only)
TX9D	9th bit of Transmit Data	X	——

Tabela 7: Configuração do registo TXSTA.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Registo RCSTA

Campo	*Parâmetro*	*Valor*	*Descrição*
SPEN	Serial Port Enable bit	1	Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
RX9	9-bit Receive Enable bit	0	Selects 8-bit reception
SREN	Single Receive Enable bit	X	——
CREN	Continuous Receive Enable bit	1	Enables continuous receive
ADDEN	Address Detect Enable bit	X	——
FERR	Framing Error bit	—	(Read only)
OERR	Overrun Error bit	—	(Read only)
RX9D	9th bit of Received Data	—	(Read only)

Tabela 8: Configuração do registo RCSTA.

Finalmente é necessário activar/desactivar as interrupções respectivas e defini-las como alta prioridade (não esquecer de activar a funcionalidade de dupla prioridade):

```
IPR1bits.RCIP=1;                // Interrupções de alta prioridade
IPR1bits.TXIP=1;
PIE1bits.RCIE=1;                // (Des)Activação das interrupções
PIE1bits.TXIE=0;
```

1.1.3.2. Funcionamento da USART no Master

As Fig. 12 e Fig. 13 apresentam os algoritmos para troca de informação entre o master e a unidade principal.

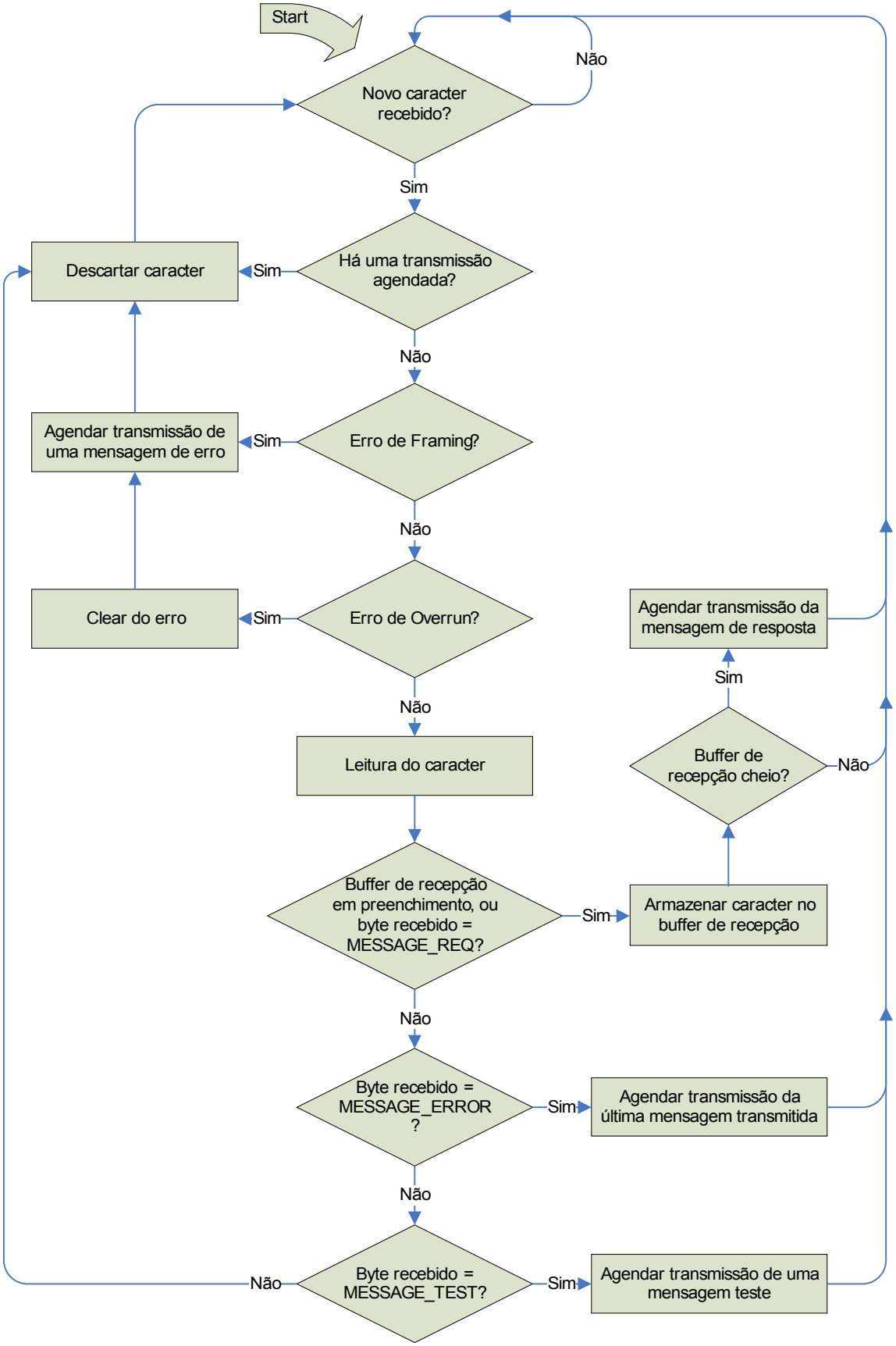


Fig. 12: Algoritmo de recepção de informação, via USART, pelo Master.

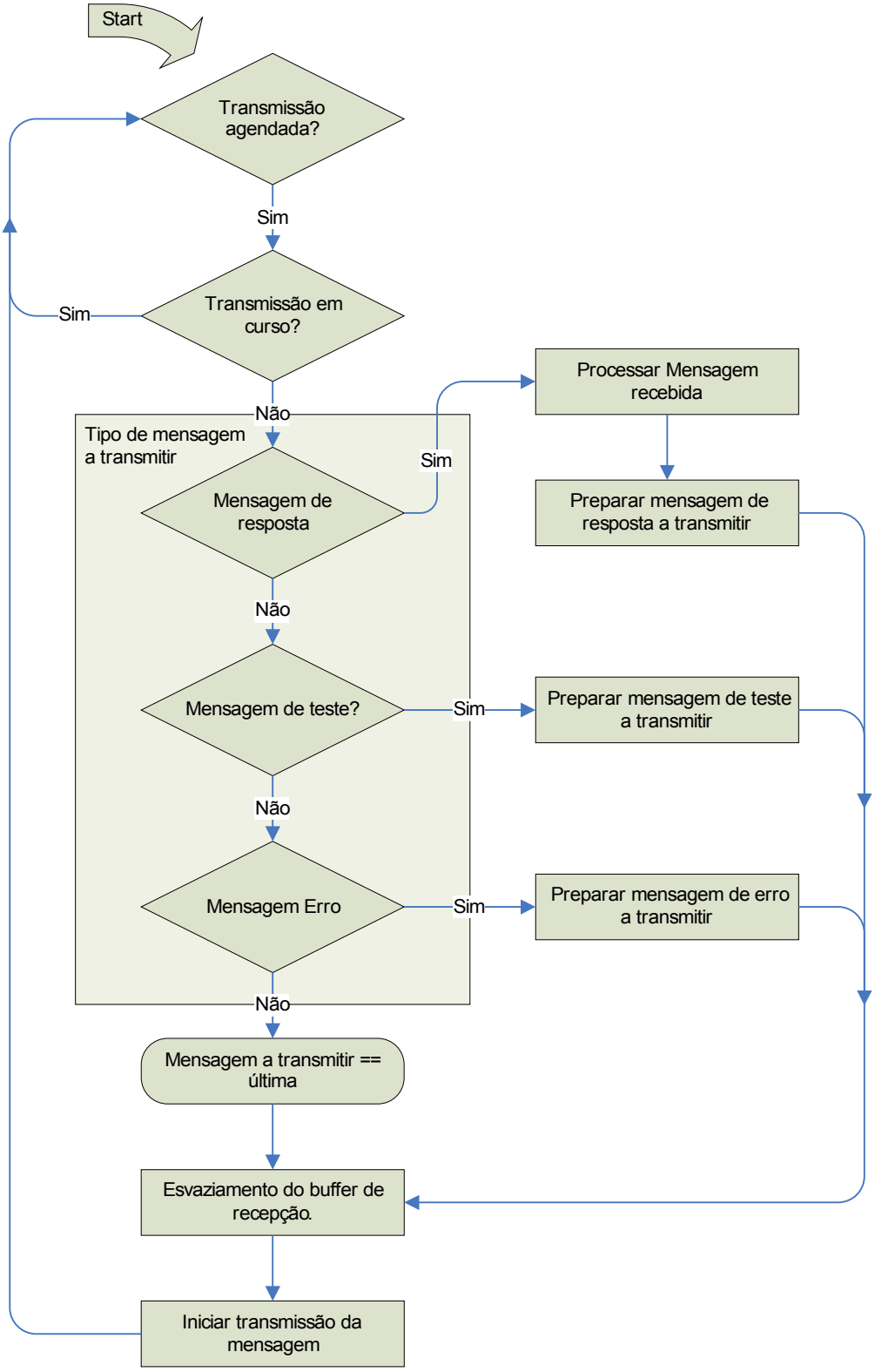


Fig. 13: Algoritmo de transmissão de informação, via USART, pelo Master.

Para evitar congestionamento de largura de banda de CPU, optou-se por uma solução baseada no agendamento de transmissão, em vez da transmissão imediata das mensagens de resposta. Tal permite a transmissão dos vários bytes sempre que for possível, possibilitando, desta forma, a execução de outras tarefas em paralelo sem terem que esperar que a transmissão se finalize. Ao mesmo tempo também é possível a recepção e a transmissão de informação simultaneamente.

Os algoritmos evidenciados na Fig. 12 e Fig. 13 relativamente à recepção e transmissão respectivamente, são executadas em paralelo entre si e, simultaneamente, com outras tarefas, como será o caso da comunicação CAN com as unidades slave. Para a implementação do protocolo, dois *buffers* implementados por software são utilizados para armazenamento dos bytes recebidos (*buffer* de recepção) e dos bytes a transmitir à

unidade principal (*buffer* de transmissão). Note que a recepção/transmissão é efectuada por interrupção, pelo que a recepção/transmissão de um *stream* de bytes implica várias entradas/saídas da rotina de serviço à interrupção. Tal implica que os dados trocados têm que ser armazenados em estruturas estáticas a que denominados por *buffers*.

1.1.3.3. Base de Dados Global do Sistema

O processamento da mensagem recebida a partir da unidade principal é baseada na análise do byte *opcode* segundo a descrição na Tabela 3. Uma vez decifrado o conteúdo da mensagem, uma base de dados global com a informação sensorial e de actuação de cada uma das unidades slave, é acedido para consulta ou actualização, dependendo se se trata de uma operação de consulta sensorial ou de actuação respectivamente. A base de dados é dividida em duas estruturas, uma para a informação sensorial e outra para a de actuação.

Estrutura sensorial

```
// Estrutura descritiva dos sensores
typedef struct {
    byte sysStatus[N_SCU];                                  // Estado do sistema
    struct_servo servo[N_SCU][N_SERVOS];                    // Sensores dos servos
    unsigned char reaction[N_FEET][N_FORCE_SENSORS];        // Sensores de força dos pés
    signed char giroscope[2];                               // Giroscópio
    signed char inclinometer[2];                            // Inclinómetro
} struct_sensors;
extern struct_sensors   sensors;                            // Sensores
```

Estrutura de actuação

```
typedef struct {
    struct {
        byte ki, kp, kd;
        bool controlOn;
    } control[N_SCU][N_SERVOS];
    struct_servo servo[N_SCU][N_SERVOS];
} struct_actuators;
extern struct_actuators actuators;                          // Actuadores
```

Estrutura descritiva de um Servomotor

```
// Estrutura descritiva de um servo
typedef struct {
    signed char position;       // Posição
    signed char velocity;       // Velocidade
    unsigned char current;      // Corrente consumida
} struct_servo;
```

Esta base de dados constitui uma imagem de todas as unidades slave, pelo que a consulta sensorial pode ser efectuada de forma imediata sem ser necessária qualquer comunicação adicional com o slave em questão. Já no que toca à actuação apenas é necessário actualizar a variável de interesse, e um mecanismo à parte encarrega-se de transmitir ao destino a nova informação. Este mesmo mecanismo também é responsável por manter a imagem sensorial de todas as unidades slave sincronizada e com o mínimo de atraso temporal.
Este mecanismo trata-se do segundo protocolo de comunicação implementado, o CAN, que é responsável pela troca de informação entre o Master e todos os Slaves. Este mecanismo trabalha de forma independente das comunicações USART e apenas tem como tarefa a actualização da base de dados sensorial e a transmissão aos slaves dos valores contidos na base de dados de actuação. Todo este processo ocorre a uma taxa de 1 KHz (1 ms) completando a actualização de todo o conjunto em apenas 8 ms (para 8 slaves).
As estruturas evidenciadas correspondem à primeira versão do software, em que apenas se considera um giroscópio e um inclinómetro (de dois valores cada um) para um único SCU. Recomenda-se, nas versões futuras, a actualização desta estrutura de modo a considerar 8 conjuntos destes valores para permitir a existência de vários sensores deste género.

1.2 – Comunicação CAN

A comunicação CAN é implementada entre o Master e as diversas unidades Slave e tem como finalidade o redireccionamento dos dados de actuação, provenientes da unidade principal, para cada unidade Slave, e, no sentido oposto, o envio dos dados sensoriais para a unidade Master, de modo a que esta actualize a base de dados com a informação sensorial de todas as unidades por forma a permitir à unidade principal a sua consulta com o mínimo de desfasamento temporal.

O CAN é um sistema de comunicações série multi-ponto que foi desenvolvido originalmente para a indústria automóvel para possibilitar as comunicações entre diversos componentes em ambientes extremamente ruidosos. Para tal, o sinal que serve de suporte à comunicação é definido em corrente e não em tensão. É um protocolo baseado na mensagem e não no endereço, o que significa que as mensagens são transmitidas em broadcasting para todos os nós existentes na rede, cabendo a cada um a decisão de a aceitar ao não. As mensagens são constituídas por um identificador que pode possuir 11 ou 29 bits, de acordo com a versão A ou B respectivamente, e até 8 bytes de dados (Fig. 14).

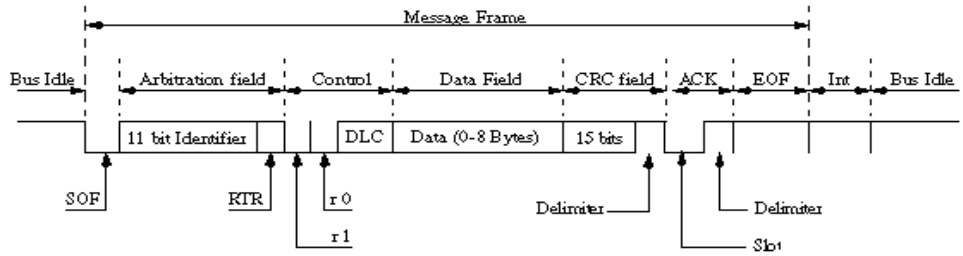


Fig. 14: Formato *standard* das mensagens CAN.

De modo a eliminar a hipótese de colisão destrutiva de pacotes (no caso de envio simultâneo de mensagens por diferentes nós) como acontece nas redes Ethernet, o CAN é dotado de um sistema de colisão determinístico bit a bit (CSMA/BA – Carrier Sense Multiple Access / Bit-wise Arbitration) que no caso de uma colisão de pacotes apenas um prevalece (o de maior prioridade), enquanto que os restantes são destruídos. No último caso, os nós associados a esses pacotes devem tentar retransmitir mais tarde. Desta forma garante-se que a largura de banda oferecida pela rede não é desperdiçada, havendo sempre o transporte de mensagens desde que pelo menos um nó transmita.

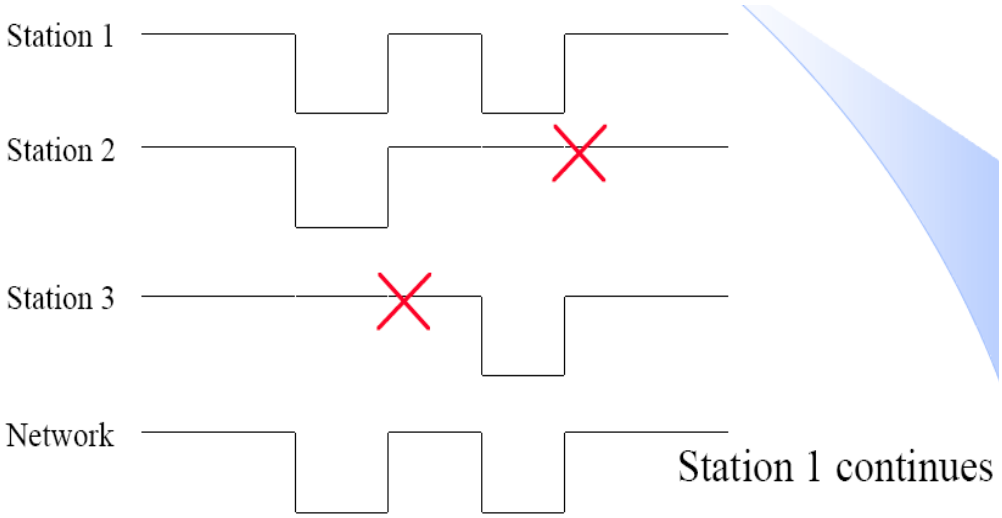


Fig. 15: Exemplo de aplicação utilizando o CSMA/BA.

Para implementação deste sistema um dos bits é definido como dominante (bit 0) e o outro como recessivo (bit 1), de modo que quando dois bits colidem um com o outro "vence" o dominante. Esta estratégia é implementada bit a bit ao nível do identificador conferindo, deste modo, prioridades de acordo com o seu

valor. Deste modo, quanto menor for o valor do identificador, maior é a prioridade da mensagem (Fig. 15). Segundo esta política é vantajoso atribuir um endereço de menor valor numérico à unidade master de modo a conferir-lhe a máxima prioridade e às restantes unidades é-lhes atribuído endereços de valor crescente à medida que a sua ordem de importância na rede diminui. Desta forma, atribuiu-se o endereço 0b0000 ao master e os endereços 0b0001 até 0b1000 para os oito slaves, dando maior prioridade às unidades respeitantes aos membros inferiores dado que requerem maior atenção ao nível do controlo.

Unidade Controladora	***Secção a que respeita***	***Endereço na Rede***
Master	Unidade mestre	0b0000
Slave 1	Perna direita	0b0001
Slave 2	Perna esquerda	0b0010
Slave 3	Anca direita	0b0011
Slave 4	Anca esquerda	0b0100
Slave 5	Tronco	0b0101
Slave 6	Braço direito	0b0110
Slave 7	Braço esquerdo	0b0111
Slave 8	Cabeça	0b1000

Tabela 9: Endereços atribuídos às diversas unidades de controlo.

Outro benefício da comunicação baseada na mensagem é o facto de se poderem adicionar outros nós à rede sem haver necessidade de reprogramar todos os nós existentes. O novo nó adicionado com base na mensagem recebida decide se deve ou não processar a sua informação.

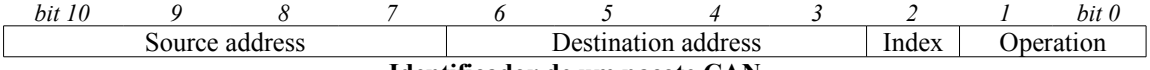
A escolha do sistema de comunicações recaiu sobre o CAN principalmente devido ao facto de este ser um protocolo baseado na mensagem. Desta forma, futuramente, a construção da rede pode ser modificada para que os controladores locais possam "ouvir" a informação uns dos outros tomando decisões (*i.e.*, dotado de poder local).

No microcontrolador em causa, é utilizada a versão *full*CAN 2.0A cujas características principais são:

- Mensagens com um identificador de 11 bits e um máximo de 8 bytes de dados;
- Filtragem completa do conteúdo do identificador (todos os bits do identificador podem ser utilizados para configuração da filtragem);
- Associação do identificador a buffers de dados;
- Múltiplos *buffers* de transmissão/recepção: 3 *buffers* de transmissão e 2 *buffers* de recepção;
- Processamento e recuperação automática de erros.

1.2.1. Protocolo CAN

De acordo com os dados a trocar entre o Master e cada Slave, e utilizando todos os 8 bytes de dados em cada mensagem, é possível efectuar trocas de informação, quer de actuação, quer sensoriais, recorrendo a apenas dois pacotes para cada instanciação (16 bytes). Para efeitos de rigor na linguagem o termo pacote será utilizado para referir a informação trocada numa transacção atómica CAN (Fig. 14) e mensagem será utilizada para descrever a troca completa de informação (dois pacotes). O protocolo é enunciado a seguir.



Identificador de um pacote CAN

Campo	*Descrição*
Source Address	Endereço do nó emissor da mensagem
Destination Address	Endereço do nó destinatário da mensagem
Index	Índice do pacote dentro de uma mensagem: ➔ 0: primeiro pacote; ➔ 1: segundo pacote.
Operation	Operação associada à mensagem: ➔ 0b00: actualização sensorial; ➔ 0b01: actualização de actuadores.

Tabela 10: Campos do identificador de um pacote CAN.

De notar que apenas o campo *destination address* é útil para a recepção de cada pacote, pois é ele quem define o destino do pacote. Já o campo *source address* está presente nos bits mais significativos com o propósito de definir a prioridade do pacote de acordo com endereço do remetente. Desta forma, mensagens enviadas pelo master possuem a máxima prioridade, e dentro dos slaves, os dos membros inferiores possuem maior prioridade de envio. Como cada mensagem é constituída por dois pacotes, o campo *index* identifica a ordem do pacote, e o campo operation refere a função dos dados transportados.

Transacção Master→Slave

As mensagens trocadas do master para slaves, têm como função a actualização dos actuadores das juntas a que dizem respeito, pelo que transportará as seguintes informações para cada um dos três servomotores associados:

- Posição a atingir;
- Velocidade média do movimento;
- Flags de activação da compensação PID;
- Parâmetros de compensação PID (Proporcional, Integrador e Derivativo);

Source address				Destination address				Index	Operation	
0	0	0	0	X	X	X	X	X	0	1

Identificador de uma mensagem de actualização de actuadores

O identificador destas mensagens possuem o endereço *0b*0000 no *source address* indicativo de que o pacote é proveniente da unidade master, e o campo *operation* contém o código *0b*01 descrevendo que o pacote tem como intenção fazer uma actualização dos actuadores.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*	*byte 8*
Position 1	Position 2	Position 3	Velocity 1	Velocity 2	Velocity 3	Control Flags	K_D 3

Dados do primeiro pacote de actualização de actuadores (*Index*=0)

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*	*byte 8*
K_I 1	K_I 2	K_I 3	K_P 1	K_P 2	K_P 3	K_D 1	K_D 2

Dados do segundo pacote de actualização de actuadores (*Index*=1)

Cada mensagem é constituída por dois pacotes em que tanto a posição final, como a velocidade média e os parâmetros de compensação PID de cada servomotor são actualizadas de uma só vez. O byte *Control Flags* tem como função indicar se o controlador PID deve estar activo ou não em cada um dos servo motores. Os restantes 5 bits poderão eventualmente ser utilizados futuramente para mais opções.

bit 7	*6*	*5*	*4*	*3*	*2*	*1*	*bit 0*
					PID Controller on/off		

Byte *Control Flags* do primeiro pacote

Transacção Slave→Master

Por outro lado é necessário actualizar a base de dados do master pelo envio de mensagens CAN com os dados sensoriais de cada slave para o master. Tais informações são as seguintes:

- Posição;
- Velocidade média;
- Corrente consumida;
- Output dos sensores adicionais;

Source address				Destination address				Index	Operation	
X	X	X	X	0	0	0	0	X	0	0

Identificador de uma mensagem de actualização sensorial

Os pacotes de actualização sensorial devem possuir no campo *destination address* o endereço da unidade master, dado que a mensagem se destina a ele, e o campo operation a *0b*00 indicativo de que a mensagem contém dados sensoriais.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*	*byte 8*
Sensor Flags	Position 1	Position 2	Position 3	Velocity 1	Velocity 2	Velocity 3	Slave Status

Dados do primeiro pacote de actualização sensorial (*Index*=0)

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*	*byte 8*
Sensor Flags	Current 1	Current 2	Current 3	Extra 1	Extra 2	Extra 3	Extra 4

Dados do segundo pacote de actualização sensorial (*Index*=1)

Os valores de saída dos sensores adicionais estão descritos nos campos *Extra* apenas podendo fazer parte apenas de uma das seguintes classes:

- Sensores de força de um dos pés (4 sensores);
- Inclinómetro e Giroscópio (2 + 2 valores);

O pacote *sensor flags* é utilizado para distinguir de que conjunto de sensores fazem parte os valores introduzidos.

bit 7	*6*	*5*	*4*	*3*	*2*	*1*	*bit 0*
Source Address				Index	Special	What	Foot

Byte *Sensor Flags*

Campo	***Descrição***
Source Address	Endereço do slave remetente.
Index	Índice do pacote na mensagem de dados.
Special	Flag indicativa da presença de sensores adicionais. (No caso de ser 0, os campos *Extra* não têm significado)
What	No caso de *Special*=1, indica qual a classe de sensores em uso: ➔ 0: sensores de força; ➔ 1: inclinómetro + giroscópio.
Foot	No caso de Special=1 e What=0, indica de que pé faz parte os valores: ➔ 0: pé direito; ➔ 1: pé esquerdo.

Tabela 11: Campos do byte *Sensor Flags*.

Já o byte *slave status* contém flags indicadoras do estado de funcionamento da unidade slave.

bit 7	*6*	*5*	*4*	*3*	*2*	*1*	*bit 0*
Deadline error	Position reset	PID Control on/off			Motion finished		

Byte *Slave Status* do primeiro pacote

Campo	***Descrição***
Deadline Error	Indicador da ocorrência de violação de deadline da execução da função *main* dentro do período de PWM (20ms).
Position Reset	Indica se ocorreu uma reinicialização da posição dos servos para os valores originais.
PID Control on/off	Indica se o controlador de posição PID está activo para cada servomotor.
Motion Finished	Indicador do término do movimento para cada servomotor.

Tabela 12: Campos do byte *Slave Status*.

1.2.2. Configuração do CAN

1.2.2.1. Inicializações

As definições de funcionamento do CAN devem ser iguais entre a unidade Master e as unidades Slave de modo a permitir o correcto sincronismo entre todas as entidades envolvidas na rede, pelo que as opções tomadas devem ser implementadas em todas as unidades.

Como condições base seguiram-se as seguintes linhas:

- ✔ Escolha da versão CAN mais universal: *full*CAN 2.0A;
- ✔ Configuração da velocidade na máxima possível (≤ 1 Mbps);
- ✔ Mascaragem das mensagens não destinadas ao próprio SCU – apenas as mensagens com o endereço destino igual ao do próprio são aceites;
- ✔ Dado o facto das mensagens serem constituídas por dois pacotes, aproveitou-se a presença de dois *buffers* de recepção para redireccionar cada uma para um *buffer* à parte – dupla filtragem;
- ✔ Igualmente, utilizou-se dois *buffers* de transmissão para transmitir cada um dos dois pacotes.

A seguir são descritos as principais operações na configuração deste recurso.

a) Modo de Configuração

O PIC 18F258 possui seis modos de operação, dos quais apenas 3 nos são úteis:

- ➢ Modo disable: o PIC não transmite nem recebe quaisquer mensagens;
- ➢ Modo de configuração: necessário antes de activar as transmissões/recepções;
- ➢ Modo de operação normal: recepção e transmissão de mensagens válidas, via CAN.

Antes de colocar o PIC a processar mensagens via CAN, é necessário, antes, colocar o PIC em modo de configuração e definir os seus parâmetros de configuração. Apenas depois deste passo coloca-se o PIC no modo de operação normal.

Para definir o modo de funcionamento, os bits REQOP do registo CANCON devem ser modificados para o modo de interesse, e esperar a mudança efectiva pelo polling aos bits OPMODE do registo CANSTAT.

b) Baudrate

Todos os nós na rede CAN devem ter o mesmo *bitrate* nominal – número de bits transmitidos por segundo – sem no entanto ser necessário que todos os nós tenham a mesma frequência de oscilação. No entanto é necessário o cuidado em programar o *baudrate prescaler* e o número de *time quanta* (explicado adiante) em cada segmento de bit, de modo a manter o *bitrate* nominal. O *bitrate* máximo depende da qualidade do transmissor e do oscilador e da presença de ressincronizações, mas poderá atingir o valor máximo de 1 Mbps numa situação ideal.

Dado que a transmissão utiliza codificação NRZ e que os osciladores e transmissores podem variar de nó para nó, é necessário introduzir *bit stuffing* para possibilitar a extracção de relógio para efeitos de sincronização. Tal é feito de modo a garantir uma alternância de bit pelo menos de 6 em 6 bits.

Uma unidade denominada por DPLL é utilizada para a sincronização dos dados recebidos e para garantir o *bitrate* nominal nos dados transmitidos. Funções de *bus timing* executadas dentro de cada *bit frame*, como a sincronização com o oscilador local, compensação do atraso introduzido pela rede e posicionamento de amostragem, implicam a particionamento de cada bit em vários segmentos definidos a partir de períodos de tempo mínimos chamados *Time Quanta* (T_Q).

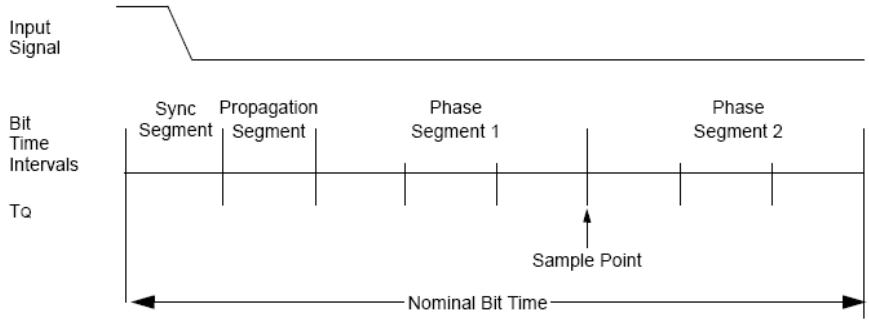


Fig. 16: Particionamento temporal de um bit.

Pela Fig. 16 podemos identificar 4 tipos de segmento:

1. Segmento de sincronização: utilizado para sincronizar dos vários nós da rede – duração fixa de 1 T_Q;
2. Segmento de propagação: utilizado para compensar os atrasos introduzidos na rede – duração entre 1 e 8 T_Q.
3. Segmento de fase 1: número de T_Q antes da amostragem do bit – duração entre 1 e 8 T_Q.
4. Segmento de fase 2: fornece um atraso antes da mensagem seguinte – duração entre 2 e 8 T_Q.

A duração dos segmentos de fase 1 e 2 não são parâmetros estáticos e podem sofrer variações para efeitos de ressincronização. O parâmetro SJW (*Synchronization Jump Width*) define a forma como se farão estas flutuações contendo o número máximo de T_Q que serão adicionados ao segmento de 1 ou subtraídos ao segmento 2.
Para definição das durações de cada segmento algumas regras deverão ser respeitadas. Elas são:

- ✔ 1 + Seg. de propagação + Seg. de fase 1 + Seg. de fase 2 ≥ 8
- ✔ Seg. de propagação + Seg. de fase 1 ≥ Seg. de fase 2
- ✔ Seg. de fase 2 ≥ SJW

Para o nosso caso, de forma a conseguir a máxima velocidade, definimos a mínima duração para cada bit, ou seja 8 T_Q. A distribuição de Time Quanta está indicada a seguir:

- Segmento de sincronização: 1 T_Q (não configurável);
- Segmento de propagação: 2 T_Q;
- Segmento de fase 1: 3 T_Q;
- Segmento de fase 2: 2 T_Q;
- SJW: 2 T_Q.

Com base nestes parâmetros, e de modo a conseguir a máxima velocidade de F_{bit}=1Mbps, o *baudrate* a definir no registo BRP seria de 1.5. Como estes registos são inteiros têm que se arredondar para o lado que diminua a velocidade, ou seja, para cima. Neste caso BRP deverá ser de 2!

$$BRP = \frac{F_{OSC}}{2 * F_{bit} * N_{seg}} - 1 \qquad \text{com} \qquad N_{seg} = Sync_{seg} + Propag_{seg} + Phase1_{seg} + Phase2_{seg}$$

$$F_{bit} = \frac{F_{OSC}}{2 * (BRP + 1) * N_{seg}}$$

Para BRP=2, o bitrate nominal efectivo será de 833 Kbps.

Máscaras e Filtros

Os filtros são utilizados para automatizar o processo de aceitação de mensagens. No caso das mensagens de actuação, apenas se destinam a um SCU particular não possuindo qualquer interesse para os restantes. Desta forma, na perspectiva de cada SCU apenas algumas das muitas mensagens que passam pelo seu porto de entrada devem ser processadas, pelo que a verificação por software, do destinatário de cada uma seria um tanto pesada em termos de consumo de largura de banda de CPU.
De modo a automatizar o processo de aceitação, uma máscara pode ser utilizada para indicar quais os bits do identificador que contém o padrão de interesse para a unidade em questão (como por exemplo o seu endereço) e filtros podem ser utilizados para fazer a selecção dos padrões e redireccionar para determinados *buffers*. Desta forma, apenas as mensagens cujos bits da secção do identificador programada coincide com o padrão especificado, são aceites e carregados para os buffers de recepção gerando uma interrupção para o posterior processamento.

Máscara do bit n	***Filtro do bit n***	***Valor do bit n***	***Resultado***
0	x	x	Rejeitado
1	0	0	Aceite
1	0	1	Rejeitado
1	1	0	Rejeitado
1	1	1	Aceite

Tabela 13: Resultado da filtragem para cada bit.

Pela Tabela 13 podemos verificar que apenas os bits definidos pela máscara serão considerados para filtragem. Todos os outros serão rejeitados à priori. Dos que são considerados para filtragem são comparados com um determinado padrão sendo apenas aceites os que coincidem com esse padrão. Apenas as mensagens em que todos os bits do filtro são aceites, são consideradas para processamento por parte do programador.

Para o PIC 18F258 é possível utilizar todos os bits do identificador na mascaragem e filtragem e definir até dois padrões de filtragem para armazenamento imediato no primeiro buffer de recepção e até 4 padrões para armazenamento no segundo buffer (Fig. 17).

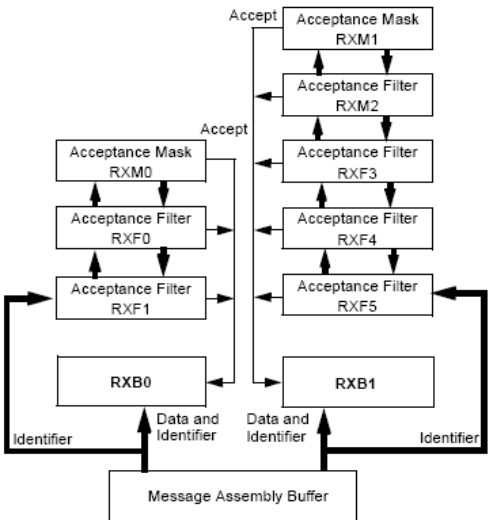


Fig. 17: Registos associados à mascaragem e filtragem.

Para o nosso caso em concreto, de acordo, com a estrutura do identificador enunciado anteriormente, a máscara a definir cobrirá os bits do endereço destinatário e do *index* do pacote, ou seja, possuirá o seguinte padrão:

bit 10	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*bit 0*
Source address				Destination address				Index	Operation	
0	0	0	0	1	1	1	1	1	0	0

Máscara a aplicar ao identificador

De seguida definir-se-ão dois filtros, de modo a aceitar unicamente os pacotes cujo endereço destino corresponda ao próprio SCU, e destes, redireccionar os de *index* 0 (primeiro pacote) para o *buffer* de recepção 0, e os de *index* 1 (segundo pacote) para o *buffer* de recepção 1 (Tabela 14).

Redirecção dos pacotes	***Registos a configurar***	***Padrão***
Buffer 0	RXF0	0 0 0 0 a_3 a_2 a_1 a_0 0 0 0
	RXF1	0 0 0 0 a_3 a_2 a_1 a_0 0 0 0
Buffer 1	RXF2	0 0 0 0 a_3 a_2 a_1 a_0 1 0 0
	RXF3	0 0 0 0 a_3 a_2 a_1 a_0 1 0 0
	RXF4	0 0 0 0 a_3 a_2 a_1 a_0 1 0 0
	RXF5	0 0 0 0 a_3 a_2 a_1 a_0 1 0 0

Tabela 14: Configuração dos filtros para redireccionamento de pacotes para os dois *buffers* de recepção (padrão a_3 a_2 a_1 a_0 = endereço do SCU).

Desta forma, redireccionando cada um dos dois tipos de pacotes para um *buffer* à parte, evita-se a perda de informação caso ocorra *overflow*, pois apenas o pacote cuja informação é equivalente (*index* igual) é substituído pelo novo pacote.

Transmissão e Recepção de Mensagens

Finalmente, também é necessário configurar a forma como as mensagens serão transmitidas e recebidas.

Para a recepção, é necessário definir a recepção de apenas pacotes válidos cujo identificador segue o formato da versão CAN 2.0A (11 bits), e deve-se assegurar que o *overflow* do *buffer* 0 para o *buffer* 1 está desactivado, dado que estamos a usar um *buffer* para cada tipo de pacote.
Finalmente devemos colocar o bit RXFULL de cada *buffer* a zero, para a sua abertura à recepção de novas mensagens, verificando também se o *interrupt flag* associado está desligado.

Para a transmissão, os identificadores das mensagens deverão estar de acordo com o formato *standard*, ou seja, deverão possuir 11 bits de comprimento, e é necessário definir prioridades para cada um dos três *buffers*, para definição da ordem de transmissão no caso de vários agendamentos simultâneos.
Por questões de similaridade de comportamento, também se reservará um *buffer* de transmissão para cada tipo de pacote, fazendo apenas uso de dois *buffers*. O *buffer* 0 utilizado para os pacotes de *index* 0 possuirá a máxima prioridade e o *buffer* 1 possuirá a prioridade mínima (Tabela 15).

Buffer	***Pacotes associados***	***Prioridade***	***Registos***
Buffer 0	Index 0	Máxima	TXB0CON=*0b*11
Buffer 1	Index 1	Mínima	TXB1CON=*0b*10
Buffer 2	——	——	TXB2CON=*0b*01

Tabela 15: Atribuição de prioridades entre cada buffer de transmissão.

Modo de Operação Normal

Para colocar o CAN a funcionar só resta definir o modo de operação normal no registo CANCON.

Resumindo...

1. Colocação do PIC no modo de configuração;
2. Definição da duração de cada segmento de bit;
3. Definição do *bitrate* para a velocidade pretendida;
4. Configuração da máscara de recepção;
5. Configuração dos dois filtros para cada um dos *buffers* de recepção;
6. Definição do formato *standard* para os identificadores dos pacotes a transmitir;
7. Definição de prioridades entre *buffers* de transmissão;
8. Configuração da recepção para apenas pacotes válidos com identificadores de 11 bits;
9. *Overflow* do *buffer* 0 para o *buffer* 1 desactivado;
10. *Clear* dos *interrupt flags* de recepção;
11. Abertura dos *buffers* para recepção;
12. Colocação do PIC no modo de operação normal.

(Os passos 6 a 11 devem ser aplicados a cada um dos buffers de recepção/transmissão.)

1.2.2.1. Troca de Pacotes via CAN

Esta secção apresenta os procedimentos a efectuar nos PICs 18F258 para a troca de pacotes utilizando a rede CAN. Estes microcontroladores oferecem uma gama diversificada de registos que permitem controlar diversos aspectos, como por exemplo o comprimento do identificador ou do conjunto de dados e no domínio da recuperação de erros.

Na recepção de um pacote que passe a máscara e um dos dois filtros, o seu conteúdo é carregado no *buffer* associado ao seu *index* n. Os procedimentos a seguir descritos devem-se aplicar relativamente a esse *buffer*:

1. Verificação da recepção de uma mensagem inválida através da *flag* IRXIF;
2. Confirmação da utilização do *buffer* n através da *flag* RXFULL;
3. Transferência do *buffer* n para o *Acess Bank Area*, de modo a se poder aceder a ele a partir de qualquer banco de memória. Para tal deve-se definir a janela de endereçamento de acordo com o *buffer* em uso (*window address bits*);
4. Verificação da ocorrência de *overflow*;
5. Leitura do identificador do pacote;
6. Leitura do comprimento de dados recebidos;
7. Leitura dos dados;
8. Libertação do *buffer* n pela recolocação da *flag* RXFULL a zero;
9. Reposição da janela de endereçamento para o seu valor original;
10. Reposição do *interrupt flag* associado ao *buffer* n (RXBnIF) a zero;

Transmissão de um pacote usando o *buffer* n:

1. Verificar se o *buffer* n está pronto para transmissão (TXREQ deve ser zero);
2. Definição da janela de endereçamento para o *buffer* de transmissão n;
3. Definição do identificador;
4. Definição do comprimento do pacote (8 bytes);
5. Definição do conjunto de dados a enviar;
6. Activação da transmissão do *buffer* n (TXREQ=1);
7. Reposição da janela de endereçamento para o seu valor original;
8. *Clear* do *interrupt flag* associado ao *buffer* n (TXBnIF).

Os procedimentos de configuração e de troca de mensagens estão definidos numa biblioteca denominada por *candrivers* para utilização por parte do programador.

1.2.3. Funcionamento do CAN na unidade Master

A troca de informação é desencadeada a partir da unidade Master, estando os Slaves programados para responder imediatamente após a recepção dos pacotes oriundos do Master. A Fig. 18 apresenta o algoritmo utilizado na unidade master para gerir a troca de informação.

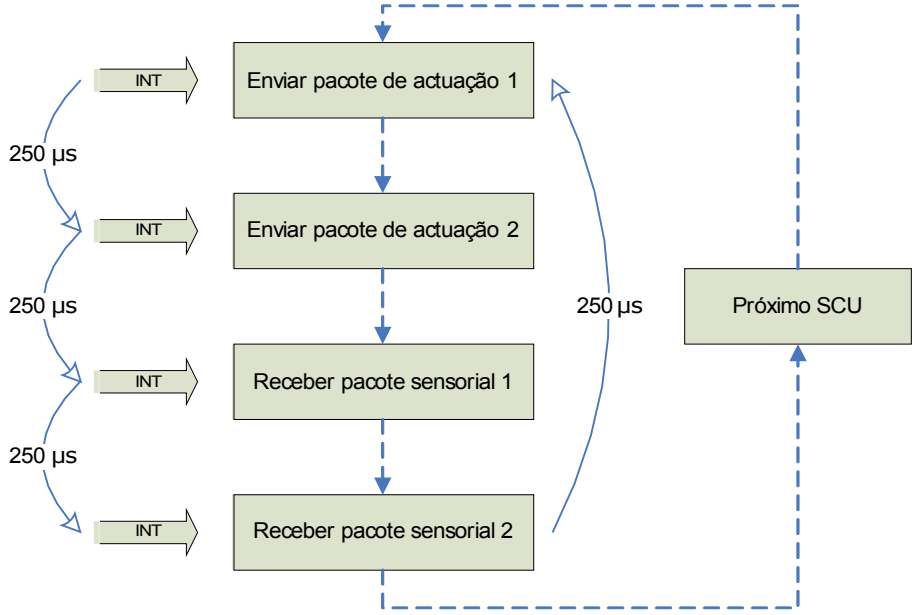


Fig. 18: Algoritmo de troca de informação pelo CAN no Master.

Todas as transmissões e recepções são desencadeadas a partir de um *timer* programado para gerar interrupções de 250 em 250µs. Começa-se por transmitir os dois pacotes relativos à actuação separados por 250 µs no tempo, e a seguir são esperados os dois pacotes sensoriais provenientes do slave para o qual se enviou a mensagem anterior separados também por 250µs. Desta forma a troca de informação para um SCU, o que inclui o envio da mensagem de actuação e a recepção da mensagem sensorial, tem a duração de 1ms. Para uma arquitectura com 8 SCUs, a actualização completa de todas as unidades dura 8ms.

Optou-se por uma política *time-triggered* para minimizar as interferências temporais com a comunicação pela USART que, por sua vez, funciona numa base *event-triggered*, uma vez que se baseia na resposta a mensagens de *request* oriundas do PC. O valor de 250µs resultou de um compromisso entre velocidade e mínima interferência temporal com o funcionamento da USART. Note que o processamento de um pacote tanto para transmissão como para recepção tem uma duração média de 100µs.

Transmissão e Recepção de Mensagens

Do ponto de vista do Master todas as mensagens a enviar são de actuação, e todas as recebidas são de carácter sensorial, pelo que, no último caso, é escusado o processamento do identificador, excepto para o conhecimento do SCU fonte e do index do pacote. No caso do endereço destino e da operação apenas são verificados para efeitos de teste do correcto funcionamento.

As mensagens de actuação a enviar vão buscar os valores a aplicar nos slaves à base de dados global que o PC periodicamente actualiza. Já as mensagens de carácter sensorial que vão chegando, actualizam constantemente as variáveis da mesma base de dados. É este mecanismo que garante a periódica actualização da base de dados sensorial e das unidades slave, no que toca à actuação.

Procedimento de envio de um pacote de actuação:

1. Construção do identificador com base no endereço destino e no *index* do pacote (endereço origem e operação conhecidos);
2. Construir o *array* de dados a enviar, a partir da base de dados de actuação, de acordo com o *index* da mensagem;
3. Enviar o pacote (identificador + *array* de dados) pelo *buffer* de transmissão adequado ao *index*;
4. Actualização do estado de erro, se ocorreu algum.

Recepção de um pacote de carácter sensorial:

1. Obtenção do identificador e do *array* de dados do pacote recebido;
2. Actualização da base de dados sensorial, a partir do endereço origem e do *index* do pacote;
3. Actualização do estado de erro, se ocorreu algum.

Operação I/O	***Causa do Erro***	***Label***	***Padrão***
Recepção/Transmissão	Erro geral	CAN_ERROR	*0b*00000001
Recepção	Mensagem inválida	RX_INVALIDMSG_ERROR	*0b*00000010
	Overflow	RX_OVERFLOW_ERROR	*0b*00000100
	Comprimento de dados inválido	RX_INVALID_LENGTH	*0b*00001000
	Identificador inválido	RX_INVALID_ID	*0b*00010000
	Dados inválidos	RX_INVALID_DATA	*0b*00100000
Transmissão	Erro de transmissão	TX_NOTPOSSIBLE	*0b*01000000
	Transmissão impossível	TX_ERROR	*0b*10000000

Tabela 16: Causas de erro na comunicação CAN.

Na ocorrência de erros, uma estrutura estática armazena a causa do erro na forma de um byte fazendo a operação booleana *or* entre os diversas códigos segundo a Tabela 16. Duas outras variáveis fazem a contagem dos erros de transmissão e de recepção.

```
// Estrutura descritiva dos erros ocorridos
struct  {
    enum CAN_ERRORS flags;          // Causas
    byte tx_count, rx_count;        // Número de ocorrências
} canErrors;
```

Quando um comando de operação *OP_READ_JOINT* e parâmetro *PARAM_CANSTATUS* é pedido pelo PC, uma mensagem é devolvida pela USART contendo a causa do erro e a quantidade de erros de transmissão e de recepção. Uma vez consultados, estes valores são reinicializados a zero.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
*0x*FB	==	Error Code	# TX errors	# RX errors	0

Mensagem de leitura do estado do barramento CAN

1.2.4. Funcionamento do CAN nas unidades Slave

Nas unidades slave, a comunicação CAN é efectuada por *event-triggered*, dado que apenas é accionada quando são recebidas mensagens do master. A Fig. 19 evidencia o algoritmo presente em cada unidade slave.

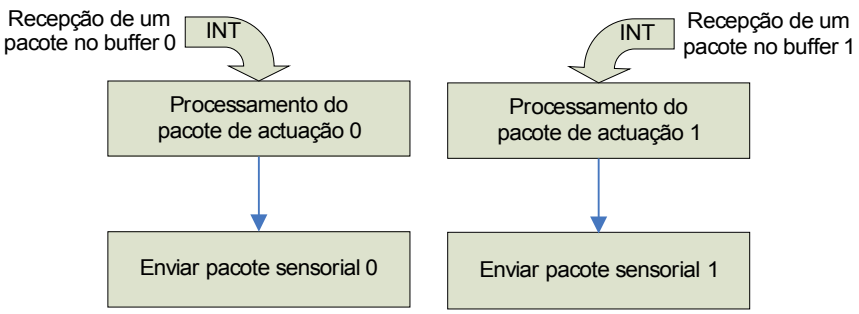


Fig. 19: Algoritmo de troca de informação pelo CAN no Slave.

Quando um pacote é recebido no *buffer* 0 (primeiro pacote de actuação) uma interrupção é gerada, e na Rotina de Serviço à Interrupção (RSI) os dados de actuação são actualizados e um pacote com informação sensorial é enviado (primeiro pacote). Por sua vez, se a interrupção corresponder à chegada de um pacote no *buffer* de recepção 1, o processo é equivalente mas respeitante ao segundo pacote de cada mensagem.

1.2.4.1. Base de Dados Local

Como as novas informações de actuação nunca são aplicadas imediatamente, torna-se necessário armazenar estes valores numa base de dados local para posterior acesso pelas tarefas de controlo local. O mesmo se passa com a informação sensorial que quando é gerada tem que ser armazenada para que a tarefa de comunicação CAN possa acedê-la mais tarde para o envio das mensagens de resposta.
Esta base de dados local é muito semelhante à base de dados apresentada na unidade master, mas agora com a diferença de ser de carácter local, ou seja, apenas armazena a informação sensorial ou de actuação de um único SCU – o próprio.

Base de dados sensorial

```
// Estrutura descritiva dos sensores
typedef struct {
    struct {
        bool reset;                                 // Motores nas home positions
        bool deadlineError;                         // Violação de deadline
        bool motionFin[N_SERVOS];                   // Movimento terminado
        bool controlOn[N_SERVOS];                   // Estado on/off dos controladores
PID
    } sysStatus;                                  // Estado do sistema
    struct_servo servo[N_SERVOS];                 // Sensores d/servos (pos,vel e
corrente)
    unsigned char reaction[N_FORCE_SENSORS];      // Sensores de força dos pés
    signed char giroscope[2];                     // Giroscópio
    signed char inclinometer[2];                  // Inclinómetro
} struct_sensors;

extern volatile struct_sensors   sensors;         // Sensores
```

```
// Estrutura descritiva do estado de um servo
typedef struct {
    signed char position;                         // Posição
    signed char velocity;                         // Velocidade
    unsigned char current;                        // Corrente consumida
} struct_servo;
```

Base de dados de actuação

```
// Estrutura descritiva dos actuadores
typedef struct {
    struct {
        byte kp, kd, ki;                        // Parâmetros Kp, Kd e Ki
        bool controlOn;                         // Controlador On/Off?
    } control[N_SERVOS];                       // Controlador PID
    struct_servo servo[N_SERVOS];              // Info d/actuação (pos,vel e corrente)
} struct_actuators;

extern volatile struct_actuators actuators;     // Actuadores
```

De forma a minimizar a interferência temporal com as restantes tarefas da unidade, as interrupções de recepção de mensagens provenientes do master (Fig. 19) foram definidas como sendo de baixa prioridade, enquanto que as restantes são de alta. Desta forma a troca de mensagens em cada slave só é feita em *background*, ou seja, nos tempos livres do processador.

1.2.4.2. Transmissão e Recepção de Mensagens

Do ponto de vista de cada slave todas as mensagens a enviar são de carácter sensorial, e todas as recebidas são de actuação, pelo que, no último caso, é escusado o processamento do identificador, excepto para o conhecimento do index do pacote. O endereço origem (MCU), endereço destino (o próprio) e da operação apenas são verificados para efeitos de teste do correcto funcionamento da troca de mensagens.

Envio de um pacote de carácter sensorial:

1. Construção do identificador com base no *index* do pacote (endereço origem, destino e operação conhecidos);
2. Construir o *array* de dados a enviar de acordo com o *index* da mensagem, a partir dos dados sensoriais obtidos directamente;
3. Enviar o pacote (identificador + *array* de dados) pelo *buffer* de transmissão adequado ao *index*;

Recepção de um pacote de actuação:

1. Obtenção do identificador e do *array* de dados do pacote recebido;
2. Actualização dos actuadores de acordo com o index do pacote;

No caso das unidades slave, dispensa-se o processamento de erros, uma vez que se pretende reduzir ao mínimo o consumo de largura de banda da rede. O processamento de erros, implicaria a transmissão de mensagens para o master do estado da unidade, o que poderia aumentar o risco de saturação da rede.

1.3 – ORGANIZAÇÃO DO SOFTWARE

1.3.1. Unidade Master

As relações de inclusão entre os diversos módulos estão esquematizadas na Fig. 20. Cada módulo representa um par de ficheiros .h e .c com o protótipo das funções externas e a sua implementação respectivamente. Apenas o módulo Master apenas possui um ficheiro .c dado que é o que contém a função *main*.

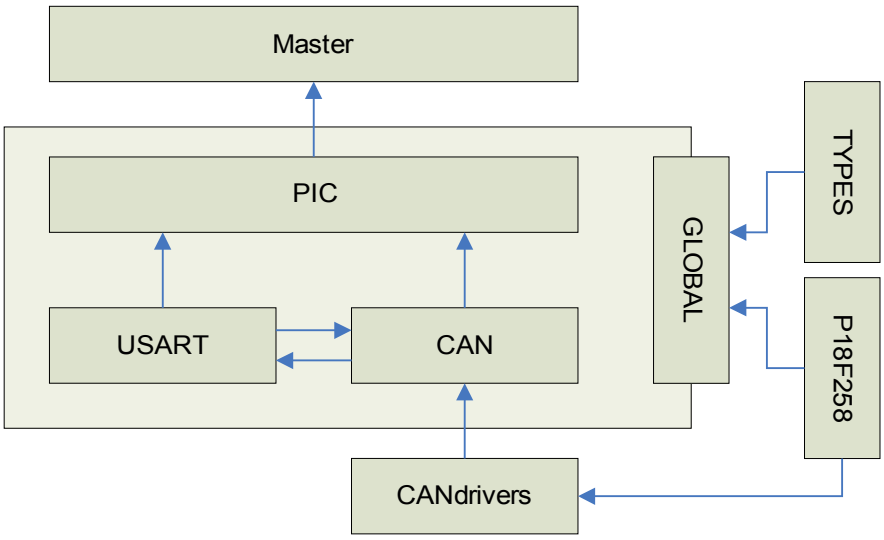


Fig. 20: Relações de inclusão dos módulos de software do Master.

Módulo Master

Este módulo é o primeiro a ser executado (função *main*) e é responsável por fazer a chamada das funções de configuração presentes no módulo PIC. Após os procedimentos de configuração, fica preso numa *dummy task* em que o único software em execução é proveniente da rotina de serviço à interrupção definida no módulo PIC.

Módulo PIC

Módulo com a implementação da rotina de inicialização do PIC e da rotina de serviço às interrupções provenientes da USART e do CAN. É neste módulo que são inicializados e implementados os mecanismos de comunicação com a unidade principal (secção 1.2.3) e as unidades slave (secção 1.2.4).

Tabela 17: Funções do módulo PIC.

Funções	***Descrição***
initPic	Inicialização dos periféricos associados às comunicações USART e CAN.
isr	Implementação dos mecanismos de comunicação pela USART com a unidade principal e da comunicação CAN com as unidades slave.

Módulo USART

Módulo com os *device drivers* para inicialização (secção 1.1.2) e implementação do protocolo USART descrito na secção 1.1.1.

Tabela 18: Funções para manipulação dos *buffers* da USART.

Funções	*Descrição*
usartInit	Inicialização da USART no PIC segundo os dados descritos na secção 1.1.3.1.
usartStreamMode	Esta função indica se o *buffer* de recepção está vazio ou não. Em caso negativo a USART está em modo de recepção de um *stream* de bytes (*true* retornado).
usartStoreRx	Armazenamento de um byte no *buffer* de recepção.
usartGetTx	Obtenção de um byte armazenado no *buffer* de transmissão.
usartResetRxBuff	Reinicialização a zero de todo o *buffer* de recepção.
usartResetTxBuff	Reinicialização a zero de todo o *buffer* de transmissão.
usartStore2extBuff	Armazenamento de um byte no *buffer* externo.

A função *usartStore2extBuff* refere a existência de um terceiro *buffer* denominado por *buffer* externo. Tal destina-se aos módulos externos que pretendam enviar informação adicional pela USART. Tal é o caso do módulo CAN que disponibiliza à unidade principal, através desta função, o estado de funcionamento do barramento CAN (estado de erro). Para o PC aceder a estes dados apenas tem de enviar um comando com o *operation*=OP_READ_JOINT e *parameter*=PARAM_CANSTATUS. O formato da mensagem de resposta está descrita na secção 1.1.1.
A Tabela 19 apresenta um conjunto de funções de mais alto nível, que fazem uso dos *device drivers* explícitos atrás, para processamento dos comandos provenientes do PC e construção da mensagem de resposta a retornar (secção 1.1.3.2).

Tabela 19: Funções de construção da mensagem de resposta para uso da Rotina de Serviço à Interrupção.

Funções	*Descrição*
usartSendTestMsg	Rotina de envio de uma mensagem com o formato FA F9 F8 F7 F6 F5 (hex). Esta mensagem tem como propósito testar o funcionamento da USART.
usartSendStatus	No caso dos comandos de actuação, dois tipos de mensagens podem ser retornados, excluindo as mensagens com erros de recepção por parte do master. Elas são a confirmação da actuação com sucesso ou parâmetros inválidos (secção 1.1.1). Em tais casos é enviada uma mensagem com todos os bytes iguais ao do comando de actuação à excepção do primeiro com a indicação do estado: MESSAGE_SUCESS (0xFB) ou MESSAGE_INVREQ (0xFC). Esta função tem o objectivo de construir uma mensagem igual à última recebida com o primeiro byte igual ao estado a retornar.
usartProcessMsg	Rotina de processamento dos comandos enviados do PC para o master para construção da mensagem de resposta.

Módulo CAN

Módulo com a implementação das funções de alto nível para a recepção e o envio de mensagens CAN (secção 1.2.3) segundo o protocolo descrito na secção 1.2.1.

Tabela 20: Funções de alto nível para troca de mensagens via CAN.

Funções	*Descrição*
canInit	Inicialização do periférico CAN e dos *timers* e interrupções associados.
canSendMsg	Envio de uma mensagem CAN com dados de actuação a aplicar a um determinado SCU.
canReceiveMsg	Recepção de uma mensagem CAN com os dados sensoriais de um determinado SCU.
canClearStatus	Reinicialização do estado de erro do barramento CAN.

Módulo CANDRIVERS

Módulo com os device drivers básicos para inicialização e transmissão/recepção de pacotes via CAN (secção 1.2.2).

Tabela 21: Device drivers da comunicação CAN.

Funções	*Descrição*
myCANInitialize	Inicialização do periférico CAN.
myCANSendMessage	Envio de um pacote através do primeiro *buffer* de transmissão vazio.
myCANSendMessage0	Envio de um pacote através do *buffer* de transmissão 0.
myCANSendMessage1	Envio de um pacote através do *buffer* de transmissão 1.
myCANSendMessage2	Envio de um pacote através do *buffer* de transmissão 2.
myCANReceiveMessage	Leitura de um pacote no primeiro *buffer* de recepção cheio.
myCANReceiveMessage0	Leitura de um pacote no *buffer* de recepção 0.
myCANReceiveMessage1	Leitura de um pacote no *buffer* de recepção 1.

Módulo GLOBAL

Módulo com a definição da base de dados com os dados de actuação e sensorial de todos os SCUs da arquitectura (estruturas descritas na secção 1.1.3.3).

Tabela 22: Funções presentes no módulo GLOBAL.

Funções	*Descrição*
initGlobal	Inicialização de toda a base de dados para os valores origem.
initSensors	Inicialização somente da base de dados sensorial.
initActuators	Inicialização somente da base de dados de actuação.

Módulo TYPES

Módulo com a definição de tipos de variáveis extra para uso nos restantes módulos. Eles são:

Tipo de variável	*Descrição*
bool	Tipo booleano (*true* ou *false*) (tipo enumerado).
byte	Tipo inteiro de 8 bits sem sinal (*unsigned char*).
word	Tipo inteiro de 16 bits sem sinal (*unsigned int*).
dword	Tipo inteiro de 32 bits sem sinal (*unsigned long*).

Módulo P18F258

Biblioteca com a definição de todos os registos e bits correspondentes do PIC 18F258 para o seu controlo (ver *datasheet* da *Microchip*, PIC18F258).

1.3.2. Unidade Slave

A Fig. 21 apresenta as relações de inclusão entre os vários módulos de *software* da unidade slave. Os módulos a verde são iguais aos utilizados na unidade master, pelo que dispensam apresentações. Quanto aos módulos a laranja não estão relacionados com as comunicações e por isso não estão cobertos neste capítulo.

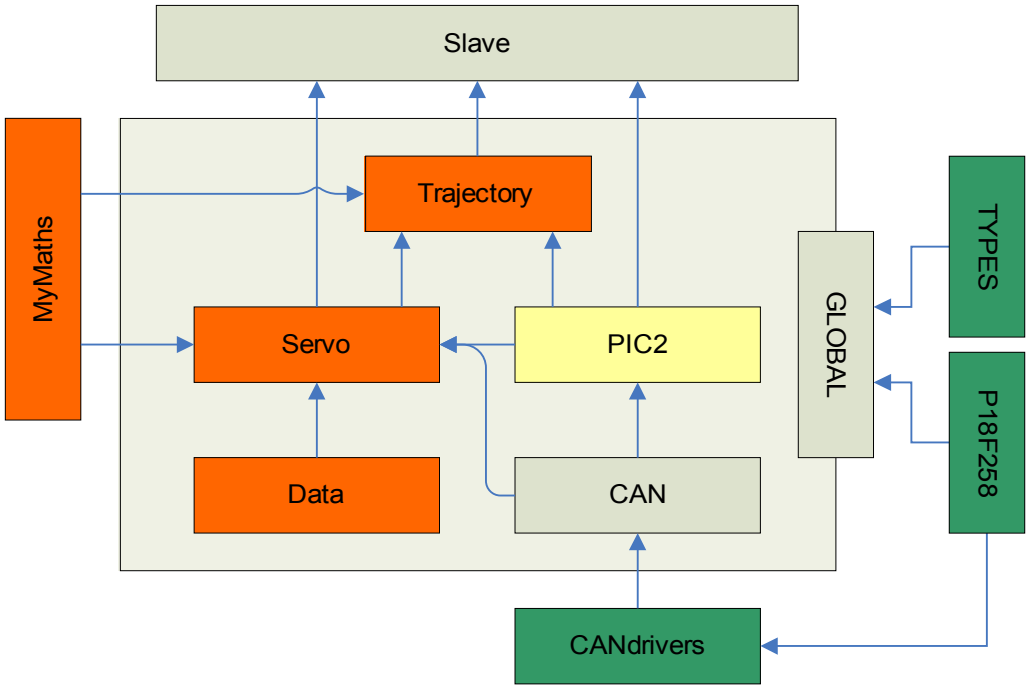


Fig. 21: Relações de inclusão dos módulos de software de cada Slave.

Módulo Slave

Módulo com a função iniciadora *main*, responsável pela chamada das funções de configuração presentes nos outros módulos e por executar tarefas em *background* durante o funcionamento normal.

Módulo PIC2

Módulo com a implementação da rotina de inicialização do PIC e da rotina de serviço às interrupções, uma das quais provenientes do CAN. É neste módulo que são inicializados e implementados os mecanismos de comunicação com a unidade master.

Tabela 23: Rotinas do módulo PIC2 responsáveis por gerir as comunicações CAN.

Funções	***Descrição***
initPic	Inicialização das comunicações CAN e de outros periféricos destinados ao controlo local.
lowISR	Rotina de Serviço à Interrupção responsável por gerir as interrupções provenientes do CAN.
...	(outras tarefas para controlo local)

Módulo CAN

Módulo com a implementação das funções de alto nível para a recepção e o envio de mensagens CAN (secção 1.2.4) segundo o protocolo descrito na secção 1.2.1, mas no que respeita à unidade slave.

Tabela 24: Funções de alto nível para troca de mensagens via CAN.

Funções	*Descrição*
initCan	Inicialização do periférico CAN e das interrupções associadas.
sendCanMessage	Envio de uma mensagem CAN com dados de sensoriais para a unidade master.
receiveCanMessage	Recepção de uma mensagem CAN proveniente do master com os dados de actuação a aplicar.
checkCan	Verificação do bloqueio do CAN, no caso do PIC se encontrar no modo *bus-off*.

A função *checkCan* é útil para verificar se a unidade ainda possui a interface CAN activa e em funcionamento. Cada nó na rede pode comportar-se de três formas diferentes durante o seu funcionamento:

- Modo *error-active*: mensagens normais e frames de erro (na ocorrência de erros) com bits dominantes são trocadas com os outros nós para lhes indicar a ocorrência de anomalias;
- Modo *error-passive*: as frames de erro passam a ser constituídas por bits recessivos para evitar a interferência destrutivas das mensagens provenientes de outros nós;
- Modo *bus-off*: o nó é bloqueado em termos de recepções e transmissões.

Na ocorrência de erros, um contador é incrementado, sendo também decrementado na ausência deles. Quando atinge um valor limite o modo de funcionamento vai alternando para o modo seguinte até atingir o modo *bus-off*, bloqueando as recepções e as transmissões com os outros nós.
Esta função foi construída devido ao facto de as unidades esporadicamente bloquearem as suas comunicações e ter assim um meio para verificar a causa.

Módulo GLOBAL

Módulo com a definição da base de dados com os dados de actuação e sensorial do próprio SCU (secção 1.2.4.1).

Tabela 25: Funções presentes no módulo GLOBAL.

Funções	*Descrição*
initGlobal	Inicialização de toda a base de dados para os valores origem.
initSensors	Inicialização somente da base de dados sensorial.
initActuators	Inicialização somente da base de dados de actuação.
reinitServo	Reinicialização das posições dos actuadores para os valores origem (definidos à priori pelo próprio SCU).

1.4 – CONCLUSÕES

A arquitectura de comunicações adoptada para a estrutura humanóide, revelou-se bastante simples de implementar e ao mesmo tempo seguindo uma lógica intuitiva. Esta arquitectura segue uma lógica distribuída, porque a cada trio de juntas (normalmente associadas a um membro físico) existe uma unidade de controlo local baseada num PIC que se responsabiliza em aplicar os sinais de controlo e ler a informação sensorial, que se interligam umas às outras por meio de um único barramento partilhado entre todos. Este conjunto de unidades locais interligam-se a uma unidade mestre (*master*) que por sua vez está conectada à unidade principal, neste caso um PC, que é responsável por enviar os comandos de actuação e de leitura sensorial a cada unidade local. Deste modo, cada unidade tem uma funções específica, deslocalizando assim todas as tarefas de controlo dos servomotores e de distribuição das mensagens entre os diversos *slaves* da unidade principal.

Implementando os protocolos de comunicação tanto série entre o PC e o *master*, como o CAN entre o *master* e os *slaves* seguindo a metodologia de funcionamento enunciada, a fiabilidade das comunicações revelou-se bastante boa com a troca de mensagens à velocidade máxima possível e sem a detecção de atrasos excessivos na sua entrega.
No entanto, esporadicamente, observa-se nalgumas unidades *slave*, o bloqueamento das comunicações CAN ao fim de uns largos minutos de funcionamento. Suspeita-se que por acumulação de erros de transmissão e de recepção, o respectivo módulo CAN entre em modo *bus-off* desligando todas as transmissões e recepções. Embora o protocolo construído faça a contagem dos erros e armazene a sua causa, para consulta pela unidade principal, ainda não é suficiente para apurar a causa dos erros. Porém verifica-se que ocorrem erros de entrega de mensagens inválidas e de *overflow*, mas sem no entanto se verificar nenhum problema a nível prático. Só ao fim de entre 5 a 10 minutos de funcionamento, uma ou mais unidades, inexplicavelmente desliga o seu porto CAN. Como a troca de mensagens isoladas funciona em óptimas condições, pensa-se que a causa pode estar por detrás da gestão da largura de banda da rede, que, com a sobrecarga de mensagens na rede, as de menor prioridade podem-se atrasar de uma forma acumulativa até provocar o bloqueio da unidade local respectiva. Tal ideia é suportada pelo facto, de até agora, as unidades que são bloqueadas correspondem às de menor prioridade. Mais investigação deve ser feita para perceber melhor o que está em causa.

Contudo, de uma forma geral, o sistema de comunicações está completamente funcional para operar directamente sobre as juntas a partir do PC como unidade principal. Utiliza-se o *software* MatLab para fazer a interface uma vez que oferece uma linguagem de alto nível e intuitiva para programação das sequências de comandos a aplicar nas juntas e também para monitorar os sinais de saída. O capítulo 2 faz uso dos *device drivers* enunciados na secção 1.1.2.2 para enviar comandos de actuação sobre os servomotores e também para leitura do sinal de posição e outros associados ao actuador.

Capítulo 2

2. SISTEMA DE CONTROLO DE BAIXO-NÍVEL

Resumo:
Este capítulo tem como objecto de estudo os actuadores presentes nas juntas: como é feita a leitura do seus sensores e a actuação. Posteriormente combinaremos estes dois procedimentos de forma a corrigir eventuais desvios ao comportamento esperado.

2.0 – CONTROLO DA PLATAFORMA HUMANÓIDE

Como já foi mencionado capítulo anteriore sobre a arquitectura de controlo, o robot humanóide é constituído por oito unidades de controlo local, que denominámos por *slaves*, que podem controlar até 3 graus de liberdade dos 22 existentes na arquitectura actual, e que se interligam com um computador (unidade principal), através de uma unidade *master*, para o envio de comandos de actuação com base na informação sensorial medida.

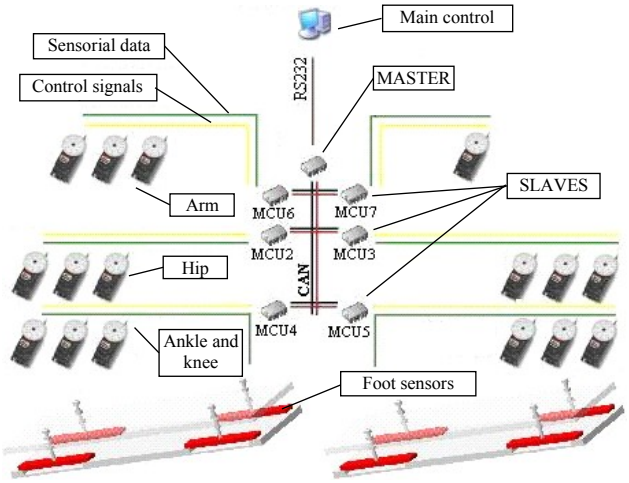


Fig. 22: Arquitectura da plataforma humanóide.

Como actuador para cada grau de liberdade, estamos a utilizar servomotores, que não são nada mais, nada menos, do que motores adaptados para efectuar controlo de posição. Como vantagens estes motores permitem-nos fazer medidas da sua posição e da sua corrente consumida o que nos possibilitará mais adiante fazer controlo externo para correcção de eventuais desvios ao comportamento ideal.

Para além do controlo sobre as juntas, também existem um conjunto de sensores adicionais destinados ao equilíbrio desta plataforma. Eles são:

- ➔ Sensores de força aplicados directamente sobre a base de cada pé, implementados a partir de extensómetros – resistências que variam o seu valor de acordo com a sua deformação.
- ➔ Inclinómetros para medição da verticalidade do tronco. Estes inclinómetros são basicamente acelerómetros que medem a aceleração da gravidade nos seus dois eixos ortogonais – na posição vertical o vector força gravítica deve coincidir com o seu eixo vertical.
- ➔ Giroscópios para medição da velocidade angular de certos pontos do corpo.

Para já, ainda só foram testados os sensores de força aplicados na base dos pés, com o intuito de assegurar a projecção do centro de massa do corpo sobre o centro de um dos pés (quando apenas um pé está assente no solo). Desta forma, para pequenas velocidades e pequenas acelerações, o robot mantém-se em equilíbrio. Este assunto será discutido de forma mais detalhada no próximo capítulo sobre controlo de equilíbrio.

Neste capítulo apenas serão discutidos os actuadores e o seu controlo de forma a permitir:

- ➔ Fazer o controlo de posição de modo a garantir que o actuador atinge sempre a posição solicitada;
- ➔ Fazer o controlo de velocidade, variando o seu valor de acordo com as necessidades;
- ➔ Que os movimentos das juntas sejam os mais suaves possíveis, sem acelerações bruscas nem velocidades muito elevadas;

Embora, os servomotores tenham sido escolhidos por possuírem um controlador interno de posição, contrariamente aos outros géneros de motores, veremos adiante que apresenta muitos problemas na concretização dos objectivos enunciados, principalmente na presença de grandes cargas no seu eixo.

2.0.1. A Unidade de Controlo Local

A Fig. 23 apresenta a constituição de uma unidade de controlo local – *slave* – com o microcontrolador PIC18F258 e a electrónica de interface para os servomotores e os diversos sensores de equilíbrio.

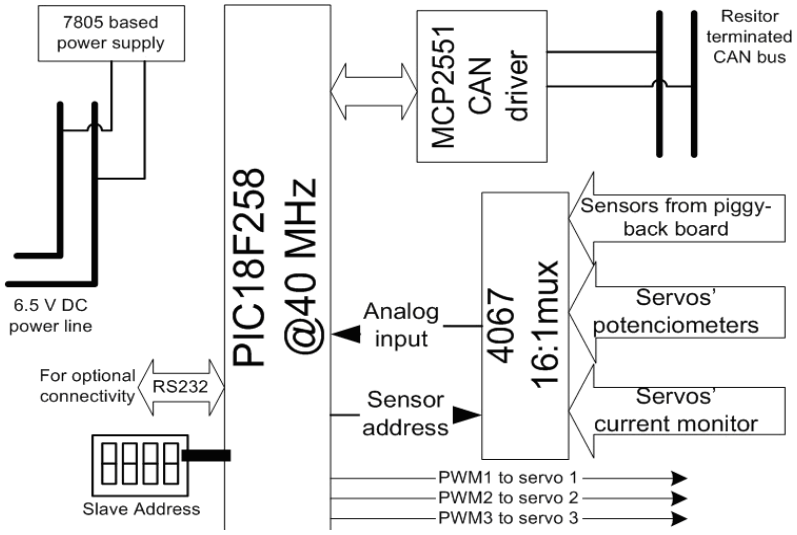


Fig. 23: Constituição de uma unidade slave.

Como se pode observar, cada slave, possui uma interface para o barramento CAN constituído pelo MCP2551 *CAN driver* uma interface adicional para comunicar pela USART e um *DIP switch* que permite via hardware configurar o endereço da unidade (*slave address*). Para a actuação sobre os servomotores, um pino por servomotor (três no total) está reservado para aplicação directa de um sinal modulado em *duty-cycle* para o controlo de posição. Adicionalmente, um multiplexer de 16 canais é utilizado para fazer a aquisição sensorial dos servomotores e dos sensores de equilíbrio.

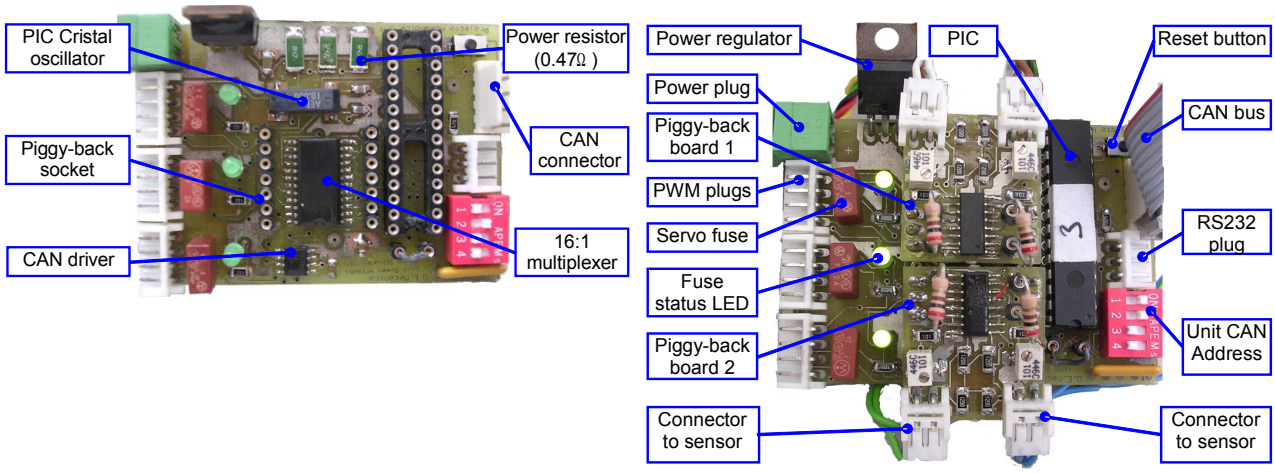


Fig. 24: Imagens de uma unidade de controlo local.

As imagens da Fig. 24 correspondem à versão implementado no ano lectivo anterior (2004/05). No que respeita a este ano, modificações foram feitas de modo a resolver alguns bugs. Eles são:

- ✔ <u>Fusíveis de 2A substituídos por de 3.15A;</u>

  Devido aos picos de corrente induzidos pelo arranque dos servomotores, provou-se ser insuficiente a corrente máxima de 2A, pelo que foi necessário aumentar para um escalão acima: 3.15A.

- ✔ Condensador electrolítico de 1μF adicionado à entrada do regulador de tensão;

  Para evitar a ocorrência de resets nos PICs adicionou-se um condensador de filtragem à entrada do regulador de tensão, para estabilização da tensão de entrada.

- ✔ Condensador SMD de 100nF colocado aos terminais da alimentação do PIC;

  De modo a conferir maior estabilidade na alimentação do PIC colocou-se um condensador de filtragem directamente sobre os pinos da alimentação. Por uma questão de boas práticas é recomendado colocar um condensador de filtragem de elevado valor à entrada do regulador e um condensador de menor valor directamente sobre os pinos de alimentação de cada circuito integrado que esteja ligado à saída do regulador. Por isso, caso uma nova versão seja construída no futuro, aconselha-se a implementação deste método.

- ✔ Resistências de potência ligadas entre a massa dos servomotores e a do SCU curto-circuitadas;

  Com a presença desta resistência, verificava-se que os níveis de corrente consumidos eram bastante maiores que na sua ausência. Como estas resistências eram utilizadas para medição de corrente, e tendo em conta que se descobriu outro método para efectuar este procedimento, estas resistências foram curto-circuitadas (ver secção 2.1.3.1).

- ✔ Adicionadas resistências de 10KΩ entre a entrada de actuação de servomotor e a massa.

  Por motivos desconhecidos, alguns servomotores, na ausência do sinal de entrada de actuação, ficavam descontrolados efectuando movimentos contínuos sem uma posição final específica. Descobriu-se que ligando uma resistência à entrada do sinal de actuação com o outro terminal à massa, este efeito deixava de se verificar.

2.1 – ACTUADORES: OS SERVOMOTORES

Para a actuação sobre as juntas é essencial o controlo de posição, e dado que em média cada junta não possui uma excursão de movimento superior a 180º, uma solução baseada em servomotores foi a mais imediata dado permite fazer controlo de posição em vez de velocidade como nos motores vulgares. Podem-se enunciar as seguintes vantagens para esta escolha:

- ✔ Excursão de posição de 180º;
- ✔ Controlador de posição incluído;
- ✔ Relativamente pequeno e compacto;
- ✔ Relativamente barato.

Fig. 26: Servomotor da HITEC

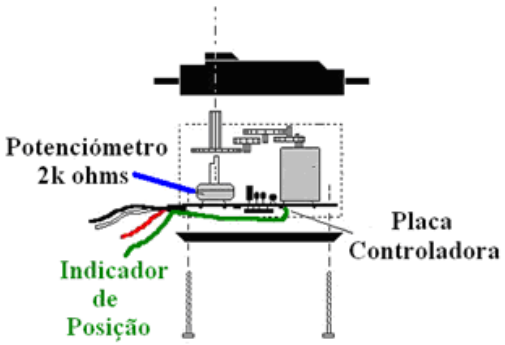


Fig. 25: Representação do interior de um servomotor.

Olhando para o interior deste actuador podemos discernir os seguintes componentes:

- ➔ Motor DC;
- ➔ Engrenagens de desmultiplicação (de velocidade);
- ➔ Mecanismo de *feedback* da posição angular do motor (potenciómetro);
- ➔ Electrónica de controlo do motor a partir de um sinal digital;
- ➔ Electrónica de controlo de posição.

Pode-se, por isso, concluir que estes servos, não são nada mais do que simples motores DC que incluem electrónica interna responsável por implementar o controlo de posição a partir de um sinal externo de referência. Este sinal externo define a posição a ser atingida é da forma de uma onda quadrada modulada em largura de impulso – PWM (*Pulse Width Modulation*) –, cuja largura é quem define a posição final. Desta forma, a posição do eixo do servo é controlada a partir do *duty-cycle* de uma onda quadrada de formato digital (Fig. 27).

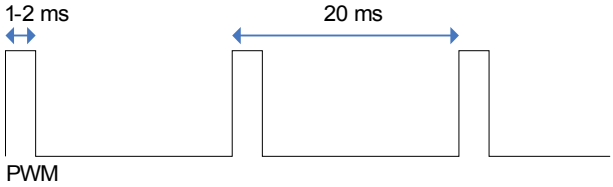


Fig. 27: Sinal de PWM aplicado no servomotor.

Spec	Values	
Control system	Pulse Width Control 1.5 ms neutral	
Voltage range	4.8V to 6.0V	
Teat voltage	@ 4.8V	@ 6.0V
Speed (no load)	60º/0.19 s	60º/0.14 s
Stall torque	1.94 Nm	2.42 Nm
Operating angle	45º /one side pulse traveling 400µs	
Direction	clockwise/pulse traveling 1.5 to 1.9 ms	
Current drain	8mA (idle); 700mA (no load running)	
Dead bandwidth	8 µs	
Dimensions	66 x 30 x 57.6 mm	
Weight	152g	

Tabela 26: Especificações do servo da HITEC HS-805BB.

Para cada modelo de servomotor este sinal deve assumir determinadas especificações, mas para um dos modelos utilizados (HITEC HS-805BB[1]) estas devem ser as características do sinal de PWM:

- ➔ Frequência de 50 Hz (Período de 20 ms);
- ➔ Duty-cycle variável entre 1 (0º) e 2 ms (180º).

Mais dados podem ser consultados na Tabela 26 com indicações do *duty-cycle* correspondente à posição neutra ou central (90º) (*control system*), gama das tensões do sinal de PWM (*voltage range*), em que observa que aceita sinais digitais de 5V provenientes do PIC, binário máximo, corrente consumida, e outros.

A escolha do modelo HITEC HS-805BB para as juntas mais exigentes, recaiu essencialmente pelo seu binário máximo de 2.42 N.m. Simulações em CATIA da plataforma humanóide realizando um passo (ano 2004/05) demonstram que no pior cenário as juntas podem estar submetidas a binários de cerca de 2.6 N.m (Tabela 27). O modelo HS-805BB destacou-se por possuir o máximo binário da gama, que mesmo assim, mostra ser insuficiente para o nosso caso. Para resolver este problema, introduziram-se correias de transmissão que multiplicam o binário máximo pela relação entre número de dentes do eixo do servo e da junta.

Motor / Junta	Θ_1 [º]	T_1 [N.m]	Θ_2 [º]	T_2 [N.m]	Θ_3 [º]	T_3 [N.m]
Pé 1 lateral	0.0	**2.37**	7.1	0.98	7.1	0.96
Pé 1 frente	4.7	0.30	10.1	0.20	10.1	0.04
Joelho 1	10.1	0.76	21.8	1.17	21.8	1.01
Anca 1 frente	5.4	0.35	11.7	0.30	11.7	0.14
Anca 1 lateral	0.0	**2.26**	7.1	**2.57**	7.1	**2.55**
Pé 2 lateral	0.0	0.00	7.1	0.00	7.1	0.00
Pé 2 frente	4.7	0.12	10.1	0.12	16.4	0.12
Joelho 2	10.1	0.17	21.8	0.23	41.9	0.30
Anca 2 frente	5.4	0.07	11.7	0.02	25.6	0.14
Anca 2 lateral	0.0	0.01	7.1	0.30	7.1	0.29

Tabela 27: Binários exigidos na simulação de um passo.

$$N = \frac{Número\ de\ dentes\ da\ polia\ da\ junta}{Número\ de\ dentes\ do\ eixo\ do\ servo}$$

$$\tau_{máx} = N \times \tau_{servo}$$

Fig. 28: Correia de transmissão aplicada a um servo.

Nas juntas dos pés e dos joelhos utilizaram-se relações de transmissão de 2:1 e de 2.2:1 assegurando um binário máximo de 5.3 N.m assumindo como 2.42 o binário máximo do servomotor, e nas juntas das ancas aumentou-se a margem para 1:3.75 dada a sua exigente natureza (máximo de 9.1 N.m).

1 Estes dispositivos podem ser adquiridos no site http://www.maxxprod.com

Olhando para o interior do servo na tentativa de perceber melhor a sua electrónica interna, a Fig. 29 apresenta o circuito de controlo de um servo semelhante aos da HITEC – o FUTABA S3003 – cujo sinal de PWM (input pulse) apresenta as mesmas especificações que as descritas atrás.

Estes dispositivos utilizam como *feedback* tanto o sinal de posição como de velocidade que, além de fornecer um controlo preciso de posição, permite estabilizá-lo de modo a conduzi-lo para a posição desejada com o mínimo de oscilação.

Em funcionamento normal, o sinal de PWM da entrada é comparado com o sinal resultante de um gerador de impulso linear controlado a partir da posição obtida pelo potenciómetro e da velocidade medida a partir da força contra-electromotriz do motor (tensão gerada entre impulsos de potência). O sinal gerado é da mesma forma que o de entrada e, para baixas velocidades, a sua largura de impulso deve corresponder à posição efectiva do motor.
A diferença de largura de impulso entre estes dois sinais, conhecido como sinal de erro, é em seguida amplificado através de um amplificador de impulso que depois é aplicado numa ponte H (BAL6686) que controla o motor. Facilmente se percebe que quando a largura de impulso do sinal de entrada é igual à resultante pelo gerador de impulso (erro zero) a diferença é de largura nula e nenhum sinal é aplicado ao motor deixando-o em repouso – note que o motor em si é controlado em velocidade pelo que se nenhum sinal for aplicado ele tende para o repouso.

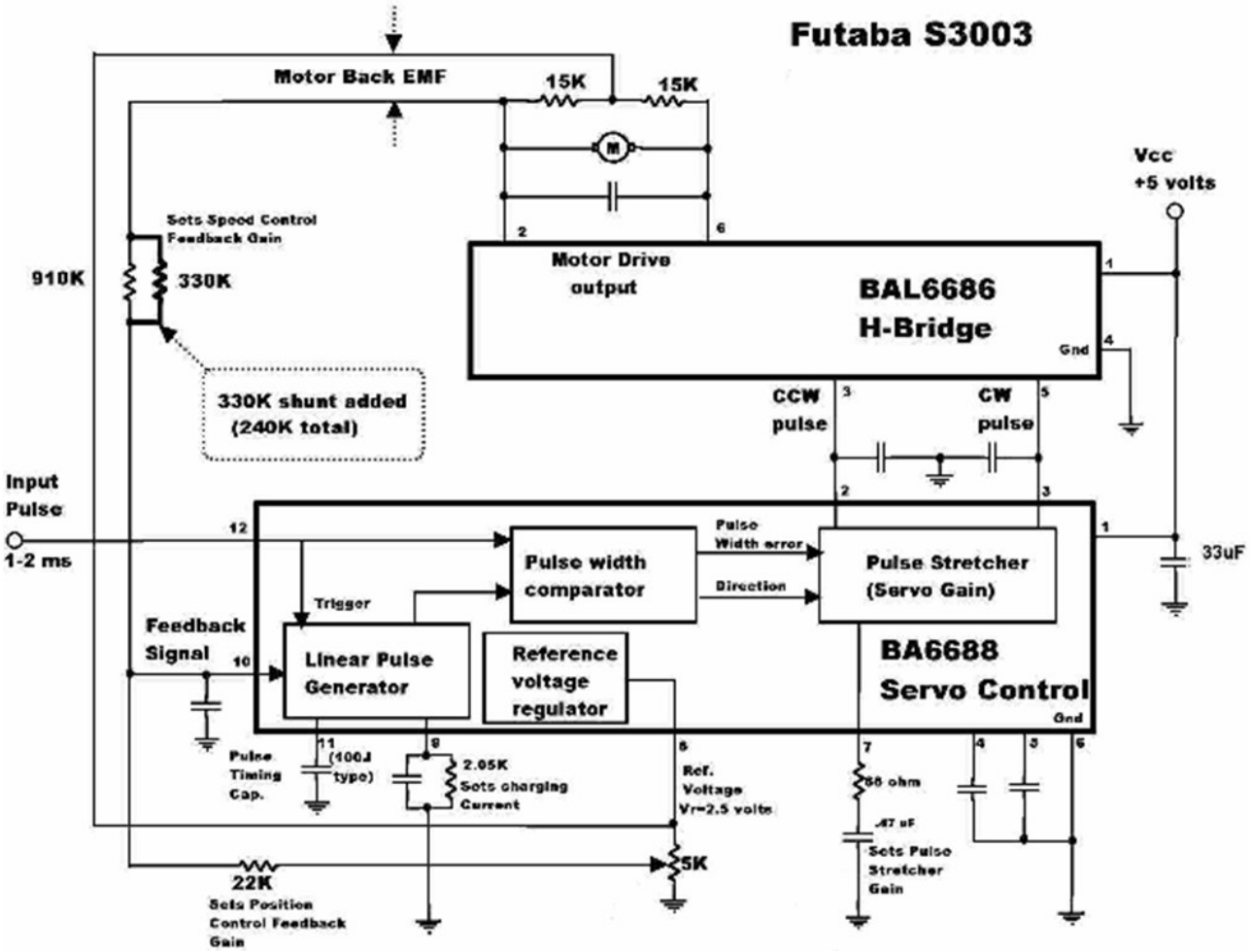


Fig. 29: Circuito do controlador de posição de um FUTABA S3003.

Representando matematicamente as operações envolvidas, verificamos que:

- À posição medida pelo potenciómetro p(t) é aplicado um ganho K_P representado pela resistência de 22KΩ que é somada à velocidade resultante da força contra-electromotriz multiplicado pelo ganho K_D representado pelo paralelo das resistências de 910K e de 330KΩ.

$$F(s)=K_P\cdot P(s)+K_D\cdot V(s)$$

$$F(s)=(K_P+s\cdot K_D)\cdot P(s)$$

- O resultado da soma f(t) é introduzido no gerador de impulso linear que produz um sinal de PWM para comparação (em termos de largura de impulso) com o sinal de entrada de referência r(t) através do comparador de largura de impulso produzindo o sinal de erro e(t) também do formato PWM.

$$E(s)=R(s)-F(s)$$

- O sinal de erro é amplificado num factor de K através de um amplificador de impulso e é aplicado ao motor através de uma ponte H.

$$U(s)=K\cdot E(s)$$

Logo, concluindo:

$$\boxed{U(s)=K\cdot[R(s)-P(s)\cdot(K_P+s\cdot K_D)]}$$

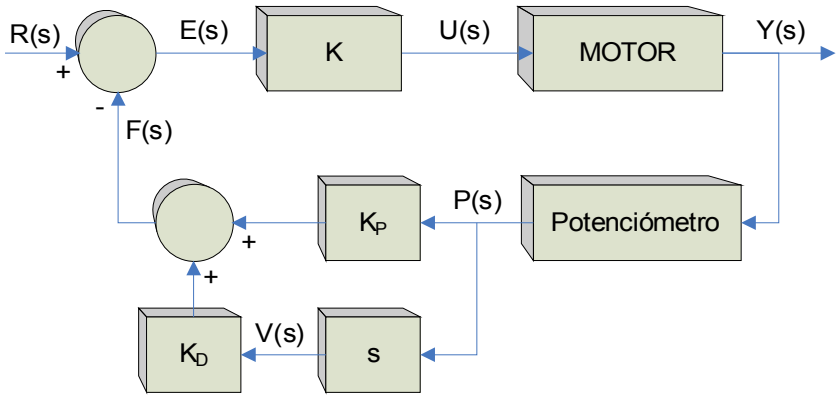


Fig. 30: Representação esquemática do controlador de posição interno.

Comparando a Fig. 29 e a Fig. 30 podemos relacionar o elemento *s* como o sensor de velocidade, a diferença R(s) – F(s) como o comparador de largura de impulso e o ganho K como o amplificador de largura de impulso. Verifica-se então dois tipos de compensação presentes: a compensação série executada pelo ganho proporcional K e a compensação paralela realizada na realimentação através de um ganho PD (proporcional+derivativo) à posição medida. Note que o sinal proveniente da realimentação f(t) para posterior comparação não depende exclusivamente da posição medida, mas também da velocidade actual o que confere uma maior estabilidade na realização do percurso para a posição desejada.

Um pormenor que vale a pena salientar é a ausência da componente integral no controlo, o que poderá originar erros em regime estacionário quando aplicadas no eixo cargas elevadas.

2.1.1. Setup Experimental

Para efeitos de testes ao actuador em questão, uma base experimental foi montada tendo em vista a realização de movimentos variando os seguintes parâmetros:

- ➔ Excursão do movimento (posição inicial e final);
- ➔ Velocidade do movimento[2];
- ➔ Carga aplicada no eixo.

... de forma a poder avaliar a performance do servomotor pela monitorização da posição efectiva que percorre ao longo do tempo e da corrente consumida.

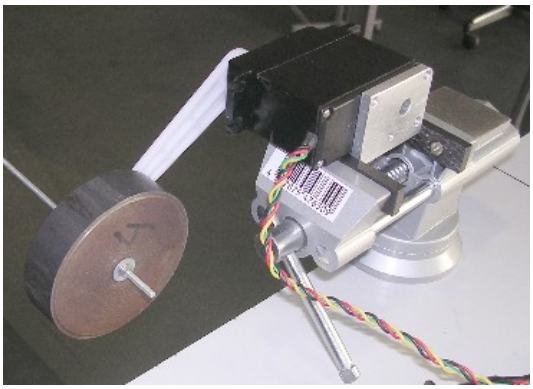


Fig. 32: Setup experimental.

Load	Mass (g)	Torque (N.m)
0	9	0.009
1	258	0.253
2	463	0.454
4	675	0.662
1+4	924	0.906
2+4	1129	1.108
1+2+4	1378	1.352

Tabela 28: Lista de cargas utilizadas para teste.

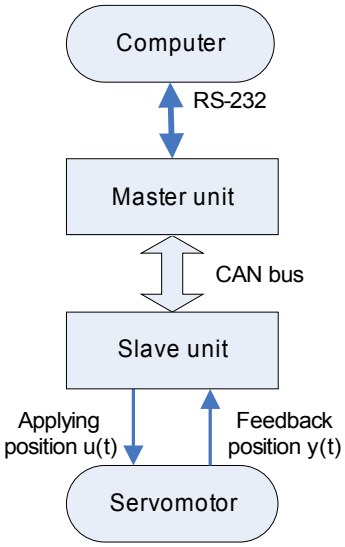


Fig. 31: Arquitectura das comunicação no setup.

Pela Fig. 31 podemos visualizar a arquitectura das ligações com a unidade principal (PC). Como só tencionámos testar um único actuador, apenas precisamos de uma unidade *slave* que liga ao PC segundo a rede de comunicações utilizada na plataforma humanóide.

O servomotor, por sua vez, encontra-se fixo num torno de fixação para poder mover cargas de uma forma segura. As cargas são presas ao servo através de um braço de massa desprezável e de 10 cm de comprimento que percorre a excursão dos 180º desde a posição vertical inferior até ao ponto vertical superior. Esta opção é muito útil dado que nos permite estimar o valor do binário resultante da força gravítica através da seguinte fórmula...

$$\tau = F_G * L * \sin(\theta) = m * g * L * \sin(\theta) \qquad \text{(Equação 1)}$$

... em que m é a massa da carga, g a aceleração gravítica (9.81 m/s^2), L é o comprimento do braço (0.1m) e θ é o ângulo entre o braço e o vector força gravítica F_G. A partir dela sabemos que nos extremos o binário gravítico é nulo e no ponto central (braço e F_G perpendiculares) ele é máximo. Por convenção atribuiu-se ao extremo inferior a posição -90º, à superior +90º, com o ponto central de posição igual a 0º.

A Tabela 28 indica a lista de massas utilizadas durante os testes. Embora as especificações destes servos indiquem um binário máximo de 2.42 N.m, na prática verificou-se que este estava muito abaixo deste valor deixando de responder para cargas superiores a 1.5 Kg (binário τ=1.47 N.m) talvez devendo-se a desgaste destes actuadores. Tal justifica o facto da massa mais elevada da lista ser de 1.38 Kg.

2 Embora não seja possível fazer controlo directo de velocidade, é possível a partir do controlo de posição induzir uma determinada velocidade ao dispositivo. Tal é explicado no capítulo seguinte.

2.1.2. Actuação sobre o Servomotor

Tal como foi descrito na secção 2.0.1 cada unidade de controlo local é capaz de gerar três sinais de PWM independentes entre si, a partir do microcontrolador PIC disponibilizando-os através de três pinos reservados que se podem ligar directamente aos servomotores. Note, contudo, a necessidade de adicionar a cada saída de PWM uma resistência de 10KΩ à massa para drenagem da carga existente aquando da ausência de PWM. Sem este elemento, nestas circunstâncias, o servo poderá deslocar-se à máxima velocidade para posições imprevistas que podem não se encontrar na gama anunciada dos 180º resultando no encrave num dos extremos com o máximo consumo de corrente. Tal comportamento imprevisto pode danificar estes dispositivos não mencionando a própria estrutura do humanóide.

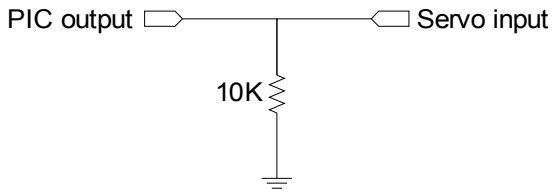


Fig. 33: Resistência a adicionar a cada saída de PWM do PIC.

Vamos agora descrever como é programado o PIC para gerar os três sinais de PWM. Tendo em mente a geração destes sinais de forma automática sem nunca colocar o CPU em espera (por *polling*) várias opções são-nos oferecidas, nomeadamente utilizando...

- ➔ Módulo de PWM (CCP);
- ➔ Módulo de comparação (CCP);
- ➔ Interrupções.

Dado o PIC 18F258 só possuir dois módulos CCP (CCP e ECCP) tal é insuficiente para gerar três sinais de PWM, com a agravante da frequência mínima configurável para o módulo de PWM ser bastante superior a 50 Hz. Por isso só nos resta a última opção recorrendo unicamente a interrupções baseadas em *timers*.
Só pensando na geração de um único PWM bastaria o uso de dois *timers*: um para a elevação do sinal a 1 com uma frequência fixa de 50 Hz e um segundo para a descida do sinal a zero depois de um determinado período de tempo após a elevação. Desta forma obtém-se um sinal de PWM cujo período de tempo a 1 corresponde ao *duty-cycle* desejado para definição da posição do servo.

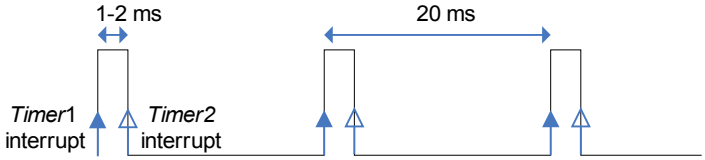


Fig. 34: Geração de um sinal de PWM através de dois timers.

A Fig. 34 esquematiza este procedimento fazendo uso dos *timers* 1 e 2:

1. O *timer* 1 é programado para gerar uma interrupção a cada 20ms (50Hz de frequência).
   A cada instanciação o pino de PWM é elevado a um;
2. Após cada interrupção do *timer* 1, o *timer* 2 é programado para gerar uma segunda interrupção após um período de tempo correspondente ao *duty-cycle* desejado (entre 1 e 2 ms).
   Após a ocorrência desta interrupção, o pino de PWM é baixado a zero.

No entanto, importa escalar este procedimento para a geração de três sinais de PWM que, embora todos tenham a mesma frequência, o *duty-cycle* gerado para os três servos devem ser independentes entre si. Uma solução poderia ser o uso de *timers* extra, mas dada a limitação de recursos, desenvolveu-se um método que permite o controlo de *N* servos usando apenas estes dois *timers*, cuja quantidade *N* apenas depende da velocidade de processamento do CPU:

1. O *timer* 1 é programado para gerar uma interrupção a cada 20ms (50Hz de frequência).
   A cada instanciação o pino de PWM é elevado a um;
2. Após cada interrupção do *timer* 1, o *timer* 2 é programado para gerar uma segunda interrupção após um período de tempo correspondente ao *duty-cycle* mínimo: 1ms.
   Na ocorrência da primeira interrupção, o timer 2 é reprogramado para gerar interrupções de alta frequência de período correspondente à variação de posição mínima do servo Δ_{pos}. Para um passo de 1º, o período será:
   $$T_{hf} = \frac{duty_{max} - duty_{min}}{180º / \Delta_{pos}} = \frac{2000\,\mu s - 1000\,\mu s}{180º / 1º} = \frac{1000\,\mu s}{180} = 5.56\,\mu s$$
3. A cada instanciação da interrupção de alta frequência (em cada passo de 5.56µs), a cada servo é verificado se o PWM correspondente deve baixar a zero nessa iteração. Em caso negativo, nada é feito e a próxima iteração é aguardada para uma nova avaliação.
   Na última iteração (correspondente ao *duty-cycle* máximo) todos os sinais de PWM devem estar a zero e o *timer* 2 é desligado. O processo é repetido na próxima interrupção do *timer* 1.

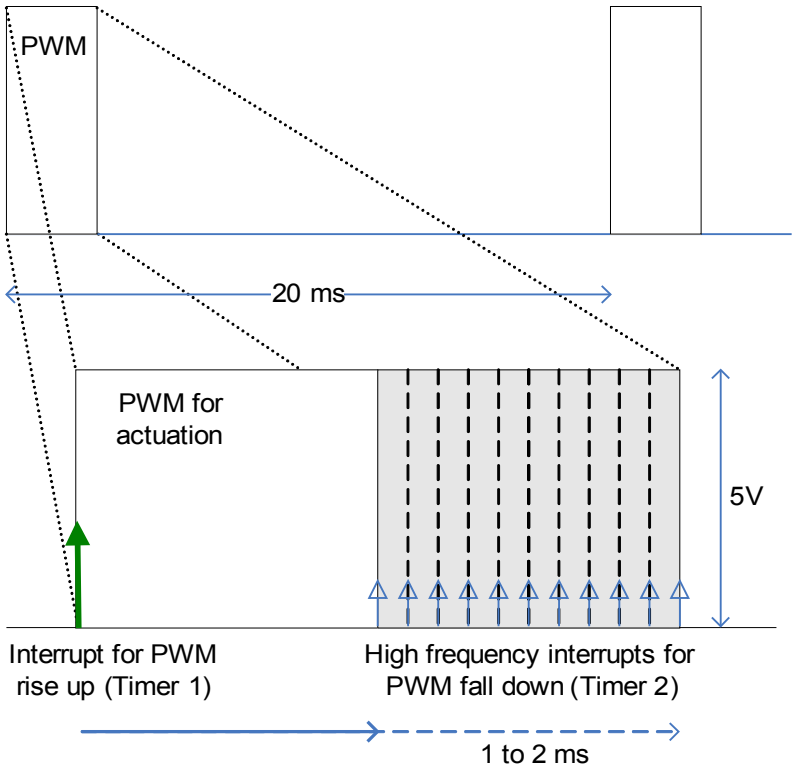


Fig. 35: Organização temporal das interrupções na geração do PWM.

A Fig. 35 demonstra bastante bem este processo. A cada servomotor é atribuído uma variável indicadora da largura de impulso a aplicar, que nada mais é o número de iterações durante as interrupções de alta frequência a manter o sinal a 1. Após a primeira interrupção um contador é utilizado para contagem das iterações que vão decorrendo entre o período de 1 a 2 ms e em cada uma delas é comparado com a variável atribuída a cada servomotor da duração do impulso. Quando o contador for igual a essa variável, o sinal de PWM correspondente é baixado a zero. Desta forma a *N* servomotores são utilizadas *N* variáveis, sendo que a única limitação é o CPU ter tempo de verificar todos os sinais de PWM e aplicar quaisquer modificações dentro do intervalo T_{hf} (5.56µs), pelo que a quantidade máxima de servos é dependente da velocidade do CPU. Para uma frequência de relógio de CPU de 10MHz a quantidade máxima de instruções *assembly* que podem ser executadas em cada iteração, é por isso:

$$N_{max} = \frac{T_{hf}}{1/f_{CPU}} = T_{hf} \cdot f_{CPU} = 5.56\,\mu s \cdot 10\text{MHz} = 55 \ \textit{instruções}$$

Esta quantidade máxima não nos dá grande liberdade para o cumprimento da *deadline* dos 5.56µs, pelo que ou aumentamos a velocidade de CPU, ou diminuímos o número de servos a controlar, ou então, na

impossibilidade de modificar estes parâmetros, aumentamos o período T_{hf} pelo aumento do passo de posição Δ_{pos}.

Para o nosso caso em concreto, pretendemos controlar três motores utilizando um CPU com 10MHz de velocidade, pelo que só possuímos, para avaliar cada servo, cerca de 18 instruções. Como para além do processamento também entram para as contas o atraso de atendimento à interrupção, mais o código extra executado até chegar à secção de código de interesse, este limite é facilmente violado. Dada a impossibilidade de aumentar ainda mais a velocidade de CPU nem de querer aumentar o período T_{hf} sob pena de perda de resolução do servo, reestruturou-se a forma de atendimento às interrupções de forma a eliminar o tempo extra resultante do tempo de atendimento à interrupção mais o código extra. Tal foi conseguido fazendo com que o atendimento às interrupções durante o período de descida do PWM, fosse feito por *polling* dentro da própria RSI, em vez do procedimento normal de saída e reentrada nesta rotina.

A Fig. 36 apresenta o algoritmo adoptado em que, na ocorrência da primeira interrupção do *timer* 2 a entrada na RSI é feita, e só volta a sair dela após o atendimento por *polling* de todas as interrupções de alta frequência até ao fim da zona de descida do PWM. Desta forma garante-se que em cada iteração, o CPU só se dedica ao bloco de código de avaliação da descida de PWM conseguindo cumprir assim a deadline dos 5.56µs imposta.

A única desvantagem desta técnica prende-se com a impossibilidade de execução simultânea de outras tarefas durante este período. No entanto tal não é problemático dado que só corresponde a 5% de tempo de cada período de PWM.

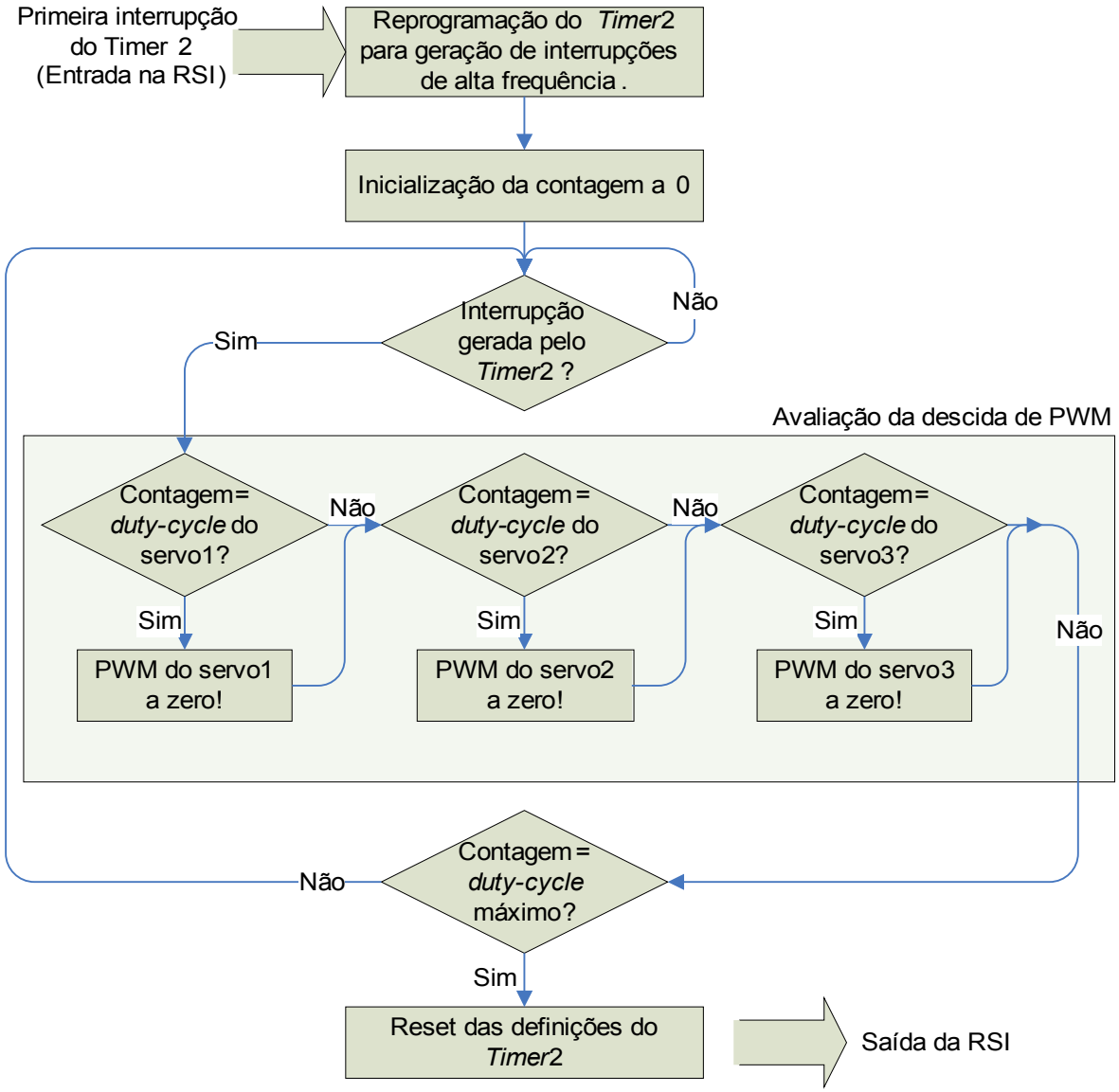


Fig. 36: Atendimento às interrupções de alta frequência.

2.1.3 – Leitura Sensorial do Servomotor

Embora, por defeito, o modelo escolhido não disponibilize externamente os sinais de posição e de velocidade para *feedback*, é possível extrair o sinal de posição dado o fácil acesso ao potenciómetro interno. Desta forma, abriram-se todos os servomotores e adicionou-se um fio extra ligado ao terminal do potenciómetro indicador da posição angular do eixo (Fig. 25). Já quanto à velocidade desconhece-se a forma de aceder a este dado. Tendo acesso à informação da posição, é possível monitorar constantemente a posição do servo a partir da unidade de controlo local e transmiti-la ao PC através da rede de comunicações, podendo avaliar continuamente a performance de cada actuador.

Deslocando o servo, na ausência de carga, verifica-se uma variação da tensão de saída entre 0.8 e 1.8 V ao longo dos 180º de excursão, podendo ser amostrado pelo PIC através da ADC. Utilizando as tensões de referência *standard* para a ADC (0 e +5V) precisamos de pelo menos 10 bits para o quantificador, tendo em conta que necessitamos de uma resolução que permita distinguir 180 posições possíveis (resolução de 1º).

$$bits_{quant} = ceil\left[\log_2(níveis\ de\ quantificação)\right] \qquad níveis\ de\ quantificação = \frac{5\text{V}}{(1.8\text{V} - 0.8\text{V})/180^\circ} = 900$$

$$bits_{quant} = ceil\left[\log_2(900)\right] = 10\ bits$$

Como a ADC do PIC oferece-nos a opção de quantificação a 10 bits tal é conveniente aproveitar.

No entanto, a medição não é tão simples como amostrar a tensão de saída quando desejado, pois na presença de cargas no eixo e/ou de velocidades elevadas surge um estranho impulso acima do nível de tensão correspondente à posição, pelo que se a medição for executada no momento do impulso o resultado será falso. Tal motivo é devido ao facto de nos modelos da HITEC, contrariamente aos da FUTABA como se verifica na Fig. 29, a tensão de referência (massa) do potenciómetro não é a mesma que para a ADC do PIC, pelo que, embora para a electrónica de controlo interna este sinal corresponderá sempre à posição do servo, para a perspectiva da ADC, este sinal possuirá oscilações na forma de impulsos ao longo do tempo, mesmo sem variar a posição do servo.

Testando diversas situações através do osciloscópio concluímos que estes impulsos possuíam propriedades não casuais:

- ➔ O impulso ocorre acima da tensão indicadora da posição;
- ➔ A amplitude do impulso é constante e apenas varia com a tensão de alimentação;
- ➔ Período de repetição coincidente com o PWM aplicado (50Hz);
- ➔ Ponto de início sincronizado com o fim do impulso de PWM;
- ➔ Largura dependente da carga e da velocidade.

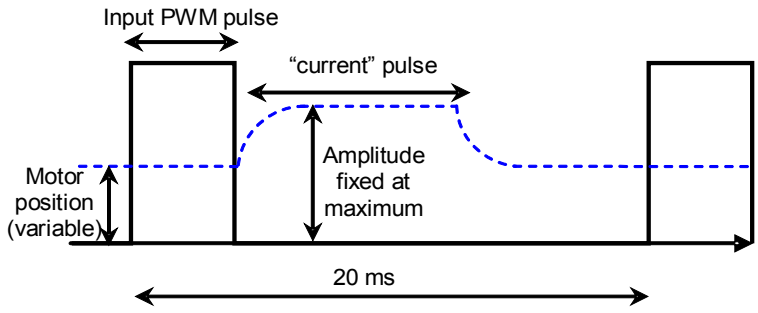


Fig. 37: Impulso de tensão medido na presença de cargas/velocidades elevadas.

O último aspecto foi o que se revelou mais interessante: quando se aumentava a carga aumentava-se a velocidade, a largura deste impulso aumentava. Tentando relacionar a largura deste impulso com a massa da carga efectuou-se o seguinte teste: fazia-se deslocar o servo para um conjunto de posições entre -90º e +90º, e para cada uma delas o actuador era deixado em repouso e media-se a largura do impulso. Desta forma, assegurávamos que a largura medida apenas era devida à gravidade (sem a interferência da velocidade). Esta experiência foi executada para duas cargas de massas diferentes, uma aproximadamente dupla da da outra.

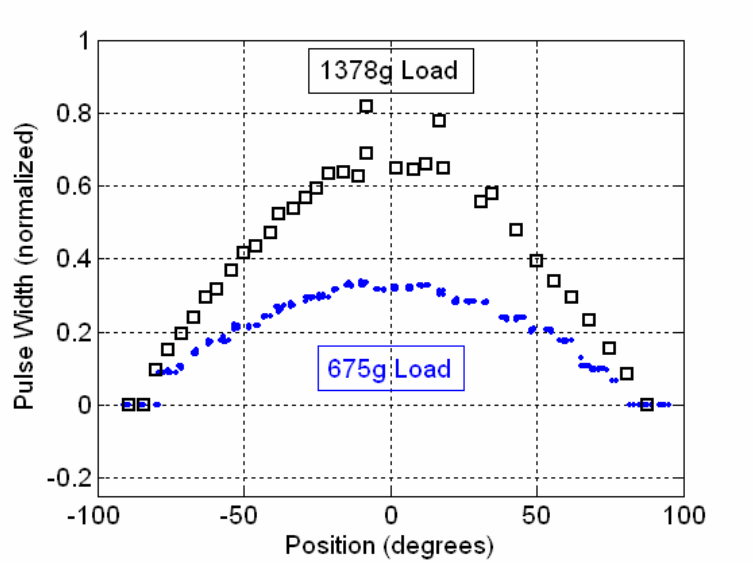


Fig. 38: Relação entre a posição e a largura de impulso.

Analisando cada carga à parte, verifica-se o comportamento sinusoidal ao longo das diversas posições sugerindo a relação com a função *seno*. Ora pela Equação 1 se conclui que a largura do impulso de tensão está directamente relacionada com o binário gravítico. Relacionando os valores entre as duas cargas, tal fundamento é reforçado dada a relação de multiplicação entre os dois conjuntos – aproximadamente de dois! Ora como se sabe a corrente está directamente relacionada com o binário...

$$I = K * \tau$$

... pelo que a largura de impulso é um indicador da corrente drenada pelo servo. Por esta razão, de agora em diante, este impulso será denominado por impulso de corrente. Este comportamento confirma a actuação sobre o motor DC de uma forma digital, também utilizando sinais no formato de PWM através da ponte H.

Note que, pelas características do sinal de posição, é possível medir tanto a posição como a corrente a partir desta única fonte, sem a necessidade de qualquer electrónica adicional para a medição de corrente!

2.1.3.1. Medição de Corrente

No ano anterior sugeriu-se um método para a medição de corrente baseado no uso de uma resistência de baixo valor em série com a alimentação do servo.

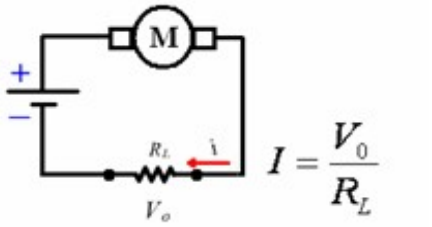


Fig. 39: Configuração possível para medição da corrente.

Embora esta forma permita uma leitura de corrente mais simplificada e de resultado sempre disponível (sem a necessidade de algoritmos de medição de largura de impulsos) acrescenta muitos mais inconvenientes do que vantagens:

- ✗ Emissão excessiva de calor nas situações de maior consumo de corrente.
- ✗ A presença da resistência provoca um aumento da corrente consumida pelo servo. Tal é explicado pelo facto de na resistência se verificar uma queda de tensão o que resulta numa diminuição da tensão de alimentação do servo, que será tanto maior, quanto maior for a corrente consumida. Com esta diminuição, o servo tem de realizar um maior esforço o que equivale a aumentar a corrente drenada.

- ✗ Dada a queda de tensão na resistência, a tensão de referência (massa) do servo varia de acordo com a corrente consumida, o que interferirá na medição de posição uma vez que a ADC não usa a mesma tensão de referência. Desta forma, além do sinal de posição do potenciómetro ser afectado de impulsos adicionais, também será afectado pelo *offset* introduzido pela queda de tensão da resistência.

O último ponto sobressai-se pelo facto de "estragar" o sinal de posição e exigir também a medição da tensão referência do servo para compensar a variação: simplifica na medição de corrente, mas dificulta na de posição.
Em alternativa, optou-se por remover estas resistências pelo seu curto-circuito nas placas *slave* e recorrendo a ferramentas baseadas em *software* medir a corrente pela determinação da largura do impulso de corrente.

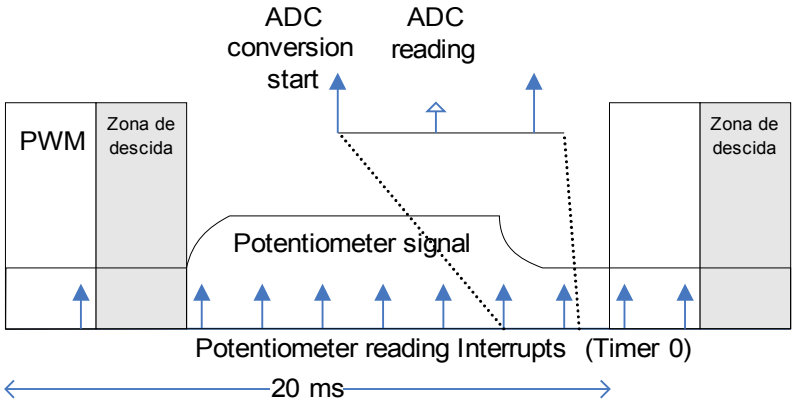


Fig. 40: Organização das interrupções (setas) na medição sensorial.

A Fig. 40 descreve bem a estratégia adoptada. Uma interrupção baseada no *timer* 0 é gerada durante todo o período de PWM com uma periodicidade de 200µs. Apenas na zona de descida do PWM ela é inibida não se pretendendo interferir com as interrupções de alta frequência. Sempre que o *timer* 0 gera uma interrupção, a tensão de saída do potenciómetro é medido e é comparado com um determinado limiar logo após o "*ADC reading*". Se estiver abaixo desse limiar considera-se a não ocorrência do impulso de corrente, mas caso esteja acima, uma variável contadora da largura de impulso é incrementada. No fim do período de PWM (considerado logo após a zona de descida de PWM) esta variável contadora indicará a largura do impulso medido no período de PWM anterior. De notar que a resolução de medição da largura de impulso é tanto maior quanto menor for a periodicidades das interrupções. No entanto não convém definir períodos muito curtos sob pena de não atribuir suficiente largura de banda de CPU para o programa principal ser executado completamente num período de PWM. A periodicidade de 200µs foi o melhor valor encontrado e permite executar no total 95 medições, ou 31 para cada servo, ao longo de um período de PWM excluindo a zona de descida de PWM, ou seja, ao longo de 19 ms.

O valor do limiar a considerar depende da amplitude do impulso, mas sabendo que este apenas varia com a tensão de alimentação, mantendo uma amplitude sempre superior a 1.0V utilizando baterias de 7.4V, definiu-se o limiar como 1.0V. Note que este limiar encontra-se acima da tensão DC pelo que é necessário em cada período de PWM medir a tensão mínima e utilizá-la para o período seguinte como sendo a tensão base dos impulsos.

Contudo este método possui uma desvantagem. Em situações de elevada exigência, em que a corrente consumida é próxima da máxima, a largura do impulso de corrente pode ocupar praticamente todo período de PWM inibindo a capacidade de leitura da posição e da largura do impulso, dado que o valor mínimo corresponde ao topo do impulso de corrente. É, por isso, de evitar que esta situação se alcance, quer impondo limites físicos, quer pela adopção de estratégias de controlo que minimizem a corrente a consumir – uma delas é a limitação de velocidade.

2.1.3.2. Medição da Posição

Já foi referido anteriormente que a componente DC da saída do potenciómetro está relacionada à posição do servo, pelo que a tarefa de guardar o valor mínimo deste sinal, executada para efeitos de detecção do impulso de corrente, é aproveitada para o cálculo da posição angular do servo.
Convencionando a tensão mínima como correspondendo à posição +90º e a tensão máxima como -90º, recorrendo a uma relação linear podemos determinar a posição com ADC_{res} igual ao resultado da ADC, m sendo a relação entre a variação de posição e a variação de ADC_{res} (declive) e b o valor referência quando ADC_{res} é nulo (ordenada na origem).

$$\boxed{pos = b - ADC_{res} * m} \qquad \text{(Equação 2)}$$

$$m = \frac{+90º - (-90º)}{ADC(1.8\text{V}) - ADC(-1.8\text{V})} = \frac{180º}{368 - 163} = 0.878$$

$$b = pos + m * ADC_{res} = 90º + m * ADC(0.8\text{V}) = 90º + 0.878 * 163 = 233$$

...com $ADC(x)$ sendo o valor convertido pela ADC correspondente à tensão x (V):

$$ADC(x) = \frac{x}{5\text{V}} * (2^{bits_{quant}} - 1) = x * \frac{2^{10} - 1}{5\text{V}} = x * \frac{1023}{5\text{V}}$$

Logo $\boxed{pos = 233 - ADC_{res} * 0.878}$

Esta conversão é implementada no PIC através da macro:

```
#define POS(volt,...)   (origin - (volt)*SLOPE/QUOCIENT)
```

... em que *origin* corresponde à ordenada na origem *b*, *SLOPE/QUOCIENT* é o declive *m*, e *volt* é o resultado da conversão da ADC (ADC_{res}). Note que o declive *m* é inferior à unidade daí a necessidade do formato *SLOPE/QUOCIENT* para poder utilizar apenas valores inteiros (*SLOPE<QUOCIENT*).

Contudo, não é possível garantir que a gama da tensão de saída esteja compreendida entre 0.8 e 1.8V para qualquer servomotor, podendo-se verificar variações até cerca de 0.2V. Tal é problemático para o cálculo da posição angular, pois estamos a considerar uma relação de correspondência constante quando tal não acontece entre dois actuadores. No entanto, note que esta diferença é mais notória para o parâmetro b do que para o m. O valor de *b* exige que os extremos de posição possuam exactamente os valores enunciados, mas para o valor de *m* só é exigido uma excursão entre extremos de 1.0V o que já é mais frequente acontecer. Este pequeno detalhe permite-nos assim a utilização de uma rotina de calibração simples de modo a acertar o parâmetro b de acordo com o servo a lidar.
Considerando o parâmetro m constante, o que é uma aproximação aceitável, sempre que o sistema é ligado, se assegurarmos que cada servo está numa posição conhecida à priori, é fácil calcular o valor de b através da Equação 2 e utilizá-lo nas medições consequentes.

A rotina de calibração é executada sempre que o sistema arranca e segue o seguinte algoritmo:

1. A primeira mensagem de actuação já chegou? Só passar para o passo 2, quando afirmativo;
2. Actuação sobre os servomotores de forma a cumprir a posição solicitada pela unidade *master*;
3. Esperar dois segundos para a cumprimento do movimento e estabilização do sinal de posição;
4. Amostragem de 25 medidas da saída do servo (25 períodos de PWM de duração);
5. Cálculo da média aritmética do valor medido;
6. A partir da seguinte equação (baseada na Equação 2) determinar o valor de *b* a partir da posição solicitada pelo master e do valor médio ADC_{res}:
   $b = pos_{master} + ADC_{media} * m$
7. Activação dos filtros de medição sensorial.

Desta forma, a medição da posição é adaptado a cada servomotor de uma forma personalizada com mínimos erros de cálculo (apenas dependem de *m*).

2.1.3.3. Organização das interrupções de medição sensorial

A medição de tensões analógicas a partir de uma ADC, como se sabe, não é imediata, demorando um determinado intervalo de tempo até o resultado estar pronto. Para se proceder à amostragem é necessário percorrer um conjunto de passos:

1. Selecção do entrada do multiplexer correspondente ao servo a ler;
2. Esperar que o multiplexer efectivamente seleccione a entrada e tenha uma saída estável (alguns microsegundos) – tempo de estabilização;
3. Esperar cerca de 20µs para a ADC possuir um valor estável na sua entrada – tempo de aquisição;
4. Iniciar conversão da ADC;
5. Esperar que a ADC termine a conversão (cerca de 40µs) – tempo de conversão;
6. **Leitura do valor convertido e processamento (cerca de 10µs);**

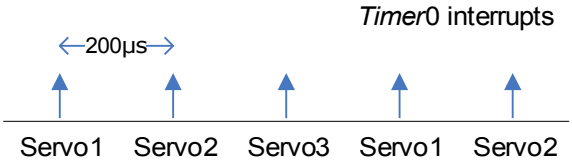


Fig. 42: Multiplexagem na leitura dos servos.

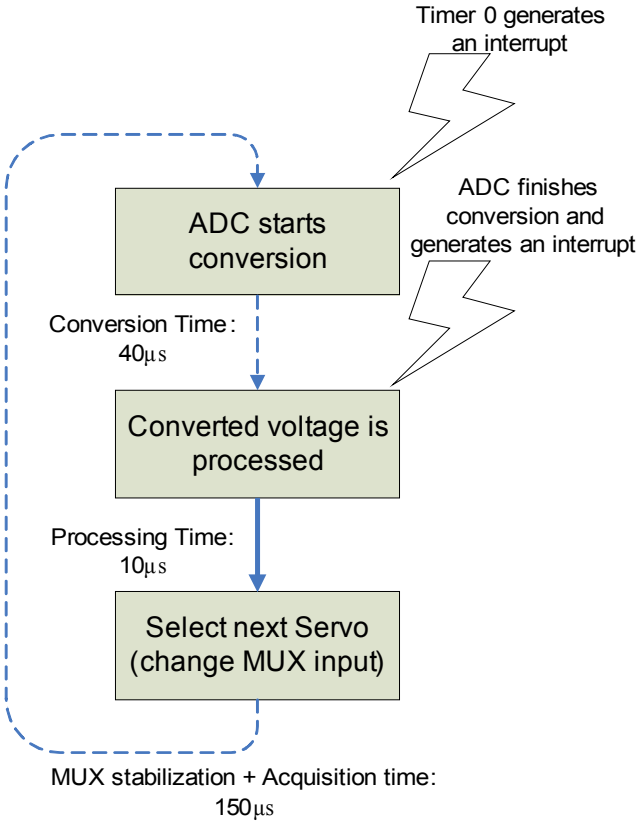


Fig. 41: Algoritmo de leitura dos três servomotores.

Só depois de executados os primeiros 5 passos, a tensão de saída do servo pode ser medida com segurança. Somando estes períodos de tempo, incluindo o tempo de execução do código de processamento do resultado, verificamos que todo o processo é executado em menos de 80µs, pelo que restam 120µs para execução de outras tarefas iniciadas pela função *main*.
No entanto, não nos esqueçamos que temos aplicar este processo a três actuadores. Para não sobrecarregar o CPU de repetir este procedimento três vezes em cada período de 200µs optou-se por efectuar multiplexagem na leitura, ou seja, em cada período de 200µs apenas um servo é lido, mas no período seguinte o servo a ler é outro, completando-se a leitura dos três servos ao fim de três períodos (600µs). Logo após a leitura do terceiro, volta ao primeiro repetindo de novo todo o processo (Fig. 42).

A Fig. 41 apresenta o algoritmo implementado, com uma gestão temporal mais eficiente, seleccionando a entrada do *multiplexer* correspondente ao servo seguinte no fim do código que processa as interrupções provenientes do *timer* 0 (RSI), e não no início, deixando livres 150µs entre o fim da execução do código e o atendimento da próxima interrupção para estabilização da saída do *multiplexer* e o cumprimento do tempo de aquisição da ADC. Desta forma, quando o *timer* 0 voltar a gerar uma interrupção poder-se-á arrancar com a ADC de imediato apenas tendo de esperar por uma interrupção por parte da ADC indicando a prontidão do valor convertido. Em termos de tempo efectivo envolvido na leitura apenas temos os 10µs do código de processamento do valor convertido, uma vez que sempre que é necessário introduzir um tempo de espera o CPU não fica bloqueado em modo de espera, mas retorna ao programa normal deixando encarregues às interrupções a tarefa de voltar prosseguir a sequência de operações.

Um outro aspecto a salientar é a coabitação entre as interrupções de medição sensorial e as interrupções para actuação (Fig. 35 e Fig. 40). Dada a igualdade, em termos de prioridade entre os dois tipos, um mecanismo que evite a interferência entre os dois torna-se fundamental para evitar atrasos que comprometam a fiabilidade de funcionamento destes dispositivos. Um atraso na interrupção para a gestão da actuação

poderia resultar num sinal de PWM modificado que alteraria sem intenção a posição do servo, ou um atraso nas interrupções envolvidas na medição poderia comprometer a validade dos resultados. Para evitar problemas deste género impuseram-se as seguintes condições:

- As interrupções gestoras do sinal de PWM nunca devem sofrer interferência por parte de outras e devem ser sempre cumpridas no instante a que foram programadas;
- As interrupções gestoras da medição sensorial nunca podem ser interrompidas a meio da sua execução, ou seja, quando a primeira operação é executada – arranque da ADC – é fundamental a sua execução sem atrasos até ao fim. Deste modo o processo de leitura sensorial deve ser considerado como uma operação atómica. Contudo este processo, como um todo, pode ser inibido de modo a não interferir com as interrupções gestoras do PWM.

O mecanismo de coabitação deve então seguir este procedimento: em cada interrupção proveniente do *timer* 0 (medição sensorial) ainda antes de se proceder à primeira operação – o arranque da ADC – deve-se verificar se há tempo para a execução completa de todo o processo de medição ainda antes da próxima interrupção gestora do PWM:

- ➔ Antes da interrupção do *timer* 1 responsável por elevar o sinal de PWM a 1;
- ➔ Antes da primeira interrupção do *timer* 2 que inicia a zona de descida do PWM a 0.

O intervalo de tempo mínimo considerado, para o processo de medição sensorial estar autorizado a iniciar deve corresponder à periodicidade do *timer* 0, ou seja, 200µs. Se para a próxima interrupção de actuação faltar mais de 200µs a medição sensorial é autorizada a arrancar, pois é garantido que finaliza antes dela chegar. Quando a zona de descida de PWM começa, as interrupções de medição sensorial devem ser inibidas de modo a poder dedicar toda a largura de banda do CPU para as interrupções de alta frequência. Quando esta zona finaliza os dados sensoriais amostrados durante o período de PWM anterior são tratados e estas interrupções são reactivadas.

Quanto ao processamento do sinal de saída do servo, a Fig. 44 descreve o algoritmo utilizado para cada servo em cada iteração ao longo de um período de PWM (caixa "*converted voltage is processed*" da Fig. 41).

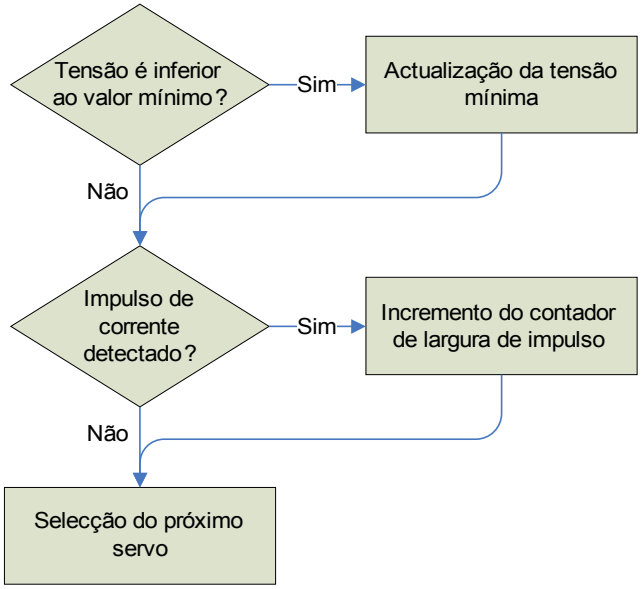


Fig. 44: Algoritmo de processamento da tensão medida para cada servomotor.

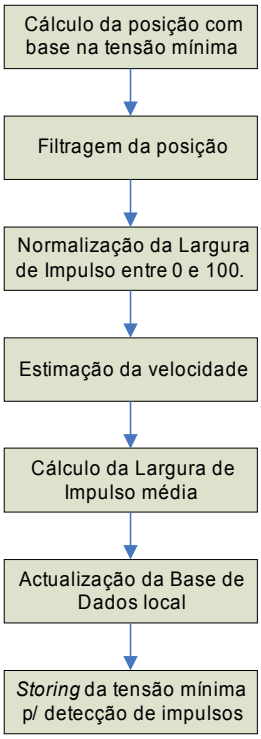


Fig. 43: Processamento final (fim do período de PWM).

No fim de cada período de PWM, os valores da tensão mínima e da largura de impulso são processados em cada servo para determinação da posição, velocidade média e corrente consumida (Fig. 43):

1. Cálculo da posição com base na tensão mínima: A posição correspondente ao valor mínimo da ADC é calculado a partir da relação linear...

$$pos = b - ADC_{min} * m$$

2. Filtragem da posição: O valor da posição obtido é filtrado de modo a evitar variações bruscas. São utilizados dois filtros:

   a) Filtro de média não linear: este filtro, muito usado em processamento de imagem, é usado apenas para remover picos nos valores medidos, sem interferir de de nenhuma forma no sinal na ausência deles. O atraso introduzido é apenas de uma amostra.

   y → amostra obtida
   y_max → máximo valor da amostra admitido
   y_min → mínimo valor da amostra admitido
   y_new → Valor filtrado
   y_prev → Valor da amostra anterior (não filtrado)

```
// Limitação da saída
if (y>y_max)        y_new=y_max;
else if (y<y_min)   y_new=y_min;
else                y_new=y;

// Actualização das amplitudes limite
if (y>y_prev)  {
    y_max=y;
    y_min=y_prev;
}
else  {
    y_max=y_prev;
    y_min=y;
}

// Actualização da saída anterior
y_prev=y;
```

   b) Filtro de média linear: este filtro é aplicado a seguir ao não linear e tem como função a suavização do sinal resultante. Apenas é feita uma média igualitária entre a nova amostra e a anterior.

   ynew_prev → Valor filtrado anterior

```
// Filtro Passa-baixo
y_new = (y_new+ynew_prev)/2;
ynew_prev = y_new;
```

3. Normalização da largura do impulso de corrente: De modo a permitir a redefinição da periodicidade do timer 0 ($timer0_{per}$) sem interferir no resultado da largura do impulso de corrente, o resultado é normalizado para a gama entre 0 e 100 (percentagem) através de uma regra proporcional:

$$Largura_{norm} = Largura * \frac{100}{19\text{ms} / (timer0_{per} * 3\, servos)}$$

4. Estimação da velocidade: Embora não seja possível medir directamente a velocidade, é possível estimá-la a partir da variação de posição. Definindo a velocidade mínima mensurável como sendo 10º/s precisamos de medir a variação de posição correspondente a 100ms uma vez que o valor mais pequeno que conseguimos medir é 1º. No entanto, é desejável que tenhamos sempre a velocidade disponível todos os 20 ms sem ter que esperar 100 ms para poder ler este dado. Tal é possível usando arrays circulares (Fig. 45):

   Um array de 5 posições pode armazenar até 5 posições antigas – *pos*(*n*-1) até *pos*(*n*-5) – em que a posição mais antiga corresponde à posição de há 100 ms atrás (20ms*5). Se em cada período de

PWM armazenarmos a posição acabada de medir no elemento mais antigo, em todos os 20 ms temos disponível o valor de posição de há 100ms atrás, ainda antes da actualização do array com o novo valor. Desta forma, a cada 20ms, podemos calcular a variação de posição verificado nos últimos 100 ms.

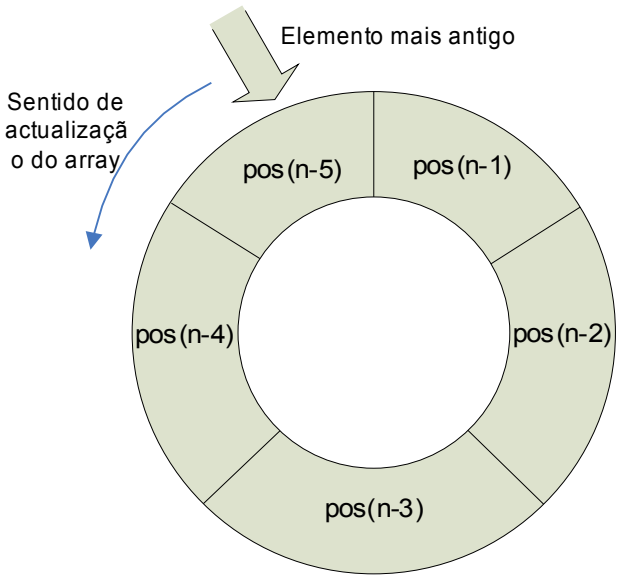


Fig. 45: Array circular para armazenamento de posições.

Este array tem o nome de circular, pois o elemento mais antigo, a que chamaremos de *index*, está sempre em contínua rotação ao longo do array: quando introduzimos um valor no elemento mais antigo, esse elemento passa a ser o mais novo e o elemento a seguir passa a ser o mais antigo.

a) Estimação da velocidade: $vel = pos(n) - pos(n-5) = pos_{medido} - array[index]$
b) Actualização do array: $array[index] = pos_{medido}$
c) Actualização do elemento mais antigo $index = remain[(index+1),5]$

5. Cálculo da largura de impulso média: Dada a instabilidade do valor de corrente medido (largura de impulso) implementou-se o cálculo da média de todos os valores de corrente medidos nos últimos 100ms. De modo a ter sempre um valor actualizado todos os 20ms seguiu-se a mesma estratégia que para a estimação de velocidade: um array circular para armazenamento dos cinco valores de corrente mais recentes. A cada 20ms é introduzido o novo valor medido e é calculada a média do conjunto considerando esse valor como a final.

$$Corrente_{media} = \frac{\sum_{index=0}^{4} array[index]}{5}$$

6. Actualização da base de dados local (sensorial): Ver secção 1.2.4.1 (base de dados local).

7. Armazenamento da tensão mínimo medida: necessário para o conhecimento da tensão base dos impulsos de corrente para o período de PWM seguinte.

2.1.4. Organização do Software de controlo básico do servo

A Fig. 46 visualiza a estrutura do software das unidades de controlo local, já apresentada no capítulo anterior sobre comunicações (Fig. 20).

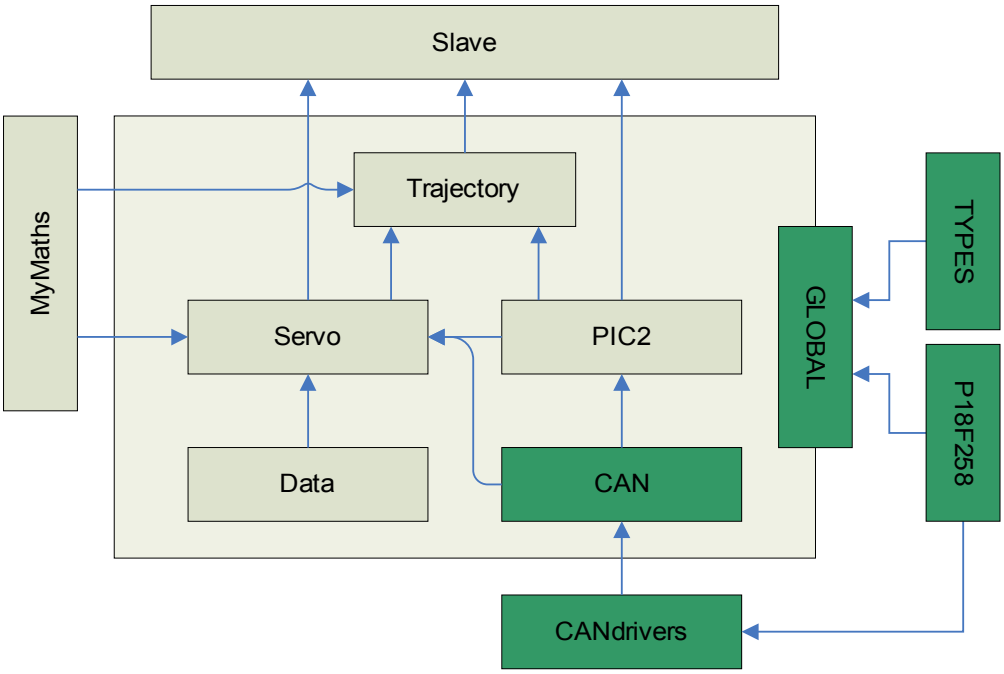


Fig. 46: Relações de inclusão dos módulos de software de cada Slave.

Este capítulo apenas aprofundou as tarefas de controlo de actuação e sensorial dos servomotores, que se encontram no módulo PIC2.

Módulo PIC2

Tabela 29: Funções de acesso externo do módulo PIC2.

Função	***Descrição***
initPic	Inicializações relativas às configurações dos periféricos do microcontrolador.
wait	Função bloqueante que gera um atraso de *n* ms (*n* passado como argumento). Esta função faz uso dos *timers* relativos à actuação.
waitTick	Função bloqueante que espera pelo período de PWM seguinte (20ms no pior dos casos).
servoActuation	Actuação directa sobre a posição dos servomotores. As variáveis com a informação do número de iterações a manter o sinal de PWM a 1 durante a zona de descida são actualizadas para as posições solicitadas.
statusPWM	Activação/desactivação dos sinais de PWM à saída do PIC.
statusFilter	Activação/desactivação dos filtros aplicados à posição medida.
limitPosition	Limitação do um valor entre os extremos de posição do servo: -90 e +90º.

A função que mais se destaca desta lista, é sem dúvida, a *servoActuation* pois é ela que define qual deve ser ao duty-cycle do sinal de PWM. É esta função que inicializa as variáveis descritas na secção 2.1.2 com a duração do impulso de PWM que posteriormente serão comparadas com um contador durante a zona de descida de PWM.

Tabela 30: Funções internas do módulo PIC2.

Função	*Descrição*
initLocal	Inicialização das estruturas de dados locais ao módulo.
sampleExtraSensors	Leitura dos sensores adicionais (coberto pelo terceiro capítulo).
updateServoMeasures	Medição iterativa do sinal de saída do servo ao longo de um período de PWM, para cálculo da tensão mínima e da largura do impulso de corrente (Fig. 44).
finalizeServoMeasures	Finalização do processamento sensorial (executado no fim do período de PWM). A posição angular do servo (em graus) e a corrente consumida normalizada entre 0 e 100 são determinados (Fig. 43).
filter	Filtragem da posição medida usando métodos lineares e não-lineares (final da secção 2.1.3.3).
delay	Geração de um atraso recorrendo somente à instrução *nop* (*no operation*).
highISR	Rotina de serviço à interrupção de alta prioridade. Nesta rotina é feita gestão da actuação e do processamento sensorial dos servomotores.
lowISR	Rotina de serviço à interrupção de baixa prioridade. Nesta rotina é feita a gestão das comunicações CAN da unidade *slave* (1º capítulo).

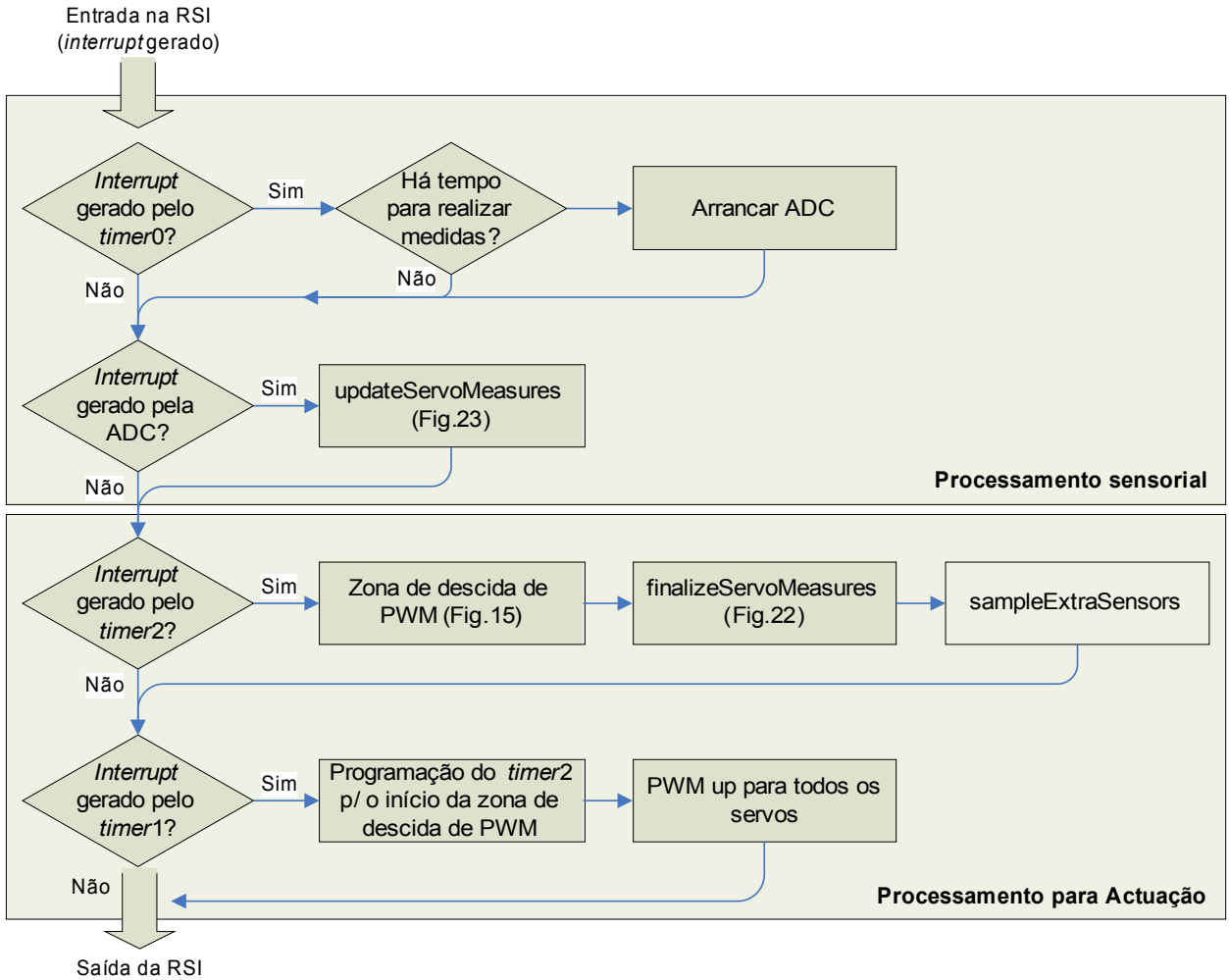


Fig. 47: Algoritmo da RSI de alta prioridade.

2.2 – ESTUDO DO SERVOMOTOR EM MALHA ABERTA

Agora que os detalhes de funcionamento do microcontrolador foram apresentados, vamos agora estudar o comportamento do servomotor sob determinadas condições de modo a avaliar a sua performance. O estudo será feito do ponto de vista de um sistema no qual aplicaremos uma entrada e queremos saber qual é a sua resposta (Fig. 48).

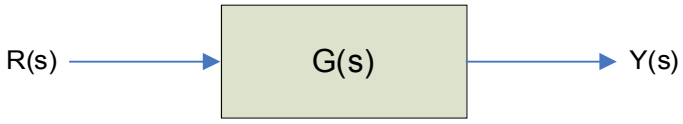


Fig. 48: Representação do servomotor por uma função de transferência G(s).

Ora, tendo sido discutido o funcionamento interno do servomotor na secção 2.1 (Fig. 29), infelizmente apenas era uma aproximação ao nosso modelo e além disso não conhecemos os valores das grandezas enunciadas. Por tudo isto iremos estudar o comportamento do sistema G(s), desconhecendo a sua função de transferência.

Embora, pudéssemos tentar encontrar esta expressão experimentalmente através de métodos bastante conhecidos como é o caso das regras de Ziegler-Nichols, tal não nos leva muito longe, pois como é sabido, as juntas da plataforma humanóide estão sujeitas a variações de inércia pelo que o comportamento dos servos torna-se bastante não linear. Teríamos, por isso, uma função de transferência diferente para cada carga aplicada.

Este capítulo tem como objectivo estudar primeiramente a resposta em malha aberta do servo, ou seja, aplicando um sinal (ou uma sequência de sinais) de PWM na entrada r(t), analisar a resposta pela leitura da posição angular y(t). Rotinas em MatLab para uso pela unidade principal foram especialmente construídas para este efeito enviando para o *slave* respectivo uma ordem de actuação, e logo de seguida monitorar a posição do servo o tempo suficiente para permitir captar informação informação importante.

Algum vocabulário não muito comum será utilizado pelo que convém, antes de tudo, esclarecer o seu significado para evitar qualquer dúvida ou ambiguidade:

- ➔ Tempo de crescimento/subida: tempo que a resposta demora a crescer entre 10% e 90% da distância total a percorrer;
- ➔ Tempo de atraso: instante que a resposta atinge 50% da distância total percorrida;
- ➔ Tempo de pico: instante em que a resposta passa pela posição máxima/mínima (depende do sentido de deslocamento);
- ➔ Tempo de estabelecimento: tempo necessário para que a resposta entre, sem voltar a sair , numa determinada vizinhança , previamente especificada , do valor final da resposta. Dois a cinco por cento é normalmente a margem especificada.
- ➔ Overshoot: oscilação verificada em torno do valor final no fim da resposta. O seu valor corresponde normalmente à relação entre a margem máxima de oscilação e a distância total percorrida.
- ➔ Erro em regime estacionário: diferença entre o valor final e o valor desejado após estabilização da resposta.

Estas definições correspondem a características da curva de resposta do sistema a testar e normalmente são usados para avaliar a sua performance (Fig. 49). No caso de um actuador ideal, todos estas características deviam ser nulas, mas infelizmente tal não existe na realidade: os sistemas levam tempo a reagir e a atingir o seu valor final, e por vezes podem entrar em oscilação (*overshoot*) quando a entrada é demasiado exigente. Este capítulo procurará perceber o quanto os servomotores se desviam da resposta ideal, e posteriormente tentaremos encontrar soluções para a sua melhoria.

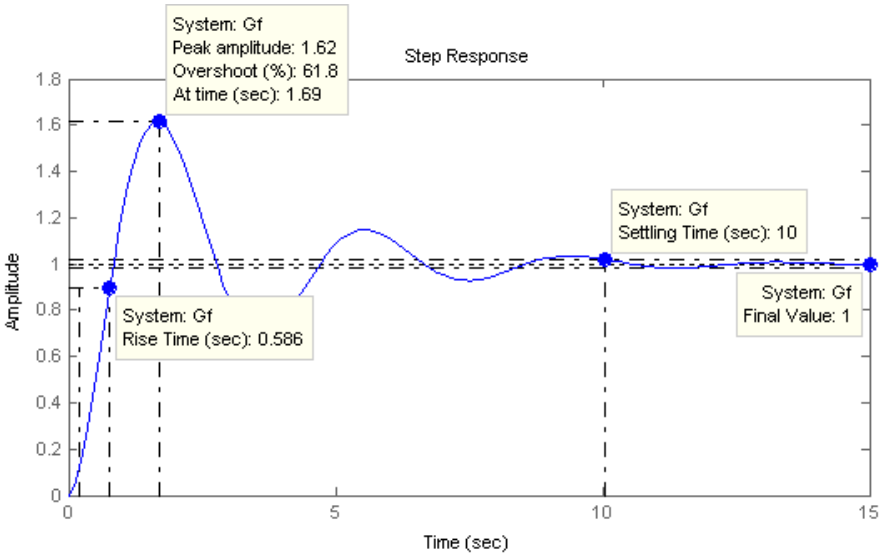


Fig. 49: Exemplo da resposta de um sistema (Gf) com a indicação das suas características.

2.2.1. Resposta ao Degrau em Malha Aberta

Com uma carga aplicada no eixo, segundo a forma apresentada na Fig. 32, inicializou-se o servo na posição inicial de -45º e enviou-se um comando de actuação para a posição de +45º – um degrau é aplicado. Monitorizou-se o percurso percorrido através do comando de leitura de posição durante 1.5s para duas cargas diferentes: 258 e 1138g (Fig. 50).

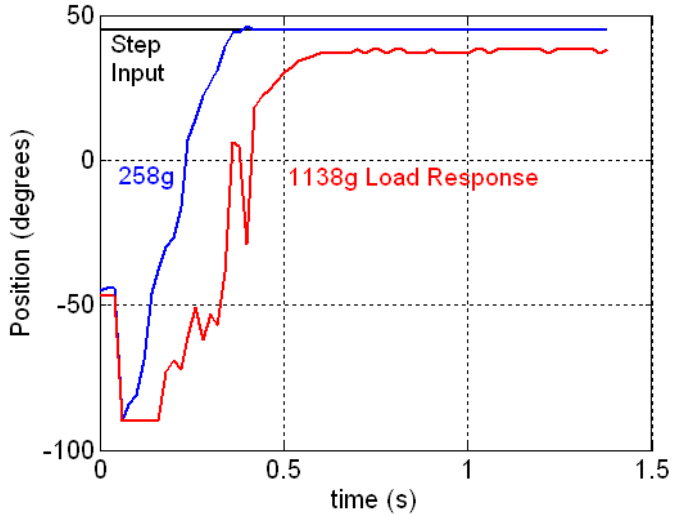


Fig. 50: Comparação das respostas ao degrau para duas cargas no percurso de -45 para +45º.

Requested position º	measured position º	Error º	Torque (Nm)
-80	-80	0	0.198
-60	-62	2	0.569
-40	-45	5	0.872
-20	-28	8	1.069
0	-9	9	1.138
+20	+11	9	1.069
+40	+33	7	0.872
+60	+55	5	0.569
+80	+80	0	0.197

Tabela 31: Erros em regime estacionário em diferentes posições para uma carga de 1138g.

Comparando as respostas da carga de 258g com a de 1138g, constata-se uma diferença dos tempos de subida evidenciando um atraso maior para cargas elevadas, o que é compreensível uma vez que o esforço dispendido é maior. Um segundo aspecto é a diferença no erro em regime estacionário: para a massa leve o erro é praticamente nulo, mas para a mais pesada já é mensurável um erro de cerca de 9º.

Adicionalmente, fez-se uma experiência tendo em vista estudar a relação do erro em regime estacionário com o binário resultante da força gravítica. Deslocando o servo para um conjunto de posições conhecidas, para cada uma delas, esperou-se pela finalização do movimento e pela estabilização do sinal de posição, anotando de seguida a posição medida pelo microcontrolador (Tabela 31). Comparando a posição solicitada com a efectiva observa-se que o erro aumenta à medida que a posição se aproxima do ponto 0º, o que demonstra que quanto maior é o binário gravítico maior é a dificuldade em atingir a posição final resultando

num erro em regime estacionário não nulo.

Outro aspecto a salientar é a visível instabilidade durante a realização do trajecto o que é mais notório para a carga elevada do que para a baixa. Além disso observa-se um salto no início da trajectória para posições inferiores a -45º seguidamente com inversão de velocidade em direcção ao valor final. No entanto este comportamento não foi o que aconteceu realmente, tendo-se observado um movimento rápido e sem oscilações durante todo o percurso. Tal sugere que a posição lida a partir do potenciómetro está a ser perturbada por algo.

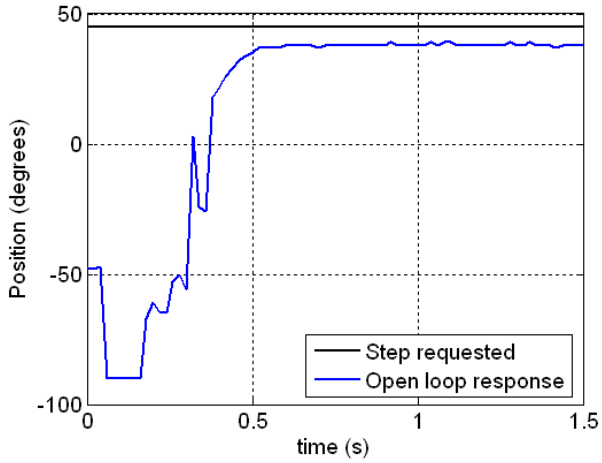


Fig. 51: Resposta ao degrau de -45 para +45º com uma carga de 924g.

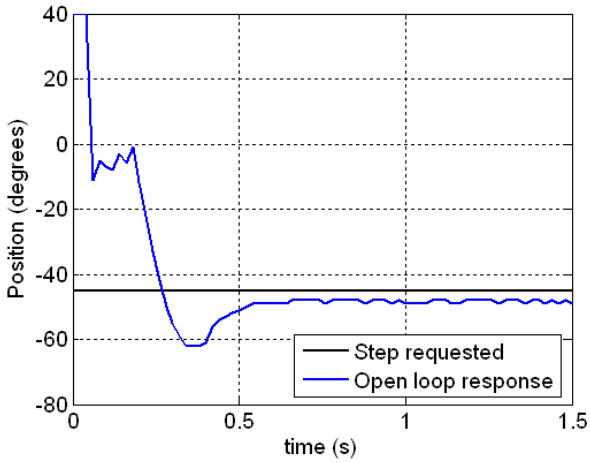


Fig. 52: Resposta ao degrau de +45 para -45º com uma carga de 924g.

Os gráficos da Fig. 51 e Fig. 52 apresentam a execução do mesmo percurso mas nos dois sentidos para uma carga de 924g. Como se pode observar, continuam a surgir os picos de corrente no trajecto inicial em ambos os casos. No caso da descida de +45 para -45º verifica-se um abaixamento brusco na posição medida até metade do trajecto onde ocorre uma recuperação.

A presença de acelerações bruscas, como acontece no arranque da trajectória, pode provocar picos de corrente, que em termos sensoriais, correspondem a um impulsos de corrente que podem ocupar toda o período de PWM. Nestas circunstâncias a posição considerada como sendo a tensão mínima deixa de poser ser medida sofrendo um aumento em tensão para o topo do impulso, que, em termos de posição angular, corresponde a uma descida brusca, tal como os gráficos nos mostram.
Neste sentido, pode-se dizer que a presença do impulso de corrente está a mascarar as medições de posição do servo agravando os resultados na presença de cargas elevadas. No entanto, tal deixa de se verificar nos últimos instantes ainda antes da finalização da trajectória e durante a fase estacionária, demonstrando que não só a massa da carga aumenta as exigências de corrente, como também a velocidade e as acelerações bruscas.

2.2.2. Controlo de Velocidade

Como se pode concluir, os servomotores são muito sensíveis a velocidades e acelerações bruscas provocando muito facilmente picos de corrente que impedem a correcta leitura da sua posição. De modo a prevenir este efeito, tentou-se introduzir algum controlo de velocidade de modo a eliminar estas variações bruscas, com a adicionante de ganharmos na capacidade de poder regular a velocidade segundo as nossas necessidades e de possibilitar movimentos mais suaves. No entanto, como não temos possibilidade de fazer o controlo directo de velocidade, utilizar-se-á o controlo de posição para executar trajectórias que no seu todo definem uma velocidade média que pode ser configurável.

Até agora temos vindo a aplicar degraus de posição aos servos tal como exemplifica a Fig. 53. Se aplicarmos uma sucessão de degraus de variação de amplitude e intervalo de tempo o mais pequenos possíveis, cuja amplitude final de cada degrau aumenta proporcionalmente até atingir a posição desejada, temos a aplicação de uma rampa de posições cujos extremos de posição e duração total definem a velocidade média do

movimento. A Fig. 54 apresenta um exemplo de uma trajectória em rampa desde -45º até +45º com uma duração de 1.8s, o que corresponde a uma velocidade média de 50º/s.

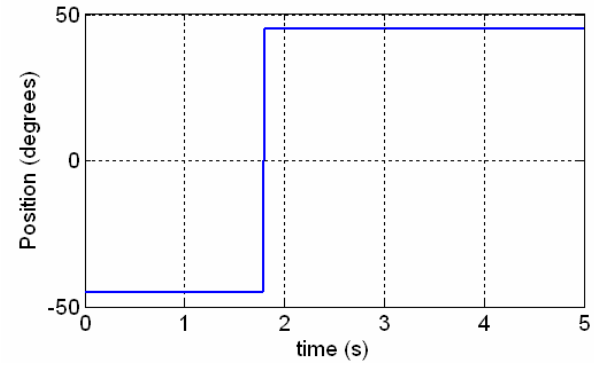


Fig. 53: Aplicação de um degrau de -45º para +45º no instante t=1,8s.

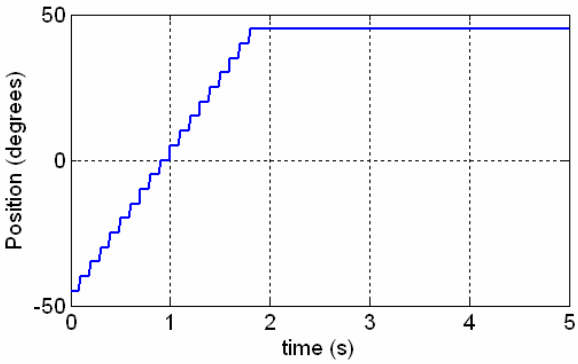


Fig. 54: Aplicação de uma rampa de posição de velocidade média 50º/s.

Enquanto que a aplicação de um degrau introduz um delta de Dirac na velocidade e na aceleração provocando facilmente picos de corrente, a trajectória em forma de rampa contêm essas variações bruscas pela aplicação sucessiva de pequenos degraus o que limita a velocidade do servo e também a sua necessidade de consumo de corrente. Esta trajectória pode ser implementada através da Equação 3.

$$pos = pos_0 + v_{med} \cdot t = pos_0 + v_{med} \cdot n \cdot Ta \qquad \text{com } T_a\text{=20ms (período de PWM).} \qquad \text{(Equação 3)}$$
$$vel = v_{med} \qquad acel = 0$$

No entanto, se quisermos optimizar ainda mais o consumo de corrente, podemos implementar trajectórias, de modo a limitar os deltas de Dirac de aceleração que continuam a verificar-se no caso da rampa. Se além de variarmos a posição, também variarmos a velocidade de modo a ser nula no início e no fim da trajectória, a necessidade de consumo de corrente decai ainda mais. Tal é exequível através de uma equação polinomial de terceira ordem que introduz velocidade zero no início e no fim de cada trajectória.

$$pos = c_0 + c_1 \cdot t + c_2 \cdot t^2 + c_3 \cdot t^3 \qquad vel = c_1 + 2 \cdot c_2 \cdot t + 3 \cdot c_3 \cdot t^2 \qquad acel = 2 \cdot c_2 + 6 \cdot c_3 \cdot t$$

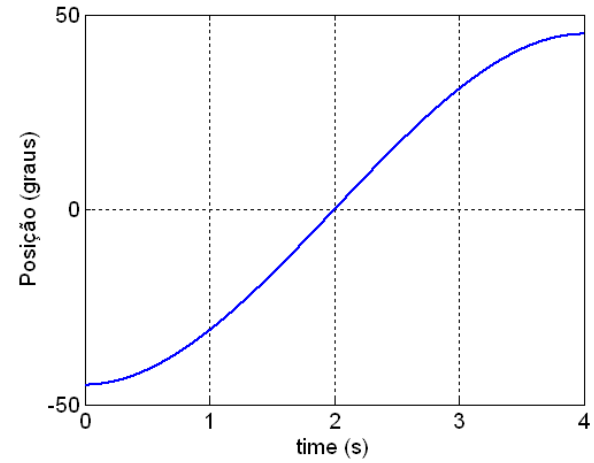


Fig. 55: Trajectória polinomial de terceira ordem.

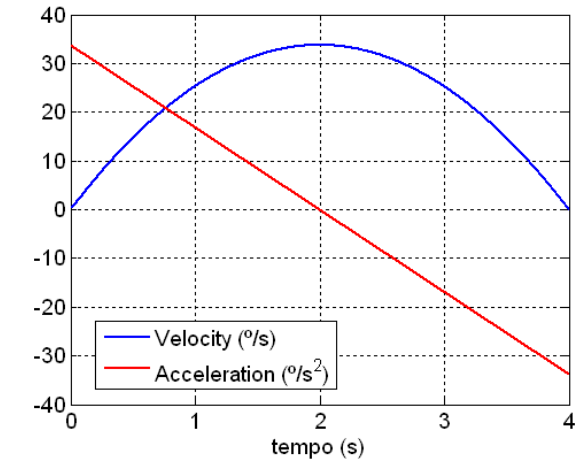


Fig. 56: Comportamento da velocidade e da aceleração na trajectória polinomial.

Como se pode verificar, deixa de se verificar deltas de Dirac até à segunda derivação da posição (aceleração). Se quiséssemos ir ainda mais longe, podíamos aumentar a ordem do polinómio de modo a garantir aceleração nula no início e no fim. No entanto tal não foi implementado dado ao facto de que quanto maior é a ordem do polinómio maior é a velocidade instantânea a meio do trajecto. Ordem três corresponde a um bom compromisso para o que é preciso.

2.2.2.1. Respostas em malha aberta

Voltando a amostrar a resposta dos servos em malha aberta, agora com a implementação de trajectórias, podemos observar, para o caso da rampa (Fig. 57 e Fig. 58), a estabilidade acrescida nas respostas qualquer que seja a carga aplicada. Os efeitos de picos de posição e as oscilações durante o percurso praticamente desapareceram demonstrando o consumo controlado de corrente com esta solução.

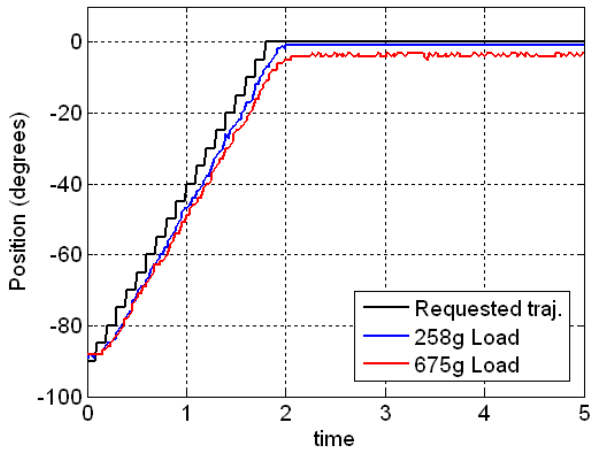


Fig. 57: Resposta à rampa com duas cargas diferentes (Δp=5º, Δt=100ms).

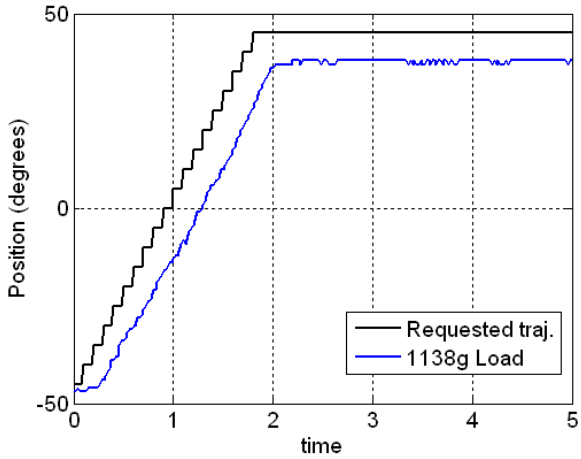


Fig. 58: Resposta à rampa com uma carga pesada (Δp=5º, Δt=100ms).

A Fig. 59 apresenta as respostas correspondentes às trajectórias polinomiais com a observação dos mesmos resultados que com a rampa.

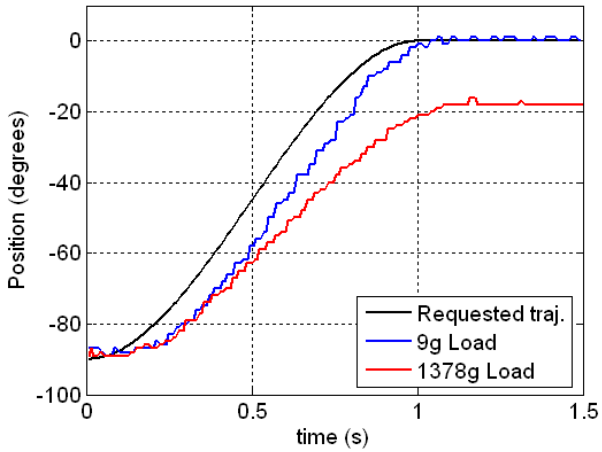


Fig. 59: Resposta ao polinómio para duas cargas diferentes (T_{traj}=1s).

No entanto, continua-se a verificar o crescente tempo de atraso e erro em regime estacionário com a carga aplicada em relação à trajectória esperada. Para corrigir estas características torna-se importante realizar alguma espécie de controlo adicional ou ao nível do controlador interno do servo, ou externamente, usando o sinal de posição como *feedback* e o sinal de PWM para controlo da posição.

2.3 – ESTUDO DO SERVOMOTOR EM MALHA FECHADA

Este capítulo descreve algumas estratégias para melhoria da resposta dos servos, em termos de:

- ➔ tempo de subida;
- ➔ tempo de estabelecimento;
- ➔ erro em regime estacionário.

O objectivo será minimizar ao máximo estes parâmetros de modo a aproximar ao máximo a resposta da trajectória solicitada.
Várias metodologias podem ser seguidas para compensação, como por exemplo a substituição da electrónica de controlo dos actuadores. No entanto, procura-se por métodos de compensação que não modifiquem estas unidades de forma a permitir a fácil substituição em caso de necessidade. Por razões de simplicidade realizar-se-á o controlo externamente ao servo usando para isso o microcontrolador para implementar a lei de controlo mais adequada. Com a implementação do controlador por software, é possível alterar os parâmetros ou a estrutura do controlador ou modificando simplesmente o código, ou por troca de informação do PC para o slave respectivo, evitando assim intervenções ao nível do hardware.

2.3.1. O Controlador

A Fig. 60 descreve a metodologia a usar: o controlador representado pelo bloco $G_C(z)$ é implementado ao nível do microcontrolador que fará uso do sinal de PWM $u(t)$ e do sinal de posição $p(t)$ como *feedback* para comparação com a posição desejada $r(t)$.

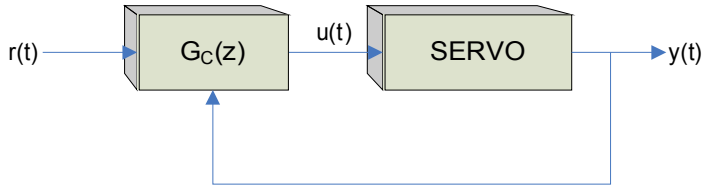


Fig. 60: Controlo externo do servomotor.

A partir daqui já não se dirá que um determinado duty-cycle do PWM corresponde a uma determinada posição, pois tal depende muito da inércia aplicada sobre o eixo, mas simplesmente é um sinal de controlo que para uma determinada carga corresponde a uma determinada posição. Para tal, a lei de controlo $G_C(z)$ seguirá um método para procurar o sinal de PWM mais adequado para que o sinal de posição medido p(t) coincida com a posição solicitada r(t). Esse método é apresentado na e é baseado num compensador clássico do tipo PID.

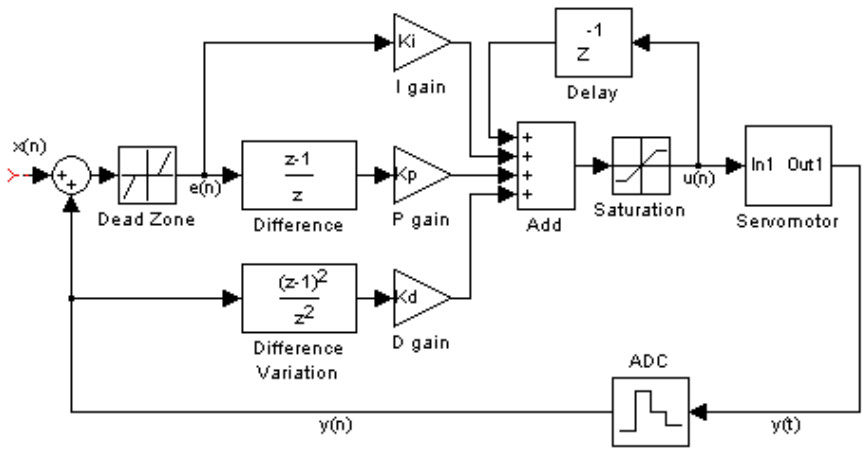


Fig. 61: Compensador PID incremental.

Este compensador é baseada na equação de compensação PI-D no domínio de Laplace, sendo convenientemente adaptada para o caso digital:

$$U(s)=\left[k_i\cdot\frac{1}{s}+k_p+s\cdot k_d\right]\cdot E(s)$$

No domínio digital (z) temos:

$$U(z)=\left[K_I\cdot\frac{1}{1-z^{-1}}+K_P+K_D\cdot(1-z^{-1})\right]\cdot E(z),\quad com\ \ K_I=k_i\cdot T_S,\ \ K_P=k_p,\ \ K_D=\frac{k_d}{T_S}\ \ (T_S=20\text{ms})$$

Colocando a equação na forma incremental:

$$U(z)\cdot(1-z^{-1})=K_I\cdot E(z)+K_P\cdot(1-z^{-1})\cdot E(z)+K_D\cdot(1-z^{-1})^2\cdot E(z)$$

Em termos numéricos:

$$\Delta u(n)=K_I\cdot e(n)+K_P\cdot\Delta e(n)+K_D\cdot\Delta\Delta e(n)$$

... em que Δ corresponde a uma variação relativamente à amostra anterior.

$$u(n)-u(n-1)=K_I\cdot e(n)+K_P\cdot[e(n)-e(n-1)]+K_D\cdot[(e(n)-e(n-1))-(e(n-1)-e(n-2))]$$

$$u(n)=u(n-1)+K_I\cdot e(n)+K_P\cdot[e(n)-e(n-1)]+K_D\cdot[e(n)-2\cdot e(n-1)+e(n-2)]$$

De modo a minimizar a instabilidade resultante do aumento da ordem da equação diferencial substituamos na secção derivativa $e(n)$ por $r(n)$-$y(n)$ com $r(t)$ constante. Desta forma a instabilidade da componente derivativa deixa de estar relacionado com as variações do sinal de referência[3].

$$\boxed{u(n)=u(n-1)+K_I\cdot e(n)+K_P\cdot[e(n)-e(n-1)]-K_D\cdot[y(n)-2\cdot y(n-1)+y(n-2)]}$$

Esta é a lei de controlo implementada no microcontrolador cujos parâmetros K_I, K_P e K_D são passados pelo PC através da rede de comunicações. Prevê-se que a componente integral resolva o problema do erro em regime estacionário e as restantes componentes lidem com a velocidade do sistema, mas que fique bem claro, que não é lícito importar "ideias feitas" provenientes de disciplinas relacionadas com o controlo de sistemas, uma vez que nem sequer iremos lidar com entradas em degrau, mas sim numa sucessão de degraus, o que pode resultar em efeitos diferentes no resultado final.

Note que o controlo é realizado de forma incremental, calculando em cada iteração o incremento a dar ao sinal de controlo $u(t)$ a fornecer ao servo. Optou-se por esta solução dadas as vantagens que oferece:

- ✔ Não são necessárias variáveis de elevada resolução para armazenar o resultado de somas;
- ✔ Protecção *wind-up*;
- ✔ Transferência *bumpless* simplificada.

A alternativa ao algoritmo incremental exigiria a actualização de um somatório a cada iteração (elemento $1/s$) o que implicaria o recurso a variáveis de elevada dimensão (*longs* ou *doubles*) para armazenar o resultado, o que nem sempre é favorável em arquitecturas baseadas em microcontroladores.

Além disso poderia ocorrer a requisição de uma posição fora dos extremos do servo (-90 e +90º) sem que a integração seja capaz de inverter a tendência do sinal de controlo uma vez que o seu incremento apenas se limita ao valor do extremo. Só ao fim de algum tempo, que não seria pouco, a soma pode ser suficiente para inverter a tendência, resultando numa perda significativa na reactividade da resposta. Este fenómeno denomina-se por *wind-up* e é devida à saturação do actuador. Embora haja bastantes soluções para este problema, a mais simples é a do algoritmo incremental, pois não existe qualquer integrador, fazendo com que numa situação de saturação o sinal de controlo $u(n)$ deixe automaticamente de aumentar resultando imediatamente na inversão da tendência.

3 Recomenda-se o teste da lei de controlo sem a inclusão do sinal de referência também na componente proporcional:
$u(n)=u(n-1)+K_I\cdot e(n)-K_P\cdot[y(n)-y(n-1)]-K_D\cdot[y(n)-2\cdot y(n-1)+y(n-2)]$

Um outro pormenor são as transferências *bumpless*, que não são nada mais do que a activação e desactivação do controlador em pleno funcionamento do servo. Pretende-se que quando o controlador é ligado ou desligado, o actuador não sofra qualquer variação brusca de posição. Embora não haja nenhum problema, por parte das duas soluções, na situação de desactivação do controlador, o problema surge na reactivação. Na solução com integrador, se a integração parar de funcionar durante a desactivação, quando reactivado, o sinal de controlo $u(n)$ não corresponderá ao sinal $r(t)$ que anteriormente era aplicado directamente, pois o resultado da integração deixou de ser actualizado, o que resulta num deslocamento brusco para uma posição desconhecida.
Por outro lado, se nunca se parar a integração corremos o risco da soma atingir valores excessivamente elevados ou mesmo de sofrer *overflow*, bastando para isso que os pedidos de actuação nunca correspondam ao valor de *feedback* – tal é frequente na presença de cargas. O resultado reflectir-se-ia em movimentos bastante oscilatórios na reactivação, acabando por provocar o fenómeno de *wind-up*!
Uma solução seria, em todas as reactivações, calcular o valor da soma de modo a que o sinal de controlo $u(t)$ correspondesse ao valor de $r(t)$ e inicializar a soma com esse valor. No caso do algoritmo incremental, se definirmos o valor $u(n\text{-}1)$ como o último valor aplicado no servo, quer com o controlador ligado, quer desligado, escusamo-nos de qualquer preocupação com este procedimento.

No entanto há uma desvantagem com este algoritmo, resultante do aumento da ordem de um para dois na remoção do integrador. Na presença de ruído, este compensador torna-se mais sensível podendo levar o sistema mais facilmente à instabilidade. Daí a necessidade do formato PI-D em que é o sinal de saída e não o de erro o utilizado na componente derivativa eliminando qualquer possibilidade de instabilidade resultante de variações do sinal referência $r(t)$. Mesmo assim recomenda-se o uso de valores baixos para o parâmetro K_D.

2.3.2. Controlo Integral (I)

Vamos agora testar o controlador para várias cargas e percursos utilizando as trajectórias em forma de rampa e de polinómio de terceira ordem, usando como referência as respostas em malha aberta para a sua avaliação.

Comecemos por utilizar a componente integral, definindo os parâmetros K_P e K_D a zero. Experimentando o valor de 0.08 para K_I para duas massas de elevado valor (Fig. 62 e Fig. 63) pode-se observar em ambos os casos a eliminação do erro em regime estacionário. No caso da Fig. 63 a diferença de dois graus é devida ao efeito da banda morta presente na entrada do controlador. Repare no sinal de saída do controlador solicitando ao servo uma posição mais elevada do que a desejada de modo a que ela seja cumprida na presença da carga eliminando assim o erro em regime estacionário.

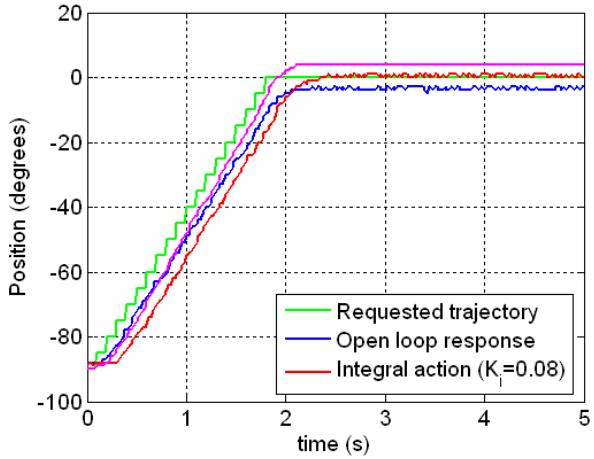


Fig. 62: Resposta à rampa com uma carga de 675g (K_I=0.08).

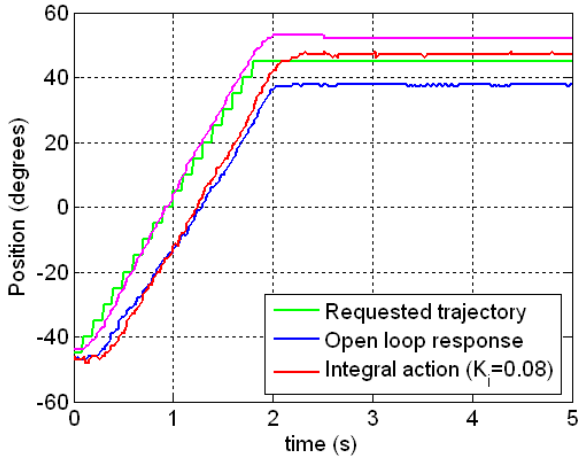


Fig. 63: Resposta à rampa com uma carga de 1129g (K_I=0.08).

Aumentando o valor de K_I para 0.20 e realizando o trajecto de -45 para +45º com uma carga de 1129g (Fig. 64 e Fig. 65), além da ausência do erro em regime estacionário, o tempo de atraso da resposta relativamente à trajectória solicitada é melhorado na presença do controlador o que beneficia o tempo de estabelecimento.

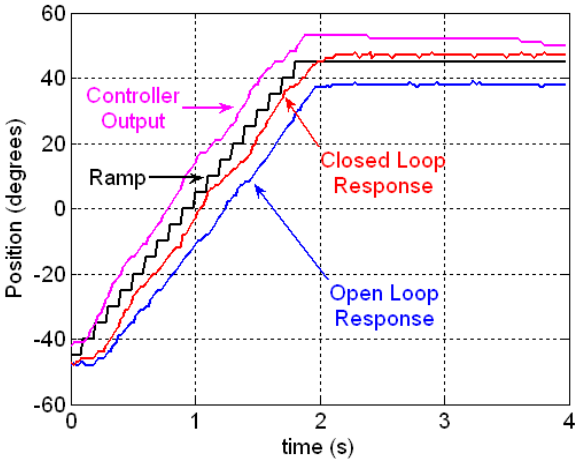


Fig. 64: Resposta à rampa com uma carga de 1129g (K_I=0.20).

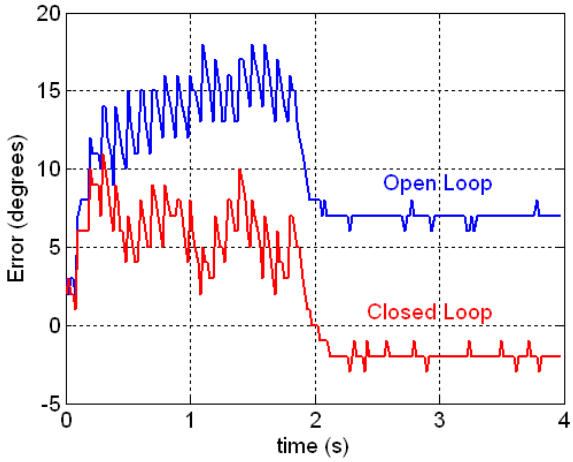


Fig. 65: Erro da resposta da Fig. 53.

De modo a estudar os efeitos do parâmetros K_I na resposta do servo, realizou-se uma experiência, no qual para uma massa elevada de 924g efectuando um percurso fixo (-45 a +45º) experimentaram-se vários valores de K_I. As respostas podem ser visualizadas da Fig. 66 à Fig. 69.

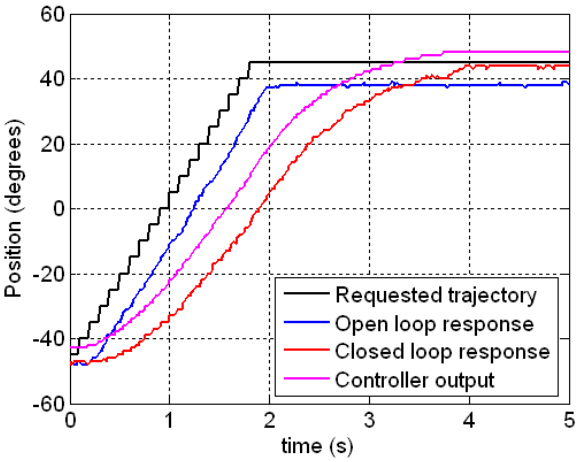


Fig. 66: K_I=0.02 (m=924g)

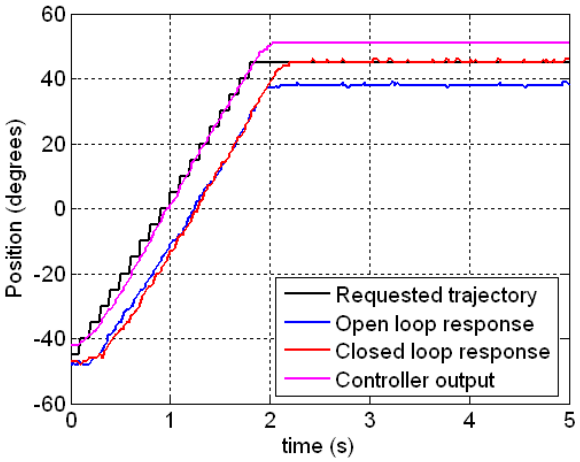


Fig. 67: K_I=0.07 (m=924g)

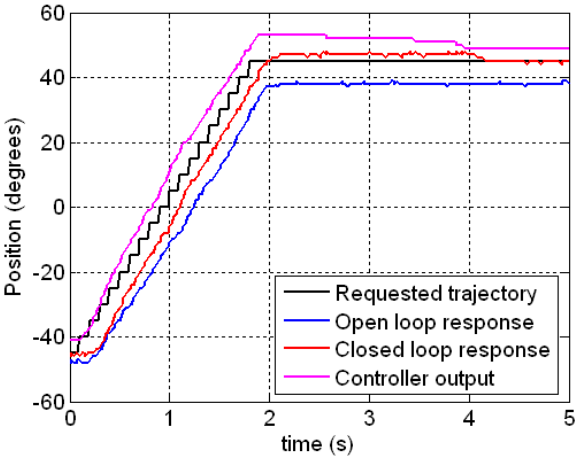


Fig. 68: K_I=0.15 (m=924g)

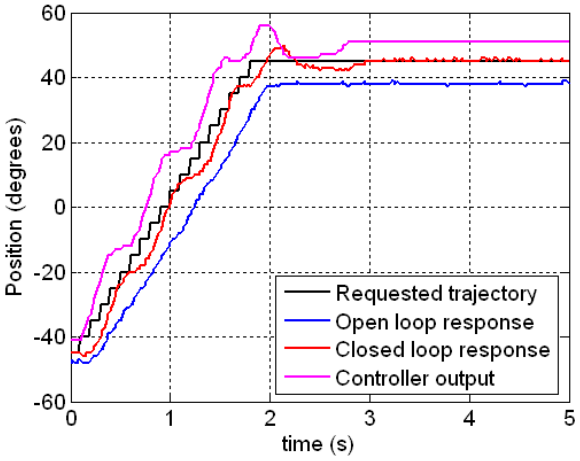


Fig. 69: K_I=0.30 (m=924g)

Por observação, constata-se que para baixos valores, a resposta tende a atrasar-se demasiado, mas à medida que vai aumentando o tempo de atraso diminui com melhorias significativas relativamente à resposta em malha aberta. O caso da Fig. 69 evidencia a resposta típica para valores excessivos do K_I: durante a fase transitória, tanto a posição medida como o sinal de controlo apresenta-se bastante oscilatório provocando no fim *overshoot*. No entanto seria desejável conter este *overshoot* de modo a melhorar ainda mais o atraso de cerca de 150ms medido pouco antes de se verificar as oscilações.

Quanto ao erro em regime estacionário, em todos os casos apresentados ele é eliminado, o que sugere que é suficiente a presença do integrador, independentemente do parâmetro K_I, exceptuando-se, obviamente, o valor nulo.

2.3.3. Controlo Proporcional+Integral (PI)

Vamos agora adicionar a componente proporcional, mantendo a integral pois é fundamental para a eliminação do erro em regime estacionário.
Para melhor percebermos as vantagens da componente proporcional, primeiramente definamos um K_I de modo a provocar um ligeiro overshoot na resposta do servo. A Fig. 70 e Fig. 71 apresentam um exemplo para uma carga de 924g com overshoot para K_I=0.10 e K_P =0.04.

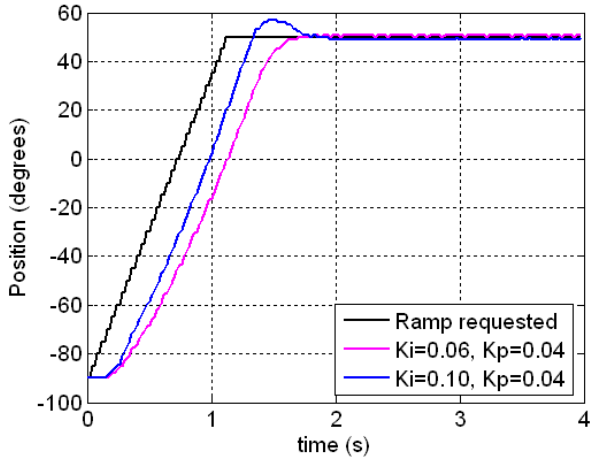


Fig. 70: Fenómeno do overshoot para valores de K_I elevados (carga de 924g)

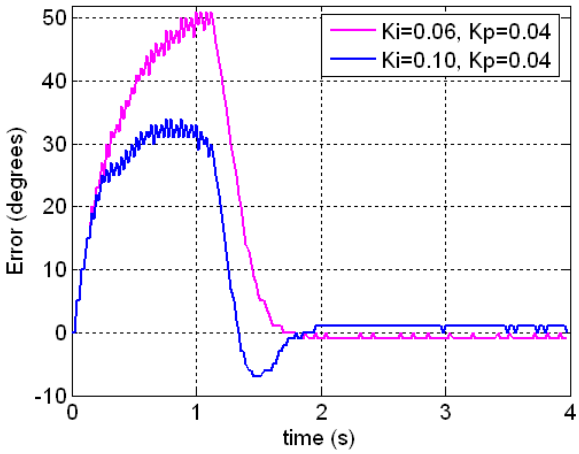


Fig. 71: Erro da resposta da Fig. 57.

Mantendo fixo o valor de K_I em 0.10 vamos aumentar o parâmetro K_P para 0.30 (Fig. 72 e Fig. 73).

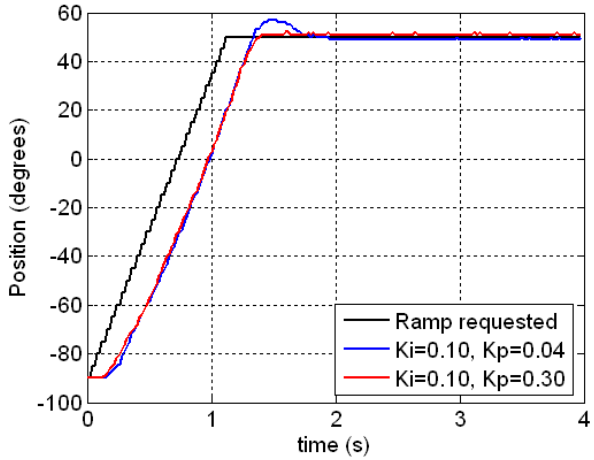


Fig. 72: Correcção do overshoot com o aumento de K_P (carga de 924g).

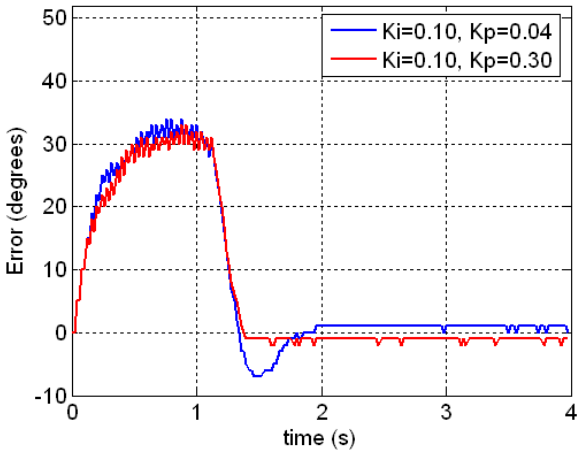


Fig. 73: Erro da resposta da Fig. 59.

Pode-se constatar que o *overshoot* desapareceu sem afectar significativamente o tempo de atraso. De modo a melhor perceber estes resultados registaram-se várias respostas para diferentes valores de K_P mantendo fixas a trajectória solicitada e a carga. A Fig. 74 e a Fig. 75 apresentam os resultados para dois valores diferentes de K_I agora utilizando trajectórias polinomiais.

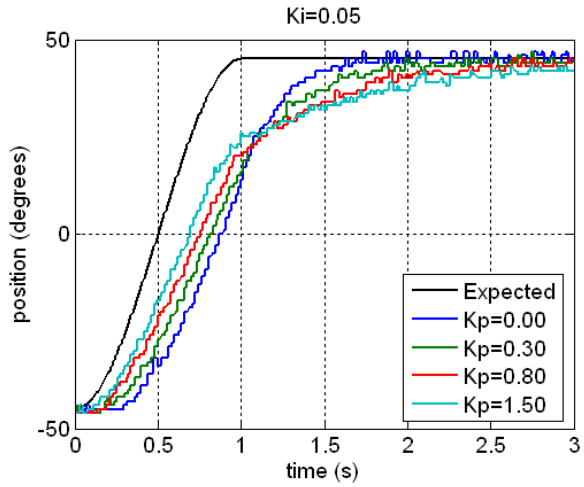


Fig. 74: Variação de K_P para K_I=0.05 (carga de 675g).

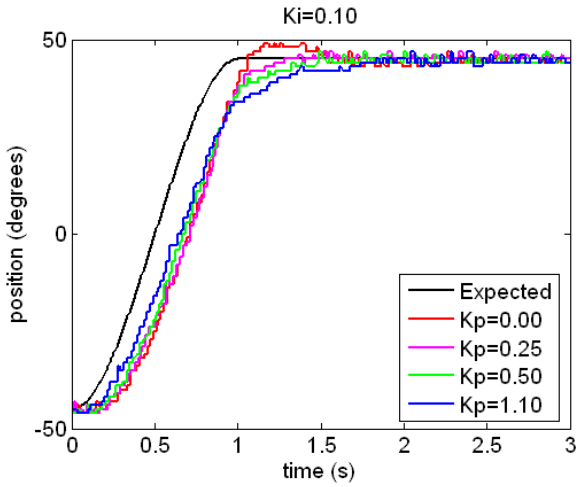


Fig. 75: Variação de K_P para K_I=0.10 (carga de 675g).

Analisando o gráfico da Fig. 75 com K_P=0.10 confirma-se o que se havia dito sobre os efeitos no overshoot: o overshoot é reduzido chegando mesmo a deteriorar o tempo de estabelecimento caso este valor seja excessivo, sem no entanto alterar significativamente o tempo de atraso.
Para um K_I mais pequeno – 0.05 – (Fig. 74), pode-se evidenciar mais claramente os efeitos de um K_P excessivo em valor: o tempo de estabelecimento é claramente prejudicado levando muito mais tempo a atingir o valor final, mas o tempo de atraso é melhorado acelerando o seguimento da trajectória no seu início. Ocorre, por isso, um ponto de cruzamento entre as diversas respostas quase coincidente com o término da trajectória.
Logo, conclui-se que deve haver um compromisso entre o tempo de atraso e o tempo de estabelecimento de modo a não perder demasiado num dos lados. O parâmetro K_P deve, por isso, ser escolhido tendo em conta estes aspectos.

2.3.4. Controlo Integral+Derivativo (ID)

Substituindo a componente proporcional pela derivativa podemos observar que praticamente não afecta a acção integral, tal como se observar na Fig. 76 e na Fig. 77, pelo que não se encontra utilidade para este tipo de controlo.

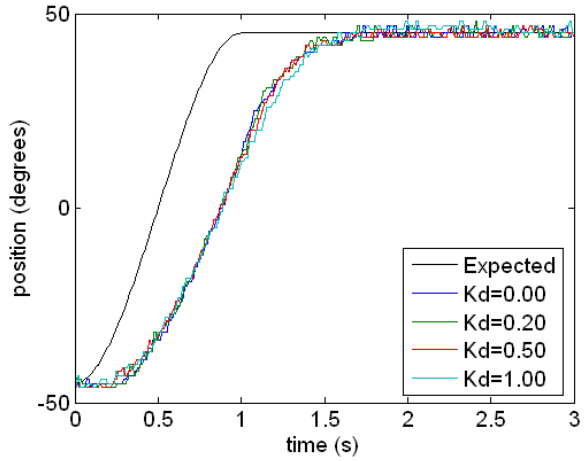


Fig. 76: Resposta ao polinómio com K_I=0.05 e K_P=0.00 (carga de 675g).

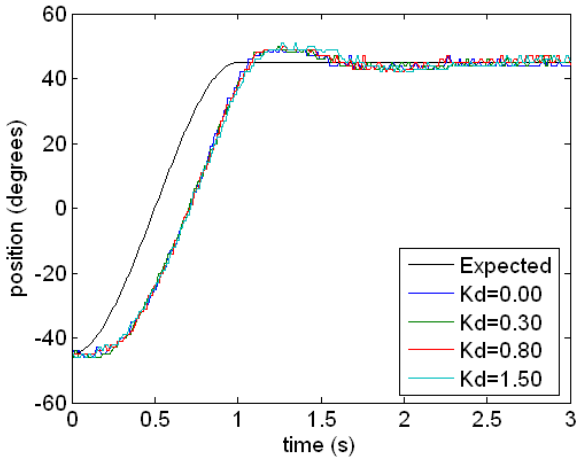


Fig. 77: Resposta ao polinómio com K_I=0.10 e K_P=0.00 (carga de 675g).

2.3.5. Controlo Proporcional+Integral+Derivativo (PID)

Introduzindo agora todas as três componentes podemos optimizar todos os parâmetros de modo a encontrar a melhor relação entre:

- ➔ Tempo de atraso;
- ➔ Tempo de estabelecimento;
- ➔ Overshoot;
- ➔ Oscilação durante a fase transitória.

Por análise de cada componente do controlador, apresentadas nas secções anteriores, já podemos relacionar estes parâmetros com cada um deles:

- ➔ A **componente integral** está directamente relacionada com o tempo de atraso, sendo tanto menor quanto maior for o parâmetro K_I. No entanto para valores excessivos começa a instabilizar com o surgimento de *overshoot*.
  Resumindo:
  - Diminui o tempo de atraso;
  - Aumenta o *overshoot* para valores de K_I elevados.
- ➔ A **componente proporcional** é utilizada para contenção do *overshoot*, sendo capaz também de melhorar o tempo de atraso se K_P possuir valores elevados, sob pena do tempo de estabelecimento se deteriorar.
  Logo, temos:
  - Redução do *overshoot*;
  - Para valores elevados de K_P:
    - O tempo de atraso é melhorado;
    - O tempo de estabelecimento deteriora-se.
- ➔ Embora ainda não tenha sido demonstrado a utilidade da **componente derivativa**, acredita-se que ela é capaz de conferir estabilidade durante a fase transitória, pelo que, para valores limite da compensação PI será interessante incluir a componente derivativa para acréscimo da estabilidade.

Combinando estas três componentes deveremos conseguir uma resposta muito próxima da ideal, com um tempo de atraso muito pequeno, um tempo de estabelecimento próxima da duração da trajectória, overshoot nulo e um comportamento suave durante a fase transitória.

Como procedimento para fazer o tunning da compensação seguiram-se os passos seguintes:

1. Aumentar K_I, de modo a optimizar o tempo de atraso, até começar a ocorrer *overshoot*;
2. Aumentar o valor de K_P o suficiente para eliminar o *overshoot*. Não convém utilizar este parâmetros para optimizar o tempo de atraso, uma vez que o tempo de estabelecimento é, ao mesmo tempo, agravado. Deixemos, por isso, essa tarefa à acção integral;
3. Se a resposta transitória ainda não for demasiado oscilante, voltar ao passo 1 para melhorar ainda mais o tempo de atraso;
4. Se começar a instabilizar durante a fase transitória, aumentar o parâmetro K_D de modo a conferir suavidade durante o percurso;
5. Voltar ao passo 1.

A Fig. 78 compara dois ensaios executados durante o tunning correspondentes a um ajuste inicial e a outro final, verificando-se uma melhoria de cerca de 6º no erro máximo em regime transitório. A Fig. 79 apresenta um caso de exagero nos parâmetros de compensação levando à instabilidade. De modo a evitar estas situações convém executar o algoritmo de *tunning* em passos pequenos, permitindo assim encontrar os parâmetros óptimos mais facilmente.

A partir de certo ponto, ao qual chamaremos de compensação limite, a melhoria já começa a ser mais exigente com variações muito mais pequenas dos parâmetros de compensação. Quando tal começa a ocorrer considere o *tunning* como terminado com os parâmetros óptimos correspondentes ao ajuste anterior.

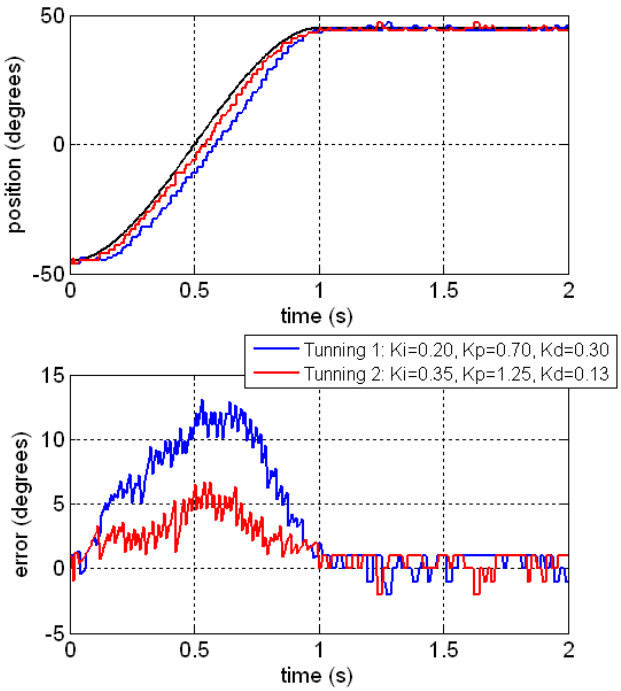


Fig. 78: Comparação entre dois conjuntos de parâmetros durante o tunning (m=675g).

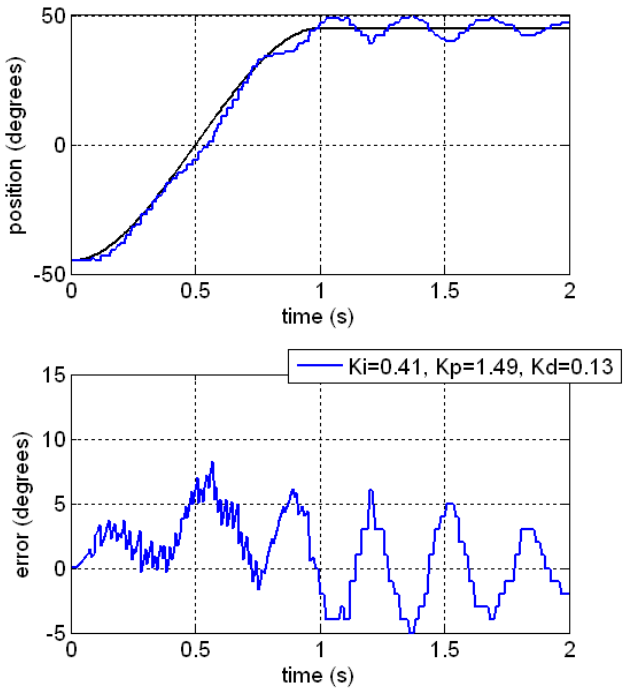


Fig. 79: Situação de instabilidade (m=675g).

Para a carga de 675g encontraram-se como parâmetros óptimos o conjunto K_I=0.39, K_P=1.46 e K_D=0.15 com um erro máximo de menos de 5º (Fig. 80). No entanto, note que nos encontramos numa situação em que o sistema torna-se muito susceptível à instabilidade face a perturbações externas. A Fig. 81 exemplifica este caso, em que o ensaio com os parâmetros óptimos é repetido verificando-se agora alguma instabilidade.

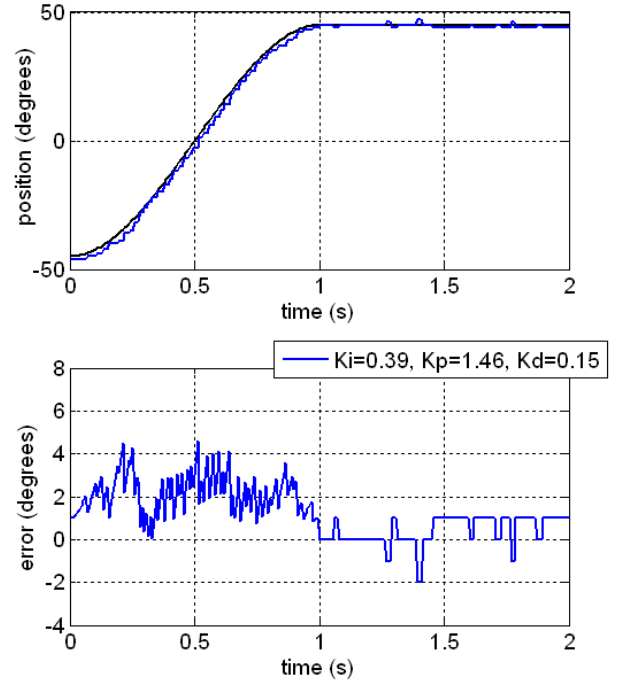


Fig. 80: PID optimizado para uma carga de 675g.

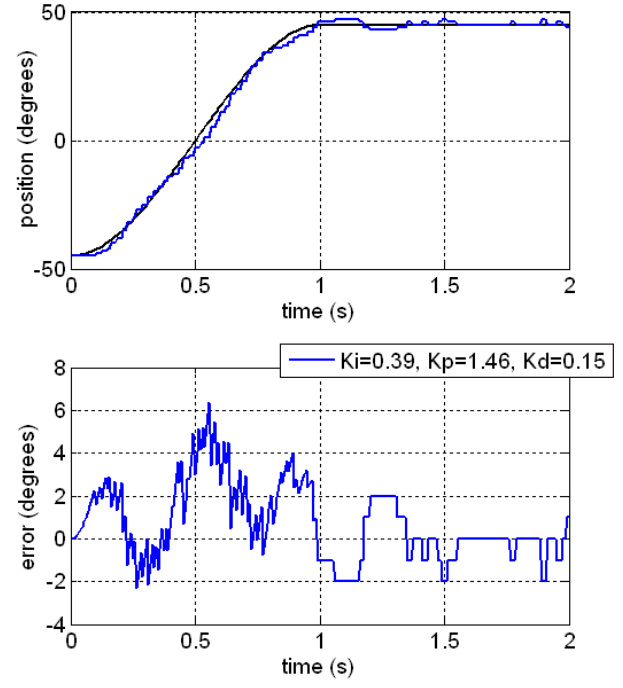


Fig. 81: Repetição do ensaio com os parâmetros de compensação óptimos.

2.3.6. Algumas Notas

Embora a solução de controlo baseada num PID aparente ter bons resultados, não dos devemos esquecer que determinado um conjunto óptimo de parâmetros para compensação dos desvios do servo, apenas se aplica nas condições em que foi feito o *tunning*:

- ➔ Massa da carga;
- ➔ Posição inicial e final, ou melhor dizendo, variação do binário ao longo da travessia;
- ➔ Período do trajecto;
- ➔ Tensão de alimentação e corrente máxima fornecida.

Mudando qualquer um destes parâmetros resulta invariavelmente na alteração da resposta do servo podendo conferir-lhe maior estabilidade ou torná-lo instável:

- ➔ O aumento da massa da carga induz à instabilidade, uma vez que o servo tem de fazer um "esforço" maior para a mover, no sentido que tem de aplicar maior binário no motor, e por isso, o tempo de atraso será acrescido. Numa situação destas em que o sinal de *feedback* tem dificuldade em acompanhar a "velocidade" do controlador (definido pelo factor de integração K_I), o sinal de controlo tende a saturar facilmente provocando fenómenos de *overshoot*. Por outro lado, se a carga diminuir de massa, o sinal de *feedback* consegue acompanhar perfeitamente o sinal de controlo conferindo uma maior reactividade à sua correcção.
- ➔ Se o percurso da trajectória for alterado, a trajectória de binário também muda fazendo com que o servo tenha de dispender mais ou menos energia, de acordo com o caso, para a realizar. Esta situação é equivalente à da modificação da carga, na medida que quando o binário a que está sujeito aumenta (aproxima-se do ponto 0º) é equivalente a aumentar a massa da carga, e vice-versa.
- ➔ Além da carga influenciar a resposta do servo, também a velocidade influencia. Tal é devido ao facto, de não só a força gravítica fazer parte do binário exercido no servo, mas também a velocidade e a sua variação (aceleração). Desta forma, diminuindo o período da trajectória, estamos a aumentar a velocidade o que interfere no binário exercido no induzindo uma resposta mais instável.
- ➔ Um outro detalhe importante são as condições de alimentação eléctrica dos servomotores. Caso a corrente máxima a fornecer seja limitada ou a tensão de alimentação é baixa, o binário a aplicar no motor para executar um determinado movimento aumenta, introduzindo atraso na resposta e logo maior instabilidade. Para minimizar estes problemas, duas baterias de Lítio de 7.4V são ligadas em paralelo de modo a fornecer uma corrente de 9600 mAh ao sistema.

Como se pode constatar, estamos a lidar com um processo altamente não linear em que as condições iniciais aplicadas nos servomotores estão sempre a mudar, o que é um problema, uma vez que o controlo clássico baseado num PID não entra em consideração com as condições iniciais.

Uma forma de dar a volta a esta questão corresponde a actualizar em tempo real o valor dos parâmetros de compensação de modo a adaptar o controlador a cada situação específica. Foi pensando nesta questão que se decidiu que a compensação via *software* seria a melhor opção, uma vez que é muito fácil mudar os parâmetros de controlo sem intervenções a nível de hardware como acontecia se o controlador estivesse implementado fisicamente. De modo a evitar a modificação do código na modificação destes valores, os parâmetros de controlo são passados a cada *slave* via barramento CAN, sendo a unidade principal, o PC, a responsável por atribuir os valores de compensação mais apropriados a cada acção.

No entanto, outro problema surge: como é que se detectam as situações mais ou menos exigentes em cada junta; e caso consigamos detectá-las, que lei de controlo seguirão os parâmetros de compensação? Para já tentaremos responder à primeira questão. Muito embora, no teste de um só servo, haja uma relação estreita entre binário aplicado e posição, tal deixa de acontecer na presença de várias juntas que se interligam em série por meio de elos, como é o que acontece com cada perna. Além disso, a velocidade da junta também afecta o binário pelo que é preciso discernir cada uma destas fontes de binário. Uma das formas de estimação do binário aplicado baseia-se na medição da corrente consumida por cada servomotor: quanto maior for o binário aplicado, maior é a corrente consumida pelo que a medição desta grandeza pode ajudar na detecção de situações de elevado ou de baixo *stress* sobre a juntas.

2.4 – MONITORIZAÇÃO DE CORRENTE

2.4.1. Estudo Estático da Corrente

Comecemos por avaliar a corrente consumida em situações de carácter estático, ou seja, com o servomotor em repouso, analisando-a de acordo com o binário aplicado. A Fig. 82 e a Fig. 83 apresentam um conjunto de medições de corrente realizadas com o servo em repouso para cada uma das posições avaliadas.

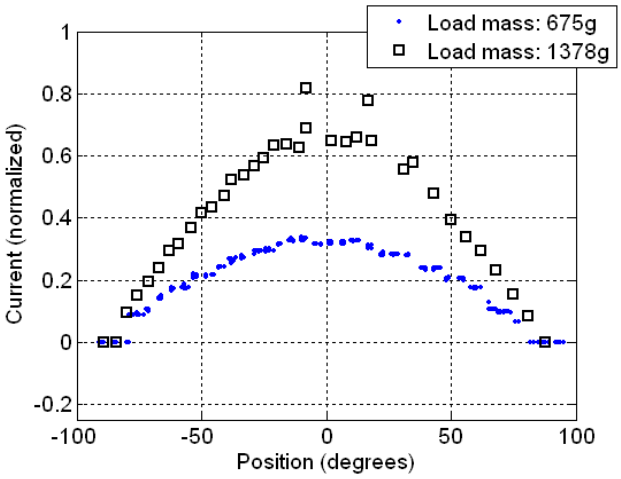


Fig. 82: Medição estática da corrente para duas cargas num percurso em subida (-90º a +90º).

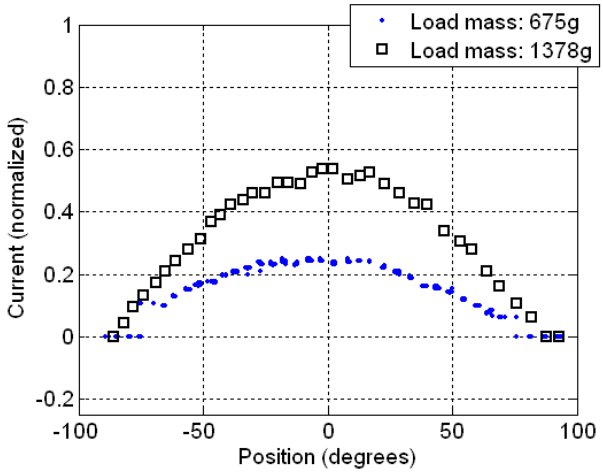


Fig. 83: Medição estática da corrente para duas cargas num percurso em descida (+90º a -90º).

Como não entram nos resultados o factor velocidade, estes apenas dependerão da força gravítica o que será de esperar que o valor máximo de corrente corresponda à posição 0º (máximo binário) e os valores mínimos aos extremos ±90º (binário nulo), tal como nos mostram os gráficos. Comparando com duas massas diferentes verifica-se que quanto maior é a massa da carga, maior é a corrente consumida evidenciando a estreita relação da força gravítica com a corrente consumida. No entanto note a diferença numérica dos resultados quando a medição se processo no sentido descendente (+90 para -90º) comparativamente ao oposto: na descida a corrente consumida é menor, talvez devido ao facto de os movimentos exigirem menor esforço que no outro caso... no entanto tal ainda não está completamente esclarecido uma vez que estamos a estudar o servo em repouso e não em movimento.

2.4.2 – Estudo Dinâmico em Malha Aberta

Introduzindo agora o elemento velocidade, fizemos vários capturas do consumo de corrente com o servo em pleno movimento – estudo dinâmico da corrente. Começámos por realizar movimentos seguindo a trajectória polinomial sem o controlador PID, ou seja, em malha aberta.
Por questões de rigor não vamos caracterizar cada trajectória a partir das posições inicial e final, uma vez que o que nos interessa é a variação de binário resultante da força gravítica. Aliás, as posições deixam de ter significado quando a configuração do servo é modificada para além da indicada na Fig. 32, pelo que referiremos o binário gravítico inicial, final e intermédio com a equivalência do percurso usando posições de referência segundo a configuração original:

- ➔ Trajectória desde o ponto de binário nulo até ao máximo: equivalente ao trajecto de ±90º para 0º em subida ou em descida;
- ➔ Trajectória desde o ponto de máximo binário até ao nulo: equivalente ao trajecto de 0º até ±90º em subida ou em descida;
- ➔ Trajectória entre pontos de binário intermédio passando pelo valor máximo: equivalente ao trajecto entre -45º e +45º.

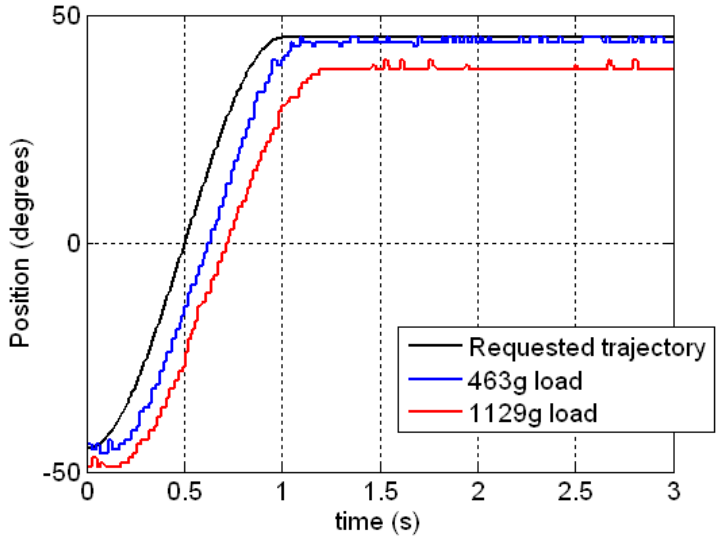


Fig. 84: Resposta em malha aberta de uma trajectória polinomial de 1s entre dois pontos de binário intermédio para duas cargas.

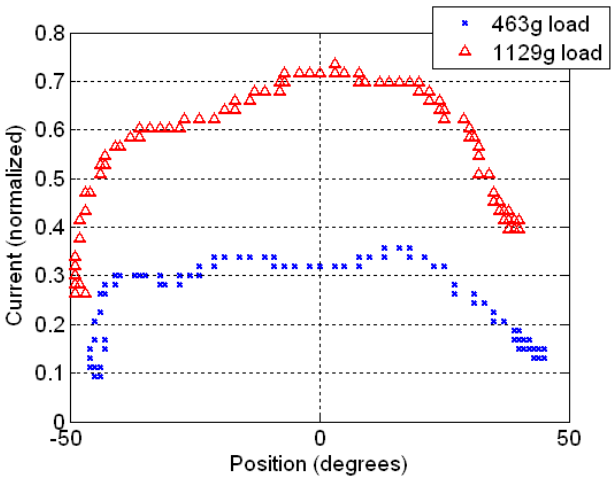


Fig. 85: Medição da corrente para cada posição do trajecto da Fig. 84.

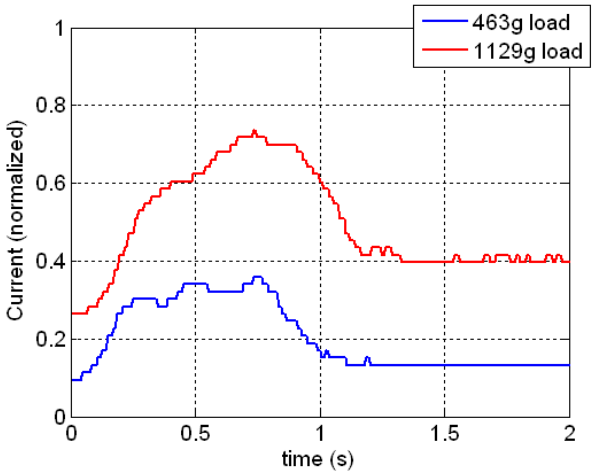


Fig. 86: Medição da corrente ao longo do tempo do trajecto da Fig. 84.

Realizando movimentos entre dois pontos de binário intermédio passando pelo valor máximo (equivalente ao percurso desde -45º a +45º na configuração original do servo) continua-se a verificar um comportamento corrente *versus* posição (Fig. 85) muito semelhante ao caso estático (Fig. 82), mas com a adicionante de ocorrer uma maior oscilação na distribuição ao longo das posições. A Fig. 86 mostra-nos o comportamento ao longo do tempo evidenciado este aspecto.

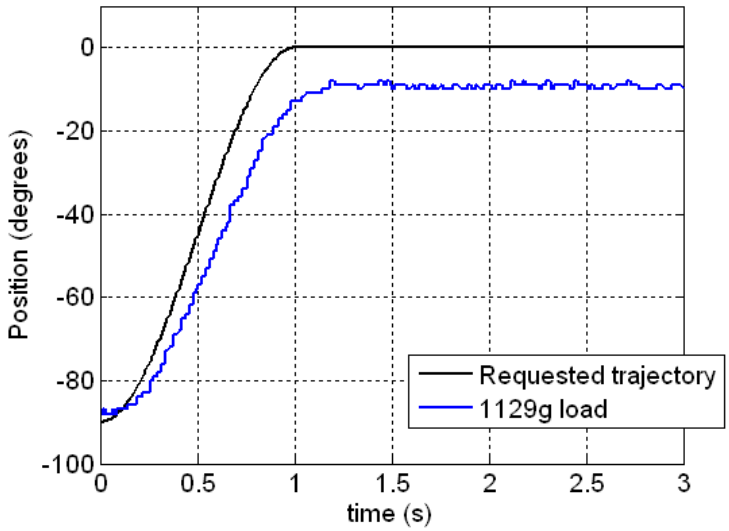


Fig. 87: Resposta em malha aberta de uma trajectória polinomial de 1s desde o ponto de binário nulo até ao máximo.

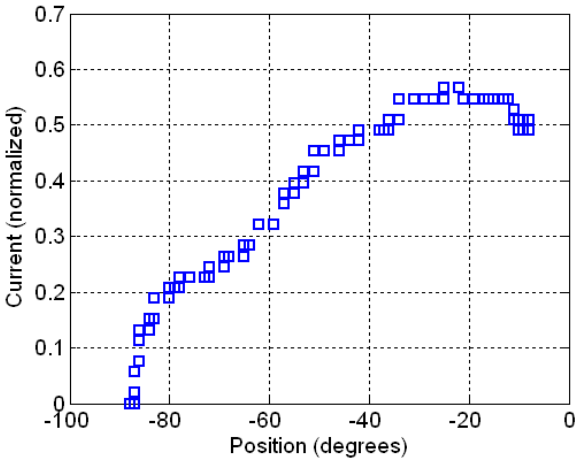


Fig. 88: Medição da corrente para cada posição do trajecto da Fig. 87.

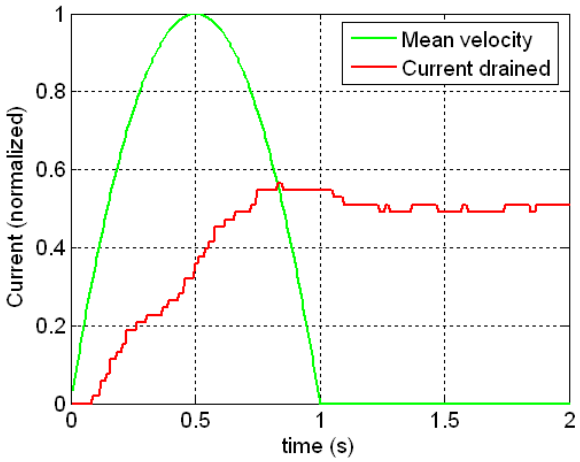


Fig. 89: Medição da corrente ao longo do tempo do trajecto da Fig. 87.

Aplicando um trajecto desde um ponto de binário nulo até ao máximo (-90º até 0º), podemos verificar na Fig. 88 e na Fig. 89 o crescente consumo de corrente à medida que o binário aumenta. No entanto, note numa ligeira descida da corrente quando a trajectória termina: como a força gravítica não diminui neste ponto, a causa provável pode estar residente na velocidade do servo. Até ao fim do percurso temos a contribuição da força gravítica e da velocidade no binário aplicado ao servo, mas no fim apenas temos a força gravítica o que pode justificar o decréscimo da corrente.

De modo a estudarmos melhor este fenómeno, realizámos uma segunda experiência, desta vez com uma trajectória iniciando num ponto de máximo binário, e terminando em binário nulo (equivalente ao percurso desde 0º até 90º na configuração original) (Fig. 90).
Seria de esperar que o início da resposta correspondesse à máxima corrente, uma vez que é neste ponto em que a componente gravítica do binário é mais significativa, e fosse descendo até zero quando chegasse ao ponto de binário nulo. No entanto, pela Fig. 92, não é isso o que se verifica: embora no instante inicial já esteja em consumo uma corrente não pouco significativa, após o arranque do movimento, ela aumenta ainda mais atingindo um máximo no ponto coincidente à máxima velocidade. Só a partir daqui é que a corrente começa a baixar continuamente até zero.

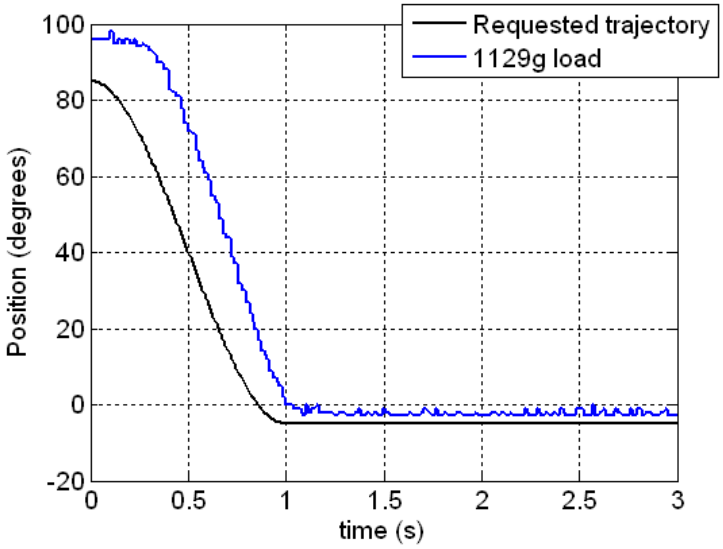


Fig. 90: Resposta em malha aberta da trajectória polinomial de 1s desde o ponto de máximo binário até ao nulo.

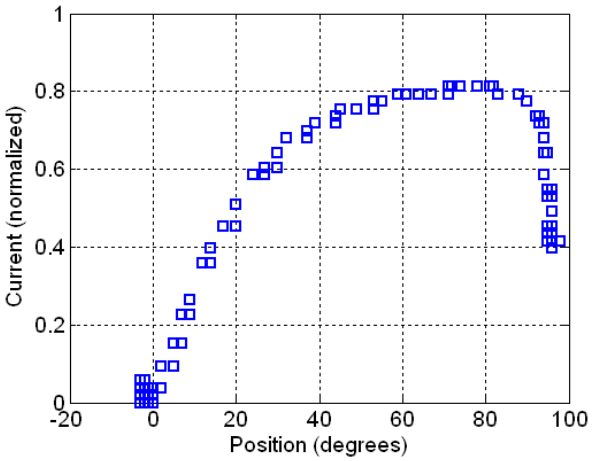


Fig. 91: Corrente consumida em cada posição do trajecto da Fig. 90.

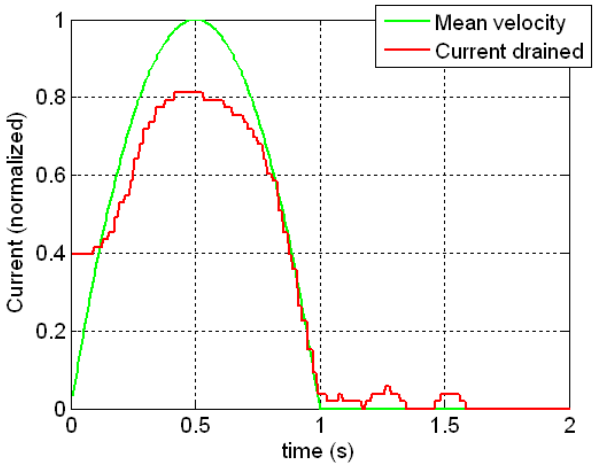


Fig. 92: Corrente consumida ao longo do tempo do trajecto da Fig. 90.

Este comportamento indica-nos claramente que a força gravítica não é a única que contribui na corrente consumida, contribuindo também a velocidade e a aceleração.

$\tau_{motor} = \tau_{gravítico} + \tau_{velocidade} + \tau_{aceleração}$ com $\tau_{motor} = K \cdot I$

Uma questão surge, entretanto. Apenas conseguimos medir o binário total aplicado ao motor, mas, no entanto, apenas nos interessa o resultante da força gravítica, pois é a componente que nos indica a inércia a que está submetida. Para conseguirmos isolar esta informação precisaríamos de conhecer a velocidade e aceleração em cada instante, uma vez que se relaciona com o respectiva componente de binário de forma proporcional:

$\tau_{gravítico} = m \cdot g \cdot L \cdot \sin(\theta)$
$\tau_{velocidade} = f_m \cdot \dot{\theta}, \quad f_m \rightarrow atrito\ do\ motor$
$\tau_{aceleração} = J_m \cdot \ddot{\theta}, \quad J_m \rightarrow Inércia\ do\ motor$

No entanto, apenas conseguimos fazer uma estimativa da velocidade em cada 100ms (muito sujeita a erro) e não podemos medir a aceleração, além que desconhecemos os parâmetros K, f_m e J_m para podermos relacionar as diversas componentes. Por enquanto este problema ainda carece de solução.

2.4.3 – Estudo Dinâmico em Malha Fechada

Nas análises seguintes vamos considerar agora o controlador PID ligado, ensaiando os mesmos trajectos que foram aplicado em malha aberta. Vemos então verificar se o consumo de corrente sofre variações significativas devido ao controlo.

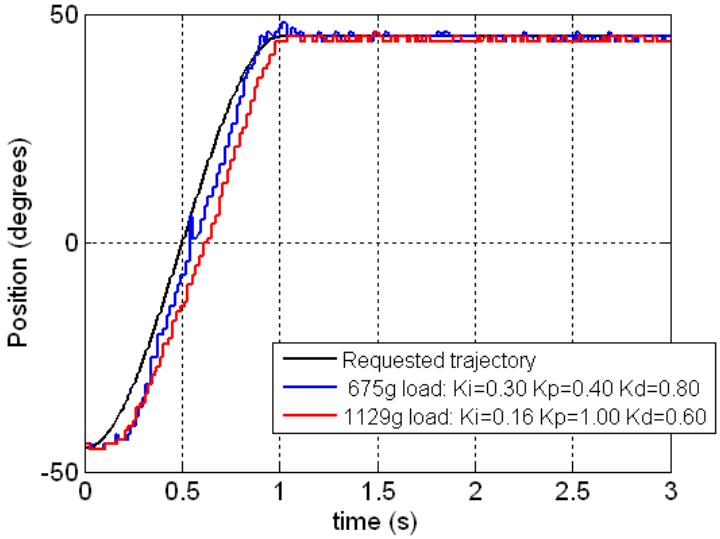


Fig. 93: Resposta em malha fechada de uma trajectória polinomial de 1s entre dois pontos de binário intermédio.

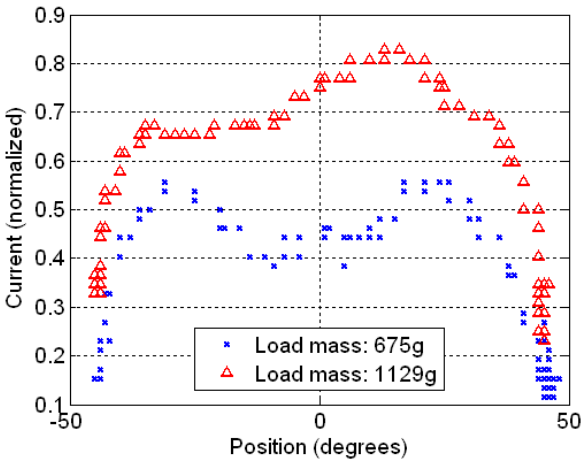


Fig. 94: Corrente consumida em cada posição do trajecto da Fig. 93.

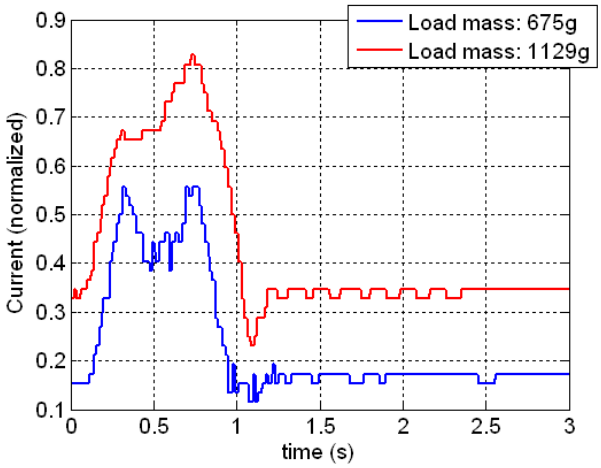


Fig. 95: Corrente consumida ao longo do tempo do trajecto da Fig. 93.

Optimizando o controlador para as condições iniciais:

- Trajectória: entre pontos de binário intermédio (-45 a +45º);
- Velocidade correspondente ao período de 1s;
- Cargas de 675g e de 1129g.

Medimos a resposta em termos de corrente, e pode-se verificar pela Fig. 94 e pela Fig. 95 uma oscilação ainda mais notória da corrente consumida. Tal é facilmente explicável pelas variações do sinal de controlo para compensar a posição solicitada: agora já não se trata de uma variação polinomial como acontecia em malha aberta, mas dependente do sinal de erro entre o sinal de *feedback* e a trajectória polinomial desejada, o que pode resultar num consumo de corrente mais irregular. A resposta ao longo do tempo evidencia claramente este consumo irregular.

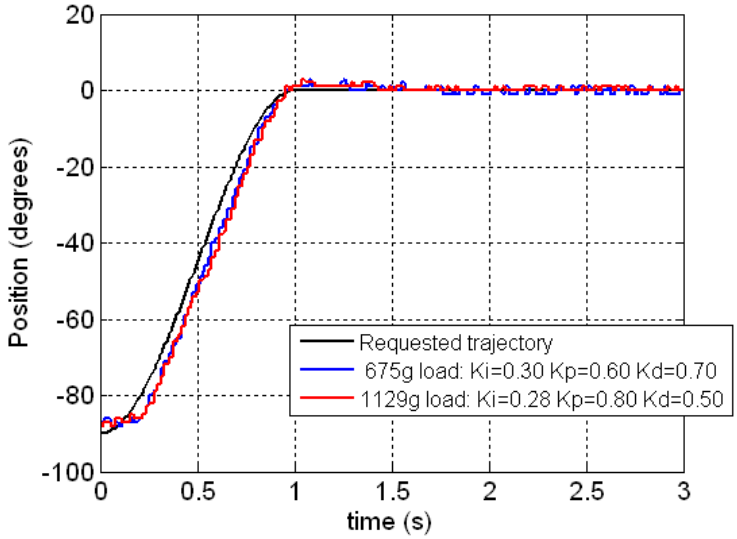


Fig. 96: Resposta em malha fechada de uma trajectória polinomial de 1s desde o ponto de binário nulo até ao máximo.

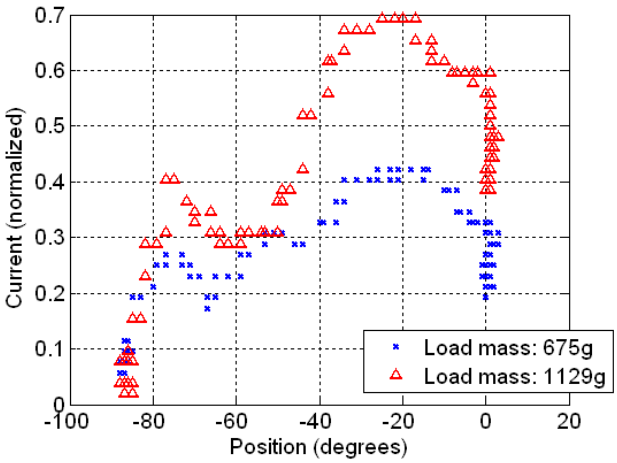


Fig. 97: Corrente consumida em cada posição do trajecto da Fig. 96.

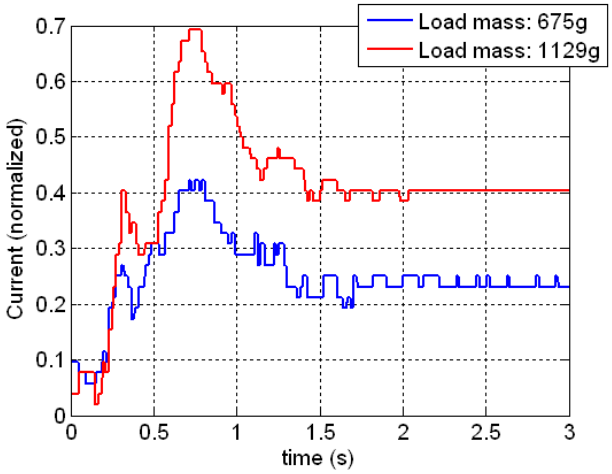


Fig. 98: Corrente consumida ao longo do tempo do trajecto da Fig. 96.

Para a trajectória do mínimo para o máximo binário (-90º a 0º) o consumo de corrente ainda é mais irregular, talvez devido ao crescente binário aplicado em toda o trajecto, fazendo com que o servo tenha mais dificuldades em acompanhar a taxa de integração do controlador. Tal resulta numa maior variação do sinal de controlo aumentando, por isso, a instabilidade do consumo de corrente.

Em jeito de conclusão, verificamos que o consumo de corrente com o controlador presente é muito mais oscilante trazendo bastantes problemas na sua interpretação, o que dificulta a procura por uma lei de variação dos parâmetros de controlo. Por enquanto este problema continua sem solução à vista, ainda mais pela dificuldade em extrair o binário resultante da força gravítica.

2.5 – APLICAÇÃO DOS ALGORITMOS AO ROBOT HUMANÓIDE

Conectando o microcontrolador, com os algoritmos de controlo implementados, a três servomotores correspondentes às seguintes juntas de uma perna humanóide:

- ➔ Servo 1: junta do pé (de rotação dianteira);
- ➔ Servo 2: junta do joelho;
- ➔ Servo 3: junta da anca (de rotação dianteira).

... executámos vários movimento, no qual apresentamos neste capítulo apenas o de flexão, dada a sua relevância na realização de um passo.

2.5.1. Movimento de Flexão em Malha Aberta

2.5.1.1. Na Ausência de Carga:

Como se pode visualizar nas figuras seguintes, na ausência de carga, o erro em regime estacionário é praticamente nulo, praticamente sem a necessidade de praticar controlo externo.

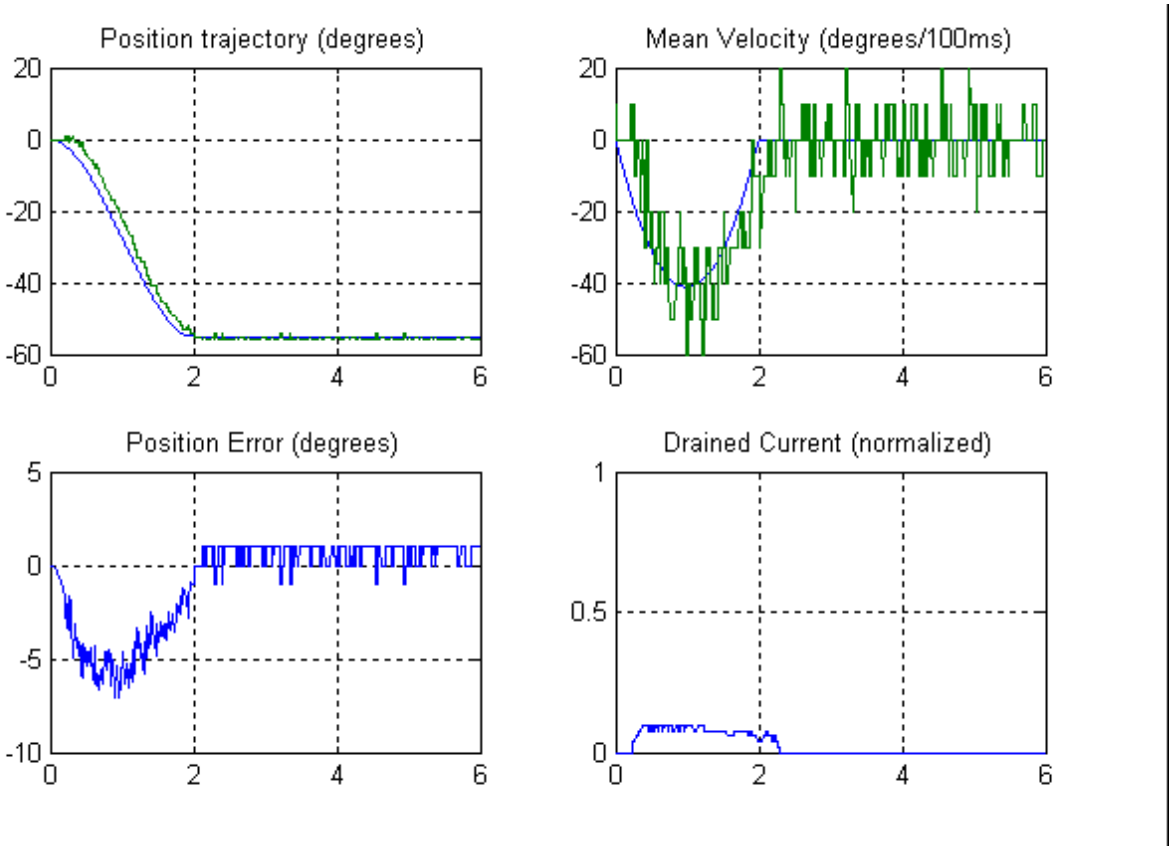


Fig. 99: Junta do pé.

Note na natureza errática na estimação da velocidade. Tal deve-se ao processo de medição baseada na variação de posição, que infelizmente introduz bastante erro.

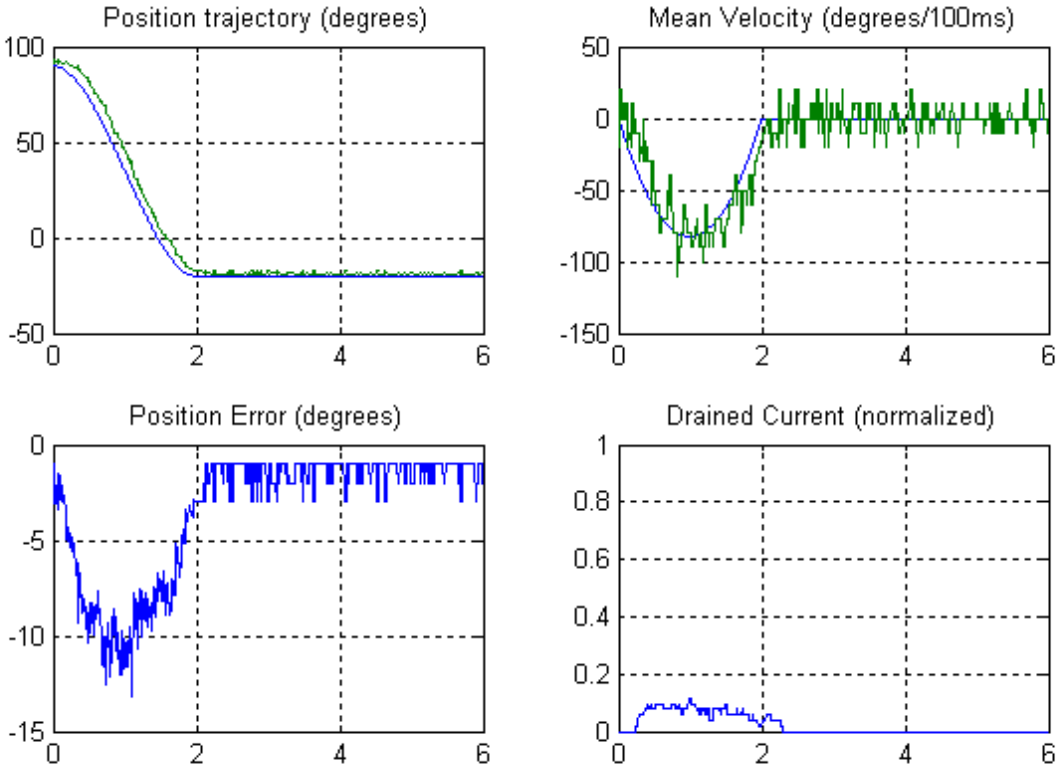


Fig. 100: Junta do Joelho.

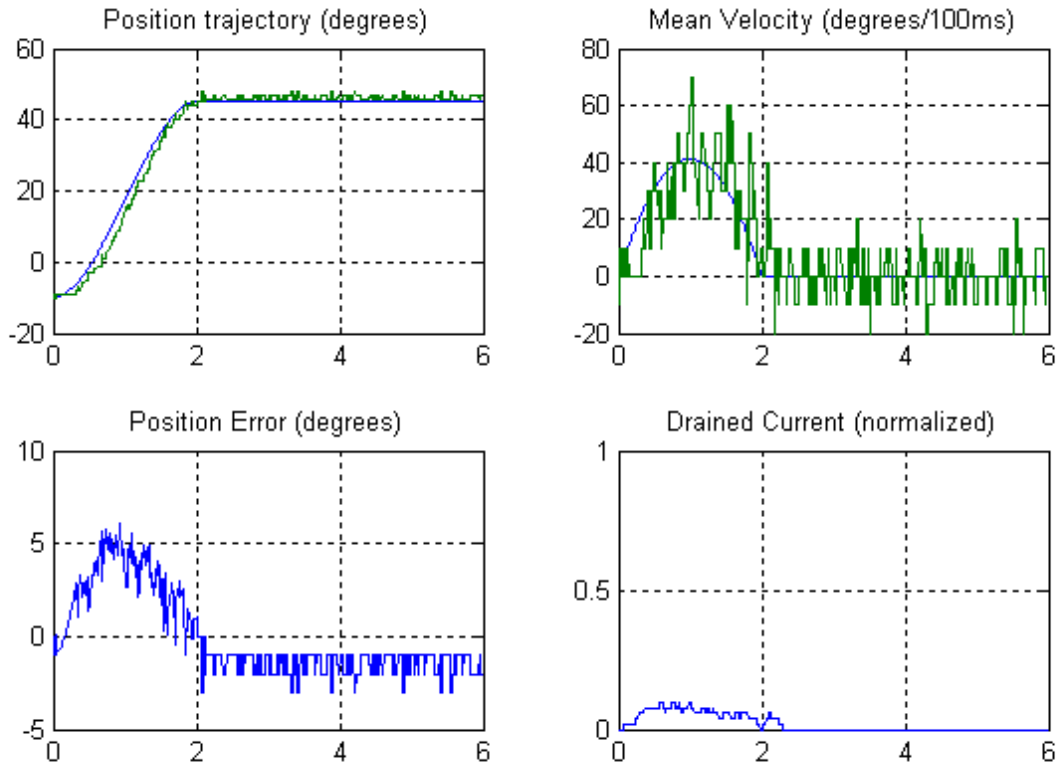


Fig. 101: Junta da Anca.

2.5.1.2. Na presença de uma Carga de cerca de 2Kg:

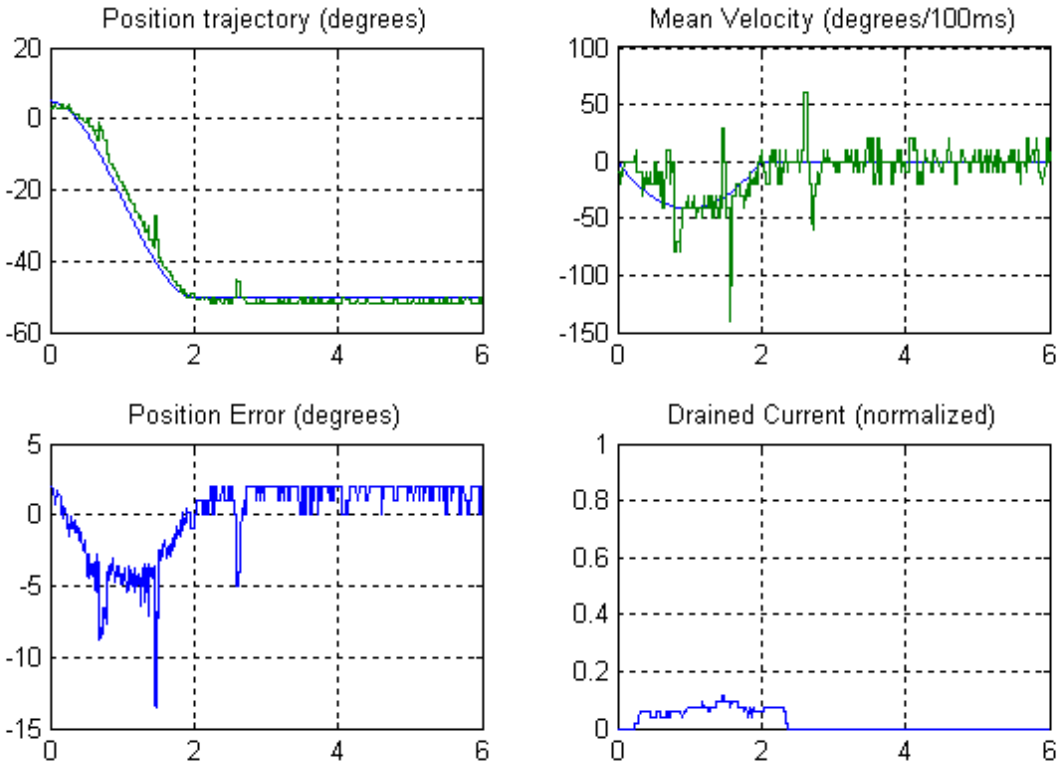


Fig. 102: Junta do pé.

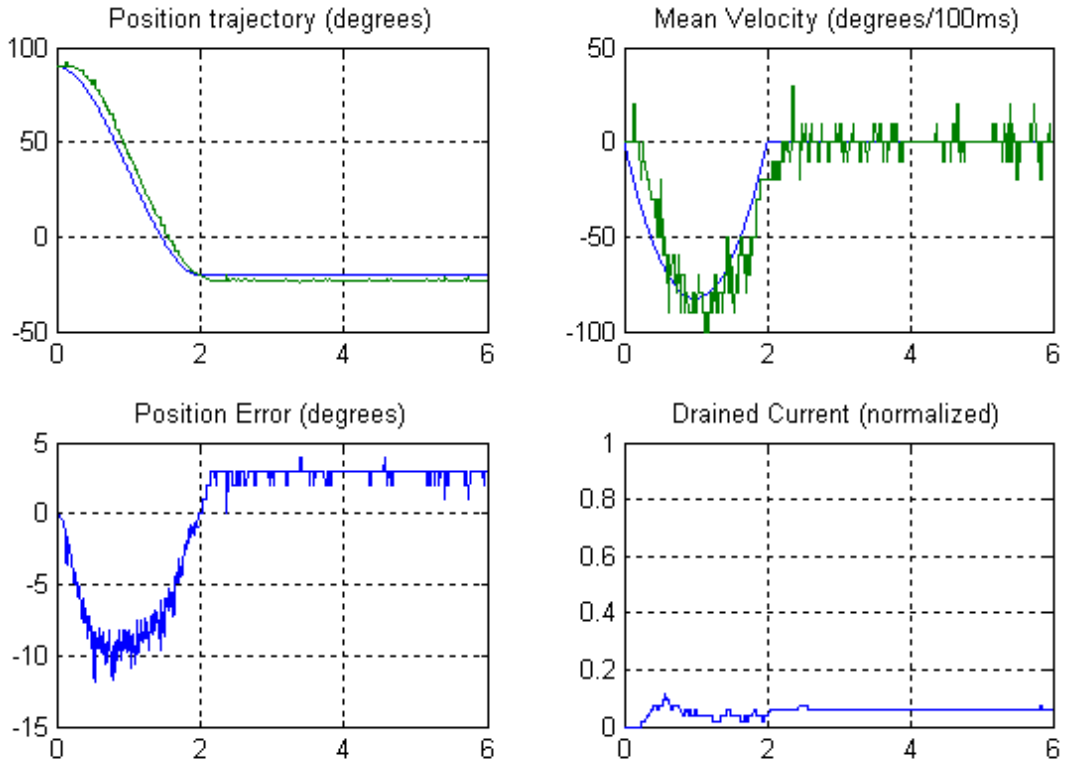


Fig. 103: Junta do joelho.

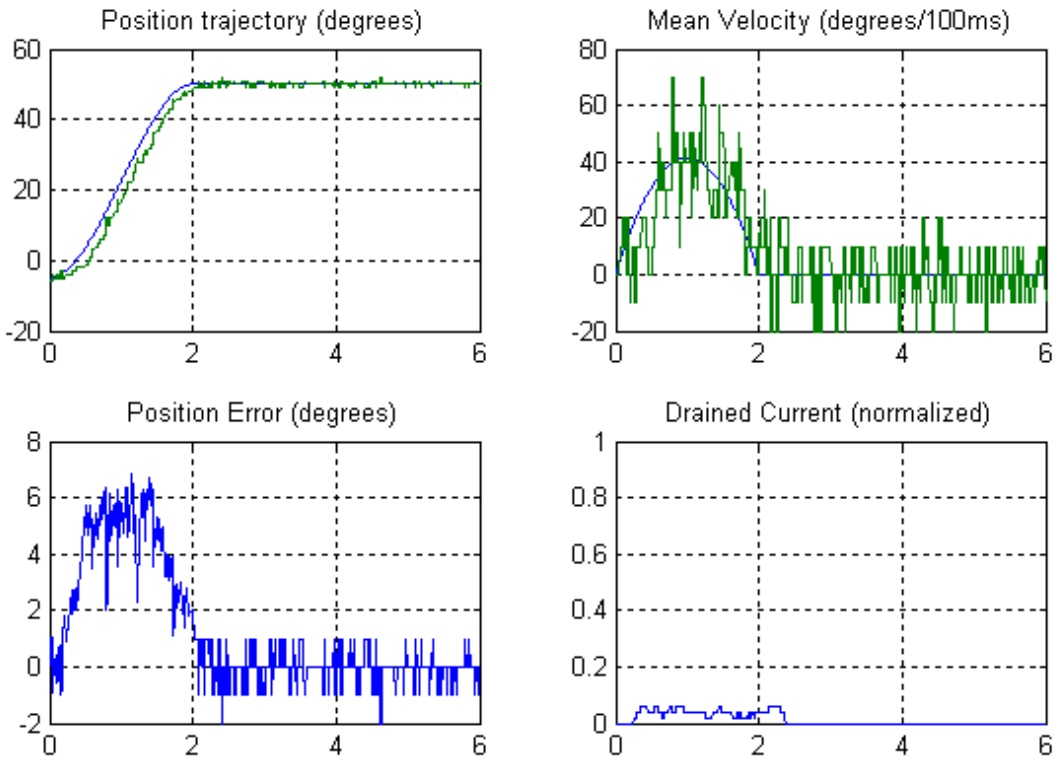


Fig. 104: Junta da anca.

Com a carga de 2Kg, surpreendentemente verifica-se pouca variação do erro em regime estacionário, com um baixo consumo de corrente. Note que agora o esforço é distribuído pelas três juntas, pelo que as condições não são tão exigentes como as utilizadas no capítulo anterior.

2.5.2. Movimento de Flexão em Malha Fechada

As figuras seguintes apresentam o comportamento das juntas, agora com o controlador ligado.
O caso mais exigente verifica-se no caso da junta do pé (Fig. 105), com erro em regime transitório de 14º e em regime estacionário de 5º. Com o controlador ligado, a resposta melhorou bastante reduzindo o erro máximo em regimen transitório para menos de metade, e eliminando o erro em regime estacionário.

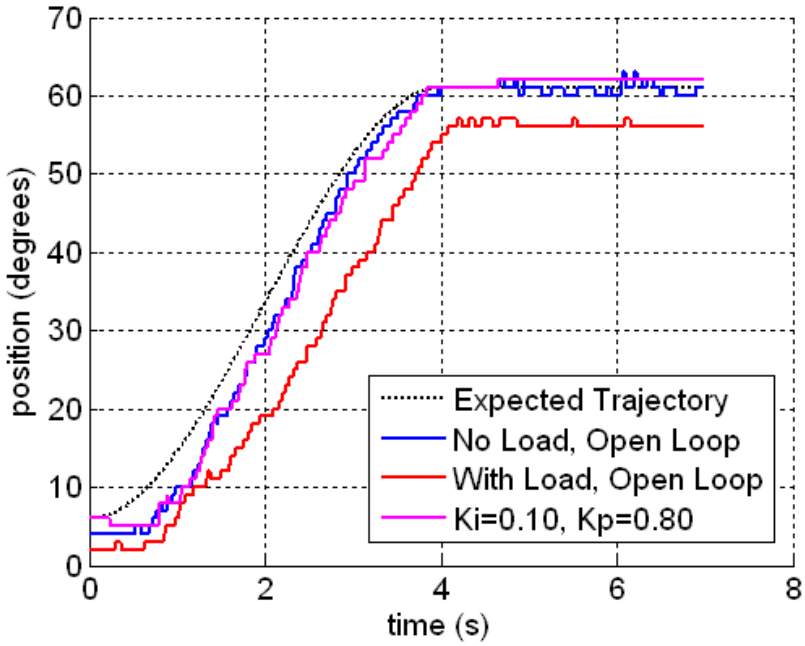


Fig. 105: Junta do pé.

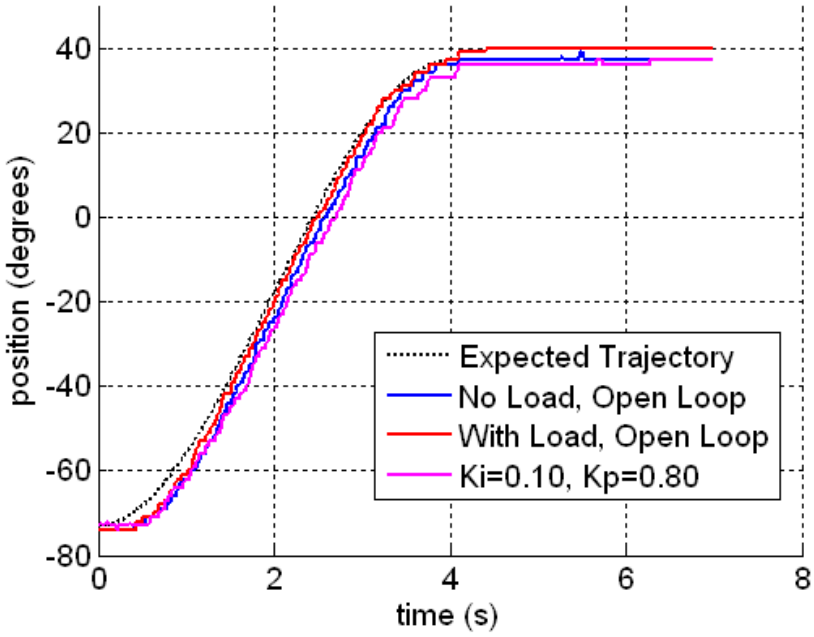


Fig. 106: Junta do joelho.

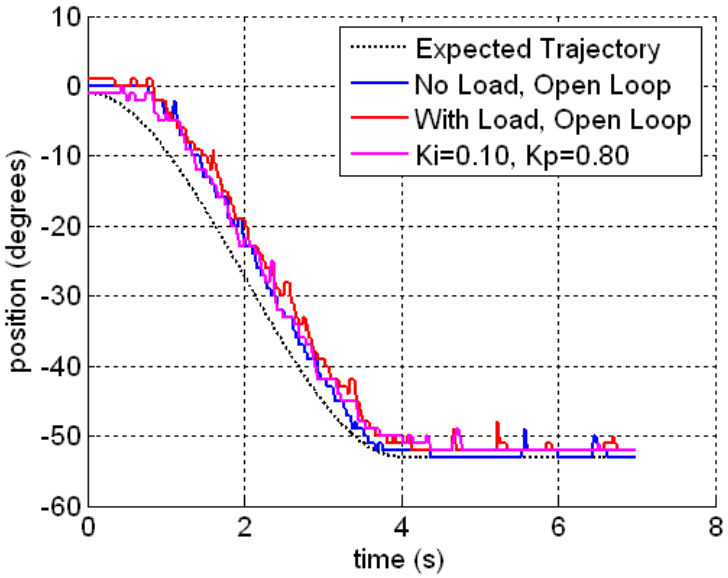


Fig. 107: Junta da anca.

2.5.3. Movimento das duas Pernas

Utilizando agora, as duas pernas juntas, com a aplicação de uma carga de massa partilhada entre os dois membros, executou-se o mesmo movimento de flexão para teste das juntas do pé, do joelho e da anca. Adicionalmente, de modo a testar a junta lateral do tornozelo do pé, realizou-se um deslocamento para o lado. As figuras seguintes demonstram os resultados referentes a apenas uma das pernas, dado que os da outra perna são bastante semelhantes.

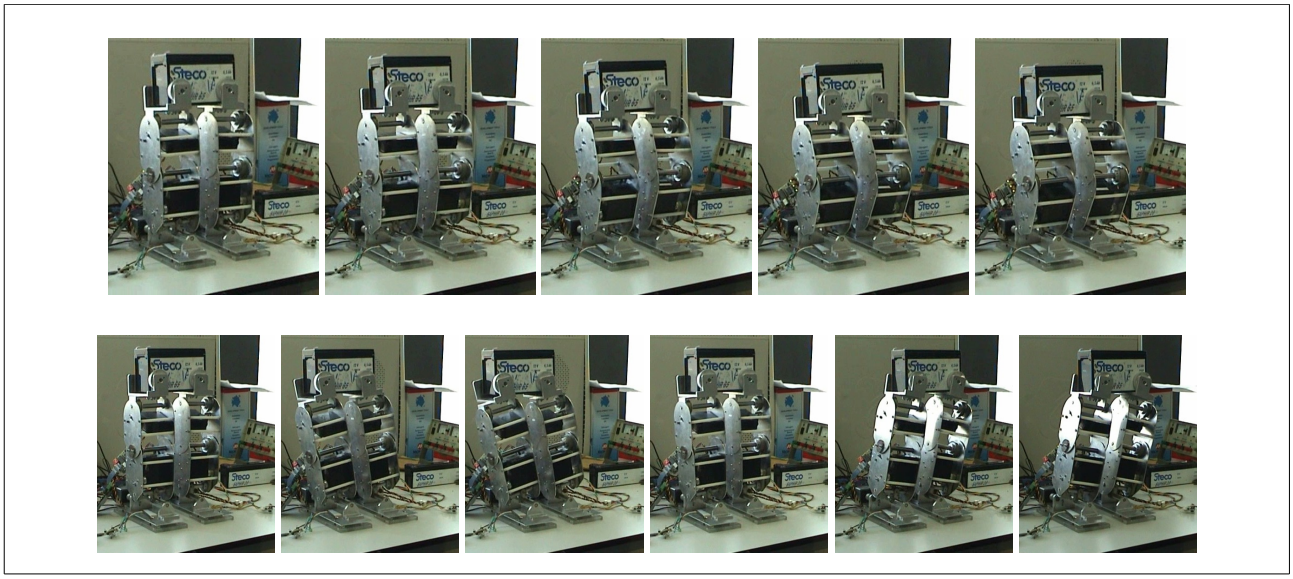


Fig. 108: Conjunto das duas pernas executando um movimento de flexão (sequência superior) e um movimento lateral (sequência inferior) com uma carga de 2.1Kg em completa sincronia de pernas.

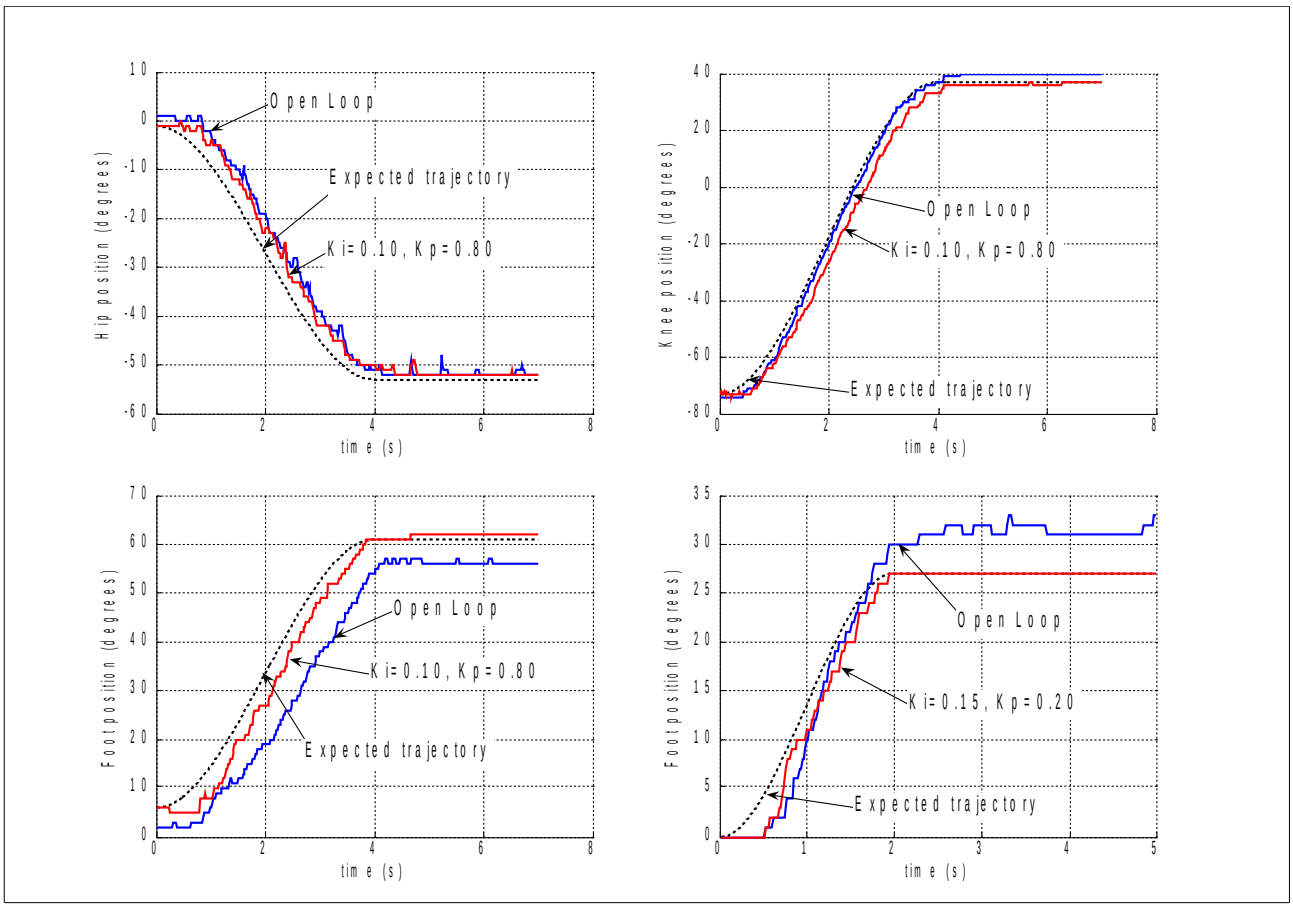


Fig. 109: Resposta ao polinómio com um controlador PI. Imagens superiores e inferior esquerda: comportamento das três juntas envolvidas no movimento de flexão; imagem inferior direita: comportamento da junta lateral do pé na realização do deslocamento lateral.

Como se pode observar, só com a presença do controlador interno, a resposta das juntas mais exigentes (juntas do pé) apresentam um comportamento com um apreciável tempo de atraso e de erro em regime estacionário, que é deveras melhorado com a introdução da compensação PI externa. Demonstra-se assim, com estes dados as vantagens da implementação do controlador.

2.5.4 Discussão de Resultados:

Embora, em função dos dados sensoriais, as respostas possuem uma boa qualidade, a nível mecânico já não é bem assim, com os efeitos elásticos das correias de transmissão e as folgas das juntas a tornaram-se significativos. Ao fim de cada conjunto de testes com uma carga elevada, verificava-se um deslocamento do parafuso de aperto de cada correia resultando no aparecimento de uma folga nas juntas mais exigentes. Tal exigia o reajuste do parafuso de aperto para corrigir a folga.
Um outro problema mais complexo prende-se com a ligação da junta do joelho com uma chaveta que garante a ligação da perna inferior com a superior. Ao fim de algum tempo começou a ocorrer folga nesta ligação, resultando numa degradação permanente dos movimentos, dado que esta folga não é reajustável. Recomenda-se por isso, a substituição do material utilizado por outro mais resistente, ou na impossibilidade desta solução, adicionar mais uma chaveta de fixação no outro lado do joelho.
Também se verificaram algumas folgas permanentes ao nível do veio do tornozelo com a presença de algumas ligações deficientes entre este veio e o veio do pé. Recomenda-se, também a revisão desta ligação.

Como se pode concluir, muito embora o comportamento dos servomotores seja considerada como muito bom pelos resultados sensoriais, em termos mecânicos ainda há muitos aspectos que precisam de ser melhorados sob pena de tornar ineficaz os algoritmos de controlo de posição e velocidade. Aconselha-se, por isso, uma séria revisão à estrutura das pernas, uma vez que são estes membros que terão de suportar os esforços mais exigentes, nomeadamente:

- ➔ no aperto das correias de transmissão: estas são as principais causas de folgas, aumentando à medida que se verifica um esforço, correndo o risco de provocar saltos de dentes. Uma solução que já foi indicada corresponde a efectuar o aperto de forma dinâmica com o recurso a uma espécie de mola, tornando automático o processo de ajuste.
- ➔ ao nível das ligações entre os veios e os elos tornando-os mais robustos quer pelo aumento da resistência dos materiais quer pelo aumento do número de ligações. Tal convêm ser estudado;

2.6 – CONCLUSÕES

Este capítulo teve como objecto de estudo, os servomotores utilizados para actuação. Como são dispositivos relativamente pequenos e baratos que já oferecem o controlo de posição, foram escolhidos dada a sua simplicidade de interface com um microcontrolador – podem-se ligar directamente – podendo controlar a sua posição simplesmente através de um sinal digital de PWM cuja largura de pulso define a posição do servo. Os modelos da HITEC, nomeadamente o modelo HS-805BB para as juntas de grande esforço, foram escolhidos dada o seu elevado binário (2.42 N.m @6V) comparativamente a outras marcas.

No entanto, cedo nos apercebemos que estes dispositivos não são perfeitos não oferendo qualquer tipo de controlo de velocidade – simplesmente se deslocam para a posição final à sua velocidade máxima – algo nada favorável quando pretendemos controlar a posição e a velocidade das juntas para que realizem movimentos suaves de velocidade limitada. Contudo, pela aplicação de um conjunto crescente (ou decrescente) de pequenos degraus de posição em forma de rampa, conseguimos implementar uma espécie de controlo de velocidade indirecto, na medida em que os extremos de posição da rampa e o intervalo de tempo da sua duração impõe uma velocidade média do movimento. Uma forma de trajectória mais exigente é o polinómio de terceiro grau que além de permitir a definição de uma velocidade média garante outros aspectos como é o caso de velocidade nula e acelerações limitadas no início e no fim da trajectória, algo importante para prevenir eventuais picos de corrente e limitar o consumo de corrente durante o percurso na realização destas trajectórias.

Mais tarde verificou-se que embora estes actuadores apresentem respostas praticamente ideais (resposta semelhante à trajectória solicitada), na presença de cargas tal deixa de ser verídico verificando um acentuado tempo de atraso no acompanhamento da trajectória solicitada e uma dificuldade em atingir o valor final devido à presença da força gravítica no binário aplicado (erro em regime estacionário não nulo). Usando o sinal de *feedback* de posição que o servo disponibiliza, por comparação com a posição desejada, tentou-se implementar uma lei de controlo que permitisse corrigir estes desvios. Esta lei de controlo baseou-se num compensador clássico do tipo PID (Proporcional+Integrador+Derivador), que por meio de diversos ensaios, conseguimos associar cada componente às características da resposta de modo a permitir-nos escrever um procedimento para rapidamente encontrarmos, através de ensaios, um conjunto de parâmetros que aproximem bastante a resposta do servo à desejada.

No entanto um conjunto de parâmetros do controlador só se adequa a uma situação específica de carga, trajectória e velocidade específicos, pelo que seria importante detectar os vários cenários possíveis para que o controlador se adaptasse. Tal seria possível pela medição do binário aplicado no motor que varia de acordo com a inércia presente.
Usando o sinal de saída do potenciómetro interno do servo, conhecido por estar relacionado com a sua posição, descobrimos que além de nos fornecer a posição também nos providencia a corrente corrente consumida pelo dispositivo, que por sua vez está relacionada com o binário aplicado ao motor. Desta forma poderíamos aplicar um controlador de força que adaptaria os parâmetros de controlo de acordo com a corrente drenada pelo motor. No entanto este sinal apresenta uma natureza bastante oscilatória, ainda mais com o controlo de posição, estando relacionada não só com o binário resultante da força gravítica como também do que resulta da velocidade e da aceleração. Para já desconhece-se qual seria o formato da lei de controlo deste adaptador, nem que parâmetros deverá assumir pela que mais investigação será necessária sobre este tópico.

Mesmo assim, nas experiências que temos realizado sobre as pernas do humanóide temos vindo a utilizar parâmetros fixos do controlador de posição, que claro, são bastante limitados para garantir a estabilidade em qualquer uma das juntas devendo, estes valores, corresponder ao pior caso. Infelizmente ficamos a perder, que em grande das juntas os parâmetros não são os mais optimizados introduzindo atrasos adicionais que por vezes podem ser piores do que se o controlador estivesse desligado (em malha aberta).
Efectuando ensaios sobre as pernas acabámos por verificar que as exigências em termos de binário são bastante inferiores às testadas com o servo isolado, com respostas de bastante qualidade mesmo em malha aberta. Apenas a estrutura mecânica das pernas se revelou incapaz de executar movimento com as cargas mais pesadas provocando folgas, que embora algumas fossem ajustáveis, outras revelaram-se sem solução com a necessidade de substituir peças. Esperemos que no trabalho futuro que se segue, estas contrariedades sejam levadas a sério não só no sentido de eliminar quaisquer folgas existentes, como também em aplicar

novas técnicas e materiais que atribuam robustez à estrutura, independentemente da solicitação imposta.

Outros aspectos que precisam de ser limados prendem-se com a alimentação dos actuadores e das unidades de controlo local. Mesmo com o uso de duas baterias de elevado fornecimento de corrente em paralelo, verifica-se esporadicamente um aparente *reset* de alguns PIC's, na medida que por vezes alguns servos deslocam-se para a posição original a alta velocidade. Como esta anomalia é muito esporádica iremos por de parte bugs do software.

- Considerando a hipótese de um *reset* por parte de um *slave*, nada deveria acontecer, uma vez que quando iniciam o seu funcionamento esperam por um comando do *master* e só depois aplicam o PWM com a posição válida, que deverá corresponder à posição actual. Se alguma coisa ocorrer é um deslocamento mínimo devido ao controlador PID. Por isso, este não deverá ser o problema.
- Se o *reset* ocorresse no *master*, todos os *slaves* dirigir-se-iam para as suas posições originais e não apenas um ou outro como se observa, pela que esta opção também é posta de parte.

Logo aparentemente, o problema não se deve a um *reset*. Eventualmente pode ser devido a informação incorrecta nas mensagens CAN. Apenas é estranho dado que as mensagens de actuação teriam de possuir valores 0 nos campos de posição e velocidade durante imenso tempo para serem visualizados os efeitos. Se se deve a isto, em princípio resolvendo bug do bloqueio do barramento CAN enunciado nas conclusões do capítulo 1, deveremos também resolver este problema.

Agradecimentos

Agradece-se toda a cooperação por parte do Departamento de Mecânica pela disponibilidade de recursos e toda a ajuda fornecida para a realização deste projecto, em especial atenção ao co-orientador de projecto Vítor Santos que nunca mediu esforços na resolução dos problemas que foram surgindo ao longo deste ano.

Bibliografia

Milton Ruas, Filipe M. T. Silva, Vítor M. F. Santos, "Parameter Measurement for Speed and Torque Control of RC Servomotors on a Small-Size Humanoid Robot", nas actas do Encontro Científico Robótica2006, Guimarães, 2006.

Milton Ruas, Filipe M. T. Silva, Vítor M. F. Santos, "Techniques for Velocity and Torque Control of RC Servomotors for a Humanoid Robot", aceite para publicação nos proceedings da 9th International Symposium on Climbing and Walking Robots and Associated Technologies, 11-14 September 2006

Milton Ruas, Filipe M. T. Silva, Vítor M. F. Santos, "A Low-Level Control Architecture for a Humanoid Robot", submetido à International Conference on Humanoid Robots 2006.

Vítor M. F. Santos, Filipe M. T. Silva, "Engineering Solutions to Build an Inexpensive Humanoid Robot Based on a Distributed Control Architecture", proceedings of the IEEE International Conference on Humanoid Robots 2005.

Vítor M. F. Santos, Filipe M. T. Silva, "Development of a Low-Cost Humanoid Robot: Components and Technological Solutions", proceedings of the CLAWAR 2005.

Luís Gomes, Mauro Silva, "Concepção e Desenvolvimento de Unidades de Percepção e Controlo para um Robot Humanóide", Relatório final de projecto 2004/05 – Departamento de Mecânica

Nuno Beça, Ângelo Cardoso, "Desenvolvimento e Integração das Sub-estruturas Inferior e Superior para a Locomoção de uma Plataforma Humanóide", Relatório final de projecto 2004/05 – Departamento de Mecânica.

K. S. Fu, R. C. Gonzalez, C. S. G. Lee; "Robotics: Control, Sensing, Vision and Intelligence", MacGraw-Hill, 1987

Índice de Figuras

Índice de Tabelas