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

1.ARQUITECTURA DAS COMUNICAÇÕES..... 3
1.1.ARQUITECTURA DO SISTEMA..... 5
1.1.1.Protocolos de Comunicação..... 6
1.2.COMUNICAÇÃO RS-232..... 8
1.2.1.Protocolo..... 8
1.2.2.Unidade Principal..... 12
1.2.2.1.Configuração da Unidade Principal..... 12
1.2.2.1.Device Drivers da Unidade Principal..... 14
1.2.2.2.Outros Device Drivers..... 18

1. ARQUITECTURA DAS COMUNICAÇÕES

Resumo:
Este capítulo descreve a organização distribuída da plataforma humanóide e como é realizada a comunicação entre os diversos nós de modo a poder trocar informação sensorial e de actuação sem o risco de colisões num barramento de natureza partilhada.

1.1. ARQUITECTURA DO SISTEMA

O sistema de controlo implementado é baseado numa configuração *master*/*slave*, e é constituído por três tipos de unidades 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 distribuir os comandos provenientes da unidade principal pelas diversas unidades locais (slaves), bem como direccionar os dados sensoriais provenientes dos slaves para a unidade principal.
- 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 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 1Mbps.

A unidade central de controlo ainda não está completamente definida, permanecendo em aberto soluções baseadas em PDA, placas de controlo genéricas (como as baseadas no padrão PC104) ou placas de controlo dedicadas (Fig. 1). 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(0) – por possuírem diversos periféricos e interfaces para redes de comunicações, incluindo o CAN (Fig. 3).

Fig. 1: Exemplo de uma board PC-104.

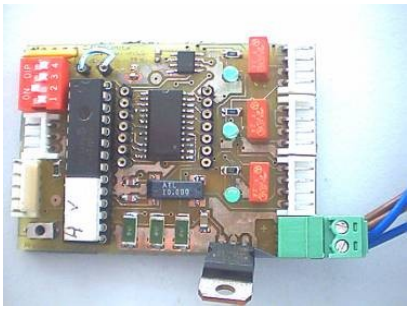

Fig. 2: Placa de controlo Master/Slave.

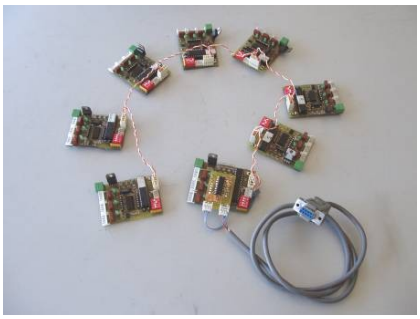

Fig. 3: Rede completa de Microcontroladores.

Até ao momento, a rede implementada é constituída por uma placa *master,* pelo qual designaremos por MCU (*Master Control Unit*), que efectua a interface entre a unidade principal e as unidades *slave*, e oito placas *slave*, designadas por SCU (*Slave Control Unit*), que efectuarão o controlo local 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. 2).

A organização enunciada na 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. Na presente data já é possível fazer a adaptação a irregularidades do solo, com as juntas a corrigir a sua posição de forma reactiva para que a projecção do centro de massa do robot não se situe fora da área de apoio dos pés. Obtém-se 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 podem ser de quatro origens diferentes:

- Potenciómetros internos aos servomotores, indicativos da sua posição;
- Extensómetros presentes na base de cada pé, para medição da força aplicada, para posterior cálculo do Centro de Pressão exercido;
- Acelerómetros/inclinómetros, que pela medição da aceleração da gravidade em duas componentes ortogonais, medem a inclinação do objecto onde se encontram localizadas;
- Giroscópios para medição da velocidade angular, para compensação das forças dinâmicas exercidas sobre o robot.

Estes valores são convertidos e registados localmente em cada unidade *slave* sendo depois enviados via CAN para o controlador *Master*. Os *Slaves* estão preparados também para receber mensagens via CAN. Estas mensagens consistem basicamente nos dados de actuação a aplicar sobre cada unidade local:

- Posições finais que os actuadores têm de tomar;
- Velocidade a que têm que se mover até atingir a posição final;
- Tipo de controlador em acção sobre as três juntas;
- Parâmetros de compensação para os algoritmos de compensação;
- Flags booleanas de controlo (ex.: PWM activados).

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) 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 *slave* respectivo.

Tabela 1: Características do Hardware.

Unidade Central de Controlo	**Computador com porta série RS-232** • Software de suporte: MatLab 7.xx • Device Drivers de comunicação série: *mini-toolbox* 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 dupla prioridade nas interrupções. • Diversos periféricos: *timers*, módulos CCP, interfaces para redes de comunicação, ADC, etc…

1.1.1. 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** estimada de cada servomotor (em graus/s);
- **Corrente** consumida por cada servomotor;
- Valores dos **sensores de força** de cada pé (quatro sensores por pé);
- Saída dos **giroscópios** (em graus/s);
- Saída dos **inclinómetros** (em graus).

Quanto aos dados de actuação:

- **Posição referência** a atingir para cada componente (ângulos para os três servomotores ou as 3 componentes x, y, z do sistema de coordenadas cartesiana);
- **Velocidade** média para a realização da trajectória até à posição final;
- **Tipo de controlador** activo (centro de pressão, inclinação, ou velocidade angular);
- **Parâmetros de compensação** para o controlador activo (K) e para o controlo local (K_P, K_I);
- **Flags booleanas** de des/activação (ex.: PWMs).

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. Tal é útil no último caso, pois cada junta pode estar sujeita a diferentes esforços, e por isso, é rentável aplicar diferentes ganhos a cada uma.

Um dado importante a ter em conta, é a multiplicidade de significados que cada dado de actuação pode possuir, dependendo do tipo de controlador activo. Assim, para cada tipo de controlador, a posição referência diz respeito ao valor final que o respectivo controlador deve atingir. Quatro tipos de controladores de primeiro nível são implementados:

- **Sem controlador**: a posição referência diz respeito às três posições angulares a aplicar a cada junta, sendo directamente aplicadas sobre o controlador local (ver abaixo);
- **Controlo de Centro de Pressão**: a posição referência diz respeito ao centro de pressão que a estrutura deve possuir no fim do movimento.
- **Controlo de Inclinação**: a posição referência especifica a inclinação que a estrutura deverá possuir.
- **Controlo de Velocidade Angular**: (análogo aos restantes).

De igual modo, a velocidade indicada diz respeito ao controlador em causa, implementado-se, desta forma, trajectórias polinomiais que tanto podem ser de posição angular, centro de pressão, inclinação ou velocidade angular.

Com base nestas posições referência o controlador de primeiro nível determina as posições angulares a aplicar a cada servomotor, que posteriormente será entregue a um controlador local do tipo PI (Proporcional+Integrador) que se responsabilizará por garantir o alcance das posições solicitadas. O ajuste do controlador de primeiro nível é feito através do ganho K, enquanto que para o controlador local o ajuste é feito através dos ganhos K_I e K_P.

No que diz respeito aos dados sensoriais enviados à unidade Master, eles são de natureza constante e independentes do controlador activo.

1.2. COMUNICAÇÃO RS-232

1.2.1. 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 colocação das juntas nas posições *A*, *B* e *C*, 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 seis bytes. O primeiro byte sinalizará a mensagem como sendo um comando de solicitação à unidade *master* – MESSAGE_REQ (ver Tabela 5); o segundo byte conterá um código (*opcode*) indicativo da operação a realizar, do SCU alvo e de parâmetros adicionais; os três bytes seguintes conterão parâmetros a atribuir às três juntas no caso de um comando de actuação, e finalmente o byte BCC indicará a validade da mensagem (como não tendo sofrido corrupção).

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
SOF: *0x*FF	OpCode	Joint 1	Joint 2	Joint 3	BCC

Mensagem USART PC→Master de actuação.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
SOF: *0x*FF	OpCode	0	0	0	BCC

Mensagem USART PC→Master de leitura sensorial.

Tabela 2: Campos das mensagens PC→Master via USART

Pacotes	Função:	Valores possíveis:
SOF	(S*tart* O*f* F*rame*) Indica o tipo da mensagem	(Ver Tabela 5).
OpCode	Código indicando o que é solicitado e a quem se destina.	(Ver Tabela 3)
Joint 1/2/3	Dados de actuação.	• Mensagem de actuação: Dados de actuação a atribuir a cada componente de um determinado SCU. • Mensagem de leitura sensorial: campos nulos.
BCC	Block Check Code Verificação da integridade da mensagem.	$BCC = \sum bytes \text{‰} 256$

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

A Tabela 3 descreve a estrutura do byte *OpCode*, bem como todos os seus valores possíveis. Os três bytes de dados (Joint 1/2/3) dizem respeito à operação indicada neste byte.

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 componentes do SCU alvo (SCU id).	*PARAM_POSITION* (*0b*00) Posição referência a ser atingida.
		PARAM_VELOCITY (*0b*01) Velocidade média do movimento a efectuar.
		PARAM_SPECIAL (*0b*11) Des/Activação do PWM aplicado aos motores e da filtragem dos valores sensoriais do servo.
	OP_APPLY_CONTROL (*0b*01) Mensagem de actuação sobre as três juntas do SCU alvo com a definição dos parâmetros dos controladores.	*PARAM_KI* (*0b*00) Ganho da componente integral do controlador local.
		PARAM_KP (*0b*01) Ganho da componente proporcional do controlo local.
		PARAM_K (*0b*10) Ganho do controlador de primeiro nível.
		PARAM_CONTROLON (*0b*11) Tipo de controlo (de primeiro nível) a aplicar na junta.
	OP_READ_SENSORS (*0b*10) Mensagem de leitura dos sensores.	*PARAM_POSITION* (*0b*00) Posição actual de cada junta.
		PARAM_VELOCITY (*0b*01) Velocidade estimada de cada junta.
		PARAM_CURRENT (*0b*10) Corrente drenada por cada servo.
		PARAM_SPECIAL (*0b*11) Saída dos sensores especiais.
	OP_READ_EXTBUFF (*0b*11) Leitura do buffer externo do Master.	*Status* do barramento CAN.

De notar que a posição referência e a velocidade média da trajectória depende do controlador de primeiro nível activo (seleccionado através do parâmetro *PARAM_CONTROLON*). Os vários tipos de controlador estão indicados na Tabela 4.

Tabela 4: Tipo de controlo a seleccionar no campo PARAM_CONTROLON.

Tipo de Controlo sobre as juntas	***Designação***	***PARAM_CONTROLON***
Sem Controlo de primeiro nível	*NO_CONTROL*	*0b*00
Controlo de Centro de Pressão	*COP_CONTROL*	*0b*01
Controlo de Inclinação	*INC_CONTROL*	*0b*10
Controlo de velocidade angular	*GIRO_CONTROL*	*0b*11

De notar, que a actuação é feita directamente às três juntas/componentes 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 7 bytes, cuja estrutura é a seguinte:

- *SOF*: possui o valor *MESSAGE_SUCESS* (*0x*FB) no caso de uma resposta com sucesso;
- *OpCode*: *opcode* utilizado pela mensagem original PC→Master;
- Data 1-4: dados sensoriais no caso de um pedido de consulta sensorial;
- BCC: Validação da mensagem.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*
*0x*FD	OpCode	Data 1	Data 2	Data 3	Data 4	BCC

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 (*OP_APPLY_**), os bytes 2-5 possuem o mesmo valor que a mensagem original com o sexto byte nulo;
- ➔ Numa leitura sensorial (*OP_READ_SENSORS*), o byte *OpCode* é igual à da mensagem original com os bytes *Data* 1-4 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 especiais, são utilizados todos os quatro bytes para conter a informação de um dos conjuntos dos sensores – sensores de força (de um pé), inclinómetros ou giroscópios.
- ➔ Se for solicitada a leitura do buffer externo do master (*OP_READ_EXTBUFF*), o estado do barramento CAN (percepcionado pelo *master*) é devolvido.

O formato das mensagens de resposta descritos apresentam-se de seguida (== significa que o byte é o mesmo do da mensagem de solicitação):

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*
SOF: *0x*FD	==	==	==	==	0	BCC

Mensagem de actuação aplicada com sucesso

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*
SOF: *0x*FD	==	Joint 1	Joint 2	Joint 3	Status	BCC

Mensagem de leitura sensorial das juntas

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*
SOF: *0x*FD	==	Data 1	Data 2	Data 3	Data 4	BCC

Mensagem de leitura dos sensores especiais

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*
SOF: *0x*FD	==	Error Code	# TX errors	# RX errors	0	BCC

Mensagem de leitura do buffer externo do Master (status 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 – posição, velocidade média ou corrente.
Quanto à leitura dos sensores especiais, note também, a limitação de quatro valores, daí a limitação imposta anteriormente de apenas usar quatro linhas analógicas dedicadas a este género de sensores: ou o conjunto dos sensores de força, ou inclinómetros, ou giroscópios.

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 SOF de código *MESSAGE_INVREQ* (*0x*FC), com todos os restantes bytes iguais à da mensagem original, é retornada ao PC.

➔ Mensagem corrompida: o código BCC não está de acordo com a estrutura da mensagem. Neste caso uma mensagem de SOF igual a *MESSAGE_INVALID* (0xFB) é retornado com os restantes bytes iguais à da mensagem recebida.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*
SOF: *0x*FC	==	==	==	==	0	BCC

Mensagem sinalizadora de um pedido inválido.

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*
SOF: *0x*FB	*0x*FE	*0x*FE	*0x*FE	*0x*FE	*0x*FE	BCC

Mensagem sinalizadora de um erro na comunicação PC→Master.

Teste das comunicações USART entre PC e *Master*:

Para confirmação da correcta comunicação entre o PC e o Master pode ser enviada uma mensagem de teste igualmente de 6 bytes com o SOF como MESSAGE_TEST (0xFE).

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*
SOF: *0x*FE	X2	X3	X4	X5	BCC

Mensagem USART PC→Master de teste.

Os bytes 2 a 5 podem possuir qualquer valor, desde que o byte BCC esteja de acordo com eles. Por sua vez, o *master* deverá responder com uma mensagem de 7 bytes possuindo os mesmos valores que a primeira:

byte 1	*byte 2*	*byte 3*	*byte 4*	*byte 5*	*byte 6*	*byte 7*
SOF: *0x*FE	X2	X3	X4	X5	0	BCC

Resposta Master→PC a uma mensagem de teste.

A Tabela 5 apresenta sintetizados os diversos tipos de mensagens apresentados nas secções anteriores, com a descrição do byte SOF utilizado para cada caso.

Tabela 5: Tipos de mensagens USART (primeiro byte de cada frame).

Tipo de Mensagem	***Designação***	***Código***	***Descrição***
Mensagem de solicitação	*MESSAGE_REQ*	*0x*FF	Solicitação de dados do PC para o *Master*.
Mensagem de teste	*MESSAGE_TEST*	*0x*FE	Envio de uma mensagem de teste das comunicações.
Mensagem de sucesso	*MESSAGE_SUCESS*	*0x*FD	Resposta a uma mensagem de solicitação (MESSAGE_REQ), indicando que o comando foi executado com sucesso.
Parâmetros inválidos	*MESSAGE_INVREQ*	*0x*FC	Pedido com parâmetros inválidos.
Mensagem inválida	*MESSAGE_ERROR*	*0x*FB	Mensagem de estrutura inválida (BCC incorrecto).

1.2.2. Unidade Principal

1.2.2.1. 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 oferece *device-drivers* para troca directa por RS-232 tanto de caracteres como de *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 utilizada: 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	2 caracteres (ms)
Tempo de timeout para a recepção/transmissão de uma mensagem	10 ms

Tabela 6: Configurações gerais do *cport*.

```
state=cportconfig(handler,'BaudRate',115200,                          ...
                          'ByteSize',8,                               ...
                          'fNull','OFF',                              ...
                          'fDtrControl','OFF',                        ...
                          'fRtsControl','OFF',                        ...
                          'fAbortOnError','ON',                       ...
                          'ErrorChar',-1,                             ...
                          'ReadIntervalTimeout',2*11/baudrate*1000,   ...
                          'ReadTotalTimeoutMultiplier',1,             ...
                          'ReadTotalTimeoutConstant',10,              ...
                          'WriteTotalTimeoutMultiplier',1,            ...
                          'WriteTotalTimeoutConstant',10);
```

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 *ASCII* (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 7: 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 é 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 7;
3. Verificar a conectividade (através de uma mensagem de teste) 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. Anotar as configurações tomadas fazendo *stat=cportconfig*(*handler*).
6. Redefinir as configurações do *cport* com estes dados (Tabela 6).

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.

Para simplificação, duas funções foram criadas para automatizar o processo de conexão e desconexão com as configurações citadas anteriormente. Elas são as seguintes:

```
% Estabelecimento de uma nova ligação via RS-232
[handler,error,errorstr]=initcom(gate,baudrate)

% Término de uma ligação RS-232 existente.
[error,errorstr]=killcom(handler)
```

1.2.2.1. 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,error,errorstr]=initcom(gate,rate)`	Criação de uma nova ligação.
killcom.m	`[error,errorstr]=killcom(H)`	Término da ligação.
testcom.m	`[error,errorstr]=testcom(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.
readspecial.m	`[special,...]=readspecial(H,scu_id)`	Leitura dos sensores especiais.
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 8: Lista de device drivers da unidade principal.

initcom

Estabelecimento de uma nova ligação via RS-232.

```
[handler,error,errorstr]=initcom(gate,rate)

Entradas:
 gate     -> Porta a utilizar (1,2,...)
 rate     -> baudrate a definir (bits/s)
Saídas:
 handler  -> ID da linha de comunicações
 error    -> Código de erro
 errorstr -> String descritiva do erro
```

killcom

Término de uma ligação RS-232 existente.

```
[error,errorstr]=killcom(handler)

Entradas:
 handler -> ID da linha série
Saídas:
 error    -> Código de erro
 errorstr -> String descritiva do erro
```

testcom

Pedido de envio de uma sequencia de teste.

```
[error,errorstr]=testcom(H)

Entradas:
 handler  -> ID da linha série.
Saídas:
 error    -> Código de erro
 errorstr -> String descritiva do erro
```

readcanstat

Leitura do estado do barramento CAN.

```
[array,state,rx,error,errorstr]=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
```

readjoint

Leitura de um parâmetro sensorial de um SCU.

```
[sensors,state,rx,error,errorstr]=readjoint(H,scu_id,param)

Entradas:
H        => Handler para comunicar com o Master
scu_id   => Identificador do SCU alvo
param    => Parametro a ler (Tabela 9)

Saídas:
sensors  => Parametros sensoriais
state    => Bits de estado dos servos
rx       => Mensagem de baixo nível recebida
error    => Código de erro, se existente
errorstr => String descritiva do erro
```

Tabela 9: Valores possíveis do parâmetro *param* na função *applyjoint*.

Campo *param*		*Descrição*	Campo *servos*
PARAM_POSITION	0	Posição angular (*pos*) de cada junta.	[pos_1, pos_2, pos_3]
PARAM_VELOCITY	1	Velocidade (*vel*) estimada de cada junta.	[vel_1, vel_2, vel_3]
PARAM_CURRENT	2	Corrente (*curr*) drenada por cada servo.	[$curr_1$, $curr_2$, $curr_3$]

readspecial

Leitura dos sensores especiais de um SCU.

```
[special,rx,error,errorstr]=readspecial(H,scu_id)

Entradas:
H      => Handler das comunicações com o Master
scu_id => SCU alvo

Saídas:
special  => Valores dos sensores especiais (4 valores)
rx       => Mensagem de baixo nível recebida
error    => Código de erro, se existente
errorstr => String descritiva do erro
```

applyjoint

Aplicação de uma ordem de actuação a cada componente de um SCU.

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

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Identificador do SCU alvo
   param    => Parametro a aplicar (Tabela 10)
   servos   => Dados a aplicar
Saídas:
   rx       => Mensagem de baixo nível recebida
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

Tabela 10: Valores possíveis do parâmetro *param* na função *applyjoint*.

Campo *param*		*Descrição*	Campo *servos*
PARAM_POSITION	0	Posição referência (*ref*) a ser atingida.	[ref_1, ref_2, ref_3]
PARAM_VELOCITY	1	Velocidade média (*vel*) do movimento a efectuar.	[vel_1, vel_2, vel_3]
PARAM_SPECIAL	3	Activação/desactivação do PWM aplicado aos motores.	[0/1, x, x]

applycontrol

Selecção do controlador de primeiro nível, e ajuste dos seus parâmetros bem como dos do controlador local.

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

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Identificador do SCU alvo
   param    => Parametro a modificar (Tabela 11)
   servos   => Dados a aplicar
Saídas:
   rx       => Mensagem de baixo nível recebida
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

Tabela 11: Valores possíveis do parâmetro *param* na função *applycontrol*.

Campo *param*		*Descrição*	Campo *servos*
PARAM_KI	0	Ganho da componente integral (*Ki*) do controlador local.	[Ki_1, Ki_2, Ki_3]
PARAM_KP	1	Ganho da componente proporcional (*Kp*) do controlo local.	[Kp_1, Kp_2, Kp_3]
PARAM_K	2	Ganho (*K*) do controlador de primeiro nível.	[K_1, K_2, K_3]
PARAM_CONTROLON	3	Tipo de controlo de primeiro nível (*Type*) a aplicar em cada junta (Tabela 12).	[$Type_1$, $Type_2$, $Type_3$]

Tabela 12: Tipos de controladores de primeiro nível (a aplicar com o parâmetro *PARAM_CONTROLON* na função *applycontrol*).

Tipo de Controlo (*Type*)		*Descrição*
NO_CONTROL	0	Sem Controlo de primeiro nível
COP_CONTROL	1	Controlo de Centro de Pressão
INC_CONTROL	2	Controlo de Inclinação
GIRO_CONTROL	3	Controlo de velocidade angular

1.2.2.2. Outros *Device Drivers*

Outras funções estão também disponíveis, não sendo nada mais que operações de nível intermédio que fazem uso dos *device drivers* apresentados na secção anterior (secção 1.2.2.1).

Ficheiro	*Função*	*Descrição*
inituc.m	`[error,errorstr]=inituc(H,scu)`	Activação do PWM para vários SCUs.
killuc.m	`[error,errorstr]=killuc(H,scu)`	Desactivação do PWM para vários SCUs.
sensor_reset.m	`[stat,error,errorstr]=sensor_reset(H,scu)`	Recalibração dos sensores de um SCU.
statmotionfin.m	`[finall,finone,fin,pwm,...]=statmotionfin(H,scu)`	Consulta do estado da execução de trajectórias por parte de um SCU.
motionstart.m	`[stat,error,errorstr]=motionstart(H,scu)`	Espera que um SCU inicie a execução do movimento.
motionfin.m	`[stat,error,errorstr]=motionfin(H,scu)`	Espera que um SCU termine a execução do movimento.
waitmotionfin.m	`[stat,error,errorstr]=waitmotionfin(H,scu)`	Espero que o processo de iniciação e término do movimento seja executado.
waitpwmon.m	`[stat,error,errorstr]=waitpwmon(H,scu)`	Espera que o PWM seja activado nas três juntas de um SCU.

Tabela 13: Lista de device drivers da unidade principal.

inituc

Activação dos PWMs de uma série de SCUs.

```
[error,errorstr]=inituc(H,scuid)

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Conjunto de SCUs alvo (endereços)
Saídas:
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

killuc

Desactivação dos PWMs de uma série de SCUs.

```
[error,errorstr]=killuc(H,scuid)

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Conjunto de SCUs alvo (endereços)
Saídas:
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

sensor_reset

Recalibração sensorial de um conjunto de SCUs.

```
[stat,error,errorstr]=sensor_reset(H,scuid)

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Conjunto de SCUs alvo (endereços)
Saídas:
   stat     => Indicador de sucesso na operação
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

statmotionfin

Leitura do estado de um conjunto de SCUs relativo à execução de trajectórias.

```
[finall,finone,fin,pwm,err,errstr]=statmotionfin(H,scuid)

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Conjunto de SCUs alvo (endereços)
Saídas:
   finall   => Todos os SCUs indicados terminaram uma trajectória
   finone   => Pelo menos um SCU dos indicados terminaram uma trajectória
   fin      => Vector indicando se a junta i (de 3) de todos os SCUs terminaram
   pwm      => Todos os SCUs indicados possuem os seus sinais de PWM activos
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

motionstart

Espera que todos os SCUs indicados na lista comecem a execução de uma trajectória. É necessário que previamente tenha sido enviado um comando de actuação para início do movimento (*applyjoint*).

```
[stat,error,errorstr]=motionstart(H,scuid)

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Conjunto de SCUs alvo (endereços)
Saídas:
   stat     => Indicador de sucesso na operação
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

Efeitos colaterais: caso a posição das juntas indicado no comando de actuação corresponda à actual, este comando ficará bloqueado. Como solução pode-se adicionar um *timer* para impor um *timeout*.

motionfin

Espera que todos os SCUs indicados na lista terminem a execução de uma trajectória.

```
[stat,error,errorstr]=motionfin(H,scuid)

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Conjunto de SCUs alvo (endereços)
Saídas:
   stat     => Indicador de sucesso na operação
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

waitmotionfin

Espera que todos os SCUs indicados na lista comecem e terminem a execução de uma trajectória. Esta operação é a conjunção das operações *motionstart* e *motionfin*. É necessário que previamente tenha sido enviado um comando de actuação para início de um movimento (*applyjoint*).

```
[stat,error,errorstr]=waitmotionfin(H,scuid)

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Conjunto de SCUs alvo (endereços)
Saídas:
   stat     => Indicador de sucesso na operação
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```

Efeitos colaterais: caso a posição das juntas indicado no comando de actuação corresponda à actual, este comando ficará bloqueado. Como solução pode-se adicionar um *timer* para impor um *timeout*.

waitpwmon

Espera que todos os SCUs indicados na lista possuam os seus sinais de PWM (actuação sobre os motores) activados. É necessário que anteriormente tenha sido enviado um comando de activação de PWMs (ex.: *inituc* ou *applyjoint*).

```
[stat,error,errorstr]=waitpwmon(H,scuid)

Entradas:
   H        => Handler para comunicar com o Master
   scu_id   => Conjunto de SCUs alvo (endereços)
Saídas:
   stat     => Indicador de sucesso na operação
   error    => Código de erro, se existente
   errorstr => String descritiva do erro
```