Instituto de Sistemas e Robótica

Pólo de Lisboa

XMOV - Controlo de Velocidade para a Plataforma Móvel ROBUTER

João Serrano Pedro Raposo Pedro Aparício
João Sequeira M. Isabel Ribeiro

22 de Novembro de 1995

RT-601-95

ISR-Torre Norte
Av. Rovisco Pais
1096 Lisboa CODEX
PORTUGAL

Resumo

Este relatório, situado na área da robótica móvel terrestre, descreve o estudo e implementação de um comando de movimento dotado de uma maior gama dinâmica do que os já existentes no sistema operativo Albatros^{TM} utilizado na plataforma ROBUTER^{TM}. Neste sistema, as instruções existentes apenas permitem impor a cada uma das suas rodas de tracção velocidades pertencentes ao intervalo -10dm/s a 10dm/s e com um grau de quantificação máximo de 25mm/s

Esta restrição impossibilita a realização de movimentos precisos essenciais à realização de operações de acostagem a mesas de trabalho em missões de transporte automático de materiais, à movimentação em ambientes com pouco espaço livre e ao desvio de obstáculos inesperados.

O novo comando desenvolvido é compatível com os comandos já existentes no sistema operativo da ROBUTER, para assegurar a sua utilização em moldes idênticos aos dos comandos nativos do Albatros^{TM}.

O relatório descreve a metodologia que presidiu à implementação do novo comando, **xmov**, e apresenta resultados experimentais da movimentação da plataforma móvel usando o referido comando.

Índice

Lista de Figuras

Lista de Tabelas

1 Introdução

O presente trabalho foi realizado no âmbito da disciplina de Projecto e Realização de Sistemas de Controlo (94/95), do ramo de Controlo e Robótica da Licenciatura em Engenharia Electrotécnica e de Computadores do Instituto Superior Técnico. O trabalho decorreu nas infraestruturas do Laboratório de Robótica Móvel do Instituto de Sistemas e Robótica (pólo do I.S.T.).

1.1 Objectivos do trabalho

Este trabalho situa-se na área da robótica móvel terrestre e o seu objectivo consiste no estudo e implementação de um comando de movimento dotado de uma maior gama dinâmica do que os já existentes no sistema operativo Albatros^{TM} utilizado na plataforma ROBUTER^{TM}. Neste sistema, as instruções existentes apenas permitem impor a cada uma das suas rodas de tracção velocidades pertencentes ao intervalo -10dm/s a 10dm/s e com um grau de quantificação máximo de 25mm/s [1]. Esta restrição impossibilita a realização de movimentos precisos essenciais à realização de operações de acostagem a mesas de trabalho em missões de transporte automático de materiais, à movimentação em ambientes com pouco espaço livre e ao desvio de obstáculos inesperados. O comando a desenvolver tem ainda que ser compatível com os comandos presentes no sistema operativo, para assegurar a sua utilização em moldes idênticos aos dos comandos nativos do Albatros^{TM}.

1.2 Plataforma móvel de teste

1.2.1 Características

A ROBUTER^{TM} é uma plataforma rectangular (ver Figura 1) com 1025mm de comprimento, 680mm de largura e 440mm de altura. A distância entre eixos é de 587mm [2]. O seu peso, com baterias, é de 150kg, possuindo uma capacidade de carga de 120kg . Possui duas rodas traseiras accionadas por dois motores DC de 300W de 48V, com 3000rpm no primário e 130rpm à saída da caixa de redução, e dois casters (rodas livres) dianteiros. A alimentação eléctrica é assegurada por baterias de 12V 60Ah. O sistema de localização por odometria é suportado por encoders ópticos com erro inferior a 1mm. As fontes de alimentação são de 5V 16A e +/- 12V 1.7A. A plataforma dispõe de dois processadores: um MC68020 (32 bits), funcionando a 16MHz, dedicado ao sistema operativo e ao processamento em geral, e um processador secundário MC68000 (16bits), funcionando a 8MHz dedicado ao controlo do sistema de ultrasons. A comunicação entre os processadores é feita através de um bus VME. A comunicação com o exterior é feita por duas portas série RS232, colocadas na carta do microprocessador principal.

1.2.2 Sistema Operativo Albatros^{TM}

Na plataforma onde irá ser desenvolvido o comando de velocidade corre um sistema operativo multiprocessamento em tempo real (SOTR), o Albatros^{TM} 4.0 [2] . O sistema foi concebido para fazer o controlo de dispositivos multi-eixos. A arquitectura do sistema não permite que sejam implementados controladores independentes para cada um dos eixos, pois têm que partilhar o mesmo processador. Este facto impossibilita a actuação simultânea dos controladores. Este sistema caracteriza-se pela sua modularidade, o que simplifica a adição de outros dispositivos. Está organizado em níveis e como as interfaces das chamadas ao sistema e dos programas de aplicação estão bem definidas, é possível adicionar novos comandos ao sistema, minimizando as modificações em níveis inferiores (ver Figura 2).

[1] Este consegue-se utilizando o comando do sistema $MOTN$ [6] .
[2] valor medido no laboratório, durante a realização do trabalho, com uma fita métrica.

Figura 1: Plataforma Robuter^{TM}

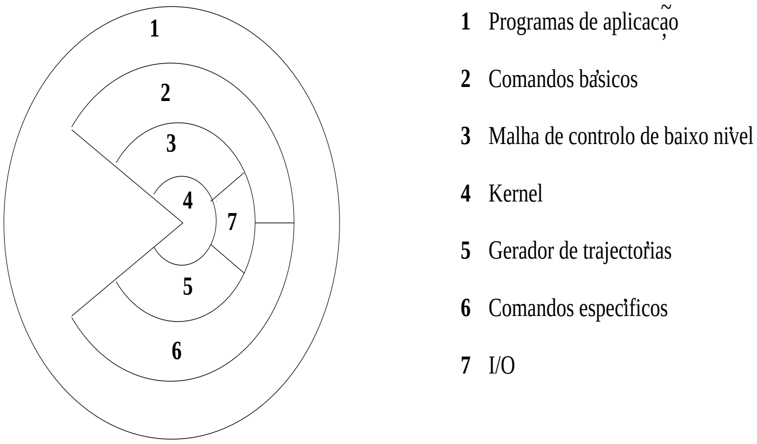

Figura 2: Organização do sistema operativo Albatros^{TM}

Este tipo de sistema operativo (SOTR) é usado, principalmente, em aplicações em que há necessidade de multiprocessamento, com várias aplicações a acederem aos mesmos recursos "simultaneamente". Nestes sistemas a importância relativa dos diferentes processos é definida pelo utilizador, podendo ser variável no tempo.

Aplicações em que é necessário o recurso a sistemas operativos em tempo real surgem em:

- Sistemas médicos, no controlo de equipamento sofisticado de diagnóstico em que se usa o sitema operativo OS9;
- Aplicações industriais, na gestão de processos complexos (p.e., projecto AIMBURN[3]);
- Aplicações militares, em sistemas de guiamento de mísseis;
- Aeronáutica, no controlo dos elementos dinâmicos de um avião (leme, flaps, reactores);
- Sistemas Robóticos, em plataformas móveis, manipuladores.

No caso presente, em que está envolvido o controlo da navegação de um robot móvel, é necessário controlar vários sub-sistemas (roda direita, roda esquerda, gerador de trajectórias, sonares, ...). Cada um destes sub-sistemas interage com o sistema operativo e exige respostas em tempo real.

O software do sistema pode ser dividido em duas partes distintas: (i) programas do sistema e (ii) programas de aplicação.

- (i) Primitivas que permitem ao utilizador aceder ao baixo nível do sistema. Esta parte inclui o kernel, o gerador de trajectórias, os módulos de leitura dos sensores e o interpretador de comandos.
- (ii) Programas desenvolvidos pelo utilizador numa linguagem de alto nível (C).

A comunicação entre os programas do sistema e os de aplicação é efectuada através do envio de comandos parametrizados em ASCII, permitindo que, do ponto de vista do sistema operativo, seja indiferente o tipo de canal de comunicação. Assim, os programas de aplicação podem utilizar o mesmo processador que o ALBATROS^{TM} ou podem ser executados num terminal, comunicando com o interpretador de comandos via modem ou cabo conectados através das portas série (ver Figura 3).

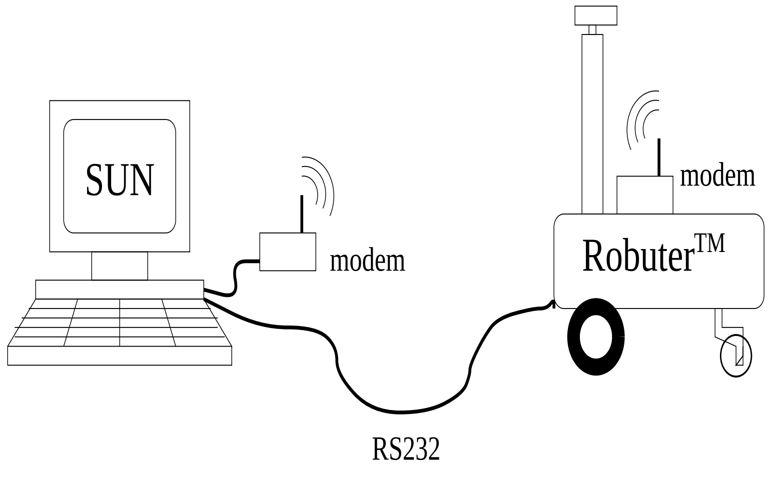

Figura 3: Disposição do sistema Plataforma-WorkStation

Utilização de processos

No Kernel da versão 4.0 do ALBATROS^{TM}, existem 3 tipos de processos:

[3]Projecto desenvolvido pelo IST para controlo de um forno de vidro.

- Processos do sistema (tipo 1)
- Processos ininterruptíveis (tipo 2)
- Processos interruptíveis (tipo 3)

Para construir aplicações, o utilizador tem à sua disposição os dois últimos.

Os processos do tipo 2, embora possam ser usados pelo utilizador, podem levar a situações de risco uma vez que, quando lançados, o sistema operativo fica sem o controlo total das operações. Com um processo deste tipo em execução, o sistema não consegue responder a situações de emergência. Os processos do tipo 3 são, portanto, aconselhados para o desenvolvimento de aplicações. Foi este o tipo de processo utilizado no presente trabalho.

As principais características dos processos do tipo 3 são:

- A possibilidade de o utilizador definir a periodicidade com que o processo deve ser executado;
- A definição de prioridades, o que permite que um processo com maior prioridade interrompa um outro com menor, quando o seu período o impõe;
- A possibilidade de vários processos poderem ser lançados na mesma instrução, o que permite fazer uma sincronização dos mesmos;
- A possibilidade de se associar um "offset" a cada processo, indicando o número de períodos de relógio que este deve aguardar antes de ser lançado;
- Poderem ser retirados na mesma instrução, havendo a garantia de que todos serão suspensos no próximo ciclo do relógio;
- Poderem retirar outro processo sem risco de interromper a aplicação em curso;
- Poderem retirar-se a si próprios.

Note-se que dois processos com a mesma prioridade são executados sequencialmente, não se interrompendo mutuamente.

A cada ciclo do sistema gestor de processos (cujo período pode variar de 5ms a 20ms, dependendo do processador), são realizadas as seguintes operações:

1. É gerada uma interrupção.
2. É accionado o sistema de gestão de processos.
3. É accionado o gestor de processos ininterruptíveis do utilizador, o que leva à suspensão dos processos em curso.
4. É activado o gestor dos processos interruptíveis, que escolhe qual o processo a lançar de seguida.

O programa *jogo.c* evidencia o uso de processos com diferentes prioridades. Neste, a função principal lança dois processos filhos, um destinado a fazer o controlo de velocidade em função dos comandos introduzidos pelo utilizador e outro com o objectivo de fazer a leitura da posição da plataforma e envio dos dados pela porta série. Os processos em causa comunicam entre si por partilha de memória (1 variável). O acesso a esta variável não necessita de semáforo uma vez que apenas um dos processos a acede para escrita. Os restantes apenas acedem para leitura.

1.2.3 Sistema de Controlo da Plataforma

O esquema de controlo de posição em malha fechada utilizado no ALBATROS^{TM} está representado na Figura 4. A entrada de referência q_f é a posição final que, por comparação com a posição actual, gera o sinal de erro e_1 que vai actuar sobre o gerador de trajectórias. Este gera um conjunto de posições intermédias q_i que, ao serem comparadas com o valor dos sensores, originam o erro e_2. Este erro é a entrada da malha de controlo de baixo nível, um controlador PID de ganhos ajustáveis, que tem, como saída, o sinal de comando dos actuadores m_i.

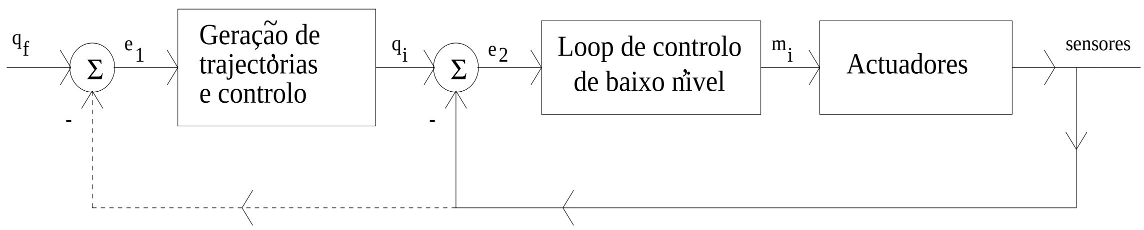

Figura 4: Esquema de controlo da plataforma Robuter^{TM}

1.3 Metodologia implementada

Inicialmente foi feito um estudo dos níveis inferiores do sistema operativo da plataforma, mais precisamente dos campos pertencentes às estruturas de dados utilizados por alguns dos processos básicos desse sistema, por forma a determinar quais os campos relevantes para a realização do trabalho. Seguidamente foram desenvolvidos programas, de teste nos quais se atribuiam valores a estes campos, com vista a fazer movimentar a plataforma utilizando os níveis inferiores do sistema operativo. Estas experiências permitiram a concepção da estratégia de controlo implementada no programa final.

1.4 Organização do relatório

Na secção 2 é descrito o desenvolvimento de software. Na secção 3 é detalhada a metodologia seguida ao longo do trabalho. Na secção 4 é descrito o método utilizado para analisar as estruturas existentes no sistema, os resultados obtidos e respectivas conclusões. A secção 5 descreve o comando implementado e como este pode ser usado pelo utilizador. Na sub-secção 5.3 encontram-se os resultados obtidos. Na secção 6 podem ver-se as conclusões. Direcções de pesquisa para trabalho futuro, sobre o comando desenvolvido encontram-se na secção 7.

2 Desenvolvimento do software

Como foi referido na sub-secção 1.1 pretende-se que o comando desenvolvido seja perfeitamente compatível com os comandos existentes de raíz e que o utilizador possa trabalhar com ele tal como faz com os comandos originais. Deste modo, o esquema da Figura 2 fica representado na forma da Figura 5.

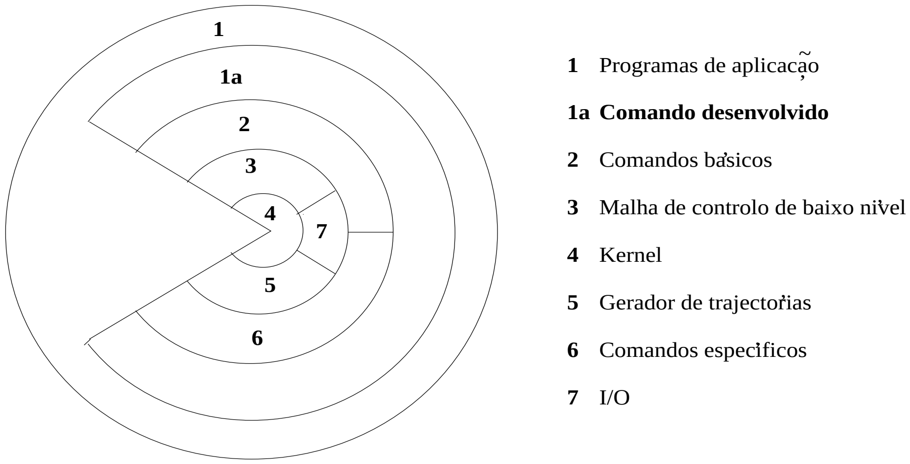

Figura 5: Organização do Albatros^{TM} com o comando implementado

O comando desenvolvido deixa o acesso aos níveis inferiores tal como estava, formando um "super-set" do sistema, totalmente *transparente para o utilizador*.

2.1 Software para a plataforma

O desenvolvimento, compilação, linkagem e carregamento do software destinado a ser executado na plataforma segue o esquema representado na Figura 6.

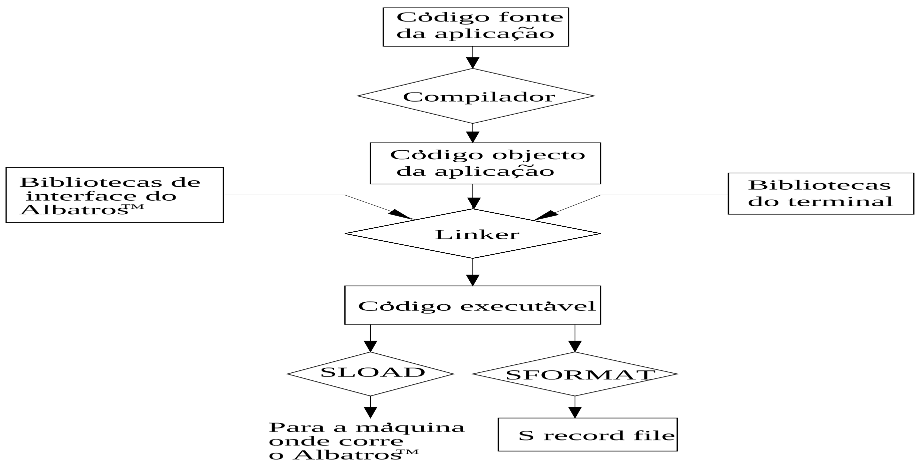

Figura 6: Esquema de desenvolvimento de Software para executar na Robuter^{TM}

Foi criado um ficheiro Makefile que gere a compilação e linkagem de todos estes módulos. Este ficheiro permite, para além da compilação e linkagem do software, que se faça a transferência do software para a plataforma utilizando uma das portas série disponíveis.

2.2 Software para a WorkStation

No caso de software destinado unicamente a ser *executado na WorkStation*, este diagrama fica simplificado, não sendo incluídas as bibliotecas do Albatros^{TM} e terminando no código executável na WorkStation (ver Figura 7). Neste caso não se faz uso de uma makefile, ficando a compilação reduzida a: **cc ficheiro_fonte.c -o ficheiro_executável**.

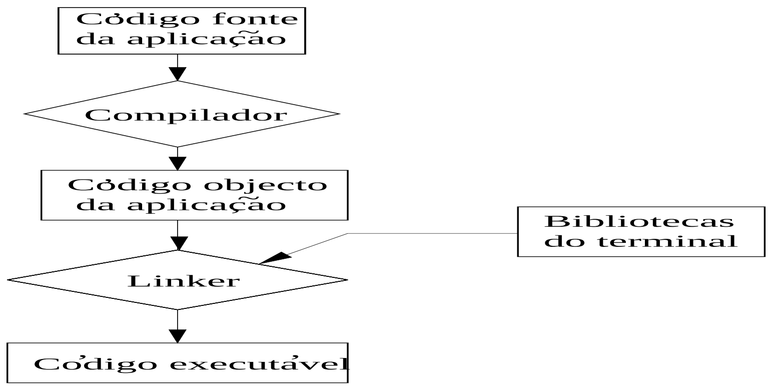

Figura 7: Esquema de compilação de software para executar na WorkStation

2.3 Comunicações Plataforma $\leftrightarrow$ WorkStation

A necessidade de verificar os resultados obtidos com o comando desenvolvido exigiu a criação de programas específicos para uso dos canais de comunicações existentes.

Como foi referido na sub-secção 1.2.1, a plataforma comunica com a WorkStation através de duas portas série RS232. Destas, uma está ligada à WorkStation via modem e a outra por um cabo (ver Figura3). As suas denominações são porta **b** e porta **a** respectivamente.

A interface **C**[4] do Albatros^{TM} tem vários comandos destinados ao controlo das portas série. Estes comandos permitem fazer, entre outras acções, leituras, escritas, testes ao seu estado, etc (Ver referência [5] para mais detalhe).

[4] Biblioteca *libio*

3 Programas implementados

3.1 Programas destinados a ser executados na plataforma

Na Tabela 1 estão descritos os programas desenvolvidos para execução na plataforma. Detalha-se em seguida o funcionamento de cada um.

Programa	**Descrição**
estruturas.c	Mostra no écran o conteúdo das estruturas (Ver secção 4) *lll_table*, *axis_table* e *gains_array_r*.
jogo.c	Controla a velocidade da plataforma através da atribuição de um valor ao campo *command* da estrutura *axis_table*.
contvel.c	Controla a plataforma em velocidade, permitindo um ajuste fino. A velocidade das duas rodas tem o mesmo valor.
xmov.c	Controla em velocidade a plataforma, permitindo um ajuste fino, velocidades independentes para as duas rodas e devolução da prompt.
teste.c	Testa o comando *xmov*, executando uma trajectória e utilizando odometria.

Tabela 1: Programas desenvolvidos para executar na plataforma

- **estruturas.c**

  O objectivo deste programa é encontrar a correspondência entre as variáveis do esquema de controlo da plataforma em malha fechada (ver Figura 4) e os campos das estruturas. Para esse efeito o programa foi executado durante um movimento imposto à plataforma, através de instruções do sistema operativo. Utilizaram-se dois tipos de movimento: um em que é activada a malha fechada de controlo de posição e outro onde é activada a malha fechada de velocidade. Os resultados obtidos e respectivos comentários encontram-se na secção 4.

- **jogo.c**

  Permite controlar a plataforma em velocidade. Um dos resultados das observações feitas sobre as estruturas (ver secção 4) foi constatar que o campo *command* das estruturas *axis_table* guarda o valor corrente da velocidade linear a ser atingida. Assim, para controlar a velocidade da plataforma basta, simplesmente, inserir o valor desejado para cada roda no campo *command* da estrutura *axis_table* correspondente. Deste modo é possível controlar as duas rodas independentemente. É este o princípio utilizado neste programa. Os valores dos campos *command* são alterados sempre que se pressiona uma das quatro teclas de comando, as clássicas o, p, q, a usadas nos jogos de computador. Consegue-se, assim, acelerar, travar e fazer mudar de direcção a plataforma. Por forma a verificar se as velocidades pretendidas são realmente atingidas recorre-se à odometria. É utilizado um processo que é executado na plataforma e envia para o terminal, em cada instante, os valores de posição lidos pelos sensores, sendo estes guardados num ficheiro, para poderem ser tratados matematicamente. Este processo, lançado por *jogo.c*, encontra-se descrito no ficheiro *rob_com.h* e pode ser o *porta1_com ou porta2_com* conforme se queira usar a porta **a** ou **b**.O controlo em velocidade é feito por outro processo descrito no ficheiro *rob_control.h*, denominado *controlo*. Dado que o campo *command* de *axis_table* guarda valores inteiros não se pode, com este programa, obter um ajuste fino de velocidade. Deste modo o *jogo.c* equivale à utilização sucessiva de comandos de movimento, já definidos no sistema operativo, em particular a instrução MOVE (a diferença reside na não definição do tempo ao fim do qual a velocidade deve ser atinguida, sendo o efeito imediato) [4].

- **contvel.c**

  Para conseguir o ajuste fino de velocidade, varia-se (continuamente), o valor da velocidade nas rodas entre dois valores inteiros (1dm,2dm..). Este conceito baseia-se na obtenção de uma velocidade média através da utilização de uma velocidade inferior e outra superior. Se, por exemplo, se quiser que a plataforma se mova a uma velocidade de 2,7 dm/s, será necessário obrigá-la a mover-se alternadamente a 2 dm/s durante um certo período de tempo e a 3 dm/s durante um outro. Ambos os períodos são calculados pelo programa e dependem da velocidade a atingir. Este processo é utilizado no programa *contvel.c*. Este programa, apesar de permitir o controlo em velocidade, não cumpre inteiramente os objectivos. O programa contém um ciclo infinito responsável pelo movimento da plataforma que é interrompido quando é percorrida uma distância previamente definida. Esta situação não está de acordo com o objectivo do trabalho porque o movimento deveria continuar até ser dada uma nova instrução. Caso não fosse definida a distância, o ciclo, sendo infinito, ocuparia o processador, não permitindo a devolução da *prompt*, o que impossibilitaria o utilizador de enviar novo comando à plataforma. Este programa tem ainda mais um problema: a impossibilidade de comandar individualmente cada uma das rodas.

- **xmov.c**

  A explicação deste programa está detalhada na secção 5.

- **teste.c**

  Este programa é usado para testar o comando XMOV. O utilizador cria uma trajectória, introduzindo, por meio do teclado, um conjunto de valores que definem as velocidades das rodas esquerda e direita para cada troço (em dm/s), e uma constante, MULT (ver secção 5) , que permite um ajuste em função do estado actual do sistema. As velocidades do primeiro troço não devem ser muito elevadas, por uma questão de segurança, já que a plataforma age de forma a atingir essas velocidades de imediato. As do último troço devem ser nulas. A trajectória fica definida quando se introduz uma velocidade superior a 10 (não fazendo o último par de velocidades parte da trajectória).

  Nota: O número de troços introduzido deve ser coincidente com o definido no programa *timer.c*.

  As mensagens enviadas por *timer.c* definem o início do movimento, o fim de cada troço e o instante. De acordo com a mensagem recebida, o *teste.c* dá início ao movimento, muda de troço ou guarda numa tabela um conjunto de dados relativos à plataforma referentes a esse instante.

  Quando a trajectória é completada, o utilizador deve executar na Workstation o programa *odom2fich.c* que recebe os dados e os redirecciona para um ficheiro.

3.2 Programas destinados a ser executados na WorkStation

Os programas apresentados de seguida são usados para testar o comando desenvolvido.

Programa	Descrição
timer.c	Funciona como um relógio que envia mensagens para uma porta série em instantes definidos pelo utilizador.
odom2fich	Lê os dados enviados pela plataforma através de uma das portas série e coloca-os num ficheiro.

Tabela 2: Programas desenvolvidos para executar na WorkStation

- **odom2fich.c**

Este programa permite que o utilizador leia um conjunto de dados (pacote) pela porta série que desejar, e que escreva esses dados num ficheiro, também ele definido pelo utilizador. No código fonte pode-se alterar o tamanho do pacote de dados (numero de elementos) e o tamanho de cada elemento do pacote (em número de carácteres). A sua sintaxe é a seguinte: *odom2fich* **porta (a ou b) nome_do_ficheiro_de_saída**.

- **timer.c**

  O programa *teste.c* necessita de um temporizador que lhe indique quando deve executar determinadas operações. A função do *timer.c* é precisamente enviar mensagens pré-definidas ("strings") segundo a segundo, via porta série, para a plataforma onde é executado o *teste.c*. O código fonte do programa contém um vector com os instantes em que deverá enviar uma mensagem que implica a alteração do movimento da plataforma.

4 Análise das estruturas de dados do sistema operativo Albatros^{TM}

Na Figura 4 é apresentado o diagrama da malha de controlo de posição fornecido pelo fabricante. As variáveis da figura têm correspondência com os campos das estruturas que se encontram no ficheiro desenvolvido pela Robosoft^{TM}, *libcalbh.h* . Nele, pode-se observar as estruturas mencionadas. No entanto, essa correspondência não é indicada em nenhum dos manuais. Deste modo, é indispensável o estudo daquelas estruturas para compreender o funcionamento a baixo nível da malha de controlo. Para esse efeito, foram criados vários programas que se encontram condensados no programa *estruturas.c* [5]. A sua função é, unicamente, mostrar o conteúdo das estruturas de dados do Albatros^{TM}.

Foram feitas experiências em situações distintas: com ou sem SERV ON [4], com ou sem MOVE, com ou sem CPOS e com controlo de posição ou de velocidade.

A instrução SERV ON é utilizada para inicializar as estruturas usadas para o controlo de movimento. Quando não é executada, o conteúdo das estruturas não tem significado. Dispõe ainda de um parâmetro, T, que permite especificar o tipo de controlo a executar. Existem quatro possibilidades:

T=OP controlo de posição em malha aberta;
T=BP controlo de posição em malha fechada;
T=OV controlo de velocidade em malha aberta;
T=BV controlo de velocidade em malha fechada.

A instrução MOVE é um comando do sistema operativo utilizado para iniciar um movimento. Esta tem três parâmetros: o primeiro, *P* ou *V*, define o tipo de movimento desejado (tem que ser concordante com a inicialização do loop de controlo), o segundo parâmetro controla o tipo de coordenadas em que se pretende trabalhar, se *absolutas* - *ac*, ou *relativas* - *rc* e o terceiro, *P=...* define o número de ciclos ao fim dos quais a plataforma atinge a velocidade ou a posição desejadas.

A instrução CPOS serve para colocar a zero os valores dos sensores de posição.

4.1 Resultados das observações das estruturas

Nesta sub-secção, apresentam-se os resultados de dois dos ensaios feitos com o programa *estruturas*.

Os resultados da parte esquerda da tabela foram obtidos durante um movimento, depois de terem sido executadas as instruções CPOS, SERV ON T=BV e enquanto estava a ser executada a instrução MOVE V RC=3,3 P=300. Os da parte direita foram obtidos também durante um movimento, depois de terem sido executadas as instruções CPOS, SERV ON T=BP e enquanto estava a ser executada a instrução MOVE P RC=1000,1000 P=300.

4.1.1 Conteúdo de axis_table

O conteúdo de *axis_table* está descrito na tabela 3

4.1.2 Conteúdo de lll_table

O conteúdo de *lll_table* está descrito na tabela 4

4.1.3 Conteúdo de gains_array_r

Foram feitas experiências para diversos tipos de controlo, sendo este indicado na primeira coluna da tabela.

Através da observação dos resultados da sub-secção 4.1 tornou-se possível tirar conclusões acerca dos campos das estruturas. De seguida encontra-se uma lista com os campos estudados e a sua descrição.

[5] A sua explicação encontra-se na sub-secção 3.1

campo	MOVE V		MOVE P	
	eixo 0	eixo 1	eixo 0	eixo 1
cmd_sat	32752	32752	32752	32752
int_sat	32767	32767	32767	32767
min_com	-32768	-32768	-32768	-32768
max_com	32767	32767	32767	32767
command	1	1	304	304
d_command	3	3	1000	1000
e_d_command	0	0	3	3
var1	3	3	1000	1000
var2	0	0	0	0
var3	9371648	9371648	4325376	8519680
cmd_flag	0	0	0	0
act_cmd	3936	2044	3997	2413
integ	0	0	2896	1321
Xk[0]	290	303	376	382
Xk[1]	290	303	376	382
Xk[2]	289	301	376	381
Xk[3]	288	300	375	380
Xk[4]	287	298	374	380
Ek[0]	300	312	382	386
Ek[1]	0	0	1	1
Ek[2]	0	0	2	2
Ek[3]	0	0	3	2
Ek[4]	1	0	0	0

Tabela 3: Resultados das observações de axis_table

campo	MOVE V	MOVE P
axis_table_size	56	56
main_flag	1	1
sema	0	0
sens_val (eixo0)	630	728
sens_val (eixo1)	648	731
m	2	2
n	4	4
ctech	141	178
nbech	300	300
type	0	0
dev_status	43	11
nfech_as	1	0
fech_as	4	4
cainc_as	V	P

Tabela 4: Resultados das observações de lll_table

t=	K_p	K_I	K_D	escala
bv	4096	0	1536	10
bp	2048	10	1024	8
op	1	0	0	1
ov	100	0	0	100
jstk on	1024	0	1536	10

Tabela 5: Resultados das observações de gains_array_r

O conteúdo de *gains_table* está descrito na tabela 5

4.2 Significados dos campos das estruturas

4.2.1 Descrição dos campos da lll_table

1. ***f_axis_table** - Ponteiro para a estrutura axis_table correspondente à roda esquerda. Com o ponteiro *(f_axis_table + 1) acede-se à roda direita.

2. **axis_table_size** - Dimensão da axis_table: 56.

3. **Main_flag** - Indicador de movimento da plataforma. Encontra-se a '1', quando esta está em movimento e a '0', caso contrário.

4. **sema** - Semáforo (no ficheiro libcalbh.h é dito que está a 1 quando a malha de controlo está activa). Não se compreendeu o seu funcionamento.

5. ***sens_val** - Ponteiro para o "array dos sensores".

6. **(*leccapt)()** - Ponteiro para uma rotina que converte coordenadas dos sensores para coordenadas generalizadas.

7. ***command** - Ponteiro para comandos de alto nível.

8. **(*asortad)()** - Ponteiro para a função que envia os comandos calculados para os actuadores.

9. **m** - Número de eixos.

10. **n** - Número de valores de epsilon, que corresponde ao tamanho dos vectores onde são guardados os valores do erro das amostragens anteriores.

11. **ctech** - Contador de passos elementares. Quando se executa um MOVE introduz-se um parâmetro P que é igualado ao número de ciclos até que se atinja a posição ou a velocidade desejadas. Ctech vai sendo decrementado desde esse valor até -1.

12. **nbech** - Número de ciclos necessários para se atingir a posição ou a velocidade desejadas. Mantem-se constante e igual ao valor inicial de ctech.

13. **type** - Tipo de loop. A '1' quando rápido e a '0' caso contrário [2].

14. ***motion_stack_p**- Ponteiro para a estrutura motion_stack que guarda a pilha de movimentos em execução e por executar.

15. ***dev_status** - Ponteiro para o estado do sistema [6]. Aponta para um byte onde cada bit tem um significado:

[6] O ficheiro libcalbh.h está errado neste campo pois falta-lhe o asterisco

- 0 a '1' quando a plataforma está ligada.
- 1 a '1' quando a malha de controlo está activada.
- 2 sempre a '0'.
- 3 a '1' quando a plataforma está em movimento.
- 5,6 define o tipo de controlo:'00' malha fechada de posição; '01' malha fechada de velocidade; '10' malha aberta de posição; '11' malha aberta de velocidade.

Para mais informações, ver manuais [4] e [5].

16. **nfech_as** - Contador de loops de baixo nível.
17. **fech_as** - Proporção constante entre loops de baixo nível e geração de trajectórias.
18. **cainc_as[2]** - Tipo de movimento. p-posição, pa-acelerado, pd-desacelerado, v-velocidade.
19. **working_table** - Não foi estudado.

4.2.2 Descrição dos campos da axis_table

Existem duas tabelas, uma para a roda direita e outra para a esquerda.

1. ***gains** - Ponteiro para a estrutura gains_array_r, onde estão os ganhos dos PID.
2. **cmd_sat** - Valor máximo que o termo proporcional pode atingir: 32752.
3. **int_sat** - Valor máximo do termo integral:32767.
4. **not_used** - Não foi utilizado.
5. **min_com** - Valor mínimo que a variável command pode atingir: -32768.
6. **max_com** - Valor máximo que a variável command pode atingir: 32767.
7. **command** - Comando corrente a ser atingido. Contém o valor da posição (em mm) ou da velocidade (em dm/s) que a roda deve ter nesse momento. Este valor vai sendo actualizado pelo processo gerador de trajectórias ao longo do movimento.
8. **d_command** - Incremento global de comando. Neste campo é guardado o valor da posição ou velocidade desejadas, o valor que foi introduzido na instrução MOVE e que será atingido quando ctech for -1.
9. **e_d_command** - Incremento elementar de comando.
10. **var1** - Igual a d_command.
11. **var2** - Igual a zero.
12. **var3** - Não utilizada.
13. **cmd_flag** - Detecta um valor de saturação. Passa de '0' para '1' quando é detectado que um valor do command a enviar para os actuadores ultrapassa o valor de saturação.
14. **act_cmd** - Resultado enviado aos actuadores. Constatou-se uma diferença muito elevada entre os valores dos dois eixos.
15. **integ** - Valor corrente do termo integral. Também se verificou a diferença muito elevada entre os valores dos dois eixos. Igual a zero, quando se trata de um comando de velocidade.

16. **Xk[5]** - Valor corrente dos Xk. Mostra os valores dos sensores de posição da roda correspondente (expressos em mm) no instante corrente e nos quatro anteriores. Xk[0] é o resultado de uma operação de divisão por, aproximadamente, 1.8, equivalente à proporção entre o campo sens_val da estrutura lll_table e o valor contido em Xk[0].

17. **Ek[5]** - Valor corrente de Ek. Mostra os valores do erro entre os sensores (em Xk) e os valores de referência (no command), expressos em mm, no instante corrente e nos quatro anteriores. O valor de Ek[0] observado foi sempre igual ao valor corrente da posição (Xk[0]).

4.2.3 Descrição dos campos da gains_array_r

Existem duas tabelas, uma para a roda direita e outra para a esquerda.

1. Kp[4] - Ganhos proporcionais.

2. Ki[4] - Ganhos integrais.

3. Kd[4] - Ganhos derivativos.

4. scale - Factor de escala.

5 Comando implementado

5.1 Descrição

A partir do estudo efectuado na secção 4, conclui-se que o campo que guarda a velocidade corrente a ser atingida, *command*, é uma variável inteira. Por este motivo, não se conseguem impôr velocidades reais directamente. Sabe-se que, se uma roda comutar entre duas velocidades, v_1 e v_2, durante os intervalos de tempo t_1 e t_2, respectivamente, a velocidade média será dada pela expressão

$$v = \frac{v_1 \times t_1 + v_2 \times t_2}{t_1 + t_2}$$

Ignorando o facto de não ser possível as rodas comutarem instantaneamente de velocidade e permanecerem a essa velocidade o tempo definido pela equação, devido à própria dinâmica do sistema que contém atrasos, inércias, atritos, outros processos compartilhados pelo mesmo processador e outros factores, implementou-se uma estratégia que consiste na obtenção da velocidade desejada por meio da comutação entre duas velocidades inteiras. Foi criado um parâmetro, MULT [7] , que pode ser definido pelo utilizador ajustando-o ao estado actual do sistema, de forma a obter uma melhor precisão na velocidade. Utilizou-se o seguinte método [8]:

1. Define-se a constante *CONT* que equivale a cada ciclo de duração t_1+t_2;
2. Guarda-se o valor da velocidade desejada na variável n[ ];
3. Separa-se a parte inteira da velocidade da parte fraccionária e guardam-se, respectivamente, em v[ ] (que corresponde à velocidade v_1) e d[ ];
4. Guarda-se na variável sgn[ ] o valor 1 ou -1, de acordo com o sinal da velocidade;
5. Limita-se a velocidade a uma gama de valores pré-estabelecida (o intervalo [-10dm/s,10dm/s]). Se a velocidade pretendida for maior que o valor máximo permitido, é-lhe atribuído esse mesmo valor, e, se for menor que o valor mínimo permitido, dá-se-lhe esse valor;
6. Calcula-se o valor de t_1 (que será guardado em lim[ ]), ou seja, o momento em que se irá mudar da velocidade v_1 para a velocidade v_2 (igual a v[ ]+sgn[ ]) através da expressão:

$$lim[\ ] = (1 - |d[\ ]|) \times cont$$

O valor de *CONT* foi escolhido experimentalmente, visto ter sido aquele que permitiu melhor precisão de velocidade.

O comando desenvolvido, **xmov**, tal como os já existentes no sistema operativo, devolve a "prompt" ao utilizador. Isso é implementado recorrendo a um programa que lê os valores desejados para cada uma das rodas, faz os cálculos atrás mencionados e lança um processo que, por sua vez, actua sobre as rodas. O programa pai, ao contrário do que é habitual, não retira o processo lançado antes de terminar. Este permanece em "background", continuando a ser executado até que se emita um novo comando **xmov**. Nesta situação, o processo em execução é substituido pelo novo. Quando este se trata de uma instrução **xmov 0 0**, o processo é executado uma última vez (para que a plataforma pare), e termina-se a si próprio.

Um processo em "background" é interrompido pelo processador sempre que haja uma tarefa a executar com prioridade superior (ver Figura 8). Em caso de interrupção, o campo *command*, cujo conteúdo indica a velocidade a ser atingida, mantém-se constante. Consequentemente, a plataforma desloca-se a uma velocidade inteira (em dm/s) constante, interrompendo a comutação entre velocidades. Este fenómeno altera a velocidade média , o que dá origem a um erro na velocidade.

Para reduzir este erro e obter uma maior precisão, utilizam-se duas soluções:

[7] Este parâmetro está descrito mais adiante nesta secção.

[8] Cada variável é um elemento de uma matriz de dois elementos referentes a cada uma das rodas.

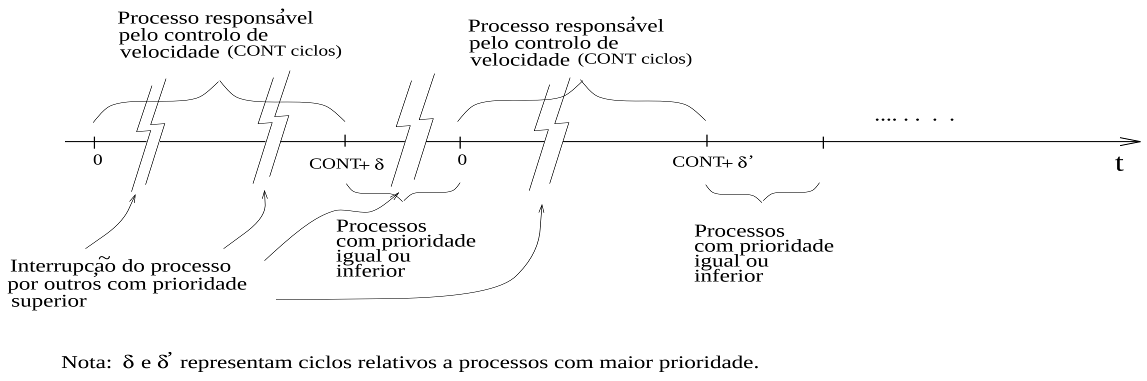

Figura 8: Ocupação do processador

1. **Aumento da duração do processo - parâmetro MULT**

   Cria-se um parâmetro denominado *MULT* que representa o número de vezes que cada ciclo de duração *CONT* é efectuado. Quanto maior for este valor, maior será o período de duração do processo responsável pelo controlo das rodas. Deste modo, a razão entre este período e aquele em que o processador se ocupa de outros processos (com prioridade menor ou igual) é maior, resultando numa diminuição do erro (ver Figura 9). Note-se que um elevado valor de MULT traz como consequência um elevado período de tempo durante o qual o processador se dedica à gestão do processo, e que este fenómeno pode provocar atrasos indesejáveis [9]. O valor do parâmetro MULT é 10 por defeito, podendo ser definido pelo utilizador de cada vez que executa um comando **xmov**. O valor 10 é o que revela uma melhor relação precisão/ocupação do processador, no entanto, o utilizador deve ajustá-lo de acordo com a dinâmica do sistema nesse instante, tendo em conta factores como a existência de cargas muito elevadas de processamento, alteração da carga das baterias, variação da carga transportada pela plataforma, etc..

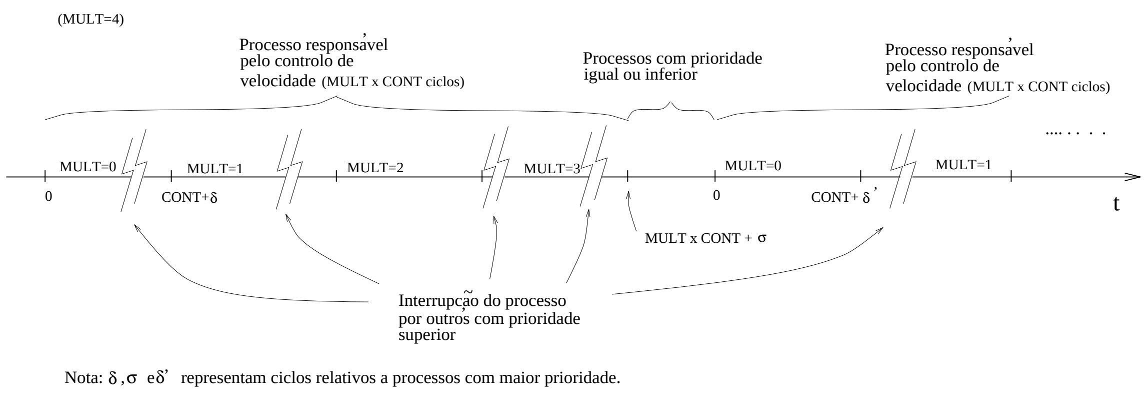

Figura 9: Efeitos da utilização do parâmetro MULT

2. **Comutação da velocidade fora do processo - variável *bool***

[9] Estes podem ser observados pelo utilizador na forma de um atraso na libertação da "prompt".

Como já foi referido, quando o processador se encontra a executar outras tarefas, deixa de haver comutação entre velocidades o que dá origem a um erro. Se a velocidade de cada roda à saída do processo responsável pelo controlo das rodas fosse sempre v[ ]+sgn[ ] (ver Figura 10), a plataforma deslocar-se-ia sempre a uma velocidade maior (em módulo) que a desejada. Para reduzir este problema foi introduzida uma variável, *bool*, que só toma os valores '0' e '1', sendo complementada de cada vez que o processo é executado. A sua função é fazer com que a velocidade presente em cada roda quando se sai do processo responsável pelo controlo de velocidade comute, i.e., quando *bool* toma o valor '0', a velocidade de cada roda à saída do processo é a de maior valor absoluto, quando toma o valor '1', a de menor valor absoluto. O utilizador não tem acesso a esta variável.

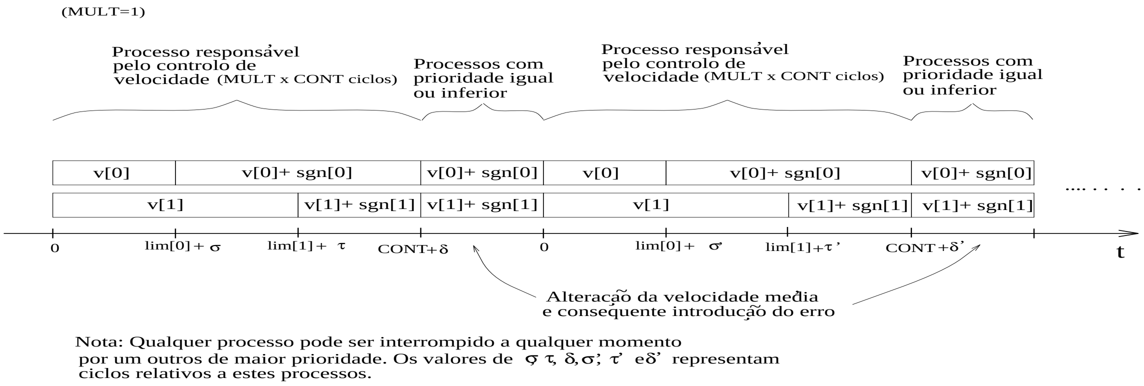

Figura 10: Erro introduzido pela interrupção da comutação entre as duas velocidades

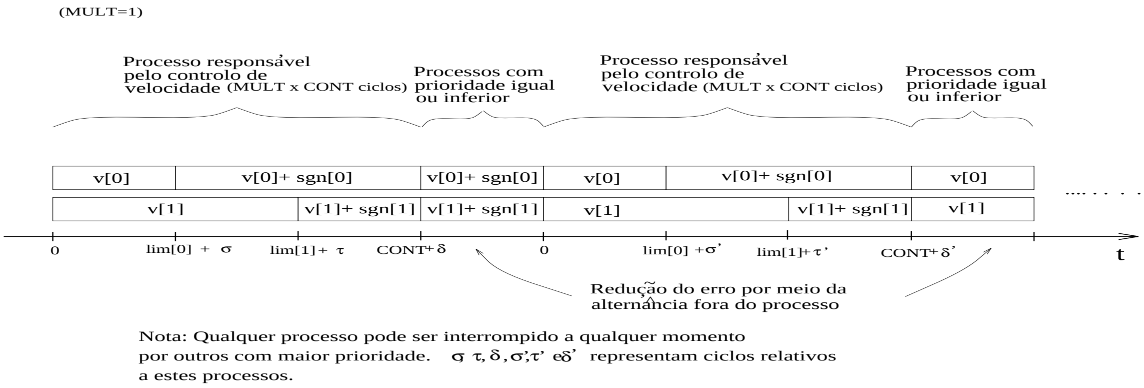

Figura 11: Efeitos da utilização da variável *bool*

5.2 Modo de utilização

O comando **xmov** pode ser utilizado de dois modos distintos. Em qualquer dos casos, para que o sistema reconheça o comando **xmov**, é necessário transferir previamente o programa *xmov.c* para o espaço de memória da plataforma a partir do endereço 20000, e após a ligação à plataforma, a execução de duas instruções para que se possa utilizar o comando (a ordem das duas é irrelevante):

- **G 20000** Esta instrução serve para adicionar o comando ao sistema Albatros^{TM}.

- **SERV ON T=BV** Esta instrução activa a malha de controlo de velocidade em cadeia fechada.

O comando pode ser utilizado:

- **Por comando directo**
  - Síntaxe

$$\text{XMOV } v_1 \; v_2 \text{ [MULT]}$$

  v_1 representa a velocidade da roda esquerda, v_2, a da roda direita e MULT é um parâmetro opcional que por defeito toma o valor 10.

- **Integrado num programa de aplicação**

  Neste caso, o utilizador cria um programa destinado a ser executado na plataforma que faz uso do comando **xmov**. Este deve ser enviado para o interpretador de comandos do sistema operativo da plataforma através do uso das instruções **SEND** ou **SENDF**. Seguem-se dois exemplos:

  - exemplo 1: SEND("XMOV 1.0 1.5\ r");
  - exemplo 2: SENDF("XMOV %f %f %d\r",vel_esq,vel_dir,mult);

  Nota: No primeiro exemplo, as velocidades das rodas esquerda e direita são 1.0dm/s e 1.5dm/s, e o valor do parâmetro MULT não é introduzido. No segundo exemplo, vel_esq, vel_dir são duas variáveis do tipo *float* onde estão guardados os valores desejados para as rodas esquerda e direita, respectivamente, e mult é uma variável do tipo *int* que guarda o valor do parâmetro MULT.

  Atenção: O utilizador deve ter o cuidado de transferir o seu programa para uma zona de memória que não interfira com a do **xmov**. Recomenda-se a transferência para a posição de memória h27000.

5.3 Resultados obtidos

Os testes efectuados incidiram sobre velocidades inferiores a 1dm/s, por ser esta a gama de valores onde o comando tem mais aplicações (operações de acostagem e movimentos em espaços limitados p.ex.).

O valor de MULT foi ajustado para a velocidade de 0.1 dm/s e com poucos processos em execução.

5.3.1 Movimento rectilíneo

Os resultados apresentados foram obtidos com o programa **teste.c**, apresentado na secção 3. Os testes realizados consistem em impor uma velocidade igual em ambas as rodas durante 30s. Apresenta-se o valor das coordenadas x, x_óptimo e y ao longo do tempo (ver 20).

Observando as figuras [12..18] verifica-se que, a menos de um pequeno erro, o comando desenvolvido cumpre as especificações de velocidade. Nota-se que a coordenada y não é sempre nula, apresentando um erro da ordem de 1% do valor da coordenada x. Este desvio ocorre também no comando *move* do sistema operativo Albatros^{TM} tal como mostra a figura 19. Este erro é, portanto, inerente ao sistema, não sendo consequência do controlo implementado. A existência de patamares nas figuras [12..18] é devida à existência de erros no processo de leitura (a plataforma nunca pára durante o trajecto). Estes erros são o resultado de interrupções originadas pelo sistema operativo que, conjuntamente com o processo de odometria levam à inibição da leitura de posição.

O erro é menor (em valor absoluto) para o caso de velocidade 0.1dm/s. Isso reflecte o ajuste de MULT para essa velocidade. Nota-se que o erro tende a aumentar conforme a velocidade se afasta de 0.1dm/s.

O erro constante que se observa na figura 19 é devido ao parâmetro p do comando *move* [4]. Se p aumentar, o atraso sentido no arranque será maior, uma vez que este parâmetro determina o número de ciclos de relógio necessários para se atinguir uma dada velocidade.

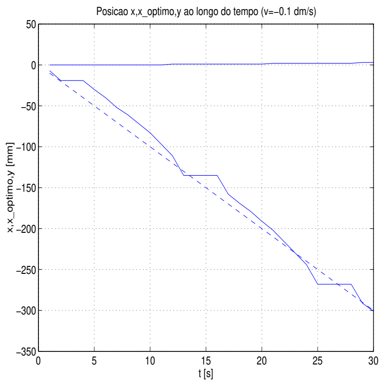

Figura 12: Resultado obtido com *xmov -0.1 -0.1*

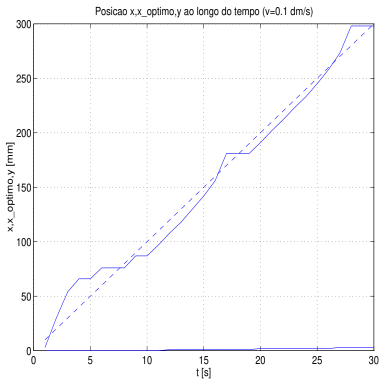

Figura 13: Resultado obtido com *xmov 0.1 0.1*

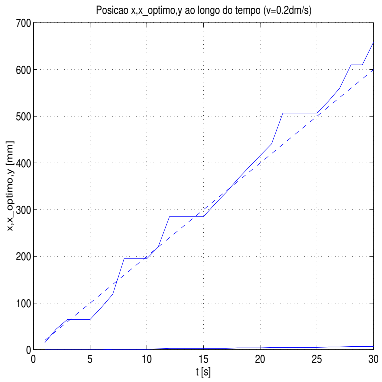

Figura 14: Resultado obtido com *xmov 0.2 0.2*

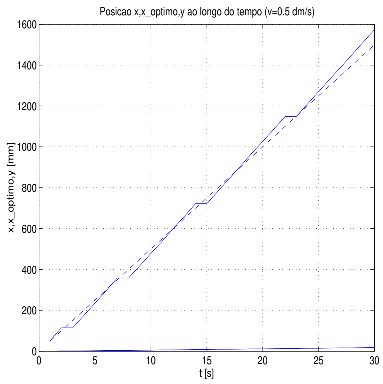

Figura 15: Resultado obtido com *xmov 0.5 0.5*

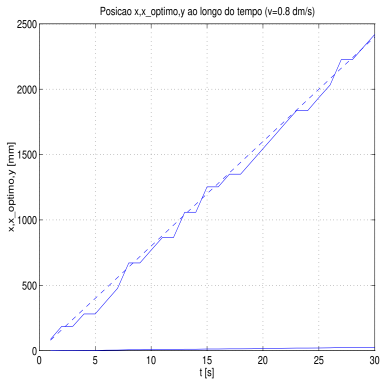

Figura 16: Resultado obtido com *xmov 0.8 0.8*

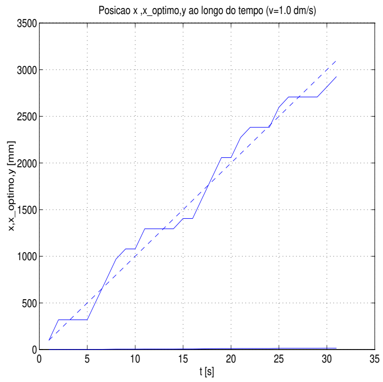

Figura 17: Resultado obtido com *xmov 1.0 1.0*

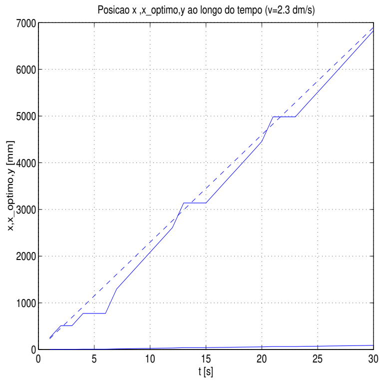

Figura 18: Resultado obtido com *xmov 2.3 2.3*

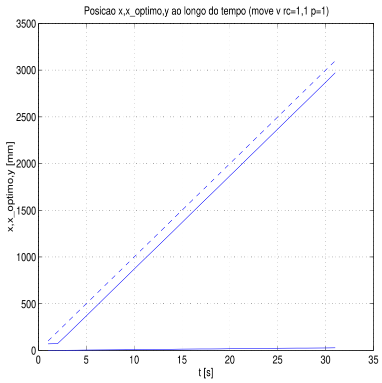

Figura 19: Resultado obtido com *move v rc=1,1 p=1*

5.3.2 Execução de uma trajectória

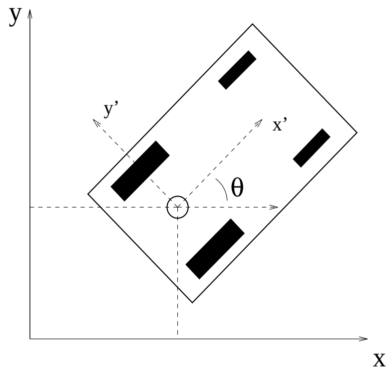

Figura 20: Sistema de coordenadas

A figura 21 apresenta a trajectória (x,y) real e a ideal, definida na tabela 6. Apresenta-se também a evolução de θ_real e θ_ideal ao longo do tempo.

t [s]	**v_esquerda**	**v_direita**
0	0.3	0.3
20	0.0	0.2
66	0.2	0.2
86	0.2	0.0
132	0.4	0.4
162	0.2	0.0
208	0.1	0.1
248	0.0	0.2
294	0.4	0.4
304	0.0	0.0

Tabela 6: Trajectória definida para testar o comando *xmov*

A trajectória realizada pela plataforma apresenta um desvio em relação à desejada que se deve à existência de erros ao longo do percurso (não se faz qualquer tipo de correcção durante o trajecto). O facto de as curvas serem iniciadas um pouco mais tarde fica a dever-se ao ligeiro atraso que por vezes existe tal como foi referido em 5.1. Note-se que o atraso ocorrido na segunda curva reflectiu-se no resto da trajectória, vindo esta subida em relação ao zero de y.

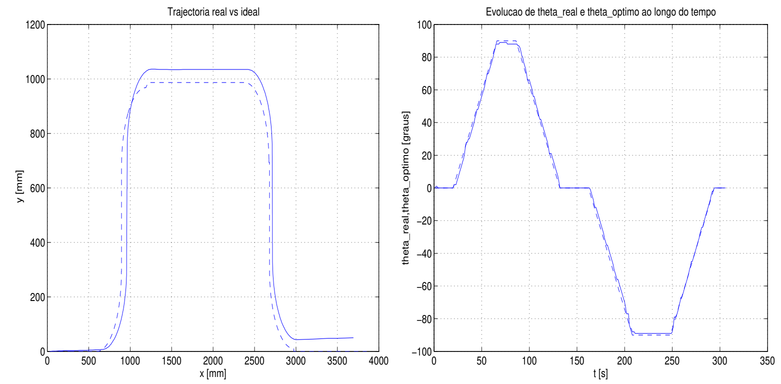

Figura 21: Trajectória realizada com o programa teste.c

6 Conclusões

O trabalho realizado conduziu à criação de um comando que visa satisfazer o objectivo proposto: a concepção de uma estratégia de controlo de velocidade de uma plataforma móvel Robuter^{TM} que possibilita ao utilizador o uso de qualquer velocidade múltipla de 0.1dm/s dentro de uma gama de velocidades compreendida entre -10dm/s e 10dm/s. Esta estratégia foi implementada na forma de um comando.

As velocidades deixam de ser condicionadas pelo grau de quantificação existente que só permitia a utilização de velocidades múltiplas de 25mm/s. O controlo das duas rodas motrizes é independente, possibilitando o seguimento de qualquer trajectória pela plataforma. O comando foi implementado de forma a ser adicionado ao sistema operativo Albatros^{TM} presente na plataforma, tornando-se, assim, equivalente aos já existentes. Este facto permite o seu uso directo ou em programas de aplicação desenvolvidos pelo utilizador, como se de um comando de raíz se tratasse. O processo responsável pelo controlo de velocidade é transparente ao utilizador, ficando a correr em “background”, sendo a “prompt” devolvida de modo a tornar possível a execução de um outro comando. Verificou-se que a *performance* do comando diminui com a carga computacional do processador. Esta limitacção deve ser considerada pelo utilizador, podendo este alterar um parâmetro (MULT), com vista a aumentar a exactidão para essa situação. Um critério para a escolha do seu valor é o número de processos em execução, sendo estes inversamente proporcionais. É de notar que, quando MULT baixa, o erro de velocidade tende a aumentar. O nível de carga das baterias é também um factor a ter em conta na obtenção de resultados uma vez que uma carga baixa pode levar a resultados errados.

Como já foi referido em 1.2.2, os controladores implementados não conseguem actuar em simultâneo em ambas as rodas. Para além disso, o processo de controlo pode ser interrompido pelo sistema, levando à existência de erros no tempo de actuação do processo.

7 Trabalho futuro

Um aspecto do comando desenvolvido que poderá ser melhorado de futuro será o de permitir velocidades superiores no arranque, evitando ao utilizador o trabalho de acelerar a plataforma através da imposição de velocidades sucessivas antes desta atingir a velocidade final desejada. Tal desenvolvimento permitiria também uma paragem suave (tal como acontece nos movimento de raíz).

O comando **xmov** recebe, como argumentos, as velocidades em coordenadas absolutas. No futuro, poderá passar a recebê-las também em coordenadas relativas.

Referências

[1] James L. Petersson, Abraham Silberschatz "Operating Systems Concepts - Second Edition" *Addison-Wesley Publishing Company, 1996*

[2] "Albatros^{TM} 4.0 Reference Manual"

[3] "Albatros^{TM} Software Development Package (ASDP) V 3.0 Users guide - Alpha release"

[4] "Albatros^{TM} User's Manual V 5.2"

[5] "Albatros^{TM} C Interface Reference Manual"

[6] "Robuter^{TM} Users Manual 3.1"