

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Modelação e Simulação de um Robot Omnidireccional de 3 rodas

Renato Miguel dos Santos Caldas

Tese submetida no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major de Automação

Orientador: António Paulo Gomes Mendes Moreira (Prof. Dr.)

Co-orientador: Paulo José Cerqueira Gomes da Costa (Prof. Dr.)

Fevereiro de 2009

Resumo

A validação de algoritmos de controlo em robótica móvel consiste, de uma forma simplificada, em comparar o desempenho alcançado com o desempenho esperado. Para que esta comparação seja possível, é necessário um ambiente experimental controlado, o que se vai tornando complicado à medida que as velocidades envolvidas aumentam.

As ferramentas de simulação apresentam-se como uma alternativa atraente à experimentação com robôs reais, já que permitem que as experiências sejam feitas num ambiente perfeitamente controlado. No entanto, a utilidade da simulação é normalmente limitada pela capacidade de os modelos usados captarem a dinâmica do processo real.

Foi desenvolvida na Faculdade de Engenharia da Universidade do Porto, no âmbito do grupo de robótica móvel *5dpo*, uma ferramenta de simulação. A presente dissertação tem como objectivo validar a sua aplicabilidade na modelação e simulação de uma classe de robôs. Mais concretamente, pretende-se uma ferramenta realista, que permita o teste de controladores antes de os aplicar em robôs reais.

Este trabalho centrou-se num dos robôs de futebol robótico. A abordagem usada passou pela identificação de um modelo do robot compatível com o simulador, sendo de seguida estimados os parâmetros do modelo, usando entre outras, uma ferramenta de medição da pose do robot baseada em visão, tendo sido feitas algumas experiências por forma a validar o modelo. Finalmente, foi estudada a possibilidade de adaptar o software do robot para que pudesse actuar transparentemente tanto no robot real como no robot simulado,

O modelo obtido revelou-se bastante realista, comportando-se de forma muito semelhante ao robot real. Por outro lado, a abordagem tomada parece ser promissora, no sentido de permitir a simulação do robot de forma transparente para o software de decisão.

Abstract

The validation of control algorithms in mobile robotics consists, simply, in a comparison between the achieved performance and the expected performance. In order for the comparison to be possible, a controlled experimental environment is required, which becomes complicated as the involved speeds increase.

Simulation tools represent an attractive alternative to the experimentation using real robots, allowing the experiences to be executed in a perfectly controlled environment. Even so, the usefulness of the simulation is usually limited by the ability of the used models to capture the real world dynamics of the process.

A simulation tool was developed at Faculdade de Engenharia da Universidade do Porto, in the sphere of the mobile robotics group *5dpo*. The present dissertation aims to validate its applicability in the modeling and simulation of a class of robots. More specifically, the intent is to provide a realistic tool, allowing controllers to be tested before applying them to real robots.

This work was centered around one of the soccer robots. The used approach comprised the identification of a suitable robot model for the simulator, followed by the estimation of the model parameters, using a vision based attitude measurement tool, while some experiments were made in order to validate the model. Finally, a possible adaptation of the robot software was studied, so that it could actuate transparently over both the real robot and the simulated robot.

The obtained model revealed itself very realistic, behaving closely like the real robot. On the other hand, the used approach seems promising, in light of allowing the simulation of the robot transparently for the decision software.

Agradecimentos

No decorrer deste trabalho foram inúmeras as pessoas que contribuíram significativamente para o seu sucesso. Gostaria de agradecer a todas elas, ainda que apenas seja possível mencionar algumas delas.

Agradeço especialmente à minha família e à Marina, por todo o apoio e motivação, que se revelou fundamental para a conclusão deste trabalho.

Gostaria também de agradecer aos meus orientadores, o Prof. Dr. António Paulo Moreira e o Prof. Dr. Paulo Costa, pela ajuda e orientação prestada.

Agradeço também a todos aqueles que me ajudaram no desenvolvimento do trabalho. Neste sentido devo agradecer em especial ao Heber Sobreira, cuja ajuda foi fundamental.

Finalmente gostaria de agradecer ao Prof. Dr. Russel Smith, autor da biblioteca *ODE*, por me permitir usar algumas das imagens da documentação da referida biblioteca.

O Autor

*“Hell, there are no rules here,
we’re trying to accomplish something.”*

Thomas A. Edison

Conteúdo

1	Introdução	1
1.1	Motivação	1
1.2	Objectivos	2
1.3	Estrutura da Dissertação	2
1.4	Revisão Bibliográfica	2
2	Descrição do robot	5
2.1	Descrição geral	6
2.2	Periféricos em Daisy Chain	7
2.2.1	Protocolo série	7
2.2.2	Módulo de controlo de motores	10
2.2.3	Módulo de controlo de chuto	11
2.3	Software de controlo	13
2.3.1	Aplicação de processamento de imagem e de interface com o <i>hardware</i>	13
2.3.2	Aplicação de controlo e decisão	14
2.4	Comunicação entre aplicações	15
2.4.1	Tipos de dados	16
2.4.2	Formato dos Pacotes	16
2.4.3	Formato dos blocos	17
3	Ferramenta de medição da pose do robot	21
3.1	Processamento de imagem	21
3.2	Marcadores	23
3.3	Cálculo da posição	23
3.4	Protocolo de rede	23
3.5	Cliente de software	25
4	Plataforma de simulação	27
4.1	Descrição Geral	27
4.2	Biblioteca de Simulação de Corpos Rígidos	29
4.2.1	Simulação da dinâmica de corpos rígidos	29
4.2.2	Simulação de colisões	31
4.3	Configuração do simulador	31
4.3.1	Mundo envolvente	31
4.3.2	Objectos	33
4.3.3	Instâncias de Robôs	33
4.3.4	Construção dos Robôs	34
4.3.5	Sólidos e Cascas	34

4.3.6	Rodas e Articulações	35
4.4	Controlo	36
4.4.1	Programa de controlo	37
4.4.2	Interacção com a simulação	37
4.4.3	Comunicação com o exterior	39
5	Modelo do robot	41
5.1	Corpo do robot	41
5.1.1	Determinação da altura do Centro de Massa	42
5.1.2	Determinação do Momento de Inércia	43
5.2	Superfícies de colisão	44
5.3	Cadeia de Actuação	45
5.3.1	Rodas omnidireccionais	45
5.3.2	Motor	46
5.3.3	Caixa redutora	46
5.3.4	<i>Encoder</i> óptico	46
5.3.5	Controlador	47
5.3.6	Modelo de atrito	48
5.4	Controlo da simulação	50
5.4.1	Sequência de referências	51
5.4.2	Emulação da aplicação de ligação ao hardware	51
6	Resultados	53
6.1	Estimação de parâmetros	53
6.1.1	Corpo do robot	53
6.1.2	Rodas omnidireccionais	54
6.1.3	Motor, caixa redutora e <i>Encoder</i> óptico	56
6.1.4	Controlador <i>PID</i> de velocidade	56
6.2	Validação do modelo	57
7	Conclusões e Trabalho Futuro	59
7.1	Satisfação dos Objectivos	59
7.2	Trabalho Futuro	60
	Referências	61

Lista de Figuras

2.1	Fotografia do robot usado nesta dissertação usando um marcador colorido	5
2.2	Ilustração do princípio de funcionamento do sistema de visão omnidireccional do robot	6
2.3	Fotografia de uma roda omnidireccional, mostrando os roletes dispostos na sua periferia	6
2.4	Ilustração da movimentação em frente do robot	7
2.5	Ilustração da movimentação lateral do robot	7
2.6	Ilustração da movimentação rotativa do robot	7
2.7	Esquema da ligação <i>daisy chain</i> entre os vários controladores	8
2.8	Ilustração da cadeia de actuação associada a cada roda	11
2.9	Captura da janela de imagem da aplicação <i>OVis</i>	14
2.10	Captura da janela de <i>zoom</i> da aplicação <i>OVis</i>	14
2.11	Captura da janela de controlo e configuração da aplicação <i>OVis</i>	14
2.12	Captura da janela da aplicação <i>MDecOmni3</i>	15
3.1	Captura da Janela do <i>SmHAL</i>	22
3.2	Robot <i>SSL</i> com marcador na face superior	23
3.3	Captura da Janela do <i>SmallClient</i>	26
4.1	Captura da janela de visualização <i>3D</i> do <i>SimTwo</i>	28
4.2	Captura da janela de configuração do controlo do <i>SimTwo</i>	28
4.3	Captura da janela de configuração da visualização do <i>SimTwo</i>	28
4.4	Captura da janela de configuração de entrada/saída do <i>SimTwo</i>	28
4.5	Captura da janela do editor de código do <i>SimTwo</i>	29
4.6	Captura da janela de gráficos do <i>SimTwo</i>	29
4.7	Imagem ilustrativa do principio de funcionamento de uma <i>joint</i> do tipo <i>hinge</i> (cortesia de Russel Smith, autor da biblioteca <i>ODE</i>)	30
4.8	Imagem ilustrativa do principio de funcionamento de uma <i>joint</i> de contacto (cortesia de Russel Smith, autor da biblioteca <i>ODE</i>)	31
5.1	Experiência de determinação da altura do centro de massa do robot	43
5.2	Experiência de determinação do momento de inércia do robot	43
6.1	Gráfico da posição angular do robot em função do tempo para a experiência de determinação do momento de inércia	54
6.2	Gráfico da velocidade angular do robot em função do tempo para a experiência de determinação do momento de inércia	54
6.3	Gráfico da posição angular do robot em função do tempo para a experiência de determinação dos parâmetros do modelo de atrito tangencial	55

6.4	Gráfico da velocidade angular do robot em função do tempo para a experiência de determinação dos parâmetros do modelo de atrito tangencial	55
-----	--	----

Lista de Tabelas

2.1	Formato do pacote usado na comunicação série	8
2.2	Resumo dos campos do pacote usado na comunicação série	9
2.3	Formato da mensagem <i>PACK_TYPE_PID_PWM</i>	12
2.4	Formato da mensagem <i>PACK_TYPE_READ_ENC</i>	12
2.5	Resumo dos campos da mensagem <i>PACK_TYPE_PID_PWM</i>	12
2.6	Resumo dos campos da mensagem <i>PACK_TYPE_READ_ENC</i>	12
2.7	Tipos de dados usados na comunicação entre Ovis e MDec	16
2.8	Blocos usados na comunicação entre Ovis e MDec	17
2.9	Formato do bloco <i>Clusters</i>	18
2.10	Formato do bloco <i>Edges</i>	18
2.11	Formato do bloco <i>Radar</i>	18
2.12	Formato dos blocos <i>Serials</i>	19
3.1	Cores lógicas usadas no <i>SmHAL</i>	22
4.1	Tipos de junção suportados pela biblioteca <i>ODE</i>	30
4.2	Secções permitidas dentro da secção <i>scene</i>	32
4.3	Objectos básicos suportados pelo <i>SimTwo</i>	33
4.4	Propriedades dos básicas dos objectos do <i>SimTwo</i>	33
4.5	Propriedades dos básicas dos robôs no <i>SimTwo</i>	34
4.6	Secções permitidas dentro da secção <i>robot</i> do ficheiro de descrição do corpo do robot	34
4.7	Componentes da cadeia de actuação do <i>SimTwo</i>	36
6.1	Resultados da experiência de determinação do momento de inércia do robot	54
6.2	Resultados da experiência de determinação dos parâmetros do modelo de atrito tangencial no eixo das rodas do robot	56
6.3	Sumário de todos os parâmetros do modelo	57

Abreviaturas

5DPO	Grupo de Futebol Robótico da FEUP
FEUP	Faculdade de Engenharia da Universidade do Porto
FPC	Free Pascal Compiler
IP	Internet Protocol
IDE	Integrated Development Environment
MSL	Middle Size League
ODE	Open Dynamics Engine
PID	Proporcional, integral, derivativo
RAD	Rapid Application Development
RGB	Red, Green, Blue
SSL	Small Size League
UDP	User Datagram Protocol
XML	Extensible Markup Language

Capítulo 1

Introdução

Nos últimos anos tem-se vindo a assistir a um grande aumento na complexidade dos algoritmos usados na robótica móvel, muito por causa da crescente disponibilidade de poder computacional. Mais concretamente, a melhoria dos algoritmos relacionados com o controlo de movimento permite dos robôs prestações cada vez mais arrojadas.

Se a complexidade destes algoritmos justifica por si só maiores cuidados na sua implementação, o aumento das velocidades dos robôs implica que estes se possam tornar perigosos para bens e pessoas. Por outro lado os fenómenos não lineares tornam-se cada vez mais importantes para o desempenho do controlo.

A validação experimental dos algoritmos de controlo consiste em comparar o desempenho obtido numa determinada situação com o desempenho esperado para essa mesma situação. Para que esta comparação seja possível, é necessário que as experiências sejam feitas num ambiente controlado, o que requer um esforço de preparação considerável. Compreende-se assim que, à medida que as capacidades dos robôs evoluem, seja cada vez mais complicado testar algoritmos e controladores em robôs reais.

1.1 Motivação

O uso de ferramentas de simulação revela-se particularmente apelativo nas primeiras fases de implementação e validação de controladores, já que permite que os vários testes sejam feitos num ambiente controlado, com elevada repetibilidade e em segurança. No entanto, a utilidade desta abordagem depende da capacidade de os modelos usados captarem a dinâmica do processo real.

Foi desenvolvido na Faculdade de Engenharia da Universidade do Porto, no âmbito do grupo de robótica móvel *5dpo* [1], um simulador vocacionado para o uso em robótica. Os simuladores são já comuns em aulas, onde seria impossível disponibilizar um robot por grupo de trabalho. O simulador em questão, o *SimTwo*, havia já dado provas neste meio, como ferramenta didáctica.

No entanto, ainda não tinha prestado provas num meio tão exigente como o do Futebol Robótico, onde a suas potencialidades podem-se provar valiosas.

1.2 Objectivos

A presente dissertação tem como objectivo validar a aplicabilidade do simulador *SimTwo* na modelação e simulação de uma classe de robôs. Mais concretamente, pretende-se validar a sua utilização como ferramenta realista para o teste de controladores, antes de os aplicar em robôs reais. Ainda que o trabalho se tenha centrado num dos robôs de futebol do grupo *5dpo*, espera-se que os resultados deste estudo sejam aplicáveis a outros robôs.

1.3 Estrutura da Dissertação

Numa primeira fase, a dissertação passou por um estudo do estado da arte, que é sumariado na secção 1.4 do presente capítulo. Foram também estudados vários aspectos do *hardware* e *software* dos robôs de futebol, tratados no capítulo 2. No capítulo 3 é descrita a ferramenta de medição da pose do robot, baseada em visão, usada extensivamente nas tarefas de estimação de parâmetros e validação de modelos. Finalmente, no capítulo 4 é tratado o software de simulação *SimTwo*.

A segunda fase do trabalho consistiu na identificação de um modelo adequado ao simulador, e na estimação dos seus parâmetros. Estes procedimentos são tratados no capítulo 5. Neste capítulo é também tratada a implementação da camada de software responsável pela comunicação transparente entre o software de decisão e o simulador.

No capítulo 6 são apresentados os resultados das várias fases do trabalho. Finalmente, no capítulo 7 são apresentadas conclusões, e propostas linhas de trabalho futuro.

1.4 Revisão Bibliográfica

A informação relevante para a elaboração do presente trabalho será revista nesta secção. Em primeiro lugar, foi necessário fazer um estudo detalhado do robot usado no trabalho. Foram também estudadas as várias ferramentas necessárias, tais como a plataforma de simulação e a ferramenta de medição da pose do robot. Finalmente foram estudados métodos de modelação e de estimação de parâmetros.

Relativamente ao estudo do robot, em [1] é encontrada alguma informação relativa à equipa à qual ele pertence, e em [2] é explicada a competição para a qual o robot foi construído. Em [3] é feita a descrição dos robôs *Middle Size League* da equipa *5dpo* para o ano de 2004. Ainda que o artigo esteja desactualizado, grande parte da informação relativa à estrutura conceptual dos robôs ainda corresponde à realidade.

Foi feito também um breve estudo dos robôs *Small Size League* da equipa *5dpo*, uma vez que parte do sistema de localização visual foi adaptado para uso no presente trabalho. Em [4] é

descrito todo o sistema associado aos robôs *Small Size League* da equipa *5dpo* para 2004. Em [5] é descrito o processamento de imagem usado para a localização dos robôs.

A documentação disponível referente à plataforma de simulação usada no trabalho pode ser encontrada em [6], sendo que para certos detalhes foi consultado o autor [?] da referida aplicação. Foi também feito um breve estudo de outras ferramentas semelhantes, para comparação, tais como o *Gazebo* [7] e o *Microsoft Robotics Developer Studio* [8].

Foi também necessário consultar a documentação da biblioteca de simulação de corpos rígidos *ODE*, usada pelo simulador, e documentada em [9, 10]. Uma biblioteca que oferece funcionalidades semelhantes, a biblioteca *Bullet*, é documentada em [11].

Para o desenvolvimento de código *Pascal* foi relevante a informação relativa ao compilador *Free Pascal Compiler* [12], bem como a relativa ao ambiente de desenvolvimento *Lazarus* [13]. A biblioteca *Pascal Script*, usada pelo simulador para interpretação de código *Pascal*, é documentada em [14].

A documentação do formato *XML*, usado pelos ficheiros de configuração do simulador, é feita em [15].

Para a modelação e estimação de parâmetros, foram estudados alguns trabalhos semelhantes [16, 17, 18, 19, 20]. Para o processo de estimação da altura do centro de massa do robot, é relevante a informação presente em [21].

Capítulo 2

Descrição do robot

O robot tratado nesta dissertação pertence à equipa de robôs de futebol do grupo *5dpo* [1]. Este robot foi construído para participar nas competições de futebol robótico *RoboCupSoccer* da *RoboCup Federation* [2]. O *RoboCup* divide as competições por modalidades, sendo o futebol robótico uma delas. Dentro da modalidade de futebol robótico existem várias ligas, sendo que este robot em particular faz parte da equipa construída para participar na *Middle Size League*, ou *MSL*. O robot pode ser visto na figura 2.1.

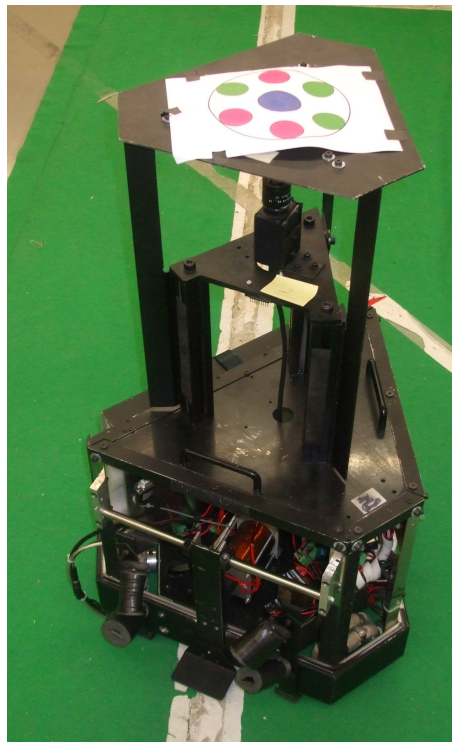


Figura 2.1: Fotografia do robot usado nesta dissertação usando um marcador colorido

2.1 Descrição geral

Cada robot da equipa está equipado com a sensorização e o poder computacional necessários para poder desempenhar autonomamente as sua tarefa, que é jogar futebol. Há no entanto a possibilidade de usar algum poder computacional externo, sob a forma de um *software* de coordenação da equipa, denominado *Treinador*.

O robot possui um computador portátil para a maior parte das tarefas de aquisição de dados, controlo e decisão. Este portátil possui um sistema operativo *GNU/Linux*, mais concretamente sob a forma de uma distribuição *Mandriva*. Este assunto será tratado em maior detalhe na secção 2.3.

Ligado a este computador está um sistema de visão omnidireccional, constituído por uma câmara com interface *IEEE1394* e por um espelho convexo. Este sistema, ilustrado na figura 2.2, permite ao robot um campo de visão completo em torno de si. A imagem captada pelo robot pode ser vista na figura 2.9.

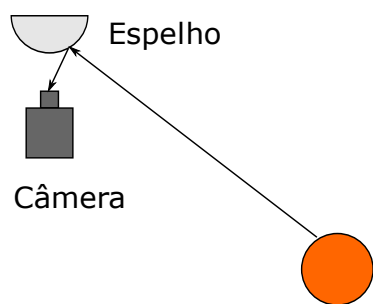


Figura 2.2: Ilustração do princípio de funcionamento do sistema de visão omnidireccional do robot

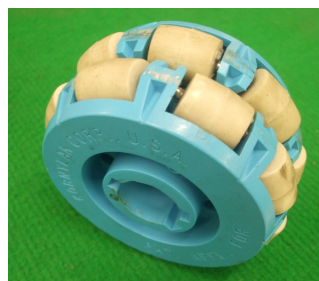


Figura 2.3: Fotografia de uma roda omnidireccional, mostrando os roletes dispostos na sua periferia

Ligados a este computador estão também alguns controladores periféricos, responsáveis pelo desempenho de algumas tarefas de controlo com requisitos temporais mais apertados. Estes módulos comunicam com o computador por intermédio de um interface em *daisy chain* especial, usando o protocolo *RS-232*. Estes módulos, bem como o protocolo usado, serão descritos em maior pormenor na secção 2.2.

O robot possui um sistema de locomoção também omnidireccional, de três rodas. Este sistema usa umas rodas especiais, ditas *omnidireccionais*, que exercem tracção na direcção perpendicular ao eixo de rotação (direcção tangencial), mas permitem a livre movimentação na direcção paralela a este eixo (transversal). Isto é conseguido por meio de roletes, distribuídos em torno da periferia da roda. Na figura 2.3 é mostrada a fotografia de uma roda deste tipo.

Este sistema permite ao robot a deslocação independente segundo as direcções longitudinal e transversal, bem como rodar sobre si próprio. Esta capacidade é ilustrada nas figuras 2.4, 2.5 e 2.6.

Cada roda está associada a um sistema de controlo de velocidade, constituído por um motor, um *encoder* óptico e um controlador de velocidade. Este sistema de controlo periférico, é descrito em pormenor na secção 2.2.2.

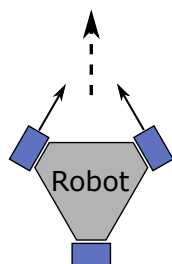


Figura 2.4: Ilustração da movimentação em frente do robot

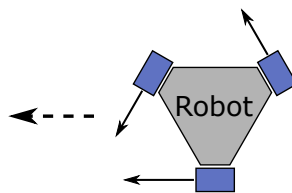


Figura 2.5: Ilustração da movimentação lateral do robot

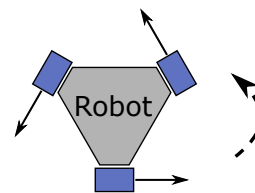


Figura 2.6: Ilustração da movimentação rotativa do robot

Da mesma forma, o robot possui um controlador periférico responsável pela função de chuto, o módulo *kicker*. Este sistema é descrito na secção 2.2.3.

2.2 Periféricos em Daisy Chain

Como já referido, o robot possui alguns controladores periféricos, responsáveis pelo processamento local de certas tarefas. Estes controladores permitem libertar o computador de tarefas simples e repetitivas, mas que requerem níveis de latência muito baixos, e por isso normalmente só disponíveis em sistemas operativos de tempo real.

2.2.1 Protocolo série

Estes controladores comunicam com o computador por intermédio de um protocolo próprio, baseado no protocolo série *RS-232*. Este protocolo adapta a ligação série *RS-232* a uma topologia em anel. Excluindo os detalhes relativos à topologia em anel, todo o protocolo eléctrico e de formato de dados obedece à norma.

Neste protocolo há apenas um dispositivo *master*, função desempenhada pelo computador portátil. Este é o único capaz de iniciar a comunicação. Como ilustrado na figura 2.7, cada dispositivo é ligado de forma a que o seu terminal de saída seja ligado ao terminal de entrada do próximo dispositivo na cadeia. A comunicação é unidireccional ao longo do anel, pelo que cada dispositivo tem forçosamente que reenviar a informação recebida para que esta percorra o anel.

Cada carácter de dados é enviado de acordo com a norma *RS-232*, com a configuração de 8 *bits* de dados, um *stop bit* e sem *bit* de paridade. A taxa de *bits* é configurada para 115200 *baud*. Por sua vez, os caracteres são organizados em pacotes com uma estrutura definida. Apenas o *master* pode gerar pacotes. Aos dispositivos apenas é permitido alterar os caracteres existentes em cada pacote, não podendo consumir ou gerar caracteres adicionais. Desta forma, o comprimento do pacote não se altera à medida que passa pela cadeia.

Cada pacote possui um cabeçalho, uma secção de dados enviados pelo *master* (denominados *data in*), e uma secção de dados destinados ao *master* (denominados *data out*). Ambas as secções

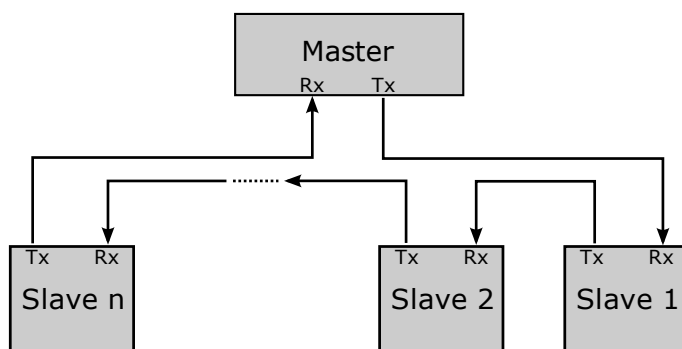


Figura 2.7: Esquema da ligação *daisy chain* entre os vários controladores

de dados possuem um comprimento variável, definido previamente pelo *master* no cabeçalho do pacote.

Byte	Bits							
	7	6	5	4	3	2	1	0
1	'D'							
2	'C'							
3	I45		O45		B	n		
4	I				O			
5	ma							
6..n	DataIn							
n..m	DataOut							

Tabela 2.1: Formato do pacote usado na comunicação série

O formato do pacote é mostrado na tabela 2.1. Os vários campos são resumidos na tabela 2.2, e descritos de seguida.

Início do pacote

Todos os pacotes são iniciados com uma sequência de dois caracteres 'D' e 'C'. Esta sequência permite aos dispositivos sincronizarem-se com o início do pacote.

Endereço do dispositivo actual

O campo *node number* é um campo de 3 *bits* que funciona como contador de saltos pela cadeia. É identificado na tabela 2.1 pela letra *n*. Este contador é enviado com zero pelo *master*, e vai sendo incrementado à medida que passa por cada dispositivo. Desta forma, é possível a cada dispositivo

Campo	Descrição
'D'	Caracter 'D', parte da sequência de início de pacote
'C'	Caracter 'C', parte da sequência de início de pacote
I45	<i>Bits</i> 4 e 5 do valor de número <i>bytes</i> de dados de entrada
O45	<i>Bits</i> 4 e 5 do valor de número <i>bytes</i> de dados de saída
B	<i>Flag</i> indicativa de modo <i>broadcast</i>
n	Endereço do dispositivo actual
I	<i>Bits</i> 0 a 3 do valor de número de <i>bytes</i> de dados de entrada
O	<i>Bits</i> 0 a 3 do valor de número de <i>bytes</i> de dados de saída
ma	Máscara contendo a informação dos destinatários do pacote
DataIn	Dados de entrada, destinados aos dispositivos
DataOut	Dados de saída, destinados ao <i>master</i>

Tabela 2.2: Resumo dos campos do pacote usado na comunicação série

saber a sua posição na cadeia, que é usada no endereçamento de pacotes. A largura de três bits permite no máximo 8 dispositivos na cadeia.

Máscara dos dispositivos endereçados

O campo *node mask* é identificado na tabela 2.1 pelas letras *ma*. O campo *node mask* é uma máscara de 8 *bits*, em que cada *bit* representa um dispositivo, de acordo com a sua ordem. O primeiro dispositivo na cadeia é representado pelo *bit* 0, o segundo pelo *bit* 1, e por aí em diante.

Se o valor do *bit* correspondente a um dispositivo for 1, esse dispositivo é destinatário da mensagem contida no pacote. Se o valor for 0 o dispositivo não é destinatário, devendo ainda assim reenviar o pacote adequadamente para o próximo na cadeia.

Esta máscara, usada em conjunto com o campo *node number*, permite endereçar vários dispositivos presentes na cadeia simultaneamente.

Número de *bytes* de entrada

O número de *bytes* de entrada é um valor binário de 6 *bits* de largura. Desta forma, são permitidos 63 *bytes* de dados de entrada, que se destinam aos dispositivos. Este valor é distribuído pelos campos *I* e *I45* da tabela 2.1. O campo *I* contém os *bits* menos significativos, de 0 a 3, enquanto que o campo *I45* contém os dois *bits* mais significativos, 4 e 5.

Dados de entrada

Este campo, de dimensão variável, contém a mensagem a ser enviada para os dispositivos, sendo designado *DataIn* na tabela 2.1. A sua dimensão é indicada no cabeçalho do pacote, como já referido, e pode variar entre 0 e 63 *bytes*. O primeiro *byte* de dados de entrada é usado normalmente para identificar o tipo de mensagem.

Número de bytes de saída

O número de *bytes* de saída é também um valor binário de com 6 *bits* de largura. Este valor corresponde ao número de *bytes* de dados de resposta, destinados ao *master*, no máximo 63. Este valor é distribuído pelos campos *O* e *O45* da tabela 2.1. O campo *O* contém os *bits* menos significativos, de 0 a 3, enquanto que o campo *O45* contém os dois *bits* mais significativos, 4 e 5.

Dados de saída

Este campo, identificado como *DataOut* na tabela 2.1, é preenchido pelos dispositivos com os dados que estes pretendem enviar para o *master*. É o *master* quem define a dimensão deste campo, como já referido, sendo este também responsável pelo preenchimento do campo com valores iniciais. Como referido em 2.2.1, os dispositivos apenas podem alterar os caracteres recebidos, não podendo gerar ou consumir *bytes*. O comprimento do conteúdo de dados de saída pode variar entre 0 e 63 *bytes*.

Broadcast

Este campo de um *bit*, identificado na tabela 2.1 pela letra *B*, permite especificar o modo de transmissão. Os modos permitidos são o modo *Broadcast*, com este *bit* a um, e o modo *Compact*, com este *bit* a zero. Este campo apenas tem significado quando a mensagem se destina a mais que um dispositivo, ou seja, quando o número de *bits* activos no *node mask* é maior do que um.

O modo *Broadcast* é usado quando se pretende enviar a mesma mensagem para todos os destinatários indicados pela *node mask*. Desta forma o *payload* de dados contém apenas uma cópia da mensagem.

O modo *Compact* é usado quando se pretende enviar uma mensagem do mesmo tipo para vários dispositivos, indicados pela *node mask*. Neste modo, o primeiro *byte* de dados de saída continua a ser usado para identificar o tipo de mensagem. No entanto, os restantes dados de entrada e de saída são particionados de acordo com o número de dispositivos endereçados. Evita-se assim, nestes casos, o envio de uma mensagem por dispositivo endereçado, o que permite minimizar o *overhead* introduzido pelo cabeçalho do pacote.

2.2.2 Módulo de controlo de motores

O robot possui 3 controladores deste tipo, sendo cada um responsável pelo controlo do motor associado a cada roda, bem como pela contagem de impulsos provenientes do *encoder* óptico associado ao motor. Esta cadeia de actuação é ilustrada na imagem 2.8.

O módulo inclui um controlador *PID* (*Proporcional, Integral, Derivativo*) em tempo discreto, usado para controlo de velocidade da roda. O período de amostragem deste controlador, implementado num micro-controlador *ATMega8*, é de 10 ms.

O controlador recebe do computador uma referência de velocidade, e calcula a tensão média a ser aplicada no motor a cada ciclo de controlo. Por uma questão de segurança, se a referência de

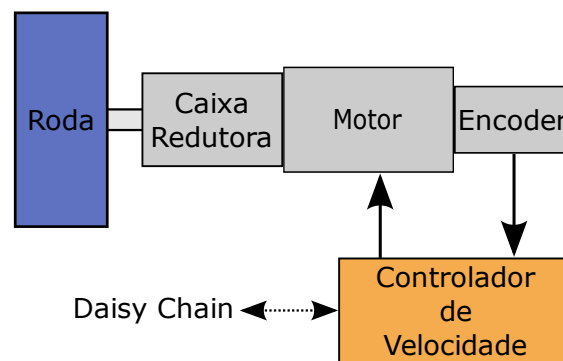


Figura 2.8: Ilustração da cadeia de actuação associada a cada roda

velocidade não for actualizada ao fim de um determinado tempo, esta é reposta a zero. Evitam-se assim acidentes no caso de falha de comunicações com o *software* de controlo, presente no computador portátil.

A tensão média aplicada ao motor é obtida por modulação *Pulse Width Modulation* da tensão de alimentação de potência, que vale 24 V. A corrente é limitada em *hardware* a cerca de 10 A, por forma a proteger o sistema.

O módulo possui também um segundo micro-controlador, responsável pela contagem de impulsos provenientes do *encoder* óptico. Desta forma é feita a realimentação de velocidade do controlador *PID*.

Mensagem *PACK_TYPE_PID_PWM*

Esta mensagem permite definir a referência de velocidade do motor, enviada em *impulsos por 10 ms* do *encoder* óptico. Para além desta referência é também enviado o valor de *PWM* a ser usado pela malha de antecipação do controlador de velocidade. O formato da mensagem é mostrado na tabela 2.3, sendo a descrição dos vários campos feita na tabela 2.5.

Mensagem *PACK_TYPE_READ_ENC*

Esta mensagem é usada para leitura do estado do controlador. Os campos de saída são preenchidos com zero pelo *master*, sendo posteriormente preenchidos com os valores correctos pelo dispositivo destinatário da mensagem. O formato da mensagem é mostrado na tabela 2.3, sendo os vários campos de resposta descritos na tabela 2.5.

2.2.3 Módulo de controlo de chuto

O robot possui também um outro controlador, desenvolvido para controlar o dispositivo electromagnético apelidado de *kicker*, usado para *chutar* a bola. Actualmente, este controlador é

Byte	7	6	5	4	3	2	1	0
I1	7							
I2	MSpeed_H							
I3	MSpeed_L							
I4	PwmRef_H							
I5	PwmRef_L							
I6	Checksum							

Tabela 2.3: Formato da mensagem *PACK_TYPE_PID_PWM*

Byte	Bits							
	7	6	5	4	3	2	1	0
I1	9							
O1	TS_H							
O2	TS_L							
O3	Enc_H							
O4	Enc_L							
O5	V							
O6	Pwm							
O7	Rm	Re	Ee	x			Dir	
O8	Checksum							

Tabela 2.4: Formato da mensagem *PACK_TYPE_READ_ENC*

Campo	Descrição
7	Valor 7, identifica o tipo de mensagem <i>PACK_TYPE_PID_PWM</i>
MSpeed_H	Byte mais significativo da referência de velocidade
MSpeed_L	Byte menos significativo da referência de velocidade
PwmRef_H	Byte menos significativo da referência de <i>PWM</i> de antecipação
PwmRef_L	Byte menos significativo da referência de <i>PWM</i> de antecipação
CheckSum	Byte menos significativo da soma dos restantes bytes da mensagem, usado para verificação de integridade dos dados

Tabela 2.5: Resumo dos campos da mensagem *PACK_TYPE_PID_PWM*

Campo	Descrição
7	Valor 9, identifica o tipo de mensagem <i>PACK_TYPE_READ_ENC</i>
TS_H	Byte mais significativo do <i>timestamp</i> da mensagem
TS_L	Byte menos significativo do <i>timestamp</i> da mensagem
ENC_H	Byte mais significativo do contador de impulsos do <i>encoder</i>
ENC_L	Byte menos significativo do contador de impulsos do <i>encoder</i>
V	Soma de 10 valores da medida da corrente no motor
PWM	Referência actual de <i>PWM</i>
Rm	Bit indicador da ocorrência de um <i>reset</i> do controlador do motor
Re	Bit indicador da ocorrência de um <i>reset</i> do controlador do <i>encoder</i>
Ee	Bit indicador de erro na contagem de impulsos do <i>encoder</i>
Dir	Bit indicador da direcção do motor
CheckSum	Byte menos significativo da soma dos restantes bytes da mensagem, usado para verificação de integridade dos dados

Tabela 2.6: Resumo dos campos da mensagem *PACK_TYPE_READ_ENC*

também responsável por algumas tarefas de aquisição de dados. Mais concretamente, faz a leitura de um sensor electromagnético inteligente, usado como bússola, e do estado de um relé de segurança, usado para desligar os motores.

2.3 Software de controlo

Como referido anteriormente, é usado um computador portátil para o controlo de alto nível. Este computador corre um sistema operativo *GNU/Linux*, que para além de ser *open source*, é robusto e fiável. Sobre o sistema operativo correm duas aplicações, que comunicam entre si por *UDP*.

As aplicações foram desenvolvidas na linguagem *Pascal*, usando o ambiente de desenvolvimento *Lazarus* [13]. Este ambiente de desenvolvimento usa, por sua vez, o compilador *Free Pascal Compiler* [12] (ou *FPC*), e ambos são *open source*. O *FPC* disponibiliza um conjunto de bibliotecas ditas de *run time*, que permitem ao programador uma abstracção relativamente ao sistema operativo para o qual o programa é compilado.

Uma das aplicações, o *OVis*, é responsável pelo processamento de imagem e pela interacção com o *hardware*, por intermédio do sistema operativo. A outra aplicação, o *MDecOmni3*, é por sua vez responsável por todo o controlo e decisão de *alto nível* (em oposição ao controlo de *baixo nível*, efectuado pelos vários módulos periféricos). Estas aplicações serão descritas brevemente adiante, sendo focadas principalmente as suas características relevantes para o trabalho, mais concretamente, a sua contribuição para o processo de controlo.

2.3.1 Aplicação de processamento de imagem e de interface com o *hardware*

A aplicação *OVis* funciona como uma camada de abstracção para a aplicação de decisão, comunicando com este por *UDP*. Na figura 2.9 pode ser vista uma captura do écran de imagem desta aplicação em execução. Na figura 2.10 é mostrada a janela de ampliação, que é usada para ampliar a janela de imagem principal. Finalmente, na figura 2.11 é mostrada a janela de controlo e configuração da aplicação.

O *OVis* desempenha funções de processamento de imagem semelhantes às desempenhadas pela ferramenta de medição da pose do robot, que será descrita mais à frente, no capítulo 3. A este nível, a aplicação permite extrair da imagem informação lógica, que é posteriormente enviada para a aplicação de decisão, o *MDecOmni3*.

Para além do processamento de imagem, o *OVis* desempenha algumas funções relacionadas com a comunicação com os vários módulos periféricos, ligados ao computador através da porta série. A este respeito, o *OVis* actua maioritariamente como um intermediário entre o *MDecOmni3* e o *hardware*.

O *OVis* é também responsável pela sincronização do ciclo de processamento, sendo ele quem inicia cada ciclo. Para isso o *OVis* recorre a uma interrupção de *software* gerada de cada vez que é recebida uma imagem (um *frame*) da câmara. Assim que a interrupção é atendida, o *OVis*

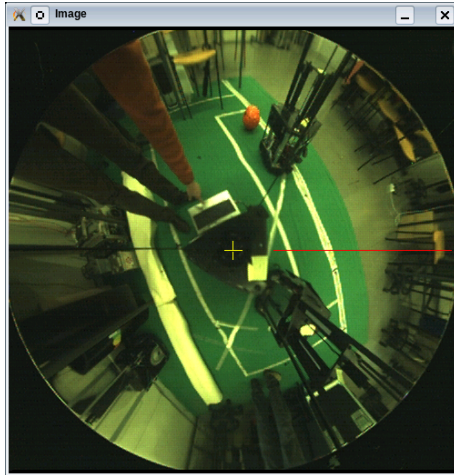


Figura 2.9: Captura da janela de imagem da aplicação OVIs

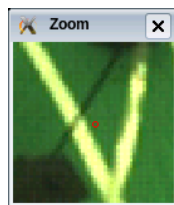


Figura 2.10: Captura da janela de zoom da aplicação OVIs

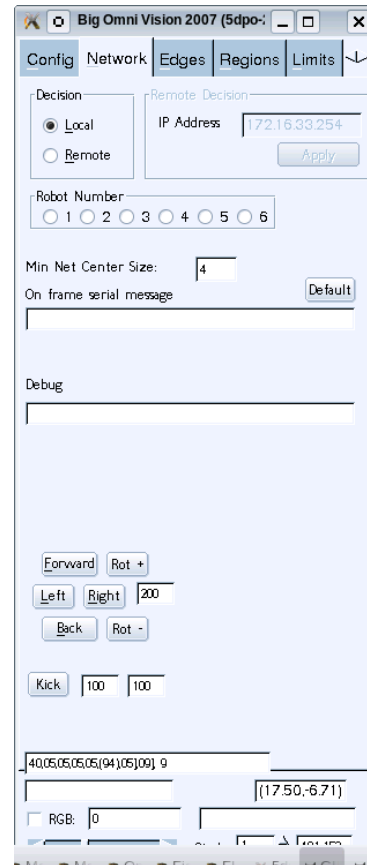


Figura 2.11: Captura da janela de controlo e configuração da aplicação OVIs

efectua o pedido dos dados referentes à odometria, enviando pela porta série um pacote do tipo *PACK_TYPE_READ_ENC*. O processamento da imagem recebida é efectuado enquanto a resposta ao pedido é aguardada.

Terminado o processamento de imagem, e chegada a resposta ao pedido, o *OVIs* procede ao envio destes dados para o *MDecOmni3*, por intermédio de um pacote *UDP*. O pacote proveniente dos periféricos é simplesmente embebido na sua secção do pacote *UDP*, não havendo qualquer tratamento desta informação por parte do *OVIs*.

O *OVIs* está permanentemente à espera de pacotes provenientes do *MDecOmni3* e endereçados à sua porta *UDP* 6969. Estes pacotes contêm a trama a ser enviada para os módulos periféricos. Uma vez recebido este pacote, a trama é enviada directamente pela porta série, finalizando desta forma o ciclo de controlo.

2.3.2 Aplicação de controlo e decisão

A aplicação *MDecOmni3* tem como responsabilidade a decisão e o controlo de *alto nível* (em oposição ao controlo de *baixo nível* efectuado pelos vários módulos periféricos). Mais concretamente, tem a seu cargo a localização do robot e da bola, o controlo da velocidade e posição do

robot, e toda a decisão e planeamento necessários ao jogo de futebol robótico.

Esta aplicação oferece uma série de mecanismos e ferramentas de teste, úteis durante o desenvolvimento e afinação da equipa de robôs. De entre estas ferramentas destacam-se um simulador rudimentar, uma ferramenta de teste e calibração dos controladores dos motores, entre outras. Na figura 2.12 é mostrada uma captura da janela da aplicação.

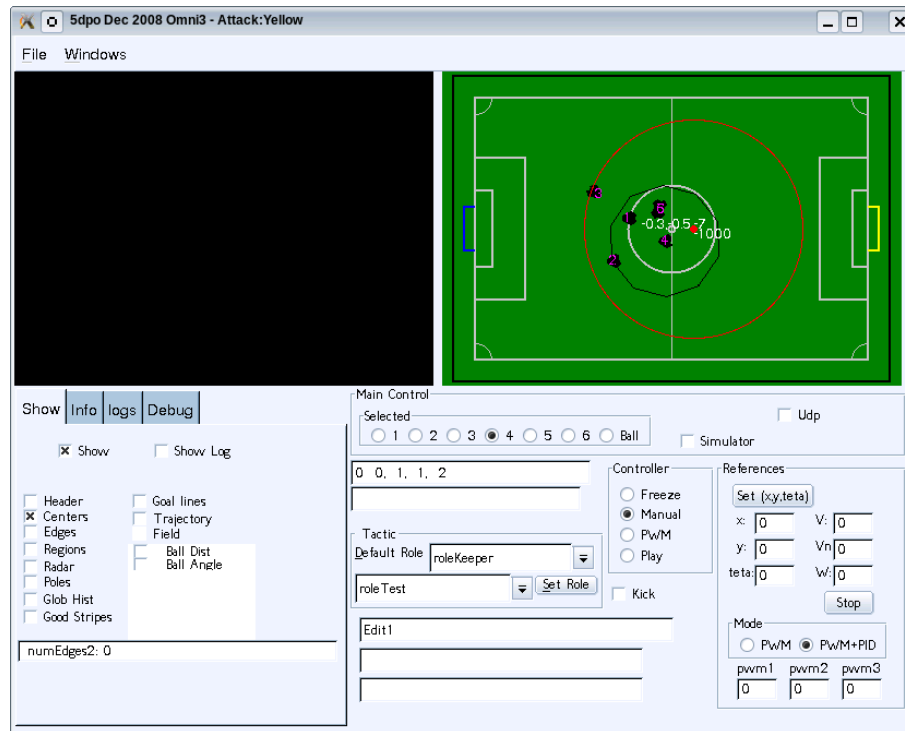


Figura 2.12: Captura da janela da aplicação *MDecOmni3*

O *MDecOmni3* participa de forma passiva no ciclo de controlo, esperando indefinidamente a chegada de um pacote *UDP* proveniente do *OVis* e endereçado à sua porta *UDP* 7171. Este pacote obedece a um formato descrito adiante, em 2.4.

Uma vez chegado o pacote, é desencadeado todo o processo de controlo e decisão. Neste processo é usada a informação proveniente do *OVis*, bem como informação interna ao robot. Se disponível, é também usada informação proveniente do *Treinador*, mencionado já em 2.1.

Entre os produtos deste processo encontram-se as referências da velocidade a aplicar em cada roda. Estas referências são usadas para a construção do pacote *daisy chain* para os controladores dos motores, que é por sua vez encapsulado num novo pacote *UDP*, enviado para a porta 6969 do *OVis*. Completado este envio, o *MDecOmni3* regressa ao estado de espera.

2.4 Comunicação entre aplicações

Como já foi referido, o controlo do robot é distribuído por duas aplicações. O *OVis*, responsável pelo processamento de imagem e pela interação com o *hardware*, foi já tratado em 2.3.1. Por

sua vez, a aplicação de controlo e decisão, o *MDecOmni3*, foi tratada em 2.3.2. Estas aplicações comunicam entre si através de uma ligação de rede *UDP*, sendo os dados enviados de uma forma estruturada. Esta estruturação simplifica a tarefa de extracção dos dados do pacote, possibilitando também a expansão futura.

2.4.1 Tipos de dados

A estrutura de dados usada para a comunicação entre o *OVis* e o *MDecOmni3* baseia-se em alguns tipos de dados básicos. A definição de tipos de dados básicos, com uma estrutura binária bem definida, é fundamental para a compatibilidade entre os intervenientes na comunicação. Todos os tipos usados ou são tipos padrão do *Free Pascal Compiler*, ou são compostos por estes tipos. A sua listagem é feita na tabela 2.7.

Tipo	Tamanho (bytes)	Detalhes
byte	1	Inteiro sem sinal, tipo básico do <i>FPC</i>
word	2	Inteiro sem sinal, tipo básico do <i>FPC</i>
angle	2	Inteiro sem sinal, obtido somando π ao ângulo em radianos, e multiplicando o resultado por 10000.
float	4	Número em vírgula flutuante, tipo básico do <i>FPC</i>
string	$1 + n$	O primeiro <i>byte</i> contém o comprimento da string (n), seguem-se n bytes com o conteúdo da string

Tabela 2.7: Tipos de dados usados na comunicação entre Ovis e MDec

2.4.2 Formato dos Pacotes

Como já referido, o pacote *UDP* é preenchido de forma estruturada, o que simplifica tanto a introdução como a extracção de dados, e permite também a futura expansão do protocolo. Por outro lado, o formato de dados é binário, permitindo um uso mais eficiente da largura de banda.

A estruturação do pacote consiste na sua divisão em blocos, ou *chunks*. Cada bloco é iniciado por um *byte* identificador, que permite distinguir os vários blocos. A organização dos dados dentro de cada bloco é específica desse mesmo bloco. Por essa razão, o comprimento do bloco não é especificado de forma unívoca, pelo que é necessário que o receptor do pacote compreenda minimamente o formato interno de todos os blocos passíveis de serem recebidos. De outro modo, não será capaz de saber onde acaba um bloco desconhecido, e onde se inicia o seguinte, o que se revela uma limitação do protocolo.

Actualmente estão definidos 6 blocos, ainda que alguns deles obedeçam à mesma estrutura interna. Os vários blocos são listados na tabela 2.8. Os identificadores 'C', 'e', 'R' e 'S' correspondem a *chunks* transmitidos do *OVis* para o *MDecOmni3*. Os identificadores '1' e '2' correspondem, por sua vez, aos *chunks* enviados do *MDecOmni3* para o *OVis*.

Identificadores	Designação	Uso
'C'	<i>Clusters</i>	Agrupamentos de píxeis da mesma cor
'e'	<i>Edges</i>	Detecção de transições
'R'	<i>Radar</i>	Detecção de obstáculos
'S', '1', '2'	<i>Serials</i>	Transmissão de pacotes <i>Daisy Chain</i>

Tabela 2.8: Blocos usados na comunicação entre Ovis e MDec

Por outro lado, ainda que a existência de um dado tipo de bloco no pacote não seja obrigatória, os blocos existentes devem obedecer a uma ordem específica. Esta é uma limitação da implementação, se bem que possa ser facilmente corrigida em caso de necessidade. A ordem obrigatória no pacote destinado ao *MDecOmni3* é *Clusters*, *Edges*, *Radar* e *Serials*. No caso do pacote destinado ao *OVis*, a ordem é *Serials* número 1, *Serials* número 2.

2.4.3 Formato dos blocos

O formato interno de cada bloco não é fixo, como já foi referido. Ainda que o significado dos vários campos de dados não seja directamente relevante para o presente trabalho, o conhecimento da estrutura dos vários blocos é importante para a correcta extracção dos dados dos pacotes. Os vários blocos serão resumidos de seguida.

Clusters

O bloco *Clusters* contém a informação referente aos agregados de píxeis determinados pelo processamento de imagem. O bloco é iniciado por um cabeçalho com 4 *bytes* de comprimento, seguido de um número variável de secções denominadas *Center*. O formato é mostrado na tabela 2.9.

Edges

O bloco *Edges* contém a informação referente a transições de cor na imagem, usadas para a detecção de linhas do campo. A detecção é feita segundo linhas radiais uniformemente distribuídas, centradas no robot. O seu número é configurável no *OVis*. O bloco é iniciado por um cabeçalho com 3 *bytes* de comprimento, seguido de um número variável de secções denominadas *Edge*. O formato é mostrado na tabela 2.10.

Radar

O bloco *Radar* contém a informação referente à presença de obstáculos nas proximidades do robot. O bloco é iniciado por um cabeçalho com 3 *bytes* de comprimento, seguido de um número variável de secções denominadas *Ray*. O formato é mostrado na tabela 2.11.

	Tipo	Conteúdo	Detalhes
Cabeçalho	byte	'C'	Identifica a secção
	byte	ncolors	Número de cores lógicas
	word	nclusters	Número de subsecções <i>center</i>
Secção <i>Center</i>	float	area	Número de píxeis do <i>cluster</i>
	angle	tetaMax	O valor máximo do ângulo horizontal do <i>cluster</i>
	angle	tetaMin	O valor mínimo do ângulo horizontal do <i>cluster</i>
	angle	roMax	O valor máximo do ângulo correspondente à distância do <i>cluster</i> ao centro do robot
	angle	roMin	O valor mínimo do ângulo correspondente à distância do <i>cluster</i> ao centro do robot
	angle	teta	O valor central do ângulo horizontal do <i>cluster</i>
	angle	ro	O valor central do ângulo correspondente à distância do <i>cluster</i> ao centro do robot

Tabela 2.9: Formato do bloco *Clusters*

	Tipo	Conteúdo	Detalhes
Cabeçalho	byte	'e'	Identifica a secção
	word	Edges	Número de subsecções <i>Edge</i>
Secção <i>Edge</i>	byte	lineNum	O número identificativo da transição
	byte	color12	4 <i>bits</i> mais significativos com o número da cor inicial, 4 <i>bits</i> menos significativos com o número da cor final
	angle	teta	O valor do ângulo horizontal da transição
	angle	ro	O valor do ângulo correspondente à distância da transição ao centro do robot

Tabela 2.10: Formato do bloco *Edges*

	Tipo	Conteúdo	Detalhes
Cabeçalho	byte	'R'	Identifica a secção
	word	nRays	Numero de subsecções <i>Ray</i>
Secção <i>Ray</i>	byte	lineNum	Número da linha de detecção
	byte	color	Cor do obstáculo
	byte	size	Tamanho do obstáculo
	angle	teta	O valor do ângulo horizontal do obstáculo
	angle	ro	O valor do ângulo correspondente à distância do obstáculo ao centro do robot

Tabela 2.11: Formato do bloco *Radar*

Serials

Os blocos *Serials* contêm a informação referente aos dados série de ou para a *Daisy Chain*, tratada em 2.2. No caso do bloco 'S', este contém os dados recebidos pelo *OVis*. No caso dos blocos '1' ou '2', estes contêm os dados gerados pelo *MDecOmni3*, e destinados a serem enviados pelas portas série principal ou auxiliar, respectivamente.

O bloco é constituído pelo *byte* identificador, seguido de uma *string* de comprimento variável, contendo os dados. O formato é mostrado na tabela 2.12.

Tipo	Conteúdo	Detalhes
byte	'S', '1' ou '2'	Identifica a secção
string	str	Cadeia de <i>bytes</i> contendo os dados série

Tabela 2.12: Formato dos blocos *Serials*

Capítulo 3

Ferramenta de medição da pose do robot

O software *SmHAL* surgiu no âmbito do projecto de Futebol Robótico *Small Size League* (*SSL*) da Faculdade de Engenharia da Universidade do Porto. Nesta modalidade é permitido o recurso a sistemas de localização externos aos robôs, pelo que a equipa *5dpo* optou por usar um conjunto de duas câmaras montadas em pórticos e direccionadas para o chão. Cada uma destas câmaras é responsável por captar imagens da sua metade do campo. O sistema tem assim acesso visual a todo o campo de jogo. Os detalhes da implementação de 2004 podem ser consultados em [4].

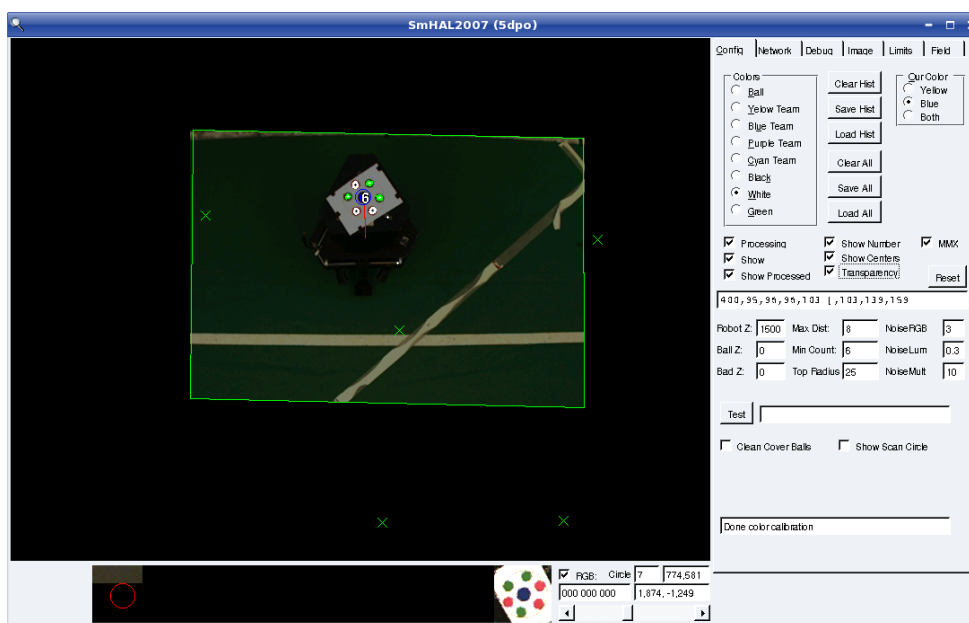
Tendo em conta as necessidades de processamento de imagem em tempo real, optou-se por atribuir a cada câmara uma instância de um software próprio. Este software, o *SmHAL*, permite aos processos de decisão uma abstracção relativamente a todo o processamento de imagem.

Cada instância do *SmHAL* associa-se a uma câmara, e é responsável pela captura de imagem e pela extracção de informação de alto nível referente à sua metade do campo de jogo. No final do processo esta informação de alto nível é enviada por *UDP/IP* ao software de decisão. Desta forma o sistema global pode ser distribuído por vários computadores de acordo com as necessidades.

É incluída neste capítulo uma breve explicação do processamento de imagem envolvido. Esta explicação é limitada ao estritamente necessário para a compreensão do funcionamento geral deste software. Uma explicação com maior detalhe pode ser encontrada em [5].

3.1 Processamento de imagem

O funcionamento do *SmHAL* assenta na sua capacidade de agregar píxeis próximos na imagem em grupos de cores lógicas. A agregação é feita com base na cor *RGB* de cada píxel e com base na proximidade entre os píxeis. São criados assim agregados (*clusters*) de píxeis, cuja cor lógica define a sua classe.

Figura 3.1: Captura da Janela do *SmHAL*

As cores lógicas permitidas no *SmHAL* são apresentadas na tabela 3.1. Estas cores lógicas, apesar de terem nomes semelhantes aos das cores reais tipicamente usadas, não têm nenhuma associação obrigatória com estas. Por outro lado, cada cor lógica tem uma função específica no programa.

A correspondência entre as cores lógicas e as cores da imagem é feita recorrendo a um processo de calibração manual. Neste processo, e para cada cor lógica usada, o utilizador selecciona na imagem alguns píxeis que lhe correspondam.

Depois da segmentação da imagem, o software *SmHAL* procura identificar na imagem os vários objectos de jogo, nomeadamente a bola e aos vários robôs. A bola é identificada unicamente pela sua cor. Para a identificação dos vários robôs são usados marcadores coloridos com um código individual.

Cor Lógica	Descrição
Ball	A bola
Yellow Team	Cor central do marcador da equipa amarela
Blue Team	Cor central do marcador da equipa azul
Purple Team	Não usada
Cyan Team	Não usada
Black	Não usada
White	Cor clara do código dos marcadores dos robôs
Green	Cor escura do código dos marcadores dos robôs

Tabela 3.1: Cores lógicas usadas no *SmHAL*

3.2 Marcadores

Os marcadores são aplicados na face superior de cada robot, sendo usados para os identificar individualmente. Estes marcadores permitem ao *SmHAL* saber não só a posição do robot, como também a sua orientação.

Um dos marcadores é mostrado na figura 3.2. Como se pode ver, este marcador possui um círculo central cuja cor corresponde à cor da equipa. À volta do círculo central estão dispostos 6 círculos mais pequenos, que correspondem a um código individual binário. É este código que permite uma identificação unívoca de cada robot, bem como o cálculo da sua orientação.



Figura 3.2: Robot *SSL* com marcador na face superior

3.3 Cálculo da posição

O *SmHAL*, como já referido, envia por rede, ao software cliente, a posição dos vários objectos visuais. Esta posição pode ser referenciada ou à imagem, ou a um referencial fixo configurável. Na maior parte dos casos interessa que a posição seja referenciada ao referencial do campo. Por outro lado, a posição dos objectos é calculada inicialmente em coordenadas da imagem. Assim, é necessário ao *SmHAL* transformar as coordenadas da imagem para coordenadas do campo. A transformação tem que ter em conta factores como a posição da câmara, a sua distância aos marcadores ou efeitos como a *distorção em barril*. Sempre que a câmara é reposicionada, é necessário calibrar o sistema para que tenha em conta todos estes factores.

3.4 Protocolo de rede

Como já referido, o *SmHAL* comunica com o cliente por intermédio de pacotes *UDP*. Antes da comunicação, o *SmHAL* preenche uma estrutura do tipo *TlanPacket* com os dados relevantes. Uma vez preenchida a estrutura, o seu conteúdo em memória é copiado directamente para o pacote.

Do lado da ligação *UDP*, no cliente, deverá ser feito o processo inverso. Este processo consiste em copiar o conteúdo binário do pacote para a posição de memória de uma estrutura do mesmo tipo. Desta forma, o cliente obtém uma cópia da estrutura original.

A estrutura é composta por algumas estruturas mais simples. A estrutura *TMyTimeVal*, usada para conter valores temporais, é definida pelo seguinte código *Pascal*:

```
TMyTimeVal=record
  tv_sec : longword ;
  tv_usec : longword ;
end ;
```

A estrutura *TLanCenter*, por sua vez, permite representar aglomerados de píxeis, contendo informação de posição e de dimensão do aglomerado. A sua definição em código é o seguinte:

```
TLanCenter=packed record
  x,y : single ;
  count : word ;
end ;
```

A estrutura *TLanRobot* permite representar os robôs da equipa. A posição do robot em questão é representada por uma sub-estrutura *TLanCenter*, sendo também incluída a informação referente ao número do robot, à sua orientação e à qualidade da sua detecção. A estrutura é a seguinte:

```
TLanRobot=packed record
  center : TLANCenter ;
  number : integer ;
  angle , badness : single ;
end ;
```

A estrutura *TLanPacket* possui também um cabeçalho, a estrutura *TLanHeaderPacket*, definida pelo seguinte código:

```
TLanHeaderPacket= packed record
  PacketType : integer ;
  SerialNumber : integer ;
  CameraID : integer ;
  FrameTime : TMyTimeVal ;
  SendTime : TMyTimeVal ;
end ;
```

Esta estrutura contém, entre outra informação, o *timestamp* correspondente ao instante da captura da imagem e o *timestamp* correspondente ao envio do pacote.

Finalmente, todas estas estruturas são usadas na construção da estrutura *TLanPacket*, que é definida pelo seguinte código:

```
TLanPacket=packed record
  Header : TLANHeaderPacket ;
  BallCenters : packed array [0.. MaxLanBallCenters -1] of TLANCenter ;
  NumBallCenters : integer ;
```

```

FoeCenters: packed array [0..MaxLanFoeCenters-1] of T LanCenter;
NumFoeCenters: integer;
Robots: packed array [0..MaxLanRobots-1] of T LanRobot;
NumRobots: integer;
end;

```

O *array* designado por *BallCenters* contém possíveis posições da bola. O *array FoeCenters* contém a posição dos robôs adversários, enquanto que o *array Robots* contém a informação referente aos robôs da equipa. A dimensão da estrutura é fixa, sendo o comprimento dos vários *arrays* definido pelas seguintes constantes:

```

const
  MaxLanBallCenters=32;
  MaxLanFoeCenters=12;
  MaxLanRobots=16;

```

Ainda que o comprimento dos vários *arrays* seja fixo, o número de posições preenchidas para cada um é variável, e dado respectivamente pelos inteiros *NumBallCenters*, *NumFoeCenters* e *NumRobots*.

3.5 Cliente de software

No âmbito desta dissertação, foi necessário efectuar algumas experiências que envolveram a movimentação do robot. Para essas experiências, descritas no capítulo 5, foi necessário determinar a posição e orientação do robot a cada instante de tempo.

O *SmHAL* foi usado nessas experiências como ferramenta de medição da pose do robot. Foi também necessário implementar um pequeno programa, cliente do *SmHAL*, que recebesse os dados de posição e orientação do marcador e os guarde num formato adequado.

Esse programa cliente foi implementado em *Lazarus*. Este *IDE* foi escolhido pelas suas qualidades de *Rapid Application Development*, bem como pela maior simplicidade de integração com as restantes ferramentas usadas pelo grupo *5dpo*, também feitas usando a linguagem de programação *Pascal*.

Na figura 3.3 é mostrada uma captura da janela da aplicação cliente. Nela pode ser visto, no canto superior esquerdo, o grupo de controlos *Robot Position*, que são constantemente actualizados com os dados de posição e orientação provenientes do *SmHAL*. Do seu lado direito estão os controlos *Logging*, que controlam a captura de dados. Finalmente, por baixo destes controlos encontra-se a caixa de texto para onde são capturados os valores de posição e orientação, bem como a informação temporal do instante da captura. O processo de captura é activado quando a caixa de selecção *Capture* é seleccionada. Carregar no botão *Save* faz com que o conteúdo da caixa de texto seja guardado num ficheiro, e carregar no botão *Clear* limpa a referida caixa.

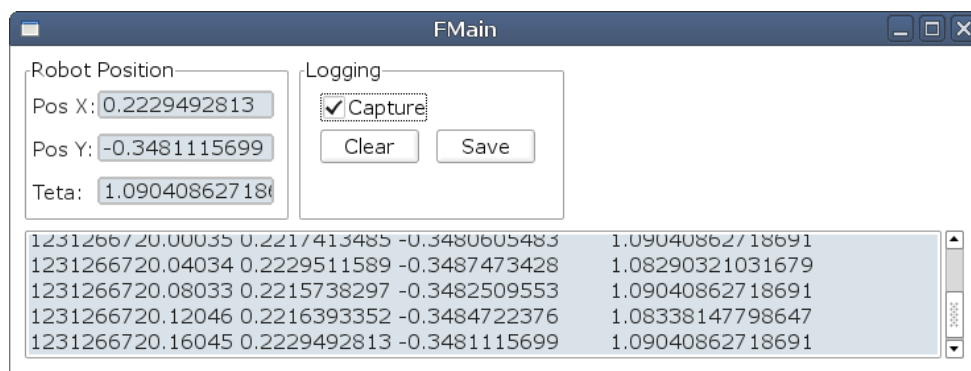


Figura 3.3: Captura da Janela do SmallClient

Capítulo 4

Plataforma de simulação

A plataforma de simulação *SimTwo* [6] foi desenvolvida no âmbito do grupo de robótica móvel *5dpo* [1], pelo Prof. Dr. Paulo Costa. Esta plataforma evidencia um cuidado especial no realismo, especialmente a nível de dinâmica e de modelos. Estes incluem características não lineares, que se revelam importantes no comportamento dos objectos reais. O simulador permite também a visualização 3D em tempo real da simulação, o que se revela útil a nível da supervisão da simulação.

4.1 Descrição Geral

Na figura 4.1 é mostrada uma captura da janela de visualização 3D do simulador. Esta janela é usada para supervisão da simulação, sendo os parâmetros da visualização definidos no separador *Graphics* da janela de configuração, visível na figura 4.3.

A nível de controlo do robot, o simulador possui apenas uma solução completamente implementada, se bem seja suficiente para a grande maioria dos casos de uso, pela sua grande flexibilidade. Esta solução consiste num intérprete de *Pascal*, incluído no simulador com recurso à biblioteca *open source Pascal Script* [14]. Com recurso a este intérprete, é possível implementar o controlo dos robôs unicamente em *Pascal Script*, ou usando o código como uma camada de interface com um controlador externo.

Os parâmetros de controlo dos robôs são configurados no separador *Control* da janela de configuração, visível na figura 4.2. Os parâmetros referentes aos interfaces de entrada e saída são configurados no separador *IO* da janela de configuração, mostrada na figura 4.4. O simulador permite também registar a progressão de algumas variáveis ao longo do tempo, sendo mostrada na figura 4.6 a janela de gráficos. Finalmente, na figura 4.5 é mostrada a janela do editor de código *Pascal Script*. Os detalhes do controlo serão abordados na secção 4.4 do presente capítulo.

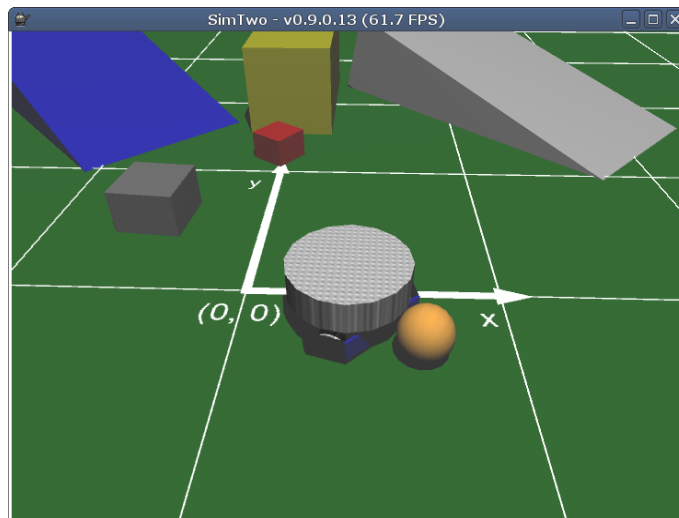


Figura 4.1: Captura da janela de visualização 3D do *SimTwo*

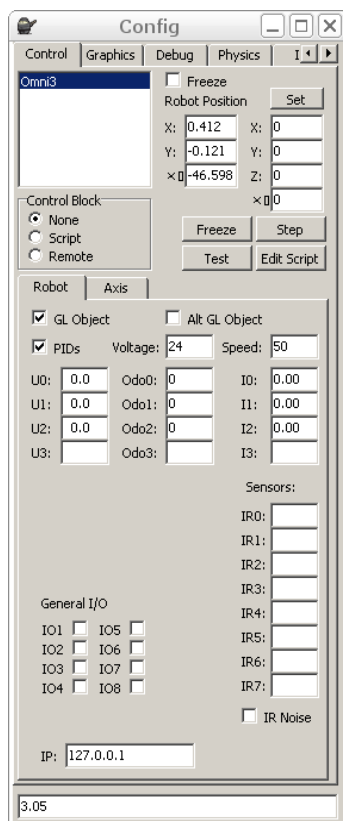


Figura 4.2: Captura da janela de configuração do controle do *SimTwo*

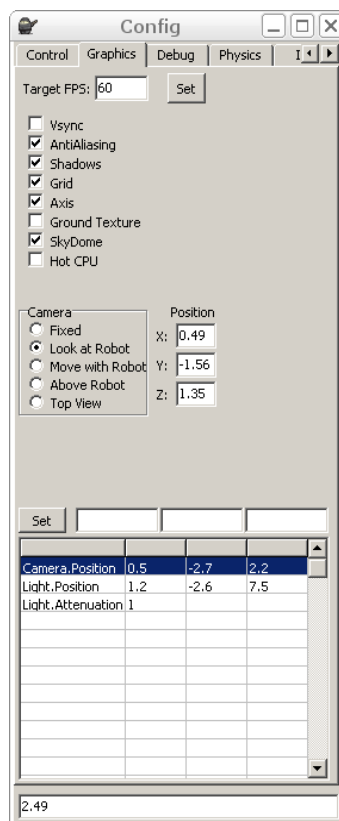


Figura 4.3: Captura da janela de configuração da visualização do *SimTwo*

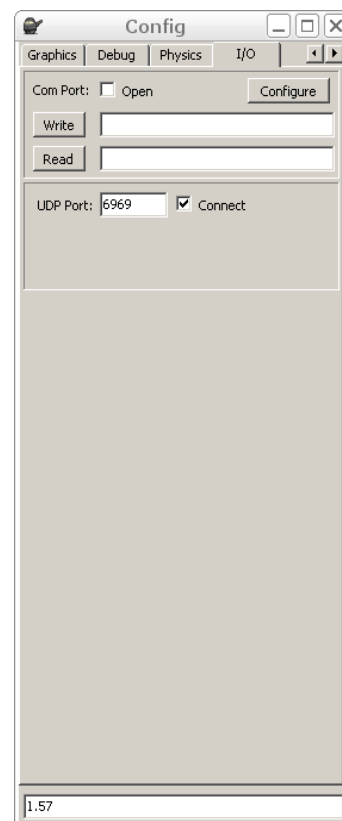


Figura 4.4: Captura da janela de configuração de entrada/saída do *SimTwo*

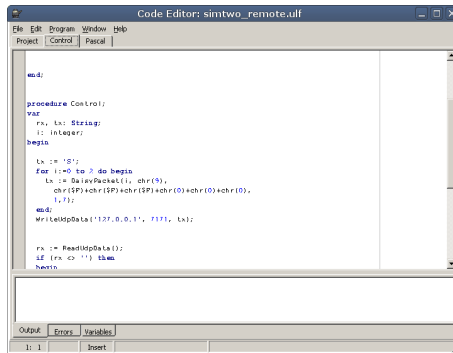


Figura 4.5: Captura da janela do editor de código do *SimTwo*

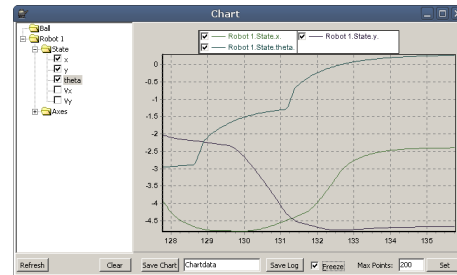


Figura 4.6: Captura da janela de gráficos do *SimTwo*

4.2 Biblioteca de Simulação de Corpos Rígidos

O *SimTwo* usa a biblioteca *ODE* (*Open Dynamics Engine*) [9, 10] para a simulação de corpos rígidos. Esta biblioteca é vocacionada para a simulação de estruturas articuladas, tais como veículos terrestres ou criaturas com pernas. Os robôs enquadram-se normalmente neste grupo.

A biblioteca é direccionada para a simulação em tempo real, pelo que é dada prioridade à rapidez e estabilidade face à precisão. Citando a documentação:

ODE emphasizes speed and stability over physical accuracy.

A biblioteca supõe dois níveis de simulação, o nível de simulação da dinâmica, e o nível de simulação de colisões. Estes níveis são propositadamente separados, a fim de maximizar a flexibilidade e utilidade da biblioteca. Desta forma, é possível usar uma biblioteca de simulação de colisões externa juntamente com a biblioteca interna de simulação da dinâmica, ou vice-versa.

A componente de simulação da dinâmica ocupa-se das propriedades dinâmicas dos corpos, tais como massas ou velocidades, enquanto que a componente de simulação de colisões ocupa-se exclusivamente da forma dos corpos. Ao nível da simulação dinâmica, os corpos são desprovidos de forma, sendo representados pelas suas massa e momentos de inércia, como é referido na documentação:

Note that the shape of a rigid body is not a dynamical property (except insofar as it influences the various mass properties). It is only collision detection that cares about the detailed shape of the body.

4.2.1 Simulação da dinâmica de corpos rígidos

Na simulação da dinâmica de corpos rígidos estão envolvidos dois tipos de entidades, as entidades *body* e as entidades *joint*. As entidades *body* representam os corpos rígidos, e correspondem a distribuições arbitrárias de massa, representadas pela sua massa e pela matriz 3×3 do seus momentos de inércia. Sobre os corpos são exercidos binários e forças, que podem ser aplicados

Tipo	Descrição	Graus de liberdade
Ball	Uma junção de rótula	3 angulares
Hinge	Uma junção do tipo dobradiça	1 angular
Slider	Uma junção de deslizamento linear	1 linear
Universal	Uma junção do tipo <i>cardan</i>	2 angulares
Hinge2	Duas junções de dobradiça em série (usada em situações especiais)	2 angulares
Fixed	União fixa entre corpos (útil apenas para teste da biblioteca)	nenhum
Contact	Uma junção de contacto	(especial)
AMotor	Junção especial que permite controlar as velocidades angulares relativas entre dois corpos	(especial)

Tabela 4.1: Tipos de junção suportados pela biblioteca *ODE*

directamente ou por intermédio de junções com outros corpos. As entidades *joint* correspondem a junções entre os corpos, permitindo a transmissão de forças e binários entre estes. As junções impõem restrições ao movimento, sendo que a cada tipo de junção corresponde a determinadas restrições.

Os vários tipos de junção suportados pela biblioteca são descritos na tabela 4.1. Para mais detalhes, consultar a documentação da biblioteca em [10].

Na figura 4.7 é ilustrado o princípio conceptual de funcionamento de uma junção do tipo *hinge*, cujas restrições são semelhantes às apresentadas por uma dobradiça. Como se pode perceber, esta junção permite apenas o movimento angular relativamente a um eixo, mas previne qualquer tipo de movimento linear.

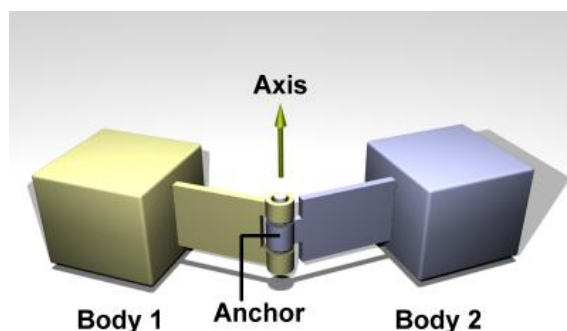


Figura 4.7: Imagem ilustrativa do princípio de funcionamento de uma *joint* do tipo *hinge* (cortesia de Russel Smith, autor da biblioteca *ODE*)

A junção *AMotor* não tem uma representação física concreta. Esta *joint* permite apenas aplicar velocidades angulares entre corpos, pelo que é habitual usá-la em conjunto com outras *joints* que

definem ligações mais restritas entre os corpos. Pode ser, por exemplo, aplicada em conjunto com uma ligação *hinge* para controlar o ângulo entre dois segmentos de um braço robótico.

4.2.2 Simulação de colisões

A simulação de colisões é feita separadamente da simulação da dinâmica, e ocupa-se da forma dos corpos. A sua função é exclusivamente a de determinar os pontos de contacto entre objectos. Estes pontos são usados para criar junções especiais, ditas de contacto, as *contact joints*, que são normalmente usadas para evitar que os objectos penetrem uns nos outros. Estas junções permitem também simular fricção entre os pontos de contacto, aplicando-se forças na direcção do atrito, perpendiculares à normal das superfícies, como representado na figura 4.8.

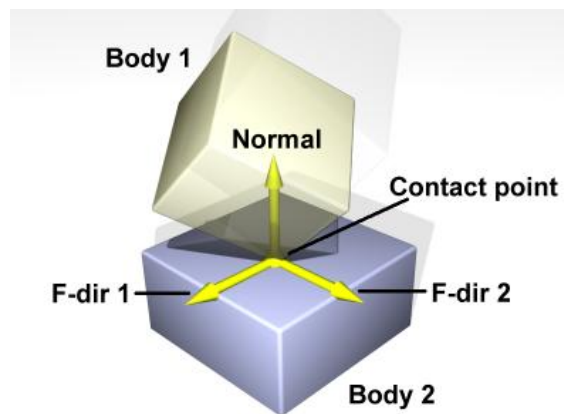


Figura 4.8: Imagem ilustrativa do princípio de funcionamento de uma *joint* de contacto (cortesia de Russel Smith, autor da biblioteca *ODE*)

4.3 Configuração do simulador

A configuração do simulador é feita por intermédio de ficheiros *Extensible Markup Language*, ou *XML* [15]. Este formato permite a definição de estruturas hierárquicas de informação. Neste caso concreto, estas estruturas tomam a forma de objectos, colocados num mundo virtual, e agrupados hierarquicamente num esquema de secções e subsecções.

4.3.1 Mundo envolvente

No topo da hierarquia de configuração está a secção *scene*, contida no ficheiro *scene.xml*. Esta secção contém todas as secções usadas na definição do *mundo envolvente*, como é chamado na documentação o ambiente virtual onde decorre a simulação.

Dentro da secção *scene* são permitidas algumas subsecções, que por sua vez dividem os elementos do *mundo envolvente* em classes. Estas secções são mostradas na tabela 4.2.

Secção	Descrição
robot	Instância de um robot
obstacles	Estruturas inamovíveis
things	Corpos livres que podem ser movidos
track	Ainda não implementada, permitirá o desenho de uma pista ou de marcadores no chão

Tabela 4.2: Secções permitidas dentro da secção *scene*

As secções *robot* serão descritas em pormenor adiante, e correspondem às entidades controláveis na simulação. As secções *things* e *obstacles* correspondem aos objectos não controláveis da simulação, sendo feita a distinção entre aqueles que podem ser movidos, *things*, daqueles que são inamovíveis, *obstacles*.

Embora toda a informação possa, em princípio, ser incluída apenas no ficheiro *scene.xml*, é mais cómodo dividi-la por vários ficheiros. A semântica usada pelo simulador permite esta definição mediante o atributo *file*. Por exemplo, o seguinte código corresponde à definição de uma secção *obstacles* cujo conteúdo está no ficheiro *obstacles.xml*:

```
<obstacles file='obstacles.xml' />
```

O código estrutural de um ficheiro *scene.xml* de exemplo, corresponde ao seguinte:

```
<?xml version="1.0" ?>
<scene>
  <robot>
    <!-- Um robot -->
  </robot>
  <robot>
    <!-- Outro robot -->
  </robot>
  <things file='things.xml' />
  <obstacles file='obstacles.xml' />
  <track file='track.xml' />
</scene>
```

Os ficheiros de *things* e de *obstacles* obedecem a uma estrutura semelhante. Como exemplo, a estrutura do ficheiro *things.xml*, referenciado no exemplo anterior, é a seguinte:

```
<?xml version="1.0" ?>
<things>
  <sphere>
    <!-- Dados relativos a este objecto do tipo sphere -->
  </sphere>
  <!-- Outros objectos -->
</things>
```

A etiqueta *sphere* corresponde a um objecto esférico, que neste exemplo é colocado no ficheiro correspondente aos objectos móveis. Os objectos serão tratados com maior detalhe em 4.3.2.

4.3.2 Objectos

As estruturas virtuais são definidas recorrendo a alguns objectos básicos. Todos os objectos possuem algumas propriedades básicas, que permitem à plataforma de simulação *SimTwo* construir internamente estruturas compatíveis com a biblioteca de simulação de corpos rígidos *ODE*. Estes objectos básicos são usados quer na definição de objectos acessórios, como os correspondentes a *things* e *obstacles*, quer na definição dos próprios robôs.

Há vários tipos básicos de objectos, que são sumariados na tabela 4.3. Embora alguns tipos sejam meramente estruturais, como por exemplo o *cuboid*, alguns representam entidades mais complexas, como por exemplo o *sensor*. As propriedades básicas dos objectos são descritas na tabela 4.4.

Tipo	Descrição
<i>cuboid</i>	Um paralelepípedo
<i>cylinder</i>	Um cilindro
<i>sphere</i>	Uma esfera
<i>belt</i>	Uma esteira rolante
<i>sensor</i>	Um sensor

Tabela 4.3: Objectos básicos suportados pelo *SimTwo*

Propriedade	Componentes	Descrição
id	value	String identificadora do objecto
mass	value	Massa do objecto, em kg
size	x, y, z	Tamanho em x, y e z, em metros
pos	x, y, z	Posição em x, y e z, em metros
rot_deg	x, y, z	Orientação sobre o eixo x, y e z, em graus
color_rgb	r, g, b	Cor do objecto na base <i>RGB</i> , com as intensidades de <i>red</i> , <i>green</i> e <i>blue</i> sendo números de 8 bit

Tabela 4.4: Propriedades dos básicos dos objectos do *SimTwo*

4.3.3 Instâncias de Robôs

Os robôs são as entidades controláveis no simulador. São permitidas várias instâncias destas entidades, sendo que a cada uma corresponde uma secção *robot* dentro da secção *scene*.

Cada instância de um robot possui algumas propriedades, listadas na tabela 4.5. A maioria destas propriedades é partilhada com objectos básicos, referidos anteriormente. Há no entanto uma propriedade específica dos robôs, a propriedade *body*. Esta propriedade corresponde à informação de construção do corpo do robot, que necessita de existir num ficheiro externo, indicado pelo

Propriedade	Componentes	Descrição
id	name	String identificadora do objecto
pos	x, y, z	Posição em x, y e z, em metros
rot_deg	x, y, z	Orientação sobre o eixo x, y e z, em graus
body	file	Caminho do ficheiro que contém a descrição do corpo do robot

Tabela 4.5: Propriedades dos básicas dos robôs no *SimTwo*

parâmetro *file*. Desta forma é possível existirem na simulação vários robôs usando o mesmo ficheiro de construção. Os detalhes deste ficheiro são discutidos na secção 4.3.4.

Um exemplo do código usado na declaração de um robot, colocado dentro da secção *scene*, é o seguinte:

```
<robot>
  <id name='Omni3' />
  <pos x='0' y='-1' z='0' />
  <rot_deg x='0' y='0' z='0' />
  <body file='f2000.xml' />
</robot>
```

4.3.4 Construção dos Robôs

O ficheiro *XML* correspondente à construção do corpo, invocado pela propriedade *body* de cada instância de robot, obedece a uma estrutura própria. A secção raiz é a secção *robot*, e contém algumas subsecções usadas para descrever os vários componentes do robot. As subsecções permitidas são descritas na tabela 4.6.

Secção	Descrição
solids	Sólidos com massa
shells	Superfícies sem massa, usadas apenas para colisões
articulations	Articulações
wheels	Rodas, usadas para locomoção
sensors	Sensores

Tabela 4.6: Secções permitidas dentro da secção *robot* do ficheiro de descrição do corpo do robot

As secções *solids* e *shells* serão descritas com maior detalhe em 4.3.5, enquanto que as secções *articulations* e *wheels* serão descritas em 4.3.6. A secção *sensors* não será discutida, por não ser relevante para o presente trabalho.

4.3.5 Sólidos e Cascas

As secções *solids* e *shells* permitem definir os vários corpos rígidos que constituem o robot. Para isso são usados alguns dos objectos básicos tratados em 4.3.2, nomeadamente os objectos

cuboid, *cylinder* e *sphere*.

Os objectos da secção *solids* possuem distribuição de massa e forma, pelo que participam tanto na simulação dinâmica como na simulação de colisões (ver 4.2). Por outro lado, os objectos de *shells* possuem somente forma, pelo que apenas participam na simulação de colisões.

O seguinte código cria uma casca cilíndrica, com 1 m de diâmetro e 1 m de altura:

```
<shells>
  <cylinder>
    <id value='1' />
    <pos x='0' y='0' z='0.5' />
    <size x='1' z='1' />
  </cylinder>
</shells>
```

Para transformar esta casca em um sólido com 5 kg de massa uniformemente distribuída, é necessário usar o seguinte código:

```
<solids>
  <cylinder>
    <id value='1' />
    <mass value='5' />
    <pos x='0' y='0' z='0.5' />
    <size x='1' z='1' />
  </cylinder>
</solids>
```

4.3.6 Rodas e Articulações

As rodas e as articulações correspondem ao sistema de actuação dos robots. A secção *articulations* permite definir ligações articuladas entre corpos, enquanto que a secção *wheels* permite simplificar a construção de um sistema locomotor de rodas, ao criar os corpos e articulações necessárias para o efeito.

Ambas as secções obedecem a uma estrutura muito semelhante, em que são definidas inicialmente propriedades comuns, sendo posteriormente criada uma subsecção para cada elemento, contendo as suas definições individuais. Em ambas as secções, as definições comuns são colocadas numa subsecção *default*. No caso da secção *wheels*, a cada roda corresponde uma subsecção *wheel*, enquanto que no caso da secção *joints*, a cada ligação corresponde uma subsecção *joint*. A estrutura de código usada para criar um robot de três rodas será semelhante ao seguinte:

```
<wheels>
  <default>
    <!-- comum a todas as rodas -->
  </default>
```

```

<wheel>
  <!-- roda 1 -->
</wheel>
<wheel>
  <!-- roda 2 -->
</wheel>
<wheel>
  <!-- roda 3 -->
</wheel>
</wheels>

```

Cada roda ou articulação pode estar associada a um sistema de controlo, que supõe um motor eléctrico de corrente contínua com uma caixa redutora, um controlador e um *encoder* óptico para realimentação do controlador, entre outros. Os vários componentes do sistema de actuação permitidos pelo *SimTwo* são referidos na tabela 4.7. Os componentes relevantes para o trabalho, permitidos na secção *wheels*, serão descritos em maior detalhe no capítulo 5. Os referentes às articulações são descritos na documentação do simulador e na documentação da biblioteca *ODE* ([6, 10]).

Componente	Descrição
motor	Parâmetros relativos ao motor
gear	Parâmetros relativos à caixa redutora
friction	Parâmetros relativos à fricção na cadeia de actuação
encoder	Parâmetros relativos ao <i>encoder</i>
controller	Parâmetros relativos ao controlador
axis	Parâmetros relativos ao eixo da roda/articulação
omni	Indica que a roda é omnidireccional (apenas para <i>wheels</i>)
tyre	Parâmetros relativos à roda (apenas para <i>wheels</i>)
spring	Parâmetros elásticos da articulação (apenas para <i>articulations</i>)
pos	Posição da articulação (apenas para <i>articulations</i>)
type	Tipo de articulação (apenas para <i>articulations</i>)
connect	Objectos ligados pela articulação (apenas para <i>articulations</i>)
limits	Limites da articulação (apenas para <i>articulations</i>)

Tabela 4.7: Componentes da cadeia de actuação do *SimTwo*

4.4 Controlo

Como referido na secção 4.1, o controlo de *alto nível* é feito por intermédio de um programa em *Pascal*, interpretado com a ajuda da biblioteca *Lazarus Script*. Esta solução, associada à disponibilidade de alguns interfaces de entrada e saída, permite bastante flexibilidade na implementação do controlo.

Embora o comando dos vários motores simulados possa ser feita directamente através deste programa, é normalmente mais vantajoso usar os controladores de velocidade locais, referidos em 4.3.6. Assim, em vez de ser responsável por gerar as tensões a aplicar nos motores, o programa em *Lazarus Script* pode limitar-se a definir a referência desejada de velocidade ou posição.

4.4.1 Programa de controlo

O código é criado com recurso a um editor de texto especial, incluído no simulador. A janela deste editor é mostrada na figura 4.5. O código é guardado num formato especial, em ficheiros com extensão *ulf*.

O utilizador deve definir dois procedimentos, o procedimento *Initialize*, que é chamado uma vez quando o programa se inicia, e o procedimento *Control*, que é chamado a cada 40 ms. A estrutura do código será, pois, a seguinte:

```
procedure Control;  
begin  
    { Ciclo de controlo }  
end;  
  
procedure Initialize;  
begin  
    { Executado quando o programa se inicia }  
end;
```

O procedimento *Control* é usado tipicamente para implementar um ciclo de controlo discreto com um período de 40 ms. Durante este ciclo é possível recolher informação de estado da planta, bem como actuar sobre esta. O utilizador pode também definir outros procedimentos e funções, de acordo com a necessidade, o que se revela útil em programas mais complexos, ou quando é necessário controlar mais do que um robot.

4.4.2 Interacção com a simulação

Ainda que sejam permitidos vários robôs na mesma simulação, o programa de controlo é único, e deve ser capaz de os controlar a todos simultaneamente. Cada robot possui um número identificativo, iniciado em zero, e atribuído pela ordem em que é colocado no ficheiro *scene.xml*.

Por sua vez, cada robot pode possuir vários eixos controlados, ou *axis*, e vários corpos, ou *solid*. Cada um destes *axis* e cada um destes *solid* possuem, da mesma forma, um número identificativo iniciado em zero, e atribuído pela ordem em que a ligação ou roda, para os *axis*, ou solido, para os *solid*, aparece no ficheiro referido pela propriedade *body* da instância do robot.

A interacção com a simulação é feita por intermédio de alguns procedimentos e funções, que levam em conta esta hierarquia de informação. As funções e procedimentos que se aplicam globalmente a um robot possuem obrigatoriamente como primeiro argumento o número identificativo do robot. Este número pode ser obtido a partir do seu *ID* com recurso à seguinte função:

```
function GetRobotIndex(R: integer; ID: string): integer;
```

Os procedimentos que permitem obter as informações de posição e velocidade do robot possuem os seguintes protótipos:

```
procedure GetRobotPos2D(R: LongInt): TState2D;
procedure GetRobotVel2D(R: LongInt): TState2D;
procedure GetRobotVx(R: LongInt): Double;
procedure GetRobotVy(R: LongInt): Double;
procedure GetRobotW(R: LongInt): Double;
procedure GetRobotX(R: LongInt): Double;
procedure GetRobotY(R: LongInt): Double;
procedure GetRobotTheta(R: LongInt): Double;
```

Da mesma forma, no caso de funções aplicadas a um determinado eixo de um determinado robot, os argumentos obrigatórios de identificação do robot e do eixo obedecem à mesma ordem. O mesmo acontece para os sólidos. As funções que permitem obter o número identificativo a partir do *ID* do objecto são as seguintes:

```
function GetSolidIndex(R: integer; ID: string): integer;
function GetAxisIndex(R: integer; ID: string; i: integer): integer;
```

Como exemplo, os procedimentos que permitem definir referências de posição, velocidade, estado ou tensão do motor para o eixo *i* do robot *R* são as seguintes:

```
function SetAxisPosRef(R: LongInt; i: LongInt; aPos: Double);
function SetAxisSpeedRef(R: LongInt; i: LongInt; aSpeed: Double);
function SetAxisStateRef(R: LongInt; i: LongInt; aState: TAxisState);
function SetAxisVoltageRef(R: LongInt; i: LongInt; aVoltage: Double);
```

Da mesma forma, as funções que permitem obter informações sobre o estado de um determinado eixo de um determinado robot são as seguintes:

```
procedure GetAxisOdo(R: LongInt; i: LongInt): LongInt;
procedure GetAxisPos(R: LongInt; i: LongInt): Double;
procedure GetAxisPosDeg(R: LongInt; i: LongInt): Double;
procedure GetAxisPosRef(R: LongInt; i: LongInt): Double;
procedure GetAxisPosRefDeg(R: LongInt; i: LongInt): Double;
procedure GetAxisSpeed(R: LongInt; i: LongInt): Double;
procedure GetAxisSpeedDeg(R: LongInt; i: LongInt): Double;
procedure GetAxisSpeedRef(R: LongInt; i: LongInt): Double;
procedure GetAxisSpeedRefDeg(R: LongInt; i: LongInt): Double;
procedure GetAxisState(R: LongInt; i: LongInt): TAxisState;
procedure GetAxisStateRef(R: LongInt; i: LongInt): TAxisState;
procedure GetAxisTorque(R: LongInt; i: LongInt): Double;
```

Em todos os casos, estas funções possuem nomes auto-explicativos e obedecem a uma estrutura uniforme. A listagem de funções permitidas no código, bem como as variáveis e estruturas pré-definidas, é feita na documentação do simulador [6].

4.4.3 Comunicação com o exterior

O simulador permite, como referido atrás, a comunicação com o exterior por intermédio de um interface de porta série ou de um interface de rede. Esta comunicação permite que o controlo seja total o parcialmente feito fora do simulador, sendo que em todos estes casos o programa *Lazarus Script* actua como uma camada de ligação.

A comunicação por porta série é configurada no separador *IO* da janela de configuração, mostrado na figura 4.4. Esta janela permite também algumas facilidades de *debug* das comunicações série. A nível de programação, estão disponíveis duas funções para comunicações série, que são as seguintes:

```
procedure ReadComPort(): String;  
function WriteComPort(s: String);
```

A comunicação por rede é efectuada por *UDP*, sendo da mesma forma configurada no separador *IO* da janela de configuração. Nesta janela é necessário configurar a porta *UDP* de recepção, sendo necessário activar a *check box Connect* para activar a recepção. As rotinas disponibilizadas para a comunicação de rede são as seguintes:

```
procedure ReadUDPData(): String;  
function WriteUDPData(ToIP: String; ToPort: LongInt; s: String);
```

A rotina *ReadUDPData* devolve ao utilizador o conteúdo do *buffer* de recepção. Por outro lado a rotina *WriteUDPData* permite enviar a *string s* para a porta *ToPort* do endereço *ToIP*.

Capítulo 5

Modelo do robot

Como referido no capítulo introdutório, pretende-se que o modelo do robot seja descrito de forma compatível com a plataforma de simulação, descrita no capítulo 4. Esta plataforma, no que diz respeito à descrição do modelo, requer que esta seja feita de acordo com a semântica particular do ficheiro de configuração.

A descrição do modelo do robot é feita por intermédio de um ficheiro de texto com formato *XML*, que supõe três secções. A secção *solids* diz respeito aos sólidos que representam o corpo do robot, a secção *shells* diz respeito às superfícies usadas para representar superfícies de contacto para o modelo de colisões, e a secção *wheels* diz respeito a todo o sistema de movimentação, desde os motores até às rodas.

Depois de identificado um modelo adequado ao simulador, foi necessário estimar os seus parâmetros. Para isso foram idealizadas algumas experiências simples, a serem realizadas no laboratório. A ferramenta de medição da pose do robot, descrita no capítulo 3, revelou-se muito útil nas experiências que envolveram movimento, já que permitiu a captura em tempo real da posição e orientação do robot.

Adiante são descritos os vários componentes do modelo, bem como os respectivos processos de estimação de parâmetros.

5.1 Corpo do robot

O corpo do robot seria normalmente modelado por uma massa pontual com momentos de inércia definidos para os três eixos cartesianos. Ainda que a plataforma de simulação use internamente esta representação, é necessário usar um sólido tridimensional no ficheiro de configuração do modelo do robot. Este objecto é definido na secção *solids*.

O corpo do robot foi representado por um cilindro, cuja massa é homogeneamente distribuída. O código usado é o seguinte:

```
|< solids>
```

```

<cylinder>
  <id value='1' />
  <mass value='25' />
  <pos x='0' y='0' z='0.185' />
  <size x='0.228' z='0.2' />
</cylinder>
</solids>

```

O parâmetro *mass* corresponde à massa do robot M_{rb} , em kg.

Os parâmetros *pos* correspondem à posição do centro de massa face à origem do modelo, em metros. Esta origem está contida no plano de contacto com o solo, pelo que o valor da coordenada z corresponde à altura do centro de massa do robot, z_{cm} . A determinação deste parâmetro será tratada adiante, na secção 5.1.1.

Finalmente os parâmetros *size* correspondem às dimensões do cilindro, sendo que x e z correspondem, respectivamente, ao raio e à altura, em metros. O raio do cilindro permite ajustar o valor do momento de inércia pela seguinte relação, em que r_{cil} é o raio do cilindro e J é o momento de inércia:

$$r_{cil}^2 = 2 \frac{J}{M_{rb}} \quad (5.1)$$

A determinação do momento de inércia será tratada na secção 5.1.2, sendo depois calculado o raio do cilindro a partir do valor determinado. Para a mesma massa, distribuída homogeneamente, altura do cilindro h_{cil} apenas terá influência nos momentos de inércia em torno de x e y . Uma vez que o modelo não supõe, para já, este detalhe, e uma vez que este valor não tem qualquer influência no momento de inércia em torno de z , foi definido aproximadamente, segundo o pressuposto de que a massa se encontra concentrada numa faixa próxima do centro de massa.

5.1.1 Determinação da altura do Centro de Massa

O centro de massa é definido em [21] como sendo o ponto sobre o qual toda a massa (do objecto) pode ser considerada “concentrada” para o propósito de calcular o “primeiro momento”, isto é, massa vezes distância:

The center of mass is the point at which all the mass can be considered to be “concentrated” for the purpose of calculating the “first moment”, i.e., mass times distance.

Para duas massas, a posição do centro de massa do conjunto, x_{cm} , relaciona-se com a posição de cada uma das massas, x_1 e x_2 , da seguinte forma:

$$(m_1 + m_2)x_{cm} = m_1x_1 + m_2x_2 \quad (5.2)$$

Se estas duas massas forem unidas por uma barra de massa nula, e se essa barra for apoiada no fulcro x_{fc} , o binário exercido sobre o conjunto é dado por:

$$\tau = (x_1 - x_{fc})m_1g + (x_2 - x_{fc})m_2g \Leftrightarrow \quad (5.3)$$

$$\Leftrightarrow \frac{\tau}{g} + (m_1 + m_2)x_{fc} = m_1x_1 + m_2x_2 \quad (5.4)$$

Usando a equação 5.2 obtemos finalmente:

$$\frac{\tau}{g} = (m_1 + m_2)(x_{fc} - x_{cm}) \quad (5.5)$$

Como se pode ver, se a posição do fulcro coincidir com a do centro de massa, então a resultante dos binários gravíticos exercidos sobre o robot será nula, mantendo-se este em equilíbrio.

Com base nesta relação, foi idealizada uma experiência muito simples com o objectivo de determinar a altura do centro de massa do robot. O robot foi colocado na horizontal, sobre uma barra de suporte, como mostrado na figura 5.1. Uma vez equilibrado o melhor possível, e confirmando que se encontrava perfeitamente na horizontal, foi medida a distância entre a linha de apoio, sobre a barra, e o plano de apoio das rodas sobre o solo. Esta distância corresponde à altura do centro de massa.

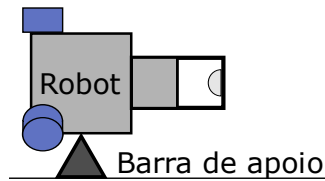


Figura 5.1: Experiência de determinação da altura do centro de massa do robot

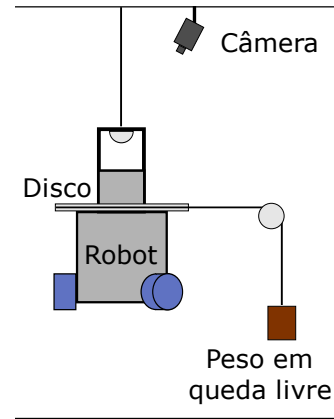


Figura 5.2: Experiência de determinação do momento de inércia do robot

5.1.2 Determinação do Momento de Inércia

Para a determinação do momento de inércia do robot, este foi suspenso no ar, a partir de um ponto no seu eixo de rotação, por um fio de massa desprezável. A torção exercida por este fio supõe-se também desprezável. Nesta situação, o atrito exercido sobre o robot é mínimo.

Foi fixo ao corpo do robot um disco ranhurado, cujos efeitos sobre o momento de inércia se supõe também mínimos. Em torno deste disco foi enrolado um fio, ao qual foi aplicada uma força constante por intermédio de uma massa em queda livre, como representado na figura 5.2. Desta forma, o robot foi submetido a um binário constante, sobre o eixo vertical de rotação.

A posição angular do robot foi medida a partir de uma situação de repouso, recorrendo à ferramenta de medição da pose do robot, descrita no capítulo 3. Desta forma, foi possível registar a evolução da sua velocidade angular.

A velocidade angular ω do robot é relacionada com o binário resultante τ nele aplicado pela seguinte equação:

$$\tau(t) = J \frac{d\omega(t)}{dt} \quad (5.6)$$

em que J é o momento de inércia do robot.

Por outro lado, o binário exercido sobre o robot, τ , e a massa do objecto em queda livre, m_q , relacionam-se da seguinte forma vectorial:

$$\tau(t) = \vec{r}_q \times \vec{f}_q(t) \quad (5.7)$$

$$\vec{f}_q(t) = m_q \vec{g} \quad (5.8)$$

em que $\vec{f}_q$ é a força aplicada pelo objecto em queda livre, $\vec{r}_q$ o seu braço e $\vec{g}$ a aceleração da gravidade.

O módulo do braço da força aplicada sobre o robot é igual ao raio do disco, R_q . Uma vez que a força $\vec{f}_q$ é constante para cada experiência, também o é o binário τ . Daí se pode obter a seguinte relação escalar:

$$\tau = R_q m_q g \quad (5.9)$$

Usando a equação 5.6, obtemos finalmente:

$$J \frac{d\omega(t)}{dt} = R_q m_q g \Leftrightarrow \quad (5.10)$$

$$\Leftrightarrow J = \frac{R_q m_q g}{\frac{d\omega(t)}{dt}} \quad (5.11)$$

A experiência foi repetida para vários valores de m_q , tendo sido medida a orientação do robot, e a partir desta, estimada a velocidade angular instantânea, ω .

Os valores da aceleração angular $\frac{d\omega(t)}{dt}$ foram estimados, para para cada experiência, usando-se o método dos mínimos quadrados. A partir destes valores, foi calculada a estimativa do momento de inércia para cada experiência.

5.2 Superfícies de colisão

A plataforma de simulação permite modelar e simular colisões entre objectos. Os sólidos, como o usado para o corpo do robot, possuem já um modelo de colisão criado automaticamente. No entanto, e para simplificar a construção de modelos, é possível usar a secção *shells* para definir superfícies de colisão mais complexas.

Os parachoques do robot, por serem relativamente complexos, foram definidos nesta secção. Foram usados *cubóides* para representar os 6 segmentos que compõem os parachoques. Para isso os parachoques do robot foram medidos, sendo posteriormente recriados para o robot simulado.

5.3 Cadeia de Actuação

O sistema de actuação engloba as rodas e os motores, com as respectivas caixas redutoras, encoders e controladores. Este sistema corresponde à secção *wheels* do ficheiro de configuração.

Abaixo está representado o esqueleto da secção referente à cadeia de actuação. Dentro desta secção há uma subsecção *default*, onde são especificados os parâmetros gerais, usados por todas as rodas. Posteriormente, é usada uma subsecção *wheel* para cada roda, onde são especificados os parâmetros específicos dessa roda.

```
<wheels>
  <default>
    <!-- Parametros gerais -->
  </default>
  <wheel>
    <!-- Parametros individuais da roda 1-->
  </wheel>
  <wheel>
    <!-- Parametros individuais da roda 2-->
  </wheel>
  <wheel>
    <!-- Parametros individuais da roda 3-->
  </wheel>
</wheels>
```

No caso concreto deste projecto, todas as definições foram especificadas de forma geral, na subsecção *default*. Apenas a orientação dos eixos das rodas necessitou de ser especificada individualmente, pelo que foi incluída na respectiva subsecção *wheel*.

5.3.1 Rodas omnidireccionais

O robot modelado é um robot omnidireccional de três rodas, pelo que usa as rodas especiais descritas no capítulo 2. O simulador permite activar este tipo de roda mediante a presença da etiqueta *omni*. As rodas possuem também outros parâmetros, definidos na etiqueta *tyre*. O código referente às rodas é o seguinte:

```
<omni />
<tyre mass='0.565' radius='0.051' width='0.042' centerdist='0.195' />
```

O parâmetro *mass* corresponde à massa da roda M_{wh} , em kg. Os parâmetros *radius* e *width* correspondem, respectivamente, ao raio R_{wh} e à espessura L_{wh} da roda, em metros. O parâmetro *centerdist* corresponde à distância R_{rb} da roda ao centro do robot, também em metros.

Todos estes parâmetros foram definidos de forma geral. Os únicos parâmetros especificados individualmente no modelo deste robot foram os referentes aos eixos das rodas. As três rodas do robot foram especificadas da seguinte forma:

```

<wheel>
  <axis angle=' -60' />
</wheel>
<wheel>
  <axis angle=' 60' />
</wheel>
<wheel>
  <axis angle=' 180' />
</wheel>

```

O número de rodas é determinado através do número de subsecções *wheel*. Em cada uma destas subsecções, o parâmetro *angle*, da etiqueta *axis*, corresponde ao ângulo do eixo de rotação da roda. Este ângulo, diferente para cada uma das rodas, é especificado em graus, e relativamente à frente do robot.

5.3.2 Motor

O modelo de motor DC incluído no simulador é definido pelo seguinte código:

```

<motor ri=' 0.316 ' ki=' 0.0302 ' vmax=' 24 ' imax=' 10 ' active=' 1 ' />

```

O parâmetro r_i corresponde à resistência dos enrolamentos, em Ω , o parâmetro k_i é a constante que relaciona o binário produzido pelo motor com a corrente que por ele passa, em Nm/A. O parâmetro v_{max} corresponde à tensão máxima no motor, em V, e o parâmetro i_{max} corresponde à corrente máxima permitida no motor, em A.

Os parâmetros r_i e k_i foram retirados da *datasheet* do motor. Ainda que não sejam exactos, estes valores são suficientes para o âmbito do trabalho. O valor v_{max} corresponde à tensão de alimentação usada, e o valor i_{max} corresponde sensivelmente ao limite de corrente imposto pelo controlador usado nos motores reais.

5.3.3 Caixa redutora

Os motores estão ligados às rodas por intermédio de uma caixa redutora, com uma relação de transmissão de 12 para 1. O simulador permite esta configuração, mediante o uso de uma etiqueta *gear*, contendo o valor da relação de transmissão, *ratio*:

```

<gear ratio=' 12 ' />

```

5.3.4 Encoder óptico

Cada motor possui um *encoder* óptico ligado ao seu veio. Este dispositivo é usado pelo controlador para realimentação de velocidade. Na plataforma de simulação, o *encoder* óptico é definido pelo seguinte código:

```

<encoder ppr=' 256 ' mean=' 0 ' stdev=' 0 ' />

```

O parâmetro ppr corresponde ao número de impulsos por revolução do veio do motor. O simulador permite também a injeção de ruído gaussiano na saída do *encoder*, ainda que não seja usado neste modelo. Os parâmetros $mean$ e $stdev$ correspondem, respectivamente, à média e ao desvio padrão desse ruído.

5.3.5 Controlador

Cada motor supõe um controlador de velocidade, que tem como objectivo manter a velocidade das rodas o mais próxima possível de um valor de referência. Este controlador é implementado no robot real como um controlador *PID* discreto. O simulador supõe também este tipo de controlador, ainda que a sua configuração seja um pouco diferente.

O controlador real toma a seguinte forma:

$$V_{out} = K_p e + K_i \int e dt + K_d \frac{de}{dt} + K_f \omega_{ref} \quad (5.12)$$

$$e = (\omega_{ref} - \omega) \quad (5.13)$$

O controlador simulado obedece a uma equação semelhante. Em ambos os controladores o valor V_{out} é medido em Volt. No entanto, enquanto que no robot real o valor de ω é medido em impulsos do *encoder* por ciclo, no caso do controlador simulado este valor é medido em rad/s . O mesmo acontece com o valor do erro, e , que num caso é medido em impulsos por ciclo e no outro em rad/s .

Assim, se o valor de ω do robot real for medido em impulsos por ciclo, e se o valor ω' do robot simulado for medido em rad/s , a sua relação é dada por:

$$\omega = \frac{NT}{2\pi} \omega' \quad (5.14)$$

em que N é o número de impulsos por revolução da roda e T é o período de amostragem do controlador.

Usando a equação 5.13 obtemos a seguinte relação

$$e = \frac{NT}{2\pi} (\omega'_{ref} - \omega') \quad (5.15)$$

Por outro lado, no caso do controlador simulado, temos:

$$V_{out} = K'_p e' + K'_i \int e' dt + K'_d \frac{de'}{dt} + K'_f \omega'_{ref} \quad (5.16)$$

$$e' = (\omega'_{ref} - \omega') \quad (5.17)$$

Usando as equações 5.15 e 5.17, obtemos:

$$e' = \frac{2\pi}{NT} e \quad (5.18)$$

Substituindo este valor em 5.16, e usando a equação 5.14, obtemos:

$$V_{out} = \frac{2\pi K'_p}{NT} e + \frac{2\pi K'_i}{NT} \int e dt + \frac{2\pi K'_d}{NT} \frac{de}{dt} + \frac{2\pi K'_f}{NT} \omega_{ref} \quad (5.19)$$

A partir desta equação podemos facilmente relacionar os ganhos do controlador real com os ganhos do controlador simulado.

No caso concreto deste projecto, todos estes valores foram obtidas a partir dos valores usados no controlador real. O período de amostragem do controlador real, T , corresponde a 10 ms. O número de impulsos por revolução do *encoder* óptico corresponde a 256, pelo que a caixa de velocidades divide este valor por 12 na roda. As constantes derivativa e de *feed-forward*, k_d e k_f , valem ambas zero.

Daí que, usando a equação 5.19, possamos escrever:

$$\frac{2\pi}{NT} K'_p = K_p \quad (5.20)$$

$$\frac{2\pi}{NT} K'_i = K_i \quad (5.21)$$

$$\frac{2\pi}{NT} K'_d = K_d \quad (5.22)$$

$$\frac{2\pi}{NT} K'_f = K_f \quad (5.23)$$

e finalmente

$$K'_p = \frac{256 \times 0.01}{2\pi} K_p \quad (5.24)$$

$$K'_i = \frac{256 \times 0.01}{2\pi} K_i \quad (5.25)$$

O controlador é definido na plataforma de simulação pelo seguinte código:

```
<controller mode='pidspeed' kp='0.245' ki='0.0112' kd='0.0'
kf='0.0' active='1' period='10' />
```

O valor de *mode* diz respeito ao tipo de controlador, que neste caso é seleccionado como sendo um *PID* de controlo de velocidade. Os valores k_p , k_i , k_d e k_f correspondem aos ganhos do modelo apresentado na equação 5.16. O valor *period* define o período de amostragem do controlador, T , em ms.

5.3.6 Modelo de atrito

Tal como foi explicado na secção 2.1 do capítulo 2, as rodas omnidireccionais exercem tracção na direcção perpendicular ao eixo de rotação (tangencial), mas permitem o movimento na direcção paralela a este eixo (transversal). Ambas as direcções de movimento supõe algum atrito, sendo que o atrito tangencial inclui os atritos de toda a cadeia de actuação até às rodas (motores, caixas

reductoras, veios de transmissão). O atrito transversal é maioritariamente da responsabilidade das rodas e do solo.

Os processos de determinação dos parâmetros do modelo de atrito consistiram na aplicação uma aceleração inicial ao robot, sendo-lhe esta posteriormente retirada, deixando-o desacelerar. Desta forma, as únicas forças em acção durante o processo de desaceleração são as forças de atrito. Pela aplicação de diferentes componentes de aceleração, foi possível isolar as componentes de atrito pretendidas para cada experiência.

Nas experiências foi usada a ferramenta de medição da pose do robot, descrita no capítulo 3, para capturar a posição e orientação do robot em cada instante. Desta forma foram recolhidos valores de velocidades instantâneas, que permitiram calcular os vários parâmetros do modelo de atrito.

5.3.6.1 Modelo de atrito tangencial

O modelo de atrito no eixo das rodas implementado pelo simulador é dado aproximadamente pela seguinte equação:

$$\tau'_i(t) = B_{tg}\omega(t) + C_{tg} \quad (5.26)$$

Os parâmetros de atrito são definidos no simulador pelo seguinte código, em que os valores de *bv* e *fc* correspondem, respectivamente, à constante de atrito viscoso B_{tg} (em $Nm/rad/s$), e ao binário de atrito de *coulomb* C_{tg} (em Nm), do modelo:

```
<friction bv='0.0' fc='0.0615' coulomblimit='1e-2' />
```

O parâmetro *coulomblimit* corresponde ao limite em que a força de atrito de *coulomb* deixa de ser exercida, sendo usado unicamente como forma de manter a estabilidade da simulação. Este valor é escolhido de forma a ser suficientemente pequeno para não comprometer o realismo da simulação, mas suficientemente grande para que esta se mantenha estável.

As experiências de determinação dos parâmetros do modelo de atrito tangencial foram realizadas de forma a isolá-los dos restantes parâmetros. Para isso o movimento do robot foi restringido, dentro do possível, ao movimento angular. Durante este tipo de movimento a componente de atrito transversal nas rodas é praticamente nula. Os motores eléctricos foram também desligados dos respectivos controladores, e deixados em circuito aberto, por forma a que a travagem electro-magnética não ocorra.

Ainda que o atrito tangencial esteja distribuído por toda a cadeia de transmissão, como já foi referido, o objectivo da experiência é determinar o atrito equivalente no veio da roda. É este atrito equivalente que é modelado pelo simulador.

Estando o robot inicialmente animado de uma velocidade angular, se as únicas forças em acção forem as de atrito, podemos dividir o binário exercido sobre o robot, τ_r , em três componentes, uma para cada roda:

$$\tau(t) = \sum \tau_i(t) \Leftrightarrow \quad (5.27)$$

$$\Leftrightarrow \tau(t) = \sum r_i(t) \times f_i(t) \quad (5.28)$$

em que $f_i(t)$ é a força aplicada no eixo roda pelo atrito, e $r_i(t)$ é o braço da força $f_i(t)$.

Na presente configuração do robot, o braço da força de atrito será a distância de cada roda ao eixo vertical de rotação do robot. Este valor é o mesmo para as três rodas, do que resulta:

$$\tau(t) = R_{rb} \sum f_i(t) \quad (5.29)$$

O binário exercido sobre o eixo de cada roda, $\tau'_i(t)$, relaciona-se com a força exercida sobre o robot, $f_i(t)$, pela seguinte equação:

$$\tau'_i(t) = R_{wh} f_i(t) \quad (5.30)$$

em que R_{wh} é o raio de cada roda.

A partir das equações 5.29 e 5.30, obtemos a relação entre o binário sobre o robot, $\tau(t)$, e o binário de atrito sobre o eixo de cada roda, $\tau'_i(t)$:

$$\tau(t) = \frac{R_{rb}}{R_{wh}} \sum \tau'_i(t) \quad (5.31)$$

Usando a equação 5.6 obtemos a relação entre o binário de atrito exercido sobre o eixo da roda, $\tau'_i(t)$, e a variação $\frac{d\omega(t)}{dt}$ da velocidade angular:

$$J \frac{d\omega(t)}{dt} = \frac{R_{rb}}{R_{wh}} \sum \tau'_i(t) \quad (5.32)$$

Por outro lado, sabemos já que o modelo de atrito no eixo das rodas implementado pelo simulador é dado aproximadamente pela equação 5.26. Assim, usando a equação 5.32, e partindo do princípio que, por questões de simetria durante a experiência, o binário de atrito em cada uma das três rodas é igual, obtemos finalmente:

$$J \frac{d\omega(t)}{dt} = \frac{3R_{rb}}{R_{wh}} (B_{tg} \omega(t) + C_{tg}) \quad (5.33)$$

A experiência foi repetida para vários valores de velocidade angular inicial, tendo sido medida a orientação do robot ao longo do tempo. A partir desta foi estimada a velocidade angular instantânea ω . Os parâmetros B_{tg} e C_{tg} foram então estimados para cada experiência recorrendo ao método de mínimos quadrados.

5.4 Controlo da simulação

Uma vez obtido o modelo físico do robot e estimados os respectivos parâmetros, é necessário proceder à sua validação. O procedimento de validação consiste conceptualmente em colocar os robôs real e simulado em condições de funcionamento semelhantes, sendo depois comparados os seus desempenhos.

No caso concreto do presente trabalho, a obtenção de condições de funcionamento semelhantes consiste em aplicar aos robôs valores de referência de velocidade semelhantes, de forma a que, numa situação ideal, quer o robot real quer o robot simulado percorram a mesma trajetória.

Para isto são consideradas duas abordagens. A primeira, descrita em 5.4.1, consiste na aplicação de uma sequência de valores de referência de velocidade definidos. A segunda abordagem, descrita em 5.4.2, consiste na emulação do *OVis* por parte do programa de controlo do simulador. Esta última solução, uma vez validado o modelo, permite que a plataforma de simulação seja usada para o teste e desenvolvimento dos algoritmos de controlo da aplicação *MDecOmni3*.

5.4.1 Sequência de referências

Esta abordagem consiste, como já referido, na aplicação de uma sequência de valores de referência de velocidade a ambos os robôs. Uma vez que desta forma não é feita a realimentação de odometria, o controlo de velocidade do robot pode ser considerado em malha aberta, sendo que a posição dos dois robôs terão tendência para divergir. Esta abordagem mais simples revela-se então a mais exigente para o modelo.

5.4.2 Emulação da aplicação de ligação ao hardware

Esta abordagem é a mais útil a longo prazo, permitindo que a plataforma de simulação emule o comportamento do robot real de forma transparente para o *software* de decisão *MDecOmni3*. O facto de permitir a realimentação de odometria significa igualmente que é a abordagem mais robusta. Ainda assim, e como referido, a validação do modelo será feita de forma mais rigorosa em malha aberta.

Como descrito na secção 2.3 do capítulo 2, a comunicação entre o *OVis* e o *MDecOmni3* é feita por *UDP*, obedecendo a um forma definida. A emulação consiste então, nesta vertente, em dotar o programa de controlo do simulador de capacidade de compreensão básica deste protocolo.

O procedimento *Control* é usado para ditar a temporização do ciclo de controlo, função desempenhada no *OVis* pela interrupção de recepção de novo *frame* da câmara. De cada vez que o procedimento é executado, é enviado para o *MDecOmni3* um pacote contendo a informação que seria recolhida de outro modo pelo *OVis*.

Como descrito em 2.4, os pacotes são divididos em blocos, ainda que a presença de um determinado bloco não seja obrigatória. Desta forma, podem ser omitidos para já todos os blocos, à excepção do bloco contendo a informação proveniente da cadeia de periféricos. Este pacote, uma vez preenchido e enviado por rede, é interpretado pelo *MDecOmni3* como se fosse proveniente de um *OVis* real.

O *MDecOmni3* envia então para o simulador um pacote contendo a trama destinada aos periféricos. Esta trama contém a informação destinada aos controladores dos motores, sendo processada pelo programa de controlo da simulação. São então obtidas as referências de velocidade a serem aplicadas aos controladores simulados, terminando desta forma um ciclo de controlo.

Capítulo 6

Resultados

Os resultados quantitativos do presente trabalho apresentam-se em dois níveis. Em primeiro lugar, foram obtidos resultados referentes ao processo de estimação de parâmetros do modelo, documentados na secção 6.1. Em segundo lugar, foram também obtidos resultados relativamente ao processo de validação do modelo e dos parâmetros estimados para o mesmo. Estes resultados serão documentados em 6.2

6.1 Estimação de parâmetros

Os parâmetros do modelo descrito no capítulo 5 foram estimados de acordo com os procedimentos descritos nesse mesmo capítulo. O sumário de todos os parâmetros e respectivas estimativas é apresentado na tabela 6.3.

6.1.1 Corpo do robot

O corpo do robot é modelado por um cilindro de massa homogeneamente distribuída, como descrito em 5.1. O parâmetro referente à sua massa, M_{rb} , foi obtido pela pesagem do robot, tendo sido subtraído a esse valor a massa do conjunto das três rodas. A altura do centro de massa do cilindro, z_{cm} , foi medida recorrendo à experiência descrita em 5.1.1.

Tal como descrito em 5.1.2, foram realizadas algumas experiências por forma a determinar o momento de inércia do robot. Nessas experiências, a posição angular instantânea do robot foi medida, e a partir desta foi estimada a sua velocidade angular instantânea. Na figura 6.1 é mostrado o gráfico da evolução temporal da posição angular do robot durante uma das experiências, sendo mostrado na figura 6.2 o gráfico da velocidade angular correspondente. Em ambos os gráficos é destacado o intervalo de interesse, em que a aceleração angular $\frac{d\omega(t)}{dt}$ é aproximadamente constante.

A partir destes dados foi calculada, para cada uma das experiências, a estimativa da aceleração angular durante o intervalo de aceleração constante. Para isso foi aplicado o método dos mínimos

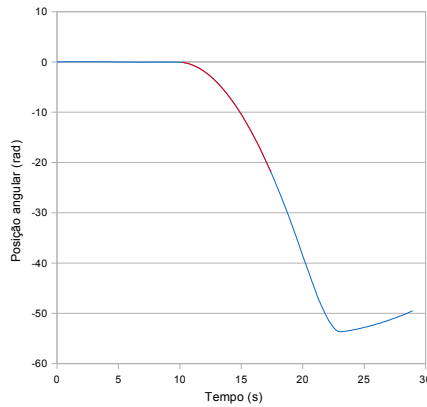


Figura 6.1: Gráfico da posição angular do robot em função do tempo para a experiência de determinação do momento de inércia

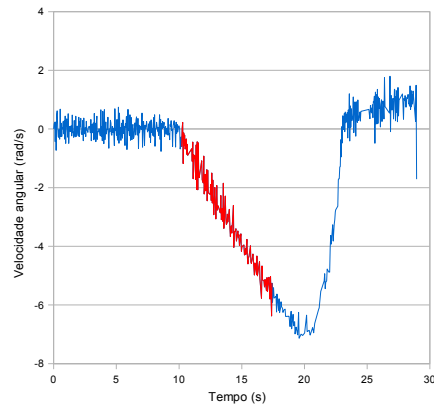


Figura 6.2: Gráfico da velocidade angular do robot em função do tempo para a experiência de determinação do momento de inércia

quadrados às amostras de velocidade. A partir destas estimativas de aceleração angular, foram calculadas as estimativas do momento de inércia para cada experiência, mostradas na tabela 6.1 juntamente com a média das amostras.

Ensaio	Massa (kg)	Momento de inércia (kgm^2)
1	0.2	0.636
2	0.2	0.662
3	0.4	0.748
4	0.4	0.784
5	0.6	0.693
Média		0.705

Tabela 6.1: Resultados da experiência de determinação do momento de inércia do robot

O raio do cilindro representativo do corpo do robot, r_{cil} , foi então determinado a partir do momento de inércia do robot, de acordo com a relação dada em 5.1.

Por sua vez, a altura do cilindro h_{cil} foi escolhida, partindo do pressuposto que a maior parte da massa do robot se encontra distribuída numa faixa de 0.2 m em torno do centro de massa do robot. Este valor apenas determina o momento de inércia em torno os eixos x e y , pelo que não afecta significativamente o comportamento do modelo na zona de funcionamento normal.

6.1.2 Rodas omnidireccionais

As rodas omnidireccionais são modeladas pelo simulador de acordo com o descrito em 5.3.1. Os parâmetros dimensionais, tais como R_{wh} , L_{wh} e R_{rb} , foram medidos. A massa da roda M_{wh} foi obtida por pesagem, enquanto que os três ângulos θ_1 , θ_2 e θ_3 são parâmetros construtivos do robot.

O modelo de atrito no eixo das rodas foi discutido em 5.3.6, juntamente com as experiências usadas para a determinação dos seus parâmetros. Nessas experiências, a posição angular instantânea do robot foi medida, e a partir desta foi estimada a sua velocidade angular instantânea. Na figura 6.3 é mostrado o gráfico da da posição angular do robot em função do tempo, durante uma das experiências. A variação inicial da posição, em sentido contrário ao movimento subsequente, é causada pelo processo manual de aceleração do robot. Na figura 6.2 é mostrado o gráfico da velocidade angular correspondente. Em ambos os gráficos é destacado o intervalo de interesse para a experiência.

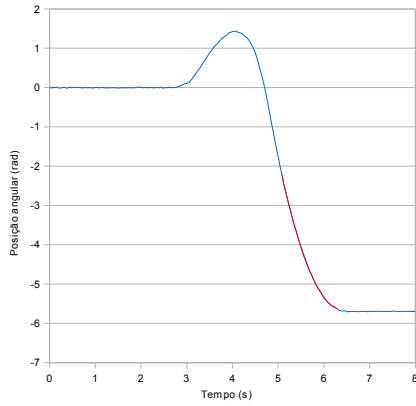


Figura 6.3: Gráfico da posição angular do robot em função do tempo para a experiência de determinação dos parâmetros do modelo de atrito tangencial

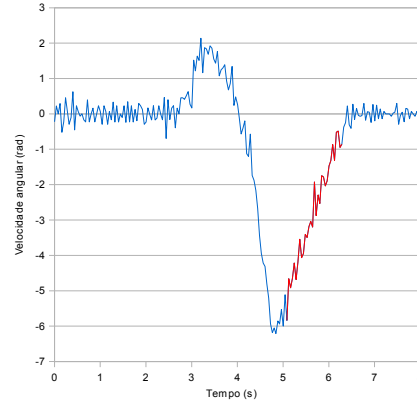


Figura 6.4: Gráfico da velocidade angular do robot em função do tempo para a experiência de determinação dos parâmetros do modelo de atrito tangencial

Como se pode verificar no gráfico da velocidade angular, o intervalo de interesse pode ser aproximado por uma recta. Tendo em conta que é atingida uma velocidade angular elevada, comparativamente à gama de funcionamento esperada do robot, é aceitável esperar que a aproximação linear seja suficiente. Desta forma, o binário de atrito aplicado pode ser assumido constante, pelo que o coeficiente de atrito viscoso B_{tg} pode ser definido como sendo nulo.

O cálculo dos parâmetros do modelo de atrito tangencial reduz-se então ao cálculo de C_{tg} a partir da aceleração angular $\frac{d\omega(t)}{dt}$. Os valores da aceleração angular foram estimados para o intervalo de interesse de cada experiência, usando-se o método dos mínimos quadrados. Adaptando a equação 5.33 obtemos então:

$$C_{tg} = \frac{JR_{wh}}{3R_{rb}} \frac{d\omega(t)}{dt} \quad (6.1)$$

Na tabela 6.2 são mostrados os valores de aceleração angular para as várias experiências, juntamente com a média das várias amostras. Os resultados são os seguintes:

$$B_{tg} = 0 \quad (6.2)$$

$$C_{tg} = \frac{0.705 \times 0.051}{3 \times 0.195} \times 3.92 = 0.0615 \quad (6.3)$$

Ensaio	Aceleração angular (rad/s ²)
1	3.95
2	3.79
3	3.81
4	4.05
5	3.92
6	3.61
7	3.86
8	4.19
9	3.95
10	4.05
Média	3.92

Tabela 6.2: Resultados da experiência de determinação dos parâmetros do modelo de atrito tangencial no eixo das rodas do robot

6.1.3 Motor, caixa redutora e *Encoder* óptico

O modelo do robot será usado para o teste dos controladores a aplicar nos robôs da equipa *5dpo*, sendo que o controlo é feito da mesma forma para todos os robôs. A variação construtiva entre cada um deles é suficientemente grande para que se justifique o uso de parâmetros não muito exactos, mas que sejam suficientemente representativos. Por esta razão, a maioria dos parâmetros destes três componentes foram obtidos a partir das respectivas folhas de características.

Os valores de tensão e corrente máximas no motor, por sua vez, correspondem a limites ditados pelo andar de potência dos controladores de velocidade dos motores.

6.1.4 Controlador *PID* de velocidade

As funções deste controlador são desempenhadas no robot real pelos controladores de velocidade dos motores, tratados em 2.2.2. Desta forma, os parâmetros dos controladores de velocidade dos motores do simulador foram obtidos a partir dos parâmetros dos controladores reais.

Como referido na secção 5.3.5, devido à diferença entre a configuração dos controladores simulados e os reais, foi necessário proceder a uma transformação dos parâmetros originais. Os resultados são os seguintes:

$$K'_p = \frac{256 \times 0.01}{2\pi} K_p = \frac{256 \times 0.01}{2\pi} \times 0.6 = 0.245 \quad (6.4)$$

$$K'_i = \frac{256 \times 0.01}{2\pi} K_i = \frac{256 \times 0.01}{2\pi} \times 0.03 = 0.0112 \quad (6.5)$$

$$K'_d = 0 \quad (6.6)$$

$$K'_f = 0 \quad (6.7)$$

Grupo	Parâmetro	Valor
Corpo	M_{rb}	Massa do robot
	z_{cm}	Altura do centro de massa do robot
	r_{cil}	Raio do cilindro do modelo
	h_{cil}	Altura do cilindro do modelo
Rodas	M_{wh}	Massa da roda
	R_{wh}	Raio da roda
	L_{wh}	Largura da roda
	R_{rb}	Distância do centro da roda ao centro do robot
	θ_1	Ângulo do eixo da roda 1 à frente do robot
	θ_2	Ângulo do eixo da roda 2 à frente do robot
	θ_3	Ângulo do eixo da roda 3 à frente do robot
	B_{tg}	Constante de atrito viscoso no eixo da roda
	C_{tg}	Binário de atrito de <i>coulomb</i> no eixo da roda
Motor	r_i	Resistência dos enrolamentos do motor
	k_i	Constante de binário do motor
	v_{max}	Tensão máxima aplicável ao motor
	i_{max}	Corrente máxima aplicável ao motor
Caixa redutora	$ratio$	Relação da caixa redutora
Encoder	ppr	Impulsos por revolução do encoder
Controlador PID de velocidade	K'_p	Constante de proporcionalidade
	K'_i	Constante integral
	K'_d	Constante diferencial
	K'_f	Constante de <i>feed forward</i>
	T	Período de amostragem

Tabela 6.3: Sumário de todos os parâmetros do modelo

6.2 Validação do modelo

Nesta secção serão apresentados os resultados da validação do modelo. Não foi possível ainda, por falta de tempo, completar o processo de validação.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Satisfação dos Objectivos

Para este trabalho foi necessário proceder a um estudo extensivo do funcionamento do robot. Esse estudo revelou-se particularmente trabalhoso, tendo sido necessário compilar informação dispersa e muitas vezes inexacta. Foi frequentemente necessário recorrer ao estudo do código fonte das aplicações de controlo, o que é bastante difícil, tendo em conta a sua dimensão e complexidade. O resultado deste estudo foi condensado no capítulo 2, estando a parte mais relevante documentada de forma clara e rigorosa. Sendo este um dos objectivos iniciais ditados para o presente trabalho, julgo que foi cumprido de forma satisfatória.

A documentação da ferramenta de medição da pose do robot, descrita no capítulo 4, foi inicialmente definida como um objectivo secundário. Se, por um lado, não existe praticamente nenhuma documentação disponível a seu respeito, que não o próprio código fonte, por outro lado o estudo da aplicação não foi muito mais além do que o necessário à sua utilização para a realização das experiências.

Foi, no entanto, necessário desenvolver uma aplicação cliente desta ferramenta, o que se traduziu numa adição aos objectivos iniciais. O protocolo de comunicações usado pela ferramenta de medição foi também documentado.

Foi também feito um estudo do simulador, sendo analisado o seu funcionamento, bem como as suas possibilidades e limitações. O simulador foi documentado no capítulo 4, sendo esta documentação muitas vezes complementar à documentação oficial. Isto porque muitas das suas características não foram ainda documentadas, tendo algumas sido inclusive desenvolvidas ao longo do trabalho, em função do *feedback* obtido. Infelizmente não foi possível incluir nesse capítulo toda a informação disponível. Ainda assim, a documentação apresentada permite compreender a posterior elaboração do modelo e cálculo dos respectivos parâmetros.

Os objectivos mais importantes do trabalho terão sido sem dúvida a identificação do modelo e a posterior estimação dos seus parâmetros. O modelo identificado comporta-se de forma realista

e estável na simulação. Os parâmetros desse modelo foram todos estimados à excepção de dois, correspondentes aos coeficientes de atrito transversal, associado aos roletes das rodas omnidireccionais. Esta característica foi adicionada ao simulador durante o trabalho, tendo inclusive sido feitas as experiências necessárias à estimação dos referidos parâmetros. Não foi no entanto possível completar o cálculo, por falta de tempo. A validação do modelo não foi concluída pela mesma razão, pelo que apenas é documentado o procedimento.

Foi também iniciado o trabalho no sentido de permitir à aplicação de decisão actuar transparentemente no modelo de simulação, como se de um robot real se tratasse. Ainda que não tenha sido completado, o estudo referente a este tópico revelou-se fundamental para o processo de validação do modelo. Por outro lado, a estrutura de código desenvolvida servirá de base para o trabalho futuro neste sentido.

7.2 Trabalho Futuro

O modelo de simulação apresenta um realismo bastante promissor, pelo que é aceitável postular que o conjunto será útil ao projecto *5dpo*. Este trabalho apresenta-se como um primeiro passo no sentido de obter uma plataforma de simulação realista, que possa ser usada para desenvolvimento e teste dos algoritmos de controlo dos robôs.

Neste sentido, será necessário desenvolver algum trabalho futuro, que passará por validar intensivamente o modelo, sendo que existe por outro lado alguma margem para eventuais melhorias. A construção de um modelo de colisão mais próximo do real, bem como a incorporação de um mecanismo de chuto no robot são duas vertentes possíveis de trabalho futuro.

Será também necessário terminar o trabalho referente à interacção entre a aplicação de decisão existente e a plataforma de simulação. A incorporação de um sistema de localização da bola, equivalente ao de visão nos robôs reais, é uma vertente do trabalho que poderá ser seguida. Neste sentido poderá ser necessária alguma intervenção no próprio simulador.

Um objectivo interessante para futuro será a adaptação do programa de controlo da simulação, de forma a permitir a simulação simultânea de vários robôs, cada um ligado ao respectivo programa de decisão. Desta forma, será possível simular um jogo *Middle Size League* entre duas equipas de robôs.

Neste sentido, será necessário criar um campo simulado, com linhas e balizas. Por outro lado, será importante a criação de uma *referee-box* virtual, semelhante à usada nos jogos reais. Pela arquitectura idealizada para o sistema, é inerentemente possível usar a aplicação de supervisão existente, o *Treinador*, sem alterações.

Referências

- [1] 5dpo. 5dpo robotics (feup, portugal), Janeiro 2009. <http://paginas.fe.up.pt/~robosoc/en/doku.php>.
- [2] The Robocup Federation. What is robocup, Janeiro 2009. <http://www.robocup.org/overview/21.html>.
- [3] António Paulo Moreira, Paulo Costa, Armando Sousa, e Luís Paulo Reis. 5dpo-2000 team description for year 2004. Em *RoboCup*. ROBOCUP2004, Julho 2004.
- [4] Paulo Costa, António Moreira, Paulo Marques, Armando Sousa, Pedro Costa, e Susana Gaio. 5dpo robotic soccer team for year 2004. Em *RoboCup*. ROBOCUP2004, Julho 2004.
- [5] António Moreira, Armando Sousa, e Paulo Costa. Vision based real-time localization of multiple mobile robots. Em *3rd International Conference on Field and Service Robotics*. FSR01, Junho 2001.
- [6] Paulo Costa. Simtwo, Janeiro 2009. <http://paginas.fe.up.pt/~paco/wiki/index.php?n=Main.SimTwo>.
- [7] The Player Project. Gazebo - 3D multiple robot simulator with dynamics, Janeiro 2009. <http://playerstage.sourceforge.net/gazebo/gazebo.html>.
- [8] Microsoft. Microsoft robotics developer studio overview, Janeiro 2009. <http://msdn.microsoft.com/en-us/library/bb483024.aspx>.
- [9] Russell Smith. Open dynamics engine, Janeiro 2009. <http://www.ode.org/>.
- [10] Russell Smith. Open dynamics engine v0.5 user guide, Janeiro 2009. <http://ode.org/ode-latest-userguide.html>.
- [11] Erwin Coumans. Bullet collision detection and physics library, Janeiro 2009. <http://www.bulletphysics.com>.
- [12] Free Pascal Project. Free Pascal - Advanced open source Pascal compiler for Pascal and Object Pascal, Janeiro 2009. <http://www.freepascal.org>.
- [13] Lazarus Project. Lazarus project :: Write once compile everywhere!, Janeiro 2009. <http://www.lazarus.freepascal.org>.
- [14] Carlo Kok. Pascal script, Janeiro 2009. http://wiki.freepascal.org/Pascal_Script.
- [15] W3C-World Wide Web Consortium. Extensible markup language (xml), Janeiro 2009. <http://www.w3.org/XML/>.

- [16] André Scolari Conceição, A. Paulo Moreira, e Paulo J. Costa. Dynamic parameters identification of an omni-directional mobile robot. Em *Proceedings of the 3rd International Conference on Informatics in Control, Automation and Robotics*. ICINCO2006, Agosto 2006.
- [17] André Scolari Conceição, A. Paulo Moreira, e Paulo J. Costa. Trajectory control and modeling of an omni-directional mobile robot. Em *Proceedings of the 3rd International Conference on Informatics in Control, Automation and Robotics*. ICINCO2006, Agosto 2006.
- [18] André Scolari Conceição, A. Paulo Moreira, e Paulo J. Costa. Model identification of a four wheeled omni-directional mobile robot. Em *7th Portuguese Conference on Automatic Control*. CONTROLO2006, Setembro 2006.
- [19] André Scolari Conceição, A. Paulo Moreira, e Paulo J. Costa. Controller optimization and modeling of an omni-directional mobile robot. Em *7th Portuguese Conference on Automatic Control*. CONTROLO2006, Setembro 2006.
- [20] Helder P. Oliveira, Armando J. Sousa, A. Paulo Moreira, e Paulo J. Costa. Precise modeling of a four wheeled omni-directional robot. Em *8th Conference on Autonomous Robot Systems and Competitions*. 8th Portuguese Robotics Open ROBOTICA2008, Abril 2008.
- [21] C. R. Nave. Center of mass, Janeiro 2009. <http://hyperphysics.phy-astr.gsu.edu/Hbase/cm.html>.