Modelação e simulação realista de sistemas no domínio da robótica móvel

José Alexandre de Carvalho Gonçalves

Dissertação submetida para a obtenção do grau de
Doutor em Engenharia Electrotécnica e de Computadores
pela Faculdade de Engenharia da Universidade do Porto

Trabalho realizado sob a orientação do
Professor Doutor Paulo José Cerqueira Gomes da Costa

Maio de 2009

The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka!" (I found it!) but "That's funny ..."

Isaac Asimov (1920 - 1992)

Resumo

O desenvolvimento de *software* de controlo, localização e navegação para *robots* móveis torna-se muito moroso devido às dificuldades associadas ao trabalho exaustivo com *hardware* real e à dificuldade de controlar o ambiente por forma a existir repetibilidade nas experiências realizadas. Estes problemas podem ser contornados caso estejam disponíveis ambientes de simulação realistas para o desenvolvimento, validação e migração de *software* para *robots* reais. Para desenvolver simuladores realistas é necessário uma prévia modelação dos *robots* móveis, incluindo os seus sensores e actuadores e a maneira como estes interagem com o meio.

Neste trabalho foi realizada a modelação e simulação de duas plataformas robotizadas. Foi modelado e simulado o Kit Lego Mindstorms NXT, sendo uma plataforma frequentemente utilizada para o ensino de fundamentos de robótica e também foi modelado e simulado um *robot* omnidireccional de três rodas equipado com motores *brushless* e sensores de distância de infravermelhos destinado a operar em ambientes estruturados.

É também apresentada uma arquitectura que permite desenvolver e validar programas para *robots* recorrendo à simulação e migração do código para o *robot* real. Esta arquitectura foi aplicada à plataforma omnidireccional com motores *brushless*. Como exemplo desenvolveu-se o *software* de localização, navegação e controlo do *robot* omnidireccional para operar num ambiente estruturado.

Abstract

Writing robot software is a time-consuming and error-prone process due to the difficulties related with the exhaustive work based only on real hardware and also due to the difficulty of controlling the robot environment in order to achieve repeatability in the performed experiments. This problems can be avoided if there are available realistic simulation environments that allow development, validation and migration of robot software to real robots. In order to develop realistic simulators it is necessary a previous robot modeling, including the sensors and actuators and how they interact with the environment.

In this work it is presented the modeling and simulation of two robotic platforms. It was modeled and simulated the Lego Mindstorms NXT Kit, being a platform frequently used for teaching robotics introductory concepts and it was also modeled and simulated a three wheel omnidirectional robot equipped with brushless motors and infrared distance sensors. The omnidirectional robot was prototyped with the goal to operate in structured environments.

It is also presented an architecture that allows to develop and validate robot software resorting to simulation and code migration to the real robot. The developed architecture was applied to the omnidirectional robot prototype equipped with brushless motors. As an example it was developed the localization, navigation and control software for the omnidirectional prototype with the goal to operate in a structured environment.

Agradecimentos

Ao professor Paulo Costa por todo o seu apoio ao longo deste trabalho.

Aos Paulo Malheiros e José Lima pelo companheirismo demonstrado.

Ao Filipe Sousa pelo seu apoio na edição de artigos científicos.

À Escola Superior de Tecnologia e Gestão do Instituto politécnico de Bragança pela disponibilização de Laboratórios e apoio financeiro.

Aos meus pais por estarem sempre presentes.

Em especial à Elisabete por tudo...

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Contexto e motivação

A robótica móvel tem sido motivo de intensa investigação à escala mundial, existindo cada vez mais aplicações de *robots* móveis na prestação de serviços [Sie03] [YAT+06] [KC06] [MMBV06]. Também se tornou um tema de eleição para o ensino de diversas áreas de engenharia devido à inerente multi-disciplinariedade associada à robótica móvel e à motivação extra do aspecto lúdico e competitivo de grande parte dos desafios propostos [rob09] [AAC+00] [AAC02] [LSR+03].

O desenvolvimento de *software* de controlo, localização e navegação para *robots* móveis torna-se muito moroso devido às dificuldades associadas ao trabalho exaustivo com *hardware* real e devido à dificuldade de controlar o ambiente por forma a existir repetibilidade nas experiências realizadas. Estes problemas podem ser contornados caso estejam disponíveis ambientes de simulação realistas para o desenvolvimento, validação e migração de *software* para *robots* reais. Para desenvolver simuladores realistas é necessário uma prévia modelação dos *robots* móveis, incluindo os seus sensores e actuadores e a maneira como estes interagem com o meio. Neste trabalho foi realizada a modelação e simulação de duas plataformas robotizadas. Foi modelado e simulado o Kit Lego Mindstorms NXT, sendo uma plataforma frequentemente utilizada para o ensino de fundamentos de robótica e também foi modelado e simulado um *robot* omnidireccional de três rodas equipado com motores *brushless* e sensores de distância de infravermelhos destinado a operar em ambientes estruturados.

O Kit Lego Mindstorms NXT é uma ferramenta educacional muito poderosa para ensinar conceitos introdutórios de robótica, principalmente de robótica móvel. Na Figura 1.1 encontram-se exemplificados dois protótipos

desenvolvidos com base no Kit Lego Mindstorms NXT. Este Kit educacional permite a prototipagem rápida de *robots* móveis e a realização de controladores com recurso a linguagens de alto nível, tendo em conta os inevitáveis e apertados requisitos de tempo real associados a desafios de robótica móvel. Verificou-se uma significativa evolução do Kit Lego Mindstorms NXT em relação ao RIS, sendo impulsionada pelas necessidades que os entusiastas deste Kit foram informalmente solicitando com o decorrer dos tempos. Uma das grandes melhorias apresentadas no Kit NXT foi a introdução de *encoders* incorporados nos motores, o que permite de uma maneira mais simples realizar o cálculo de medidas relativas, não tendo que ser realizadas extensões ao *hardware* tal como exemplificado na Figura 1.2 [GLC06].

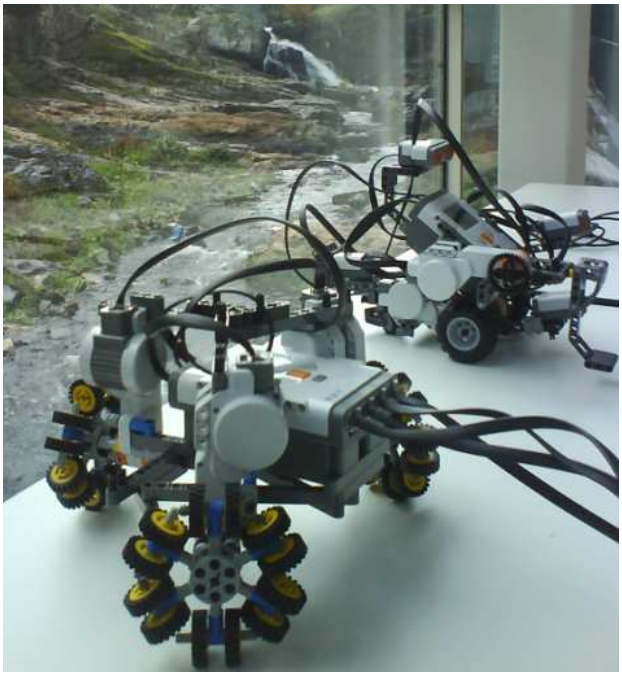

Figura 1.1: Protótipos desenvolvidos com o Kit Lego Mindstorms NXT.

A plataforma robotizada prototipada com motores *brushless* (tecnologia utilizada pela equipa 5DPO na *Small Size League*) usa uma configuração cinemática omnidireccional, tendo sido previamente utilizada na análise do desempenho de *robots* omnidireccionais de três e de quatro rodas [OLi07]. A opção de utilizar motores *brushless*, neste tipo de aplicações, prende-se com o facto de permitir melhores desempenhos que os tipicamente utilizados motores DC.

O sistema de locomoção do protótipo com motores *brushless* é baseado na configuração omnidireccional de três rodas, tendo-se acrescentado sensores de

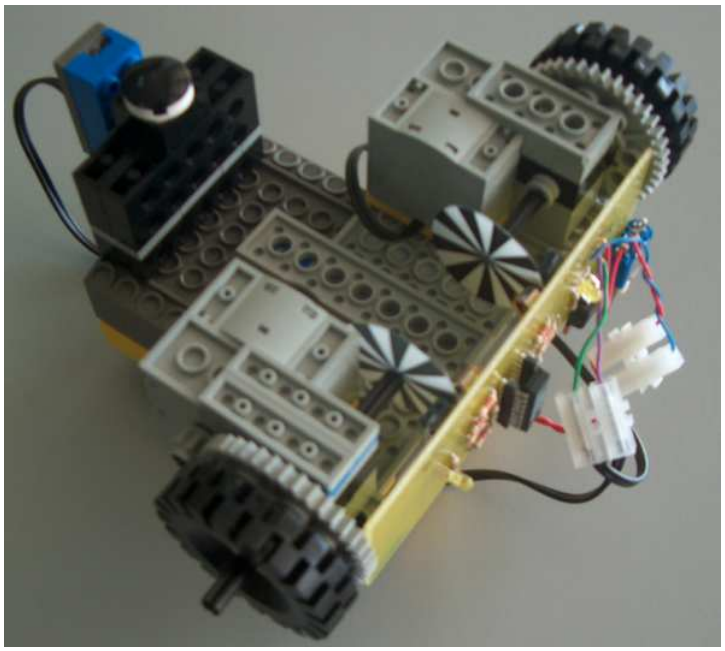

Figura 1.2: *Robot* diferencial prototipado com o Kit Lego Mindstorms RIS.

distância de infravermelhos ao protótipo original. Este protótipo destina-se neste trabalho a operar em ambientes estruturados, estando ilustrado nas Figuras 1.3 e 1.4. O algoritmo de localização está dependente da navegação do *robot*, pois este tem que recolher informação do ambiente para se localizar enquanto executa uma missão. O facto de o *robot* ser omnidireccional permite obter mais facilmente, enquanto navega, melhores medidas absolutas quando comparado com outras configurações que têm mais restrições de locomoção. A abordagem proposta tira partido do facto de o *robot* omnidireccional se poder mover com velocidades lineares e angular independentes, com o objectivo de os sensores de distância estarem sempre virados com a orientação que lhes permita uma melhor estimativa dos parâmetros de localização.

Um exemplo de um *robot* omnidireccional prototipado com motores DC encontra-se ilustrado na Figura 1.5, o protótipo ilustrado foi utilizado no trabalho intitulado "Controlo de *robots* omnidireccionais" [Gon05].

Para a plataforma robotizada com base no sistema de locomoção omnidireccional foi desenvolvida uma arquitectura que permite a migração de *software* para o *robot* real, sendo desenvolvido como exemplo o *software* de localização e navegação do *robot* omnidireccional para operar num ambiente estruturado. O algoritmo de localização é baseado na fusão sensorial de dados de distância de sensores de infravermelhos e de odometria, recorrendo a

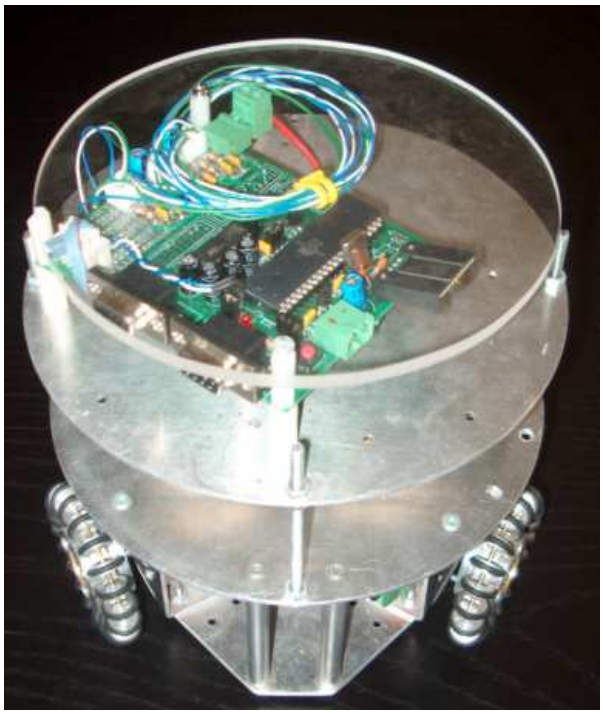

Figura 1.3: Protótipo do *robot* omnidireccional (1).

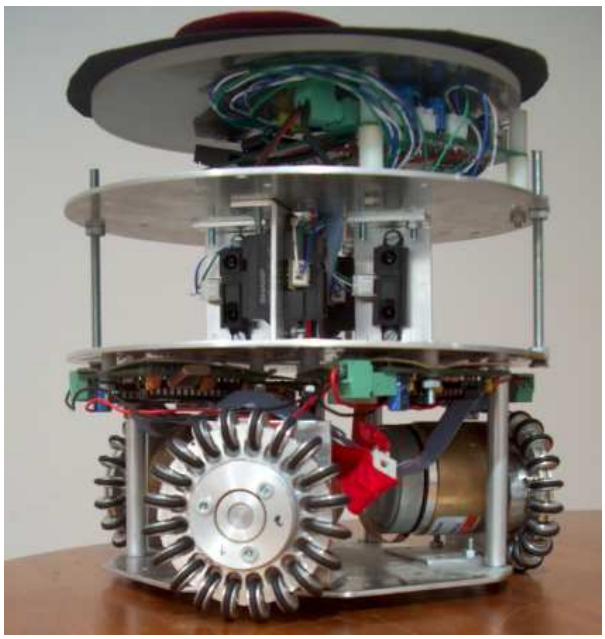

Figura 1.4: Protótipo do *robot* omnidireccional (2).

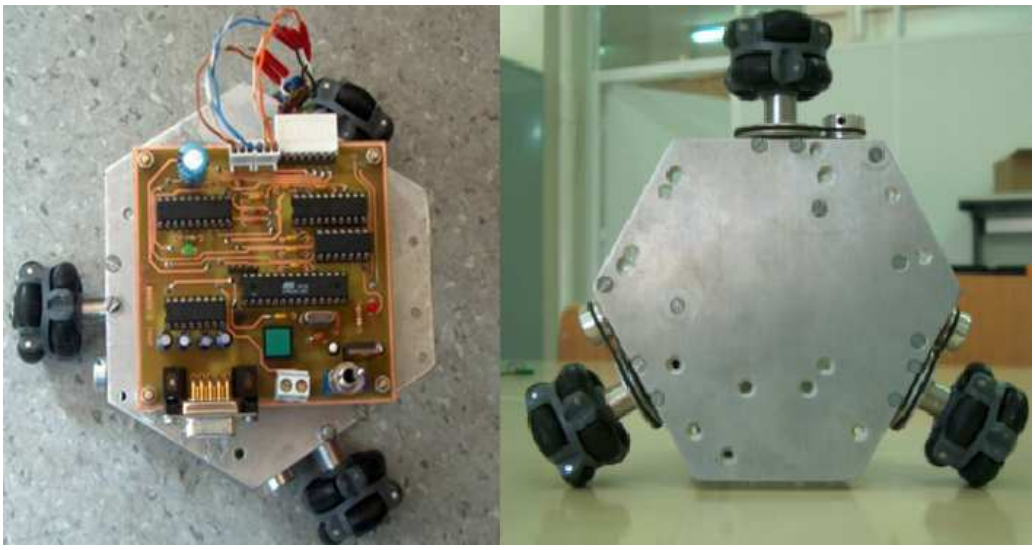

Figura 1.5: *Robot* omnidireccional prototipado com motores DC.

um filtro de Kalman estendido. Para realizar a validação física do sistema de localização e navegação do *robot* móvel foi também desenvolvido um sistema de localização em tempo real de baixo custo baseado em visão global.

Foram utilizados *robots* industriais para auxílio na modelação dos sensores de distância de ambas as plataformas e para modelar o sensor de toque do Kit Lego Mindstorms NXT. A utilização de *robots* industriais no processo de recolha de dados de alguns dos sensores modelados permitiu um aumento de velocidade e repetibilidade, contribuindo para a redução de erros [Pir06] [Gro86] [Col05] [RN96].

1.2 Contribuições

As principais contribuições de esta tese centram-se na modelação e simulação de plataformas robotizadas. O objectivo de modelar e simular sensores e actuadores de plataformas robotizadas é de a partir de simulações realistas se poder realizar a produção e validação de programas de controlo, localização e navegação de *robots* móveis sem acesso a *hardware*. Para o efeito foram modeladas e simuladas duas plataformas robotizadas sendo uma delas de cariz didáctico (Kit Lego Mindstorms NXT) e a outra sendo adaptada de um *robot* de alto desempenho prototipado com a tecnologia utilizada pela equipa 5DPO na *Small Size League* no Robocup.

Para a plataforma robotizada baseada no Kit Lego Mindstorms NXT foi apresentada a modelação e simulação de um protótipo com a configuração

cinemática diferencial e com um sensor de luz. Foi também apresentada a modelação do sensor de toque e do sonar.

Também foi modelado e simulado um *robot* omnidireccional de três rodas equipado com motores *brushless* e sensores de distância de infravermelhos destinado a operar em ambientes estruturados. É também apresentada uma arquitectura que permite desenvolver e validar programas para *robots* recorrendo à simulação e migração do código para um *robot* real sem que tenham que ser feitas alterações aos programas desenvolvidos e testados no simulador. Esta arquitectura foi apenas aplicada à plataforma omnidireccional com motores *brushless*. Como exemplo desenvolveu-se o *software* de localização, navegação e controlo do *robot* omnidireccional para navegação num ambiente estruturado.

1.3 Estrutura da tese

Inicialmente, no Capítulo 2, é apresentado o estado da arte abordando os temas de modelação e simulação no domínio da robótica móvel e localização de *robots* móveis recorrendo a algoritmos de fusão sensorial com base em filtros de Kalman. No Capítulo 3 é apresentada a descrição de um sistema de visão global. A ferramenta desenvolvida é versátil para a sua utilização em várias etapas de aprendizagem no domínio da robótica móvel. Pode ser utilizada para validação de controladores e tácticas (monitorizando em tempo real jogos segundo as regras do Robocup Junior), permite a validação de sistemas de localização e navegação para ambientes estruturados (monitorizando o desempenho do sistema de localização e navegação de um *robot*), sendo também também utilizada no estudo de controlo e técnicas de fusão sensorial no domínio do Robocup. No Capítulo 4 é descrita a modelação e simulação do Kit Lego Mindstorms NXT. No Capítulo 5 é apresentada a modelação e simulação de um *robot* omnidireccional de três rodas equipado com motores *brushless* e sensores de distância de infravermelhos, destinado a operar em ambientes estruturados. No Capítulo 6 é apresentado um sistema de localização e navegação desenvolvido para a plataforma robotizada descrita no Capítulo 5. O sistema de localização foi desenvolvido com base na sua simulação, sendo apresentada uma arquitectura que permite a migração do código desenvolvido para o *robot* real. O sistema de localização e navegação aplicado ao *robot* real foi validado recorrendo ao sistema de visão global apresentado no Capítulo 3. Finalmente são apresentadas conclusões e trabalho futuro.

Capítulo 2

Estado da arte

2.1 Simulação no domínio da robótica móvel

A simulação está a tornar-se uma área cada vez mais importante para a robótica móvel tornando possível a produção de código sem acesso ao *hardware* do *robot*, permitindo rápida reconfigurabilidade e acesso a muitas variáveis de interesse, as quais seriam mais difíceis de monitorizar caso se trabalhasse com *hardware* real. Para simular um *robot* é necessário construir um modelo dinâmico, validar o modelo dinâmico, modelar os sensores e modelar a sua interacção com o ambiente. Para tornar útil a simulação é impreterível que os modelos sejam o mais realistas possível. Após a fase de modelação é necessário que se possa visualizar a três dimensões e em tempo real o sistema modelado.

Um dos mais populares programas comerciais de simulação para robótica móvel é o Webots [Web] [Mic04], tem a dinâmica baseada no ODE (Open Dynamics Engine) [ode09] e utiliza o OpengGL [OGL09] para a visualização. Algumas das suas funcionalidades são apresentadas na Figura 2.1.

Uma outra ferramenta comercial para simulação de *robots* móveis, que também utiliza o ODE, é o Marilou Robotics Studio da empresa AnyKode, a AnyKode desenvolve aplicações e ferramentas com aplicação na robótica móvel [any09]. Um exemplo do Marilou Robotics Studio pode ser visto na Figura 2.2 onde está exemplificada a simulação de corpos rígidos.

O Microsoft Robotics Studio é também é uma ferramenta comercial para a simulação e controlo de *robots* móveis. Tem como objectivo ser uma ferramenta académica, para entusiastas e para desenvolvimento de aplicações comerciais para uma grande variedade de *robots* móveis (Figura 2.3) [mic09].

Existem também vários simuladores para robótica móvel que são projectos *Open source*, como por exemplo o Khepera Simulator (Figura 2.4)

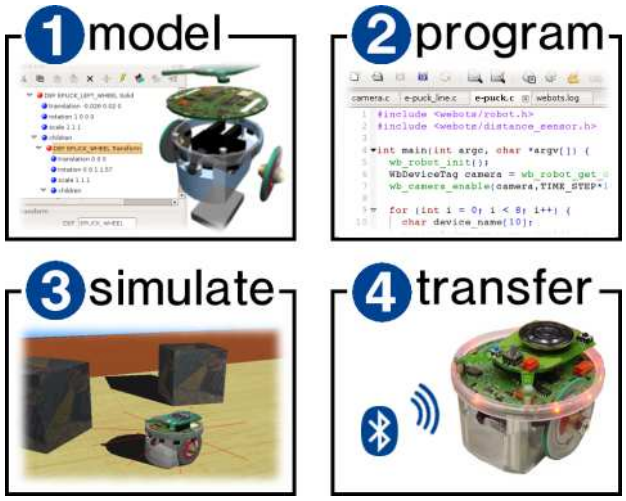

Figura 2.1: Funcionalidades do programa Webots.

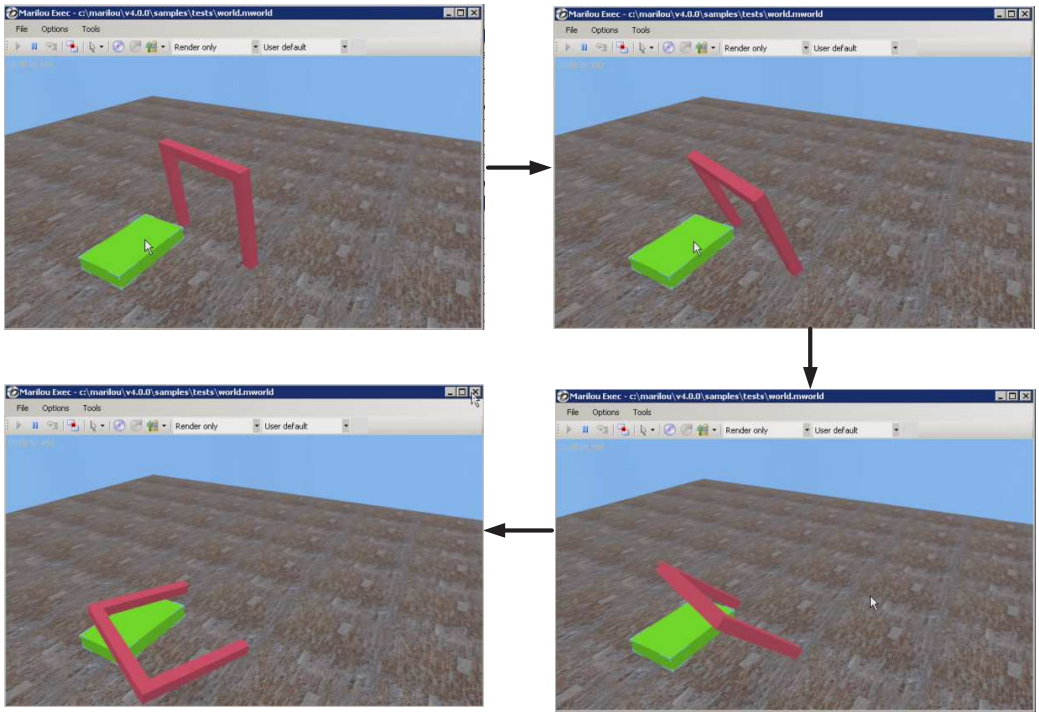

Figura 2.2: Marilou Robotics Studio.

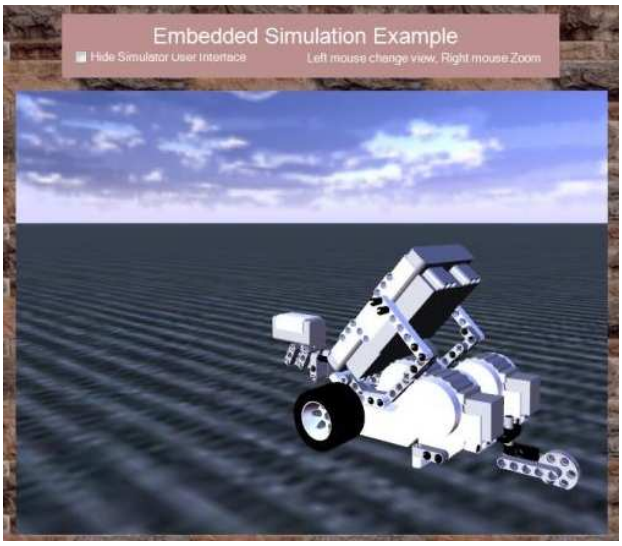

Figura 2.3: Microsoft Robotics Studio.

[O.M09], sendo o predecessor do Webots antes de este se tornar uma ferramenta comercial.

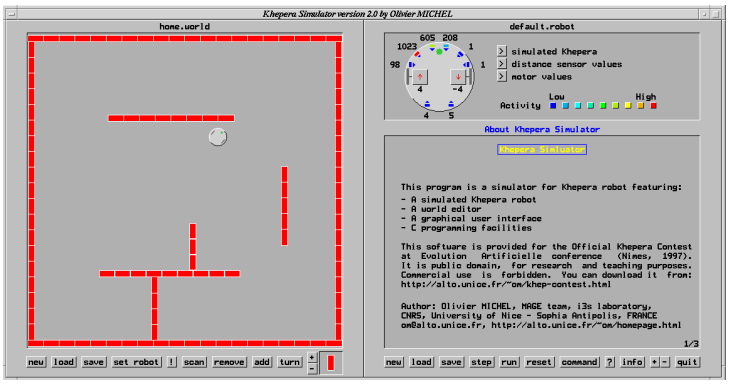

Figura 2.4: Khepera Simulator.

Outros projectos *Open source* são por exemplo o Gazebo [gaz09] que é um simulador 3D com dinâmica, suportando múltiplos *robots* móveis navegando em ambientes externos. As Figuras 2.5 e 2.6 mostram a modelação de um sistema com base no programa Gazebo. A Figura 2.5 mostra os sólidos que são responsáveis pela dinâmica, mas para uma visualização mais amigável foram associadas texturas e outros desenhos tornando a aparência muito mais realista, como se pode ver na Figura 2.6.

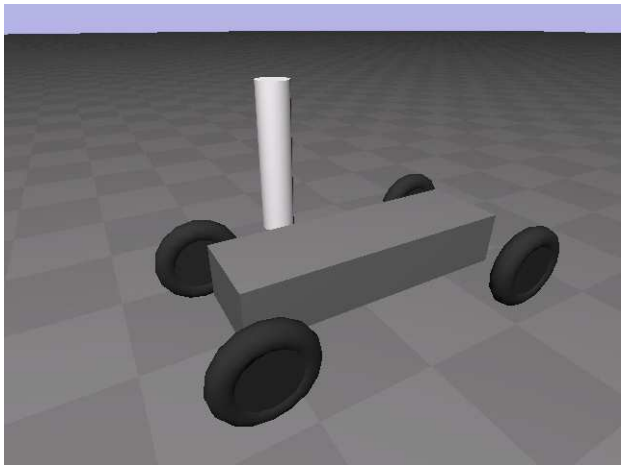

Figura 2.5: Objectos responsáveis pela dinâmica.

Figura 2.6: Objectos com aspecto realista.

Também o OpenSim é um projecto *Open source*, sendo um simulador para sistemas articulados e *robots* móveis com rodas (Figura 2.7) [ope09].

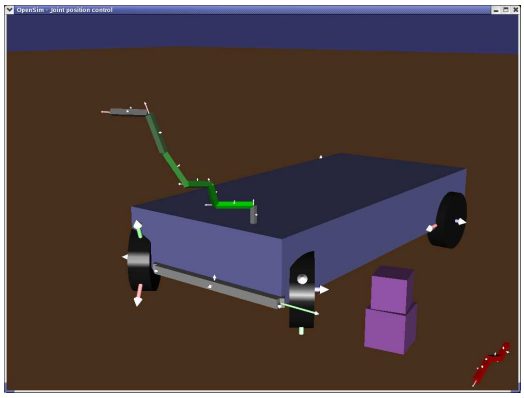

Figura 2.7: OpenSim.

2.2 Modelação no domínio da robótica móvel

É essencial, por forma a desenvolver controladores que permitam melhores desempenhos, obter modelos dinâmicos precisos. Os modelos são baseados em sistemas dinâmicos lineares e não lineares, sendo a sua modelação realizada com base na leis da física e dados experimentais. Devem também ser modeladas todas as não linearidades e limitações físicas de cada modelo por forma a este melhor se aproximar da realidade. A estimação dos parâmetros dos modelos tem sido sujeita a uma contínua investigação [CBDN96] [Kho89] [WCGR02] [CaMC06], sendo um dos métodos mais comuns para a identificação de parâmetros de sistemas o método dos mínimos quadrados.

Uma abordagem com base em motores de física permite o desenvolvimento de modelos de sistemas de corpos rígidos articulados simulando todas as forças, atrito e colisões. Alguns exemplos de motores de física são:

- Bullet
- Havok
- Newton Dynamics
- YADEYet Another Dynamic Engine
- ODE Open Dynamics Engine

O Bullet é um motor de física que permite a simulação 3D de corpos rígidos articulados com tratamento de colisões. Este motor é *Open source* licenciado sob a *zlib*, sendo livre para uso comercial e não-comercial. O Bullet é tipicamente utilizado para animações e para jogos de computador [bul09]. O Havok (também conhecido como Havok Physics) é um motor de física desenvolvido pela companhia Havok. Foi desenvolvido para jogos permitindo interacção entre objectos em tempo real, dando propriedades físicas aos objectos [hav09]. O Newton Game dynamics é uma solução integrada para simulação em tempo real de ambientes dinâmicos, a sua API disponibiliza detecção de colisões e comportamento dinâmico, permitindo a simulação 3D [new09]. YADE é um motor de física *Open source* que permite a simulação discreta de sistemas [yad09]. O ODE (Open Dynamics Engine) é um motor de física *Open source*, os seus dois principais componentes são o motor de dinâmica de corpo rígidos e o motor de detecção de colisões [ode09]. O ODE foi o motor de física utilizado neste trabalho.

Em robótica móvel é também muito importante a modelação realista dos sensores, sendo um dos factores mais importantes a correcta caracterização do ruído em função das diferentes condições de uma medida. O ruído de um sensor muitas vezes pode ser caracterizado por uma distribuição normal, à qual está associada uma média e uma variância. A variância representa a confiança que se tem numa medida, sendo essencial o seu correcto conhecimento por forma a obter um valor óptimo na fusão sensorial de várias medidas [GLC08a] [BNMS05] [Bre07]. O sucesso das missões dos *robots* está dependente de uma boa estimativa da localização, da qual dependem muitos factores, sendo um deles a correcta modelação dos sensores [Eve95] [Bra03] [FGGE99]. O problema relativo à localização é a chave para que os *robots* móveis se tornem verdadeiramente autónomos. Se um *robot* não sabe onde se encontra, muito dificilmente poderá decidir o que vai fazer em seguida. Por forma ao *robot* se localizar, este tem acesso a medidas relativas e absolutas recebendo *feedback* do meio envolvente. Dada esta informação o *robot* deve estimar a sua posição com a maior exactidão possível, esta informação deve ser combinada de uma maneira óptima [Neg03]. Uma maneira possível para a realização de fusão sensorial é o recurso a filtros de Kalman, sendo uma técnica proveniente da teoria de estimação, que combina a informação de diferentes fontes para obter o valor de variáveis de interesse, bem como a incerteza nelas contida [Cos99][Rib04b][SCM04]. A estimação da posição de um *robot* móvel pode, em alternativa aos filtros de Kalman, ser realizada usando um filtro de partículas, representando cada partícula uma posição possível do *robot* móvel. A cada amostragem um conjunto de possíveis posições é obtido, sendo estas partículas classificadas de acordo com a qualidade da medida em relação à posição real do *robot*, considerando o modelo probabilístico dos

sensores e a posição actual. A posição real do *robot* é estimada calculando a média ponderada da nuvem de partículas tendo em conta a confiança em cada uma delas [TBF05][Rek03][BVR06].

2.3 Localização com base no filtro de Kalman

Localização probabilística consiste num algoritmo probabilístico: em vez de se assumir uma única hipótese de onde está o *robot*, assume-se uma distribuição de probabilidade dentro do espaço de todas as hipóteses possíveis, tendo em conta a incerteza na locomoção do *robot* e tendo em conta que as leituras dos sensores são ruidosas [BEF96]. A distribuição de probabilidade pode ser multi-modal caso haja ambiguidade na localização absoluta do *robot* móvel [CLB+04]. No caso típico o *robot* tem bastante certeza da posição em que se encontra, sendo a estimação da sua posição uma distribuição de probabilidade centrada próximo da posição real do *robot* e sendo aproximada a uma distribuição gaussiana. Sendo este último caso o do sistema usado para validar o sistema de localização desenvolvido, sendo apresentado no Capítulo 3, o qual permite um erro muito pequeno e sem que existam leituras ambíguas.

Um dos métodos mais utilizados na localização de *robots* móveis é a integração dos comandos de velocidade permitindo a estimação da sua posição recorrendo a uma posição inicial conhecida, sendo esta medida relativa muito robusta a curto prazo. O seu erro é cumulativo, deteriorando-se progressivamente a estimativa da localização, isto acontece porque a posição inicial não é perfeitamente conhecida e também porque os comandos nunca são perfeitamente executados [BNMS05]. Outra solução alternativa para a localização relativa recorre à utilização de sensores inerciais, não dependendo do contacto com o chão. A utilização de sistemas inerciais em submarinos [PMC03] e veículos aéreos autónomos [FBS+04] encontra-se neste momento bastante difundida [GWA01]. As medidas relativas apresentam um erro cumulativo, logo é necessário injectar informação no sistema proveniente de sensores que recolhem informação do ambiente envolvente ao *robot* para compensar a informação perdida na integração dos comandos de velocidade. A informação disponibilizada pelos sensores corresponde também a uma distribuição de probabilidade tipicamente gaussiana centrada na leitura do sensor.

Geralmente qualquer método de estimação pode ser pensado em duas etapas, estimação e correcção. Na fase de estimação sendo conhecida a distribuição de probabilidade associada à posição actual, conhecendo o modelo e os comandos de velocidade do *robot* é possível estimar a nova distribuição de probabilidade de uma nova posição. Na fase de correcção são incorporadas

as medidas fundindo-se de uma maneira óptima a informação disponibilizada pelos sensores com a informação relativa à predição, compensando-se a informação perdida no passo anterior.

O filtro de Kalman estima um processo usando uma forma de realimentação: o filtro estima o estado de um processo e após esta predição obtém *feedback* do processo na forma de medidas ruidosas. Deste modo as equações de um filtro de Kalman dividem-se em dois grupos predição e correcção. As equações de predição projectam no tempo o estado do processo (estimativa à *priori*) e a covariância do erro da estimação. As equações de correcção realizam a realimentação, incorporando as medidas à estimativa à *priori*, obtendo-se a estimativa à *posteriori*. As equações que dependem do tempo podem ser pensadas como fazendo parte da predição e as que envolvem medidas fazendo parte da correcção.

O método utilizado neste trabalho para fusão de informação é o filtro de Kalman estendido, sendo essencialmente um conjunto de equações matemáticas que implementam um estimador do tipo predição-correcção que é óptimo no sentido que minimiza a covariância do erro dos parâmetros estimados, desde que algumas condições sejam alcançadas. O filtro de Kalman estendido é uma variante do filtro de Kalman apropriada para sistemas não lineares, sendo o caso do modelo de cinemática do *robot* omnidireccional de três rodas. O algoritmo do filtro de Kalman estendido encontra-se exemplificado na Figura 2.8 [WB01], sendo explicado em detalhe no Capítulo 6.

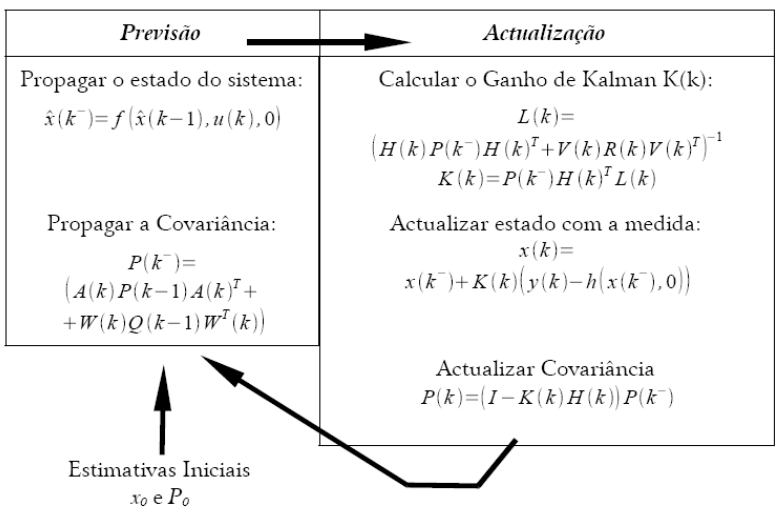

Figura 2.8: Algoritmo do filtro de Kalman estendido.

Capítulo 3

Sistema de localização baseado em visão global

3.1 Introdução

Esta secção descreve detalhadamente um sistema inspirado no que é utilizado pelas equipas participantes na *Small Size League* (SSL) do Robocup, o qual permite a localização em tempo real recorrendo a visão artificial. Certos aspectos relativos ao desempenho foram relaxados para permitir uma maior simplicidade no projecto e assim torná-lo adequado a um trabalho didáctico, assim como permitir o uso de material (câmaras, sistema de aquisição, PC para o processamento) mais comum e barato. O sistema descrito enquadra-se neste trabalho como ferramenta para a validação do sistema de localização e navegação desenvolvido, comparando-se a trajectória estimada pelo *robot* com a trajectória real disponibilizada pelo sistema de visão global. Tratando-se o futebol robótico de um ambiente dinâmico, é necessário que a informação da posição do *robot* e da bola esteja disponível com o mínimo de atraso possível [Sou03] [GPLC07], sendo que para a aplicação descrita, se está a falar de requisitos de tempo real a 25 hz, ou seja, 25 *frames* por segundo. A ferramenta desenvolvida é versátil para a sua utilização em várias etapas de aprendizagem no domínio da robótica móvel. Pode ser utilizada para validação de controladores e tácticas (monitorizando em tempo real jogos segundo as regras do Robocup Junior) [LP00], permite a validação de sistemas de localização e navegação para ambientes estruturados (monitorizando o desempenho do sistema de localização e navegação de um *robot*), sendo também utilizado no estudo de controlo e técnicas de fusão sensorial no domínio do Robocup [GLC08a] [GLC08b].

3.2 Sistema de visão global

Fisicamente, o sistema representado na Figura 3.1, é constituído pelos seguintes componentes:

- **Câmara**: Sony Video8 XR SteadyShot
- **Placa de Aquisição de Vídeo**: Pinnacle - chipset Bt878 (driver: Conexant's BtPCI WDM Video Capture)
- **PC**: AMD Athlon 64BIT 3500+, 1024 MB Ram, placa gráfica GECUBE ATI RADEON 512MB DDR e Windows XP Profissional SP2

Figura 3.1: Sistema de captura de imagem.

Assim, este sistema consiste em:

- Colocar uma câmara perpendicular à zona de acção da bola e do *robot* (tapete verde, que simula um relvado). A câmara está presa no centro de uma estrutura metálica, que permite uma altura máxima de 3 metros. Neste caso a câmara está colocada a 2 metros.
  Quanto maior for a altura da câmara em relação ao relvado, menor será o erro devido à paralaxe e o campo de visão angular da câmara aumenta, porém a qualidade de imagem baixa e a distorção em barril é maior. Como no sistema que se está a tratar a área de acção é de 1 m^2 e a câmara está colocada a uma altura de 2 metros, o efeito da distorção em barril é visível, mas por ser uma área de dimensões reduzidas, este efeito não acrescenta um erro de localização significativo, sendo por isso desprezado. Estes aspectos serão discutidos em pormenor nas subsecções seguintes.

– **Paralaxe**:

O efeito de paralaxe é minimizado aumentando a altura da câmara. Na Figura 3.2 pode-se ver que para uma altura $h_2 > h_1$ o erro de paralaxe é reduzido consideravelmente [Cos99].

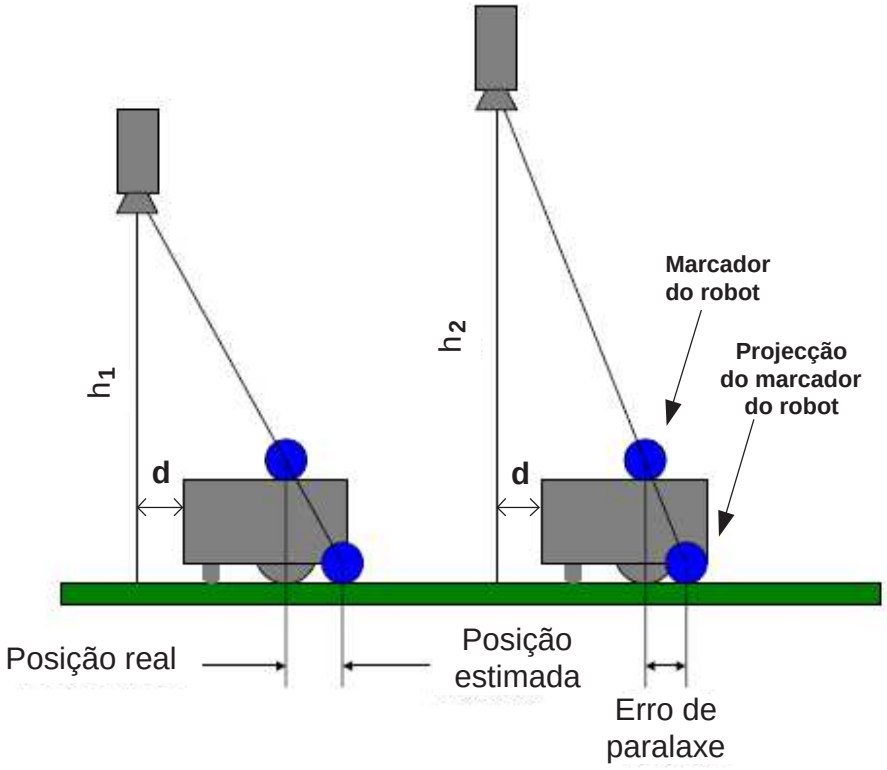

Figura 3.2: Erro de paralaxe para diferentes alturas da câmara.

Este factor pode ser compensado se a altura a que estão colocados os marcadores (círculos coloridos) no *robot* for conhecida. Este mecanismo de compensação foi implementado, permitindo uma maior precisão na determinação da posição quer da bola quer do *robot*. Os marcadores são utilizados para extrair informação sobre a localização absoluta do *robot* móvel.

– **Qualidade da imagem**:

O conceito de qualidade de imagem, neste caso, prende-se com o facto de quanto mais alta estiver colocada a câmara menos serão os pixeis que servirão para identificar os marcadores. Sendo estes de dimensões reduzidas poderá haver situações em que os marcadores não tenham um número suficiente de pixeis que o permitam identificar e consequentemente localizá-lo.

Assim sendo, o sistema de visão deve possuir as seguintes características:

– **Precisão**:

Uma vez que estamos a lidar com um *robot* com 18 cm de diâmetro (forma circular) e uma bola com 4 cm de diâmetro, é necessário que o erro na sua localização não exceda alguns milímetros.

– **Rapidez**:

Tratando-se de um ambiente dinâmico, é necessário que a informação da posição do *robot* e da bola esteja disponível com o mínimo de atraso possível. Por forma a perceber a importância desta característica, descreve-se o seguinte cenário: atendendo a velocidades na ordem dos 2 ms^{-1}, quer para a bola quer para o *robot*, é fácil de perceber que um atraso de 100 ms pode ser considerado muito grave. Este é um intervalo de tempo suficiente para que o *robot* e/ou a bola se desloquem cerca de 20 cm. Para que estas situações não aconteçam, os requisitos de tempo real têm de ser cumpridos, sendo que neste caso, se está a falar de requisitos de tempo real a 25 hz, ou seja, 25 *frames* por segundo.

– **Robustez**:

A interpretação da imagem pode, por vezes, dar origem a medidas de posição completamente erradas devido a uma má identificação dos marcadores. Uma das condições mais importantes, senão mesmo a mais importante, é a iluminação. Por vezes, a presença de sombra, por mais ténue que seja, pode alterar significativamente a cor, contribuindo assim, para eventuais erros, quer na calibração da cor, quer na detecção e localização de marcadores.

3.2.1 Dos pixeis para o estado do sistema

Considerando que a partir deste momento se dispõe da representação digital da imagem do relvado, esta será constituída por uma matriz de elementos de imagem p_{xy}, designados por pixeis, contendo cada um uma representação da área de imagem abrangida. Cada pixel corresponde a um vector com três componentes que representam a intensidade das três componentes de cor: vermelho, verde e azul (normalmente designadas de R, G e B). Assim, pode-se definir a seguinte função:

$$\begin{aligned} RGB : I &\longrightarrow \Re^3 \\ p_{xy} &\longrightarrow (r, g, b) \end{aligned} \tag{3.1}$$

que, para cada pixel, indica a cor associada, sob a forma de um vector com as intensidades das três componentes.
Estas componentes são normalmente discretizadas em, respectivamente, n_r, n_g e n_b bits. Tipicamente, para $n_r = n_g = n_b = 8$, num total de 24 bits, o que já permite uma representação em que o efeito da discretização é visualmente indetectável.

3.2.2 Sistema de coordenadas

Como o objectivo é localizar o *robot* e a bola no relvado é necessário fixar a origem e a orientação dos eixos do sistema de coordenadas a utilizar. Para a origem foi escolhido o centro do relvado e considera-se o eixo dos xx alinhado longitudinalmente em relação ao relvado.

Cada pixel p_{xy} pode ser encontrado numa matriz em que x indica a linha e y a coluna do pixel. Neste caso, o pixel com coordenadas (0,0) será o pixel do canto superior esquerdo e o pixel com coordenadas (x_m - 1, y_m - 1) encontra-se no canto inferior direito, onde x_m e y_m definem a dimensão da imagem.
Este sistema de coordenadas reflecte o de uma câmara PAL, explorando-se a imagem em linhas que a varrem da esquerda para a direita, numa sequência em que a primeira linha se encontra no topo e as seguintes vão descendo.

Para facilitar a distinção entre as coordenadas (x, y) correspondentes a uma localização no mundo e as correspondentes ao pixel de coordenadas (x, y), esta última passará a ser definida pelos conjuntos Nx e Ny:

$$Nx = \{0,\ 1,\ ...,\ x_m\ \text{-}\ 1\}$$
e
$$Ny = \{0,\ 1,\ ...,\ y_m\ \text{-}\ 1\}$$

3.2.3 Mapeamento imagem - mundo

Para extrair as localizações dos objectos observados, a partir da imagem, tem que se construir a seguinte função:

$$\begin{aligned} m : Nx \times Ny &\longrightarrow \Re^2 \\ (u, v) &\longrightarrow (x, y) \end{aligned} \tag{3.2}$$

A equação (3.2) mapeia coordenadas 2D da imagem em coordenadas do mundo. Esta função fornece as coordenadas x e y assumindo que a componente em z é zero. Como os objectos em questão (*robot* e bola), não estão colocados à mesma altura, haverá erro devido ao fenómeno de paralaxe. Este erro pode ser compensado caso a altura dos objectos seja conhecida. Com este valor pode-se construir uma função que ajusta x e y tendo em conta a altura dos objectos e da câmara. Este ajuste só pode ser realizado após uma localização do objecto.

3.2.4 Correcção da paralaxe

Na prática, com a imagem adquirida, apenas se consegue extrair informação relativa aos objectos no plano xy. Conhecendo z pode-se corrigir o valor de $\tilde{x}$, $\tilde{y}$ para obter x, y já devidamente compensados no que diz respeito à paralaxe.
A função L que implementa esta compensação pode ser descrita por:

$$\begin{aligned} L : \Re^3 &\longrightarrow \Re^2 \\ (\tilde{x}, \tilde{y}, z) &\longrightarrow (x, y) \end{aligned} \tag{3.3}$$

com

$$x = L_x(\tilde{x}, \tilde{y}, z) = \tilde{x} - \frac{z}{h} \tag{3.4}$$

$$y = L_y(\tilde{x}, \tilde{y}, z) = \tilde{y} - \frac{z}{h} \tag{3.5}$$

onde h representa a altura a que está colocada a câmara [Cos99].

3.2.5 Calibração da cor

Escolha dos marcadores:

Um dos factores mais importantes na calibração da cor, é a escolha dos marcadores que serão utilizados. É através destes que se consegue detectar e localizar quer a bola quer o *robot*. No que diz respeito à bola a decisão recaiu em usar a cor laranja, uma vez que é a cor mais usada neste tipo de aplicações (caso do futebol robótico, quer na liga SSL quer na MSL [rob09]).
Para o *robot* a escolha já foi mais cuidadosa. Sabendo à partida que seriam necessários dois marcadores diferentes, um para identificar o centro do *robot* e outro que permitisse saber o ângulo que o *robot* assume em cada instante. Sendo o relvado de cor verde e a bola de cor laranja, as cores para o *robot* tinham que ser as mais afastadas possível no cubo RGB. Assim, as cores escolhidas foram o azul para o centro do *robot* e o amarelo para o ângulo, sendo estas as cores oficiais para a diferenciação das equipas na liga SSL [rob09]. O *robot* utilizado está ilustrado na Figura 3.3.

Figura 3.3: Protótipo do *robot* omnidireccional.

Calibração de marcadores:

A calibração de marcadores, de uma forma geral, consiste em "dizer" ao sistema que um determinado marcador fica associado a uma determinada cor. Desta forma, torna-se muito simples encontrar um determinado objecto numa imagem, bastando para isso percorrer o cubo RGB e procurar a cor com que esse objecto foi calibrado.
Neste caso, é necessário efectuar a calibração da bola e dos dois marcadores do *robot*, que ficarão associados às seguintes cores:

- **Bola**: mapeada com a cor vermelha
- **Centro do *Robot***: mapeada com a cor verde
- **Ângulo do *Robot***: mapeada com a cor azul

A calibração é efectuada com o rato, escolhendo o objecto a calibrar e clicar nesse mesmo objecto na imagem de manipulação. Quantas mais vezes se clicar na mesma zona mais pontos ficam calibrados, aumentando assim a qualidade de calibração. Estes pontos ficam guardados num cubo RGB (array tridimensional de dimensões $(255 \times 255 \times 255)$).
Outro factor importante na calibração de marcadores é a incidência de luz. Assim deve-se calibrar os marcadores em diferentes locais para que, desta forma, possa ficar com mais informação sobre os marcadores, para diferentes intensidades de luz.

3.2.6 Selecção da área activa

Outra das vantagens da calibração de posições, é a possibilidade de se poder seleccionar somente a área onde se pretende localizar os marcadores, isto porque, haverá quase sempre zonas da imagem que representam áreas para as quais não há interesse em procurar marcadores. Como o relvado não ocupa toda a imagem, existe sempre uma banda em volta da imagem que mostra o chão que o rodeia.
Este processo permite que os algoritmos de pesquisa de marcadores sejam mais rápidos, consequentemente baixando o tempo de processamento, uma vez que, por norma, a área activa vai ser de dimensões menores em relação à imagem adquirida.

Assim, para cada linha y, são indicadas duas coordenadas x_1 e x_2 que marcam o início e o fim da área activa para essa linha. Isto equivale a

construir uma função:

$$\begin{aligned} V_A : Ny &\longrightarrow Nx \times Nx \\ y &\longrightarrow (x_1, x_2) \end{aligned} \tag{3.6}$$

A principal vantagem desta representação é que já não obriga a interrogar cada pixel para verificar se ele deve ser ou não processado. Considerando um cenário em que o processamento está a ser feito segundo um varrimento de linhas, desde o canto superior direito da imagem e pesquisando os pixeis da esquerda para a direita:

- Uma linha que não contenha pixeis para serem processados pode ser imediatamente descartada. Para isso basta a avaliação da função V_A para essa linha indicar $x_2 < x_1$.
- Numa linha que tenha alguns pixeis para tratar, pode-se avançar de imediato para o pixel x_1 descartando todos os pixeis anteriores e pode-se parar o processamento quando se atingir x_2. Assim serão descartados automaticamente os pixeis que faltam para acabar a linha.

Outra vantagem, que por si só já justificaria a restrição do processamento da imagem a uma zona activa, é que poderá haver no cenário todo o tipo de objectos, que não haverá interferências mesmo que as suas cores coincidam com as dos marcadores.

3.2.7 Detecção e localização dos marcadores

É importante referir que, apesar de existirem referências constantes à bola, esta, não deixa de ser um marcador tal como os marcadores do *robot*. Nesta secção explica-se o algoritmo que permite, não só, detectar os marcadores como localizar a posição em que estes se encontram. Convém relembrar que os marcadores têm todos a mesma forma geométrica (circulares), sendo por isso este algoritmo aplicável a qualquer marcador, mudando apenas o objecto que se pretende localizar.

Detecção dos marcadores:

Estando os marcadores mapeados com uma determinada cor na matriz de calibração, basta efectuar uma pesquisa nesta mesma matriz pelo

código RGB correspondente ao marcador em questão. A partir do momento em que se encontram pixeis com este código pode-se concluir que o objecto está a ser detectado.

Localização dos marcadores:

Uma vez detectado um pixel calibrado com a cor do objecto que se pretende localizar, é guardada a coordenada em x e y da imagem. Este processo é repetido para toda a área activa, sendo que as coordenadas são acumuladas. No fim de percorrer esta área é calculada a média da soma de todas as coordenadas, o que equivale a dizer que se fica com a localização do centro da circunferência, ou seja do marcador.
Assim, sendo I a imagem que contém os dados da calibração dos objectos e admitindo que pretendemos localizar o marcador que foi mapeado com o código $r = 255$, $g = 0$ *e* b *=0*, ou seja, a bola, pode-se definir a seguinte função:

$$\begin{aligned} I_{i,j}.R &= 255 \\ \tilde{x} &= \tilde{x} + i \\ \tilde{y} &= \tilde{y} + j \\ N_P &= N_P + 1 \end{aligned} \tag{3.7}$$

onde N_P é o número de pontos.

da qual resulta o total acumulado para as duas coordenadas, bem como o número total de pontos que foram detectados. Pode-se então admitir que a posição em que o marcador se encontra é dada por:

$$x = \frac{\tilde{x}}{N_P} \tag{3.8}$$

$$y = \frac{\tilde{y}}{N_P} \tag{3.9}$$

Contudo, e por este processo ser simples, é preciso ter especial atenção em alguns aspectos. Um deles, foi anteriormente descrito e já está resolvido, que é o facto de objectos que estão fora da área activa não poderem entrar no calculo da média.

Agora considerem-se os seguintes cenários:

– **Cenário 1**:

Dada a altura da câmara e as reduzidas dimensões dos marcadores, podem ocorrer situações em que para um marcador não existam mais que dois ou três pontos calibrados. Será que se pode confiar na posição que é retornada? E se estes pontos nem se encontram juntos, ou muito próximos, será que se pode admitir que estes pontos pertencem mesmo ao marcador?

– **Cenário 2**:

Admitindo que é sempre possível acontecerem ocorrências de ruído aleatório (por inúmeras causas), podem ocorrer situações em que existem bastantes pixeis a caracterizar um marcador e existirem pixeis com o mesmo código RGB espalhados pela imagem. Deverão estes pontos entrar no cálculo da média?

De seguida serão apresentadas as soluções que foram adoptadas para solucionar estes possíveis aspectos.

- **Solução para cenário 1**:

A solução para este caso, passa por admitir que o marcador não está dentro da área de acção activa. Assim, se forem encontrados menos que 5 pixeis admite-se que estes pixeis indicam uma posição, que depois de ser sujeita ao mapeamento imagem - mundo, dará uma posição fora da área activa.

- **Solução para cenário 2**:

A solução para este caso, passa por calcular uma nova posição, contudo só serão incluídos para o cálculo da média os pixeis que estejam numa vizinhança em relação à posição anteriormente calculada. Desta forma todos os pixeis que estejam espalhados pela imagem e que não pertençam ao marcador não serão incluídos, corrigindo o erro fornecido pela posição anterior.
Como as dimensões dos marcadores são reduzidas, a vizinhança estabelecida para definir os pixeis pertencem ao marcador, tem uma dimensão de 20×20.

Desta forma às funções descritas anteriormente em (3.7), (3.8) e (3.9) serão acrescentadas as seguintes funções :

$$\begin{aligned} I_{i,j}.R = 255 \ \wedge \ i \ \exists \ [x-20, x+20] \ \wedge \ j \ \exists \ [y-20, y+20] \\ \tilde{x} = \tilde{x} + i \\ \tilde{y} = \tilde{y} + j \\ N_P = N_P + 1 \end{aligned} \tag{3.10}$$

$$x = \frac{\tilde{x}}{N_P} \tag{3.11}$$

$$y = \frac{\tilde{y}}{N_P} \tag{3.12}$$

3.2.8 Determinação do ângulo do *robot*

Como já foi referido anteriormente na subsecção (3.2.7), os marcadores de cor azul e amarelo são utilizados para detecção e localização do *robot*. Assim, o marcador azul permite saber a posição em que o *robot* se encontra, e o marcador amarelo permite saber a orientação que o *robot* toma num determinado momento.
A informação útil que se pode retirar dos marcadores (para este caso em concreto), é a posição do centro de (x,y) de cada um.
Considerando $P_r(x, y)$ a posição fornecida pelo marcador azul e $P_a(x, y)$ a posição fornecida pelo marcador amarelo, pode-se definir o vector que une P_r a P_a tendo como parâmetros a e b, tal como exemplificado na Figura 3.4.

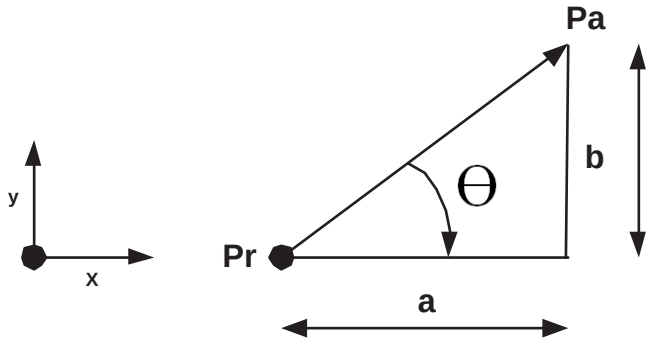

Figura 3.4: Ângulo do *robot* em relação ao sistema de eixos.

Assim, as componentes do vector em x e y são dadas por a e b respectivamente:

$$a = P_a(x) - P_r(x) \tag{3.13}$$
$$b = P_a(y) - P_r(y) \tag{3.14}$$

Desta forma é possível saber o ângulo do *robot*, recorrendo à operação trigonométrica:

$$\theta = atan2(b, a) \tag{3.15}$$

Sendo esta operação realizada pela função atan2(b,a), a qual recebe 2 parâmetros, resolvendo o problema relacionado com divisão por zero e calculando o ângulo independentemente do quadrante em que se encontre.

3.2.9 Estudo do erro na localização do *robot*

Foi realizado para o sistema de localização uma análise das distribuições de probabilidade do erro, aproximadas a distribuições normais [Rib04a], apresentado-se os resultados em unidades SI. Foi analisada a variância da estimativa da posição do *robot* em função do número de pixeis observados. Para cada intervalo apresentado foram analisadas 256 amostras da estimativa de localização. O número de pixeis obtidos para o marcador azul ($Q1$), afecta a variância do erro na localização em x e y, tal como exemplificado na Tabela seguinte:

$Q1$	x	y
5-10	1,5E-05	1,9E-05
10-20	9,25E-06	7,36E-06
20-30	4,84E-06	4,86E-06
30-40	4,15E-06	3,80E-06
≥ 40	1,96E-06	2,21E-06

Por sua vez a variância do erro do ângulo é afectada pelo número de pixeis obtidos para ambos os marcadores, azul ($Q1$) e amarelo ($Q2$), tal como exemplificado na Tabela seguinte:

$Q1$	5-10	10-20	20-30	30-40	≥ 40
5-10	0,14	8E-02	1,2E-02	1E-02	6,2E-03
10-20	1,6E-02	9,9E-03	1,3E-02	6,6E-03	4,6E-03
20-30	1,5E-02	9,9E-03	7,2E-03	4,9E-03	3,9E-03
30-40	1,4E-02	9,5E-03	5,9E-03	4,4E-03	2,9E-03
≥ 40	1,4E-02	7,2E-03	5,77E-03	3E-03	3E-03
$Q2$					

A variância para menos que 5 pixeis não é apresentada, pois para esta situação considera-se que a confiança no sensor é nula. Dois exemplos de distribuições de probabilidade do erro são mostrados nas Figuras 3.5 e 3.6, sendo aproximadas a distribuições de probabilidade gaussianas.

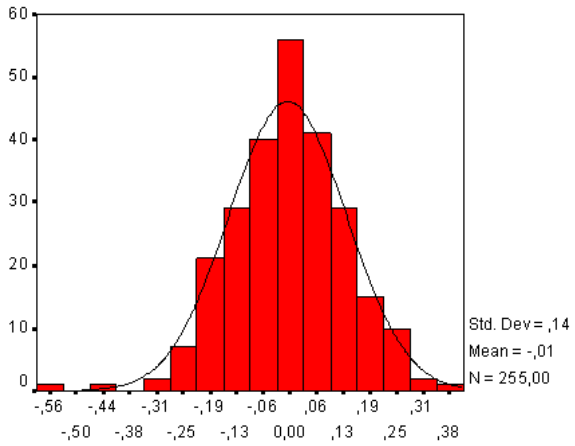

Figura 3.5: Distribuição de probabilidade para Q1 e Q2 entre 5 e 10 pixeis.

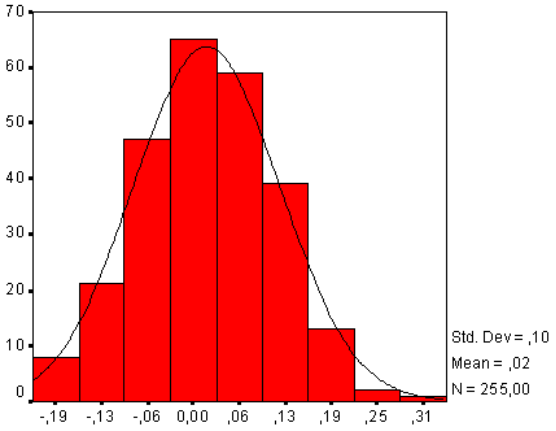

Figura 3.6: Distribuição de probabilidade para Q1 e Q2 entre 10 e 20 pixeis.

Capítulo 4

Modelação e simulação do Kit Lego Mindstorms NXT

4.1 Introdução

O Kit Lego Mindstorms NXT é uma ferramenta educacional muito poderosa para ensinar conceitos introdutórios de robótica, principalmente de robótica móvel. Este Kit educacional permite a prototipagem rápida de *robots* móveis e a realização de controladores com recurso a linguagens de alto nível, tendo em conta os inevitáveis e apertados requisitos de tempo real associados a desafios de robótica móvel. Verificou-se uma significativa evolução do Kit Lego Mindstorms NXT em relação ao RIS, sendo impulsionada pelas necessidades que os entusiastas deste Kit foram informalmente solicitando com o decorrer dos tempos. Uma das grandes melhorias apresentadas no Kit NXT foi a introdução de *encoders* incorporados nos motores, o que permite de uma maneira mais simples realizar o cálculo de medidas relativas. Outra grande vantagem do Kit NXT face ao seu predecessor foi a inclusão de peças que permitem uma maior flexibilidade para a realização de protótipos de *robots* omnidireccionais apenas com peças Lego, dada a grande dificuldade de realizar montagens de peças com ângulos diferentes de 90º [GCM04]. Na Figura 4.1 encontram-se dois protótipos que foram utilizados numa demonstração no centro de Ciência Viva de Bragança, sendo um dos *robots* omnidireccional e sendo ambos prototipados apenas com peças Lego.

Outra das grandes melhorias foi o facto de ser acrescentado o sensor de distância de ultra-sons que é muito útil para obter medidas absolutas de localização. É também relevante acrescentar que a capacidade de processamento subiu consideravelmente permitindo a realização de programas de controlo mais elaborados. O *display* do *Brick* NXT é também uma característica nova

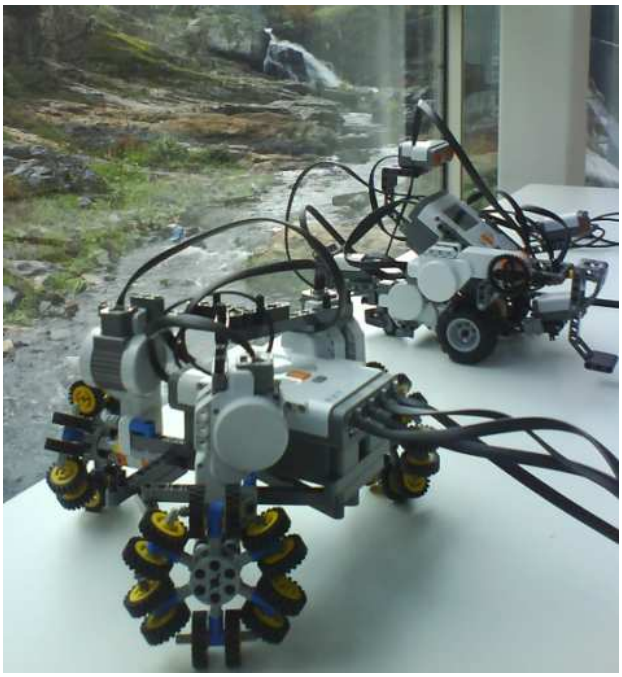

Figura 4.1: Demonstração no centro de Ciência Viva de Bragança.

que permite realizar mais facilmente o *debug* de programas, podendo disponibilizar muita informação em simultâneo e em tempo real. Para além de estas características inovadoras a introdução de comunicação por Bluetooth abriu novas possibilidades para a realização de controladores externos ao *Brick* e desafios de robótica cooperativa.

Neste Capítulo será descrita a modelação e simulação realista de um *robot* prototipado com base no Kit Lego Mindstorms NXT [GLMC09b]. O protótipo desenvolvido, apresentado na Figura 4.2, é um robot diferencial, composto por duas rodas e um sensor de luz. Também será apresentada a modelação do sensor de toque e do sonar do Kit Lego Mindstorms NXT.

4.2 Cinemática do protótipo desenvolvido

O *robot* diferencial é provavelmente o *robot* móvel mais usado, sendo representado na Figura 4.3. O *robot* diferencial é composto por duas rodas, cujos veios passam pelo mesmo eixo, sendo o seu movimento controlado variando independentemente a velocidade de cada uma das rodas.

A estrutura do *robot* diferencial, representada na Figura 4.3, impede que sejam feitos movimentos de translação segundo o eixo que passa pelos veios

Figura 4.2: Protótipo desenvolvido.

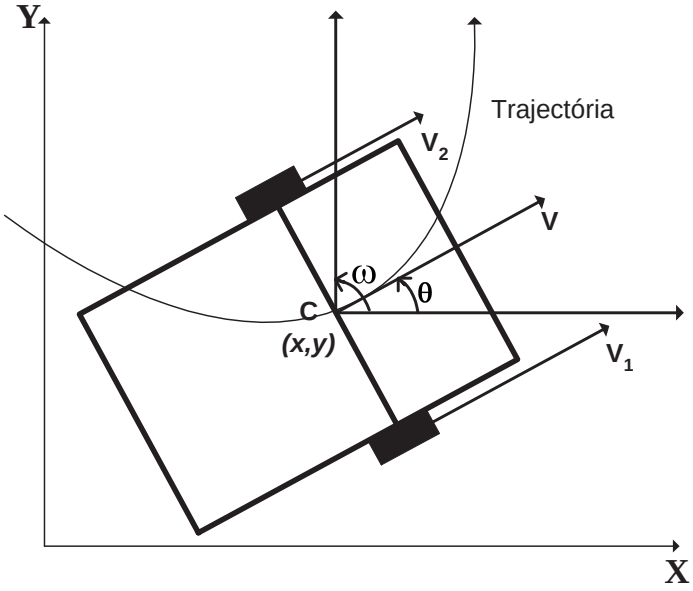

Figura 4.3: *Robot* diferencial.

dos motores [DJ00], considerando que não existe escorregamento lateral, isto é, que a velocidade das rodas no ponto de contacto com o chão seja sempre perpendicular ao eixo que passa pelas mesmas, obtemos o vector de estado representado pela seguinte equação:

$$X(t)^T = \begin{pmatrix} x(t) & y(t) & \theta(t) & v(t) & w(t) \end{pmatrix} \tag{4.1}$$

Em que $x(t)$, $y(t)$ e θ(t) representam a posição do ponto C no plano e $w(t)$ representa a velocidade angular (velocidade de rotação do *robot* segundo o eixo vertical que passa por C). Uma outra possibilidade para a escolha das variáveis de estado seria a utilização da seguinte equação:

$$X(t)^T = \begin{pmatrix} x(t) & y(t) & \theta(t) & V_1(t) & V_2(t) \end{pmatrix} \tag{4.2}$$

Neste caso $V_1(t)$ e $V_2(t)$ são velocidades medidas do ponto de contacto entre o chão e as rodas. Existem estas duas representações possíveis podendo-se passar de uma para outra usando a equação (4.3) e a equação (4.4). Na equação (4.4) b representa a distância entre os pontos de contacto das rodas com o chão.

$$V(t) = \frac{V_1(t) + V_2(t)}{2} \tag{4.3}$$

$$w(t) = \frac{V_1(t) - V_2(t)}{b} \tag{4.4}$$

Considerando a condição de não escorregamento a cinemática do *robot* diferencial está descrita pela seguinte equação:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} v(t)cos(\theta(t)) \\ v(t)sin(\theta(t)) \\ w(t) \end{pmatrix}$$

Esta equação permite usando as equações (4.3) e (4.4) exprimir as velocidades lineares $\dot{x}$ e $\dot{y}$ e a velocidade angular $\dot{\theta}$, em função das velocidades de cada uma das rodas, podendo estas ser medidas [Cos95].

4.3 Modelação e simulação do protótipo

É possível realizar a simulação realista de comportamento sem acesso a *hardware* com recurso a motores de física. A simulação da dinâmica do *robot* foi feita recorrendo ao ODE (*Open Dynamics Engine*), sendo uma biblioteca livre para a simulação da dinâmica de corpos rígidos [ode09]. A arquitectura de simulação é baseada no *robot* real prototipado com o Kit Lego Mindstorms NXT. As massas e dimensões foram seguidas à risca por forma a construir um *robot* simulado igual ao real. O *robot* pesa 0.540 Kg, sendo o peso de cada uma das suas partes constituintes especificado na Tabela 4.1. A migração de código de simuladores realistas para sistemas reais é a chave para reduzir o tempo de desenvolvimento *software* de localização, navegação e controlo de *robots* móveis. Com este propósito foi desenvolvido o simulador realista SimTwo (disponível para descarregar em [sim09]). Devido à inerente complexidade de desenvolver modelos realistas de *robots*, de sensores, de actuadores e a sua interacção com o mundo, não se torna uma tarefa fácil desenvolver simuladores para *robots*. Nesta secção é descrita a modelação de um *robot* baseado na plataforma educacional Lego Mindstorms NXT e a sua simulação realista. Por forma a validar a abordagem proposta foi comparado o desempenho do *robot* real e da sua simulação através de um desafio.

Peças	Peso (Kg)
Motor	0.079
Brick	0.255
Sensor de luz	0.013
Rodas	0.017
Restantes peças	0.080

Tabela 4.1: Pesos das peças do *robot*.

4.3.1 Modelo do motor DC

O kit Lego Mindstorms NXT disponibiliza três servomotores tendo *encoders* embutidos e uma caixa redutora de 1:48. Um servomotor da Lego Mindstorms encontra-se ilustrado na Figura 4.4.

Um servomotor pode ser resumido ao modelo de um motor DC, apresentado na Figura 4.5, onde U_a é a tensão de saída do conversor, R_a é a resistência equivalente, L_a é a indutância equivalente e e é a tensão gerada pela força contra-electromotriz, tal como expresso na equação (4.6) [CMC06].

Figura 4.4: Servomotor do kit Lego Mindstorms NXT.

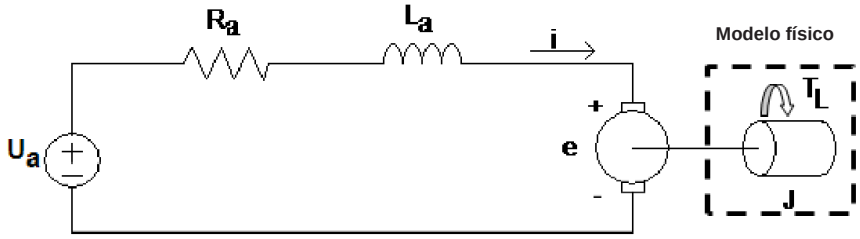

Figura 4.5: Modelo eléctrico do motor DC.

$$U_a = e + R_a i_a + L_a \frac{\partial i_a}{\partial t} \tag{4.6}$$

O motor pode disponibilizar um binário T_L e a carga tem momento de inércia J o qual será simulado computacionalmente pelo modelo físico criado com base no ODE. A corrente i_a pode ser relacionada com o binário desenvolvido T_D através da equação (4.7) onde K_s é um parâmetro do motor que pode ser encontrado através da preparação experimental apresentada na subsecção 4.3.2 [Bis02]. A tensão gerada pela força contra-electromotriz está relacionada com a velocidade angular através da equação (4.8).

$$T_D = K_s i_a \tag{4.7}$$

$$e = K_s \omega \tag{4.8}$$

De facto, o binário útil que será aplicado à carga (T_L) é o binário desenvolvido subtraído do binário gerado pelo atrito (T_F), tal como exemplificado na equação (4.9).

$$T_L = T_D - T_F \tag{4.9}$$

O valor de T_F será a soma de duas componentes o atrito estático T_C e o atrito viscoso T_ω. O atrito estático trata-se de uma constante e o atrito viscoso é directamente proporcional à velocidade angular do motor.

4.3.2 Medidas para modelação do motor DC

O servomotor do Kit Lego Mindstorms NXT foi utilizado como base para o simulador. Os valores de R_a e L_a foram medidos directamente (R_a=7.6 Ω e L_a= 4.88 mH). O parâmetro K_s do motor pode ser encontrado através de uma medida indirecta. Para várias medidas angulares pode ser medida a tensão gerada pela força contra-electromotriz do motor estando este em circuito aberto. A Tabela 4.2 apresenta os dados para 7 medidas.

rot (rto/min)	rot (rad/s)	e (V)	k=e/rot
0	0	0	
66	6.91	3.4	0.491
115	12.04	6.0	0.498
155	16.23	7.9	0.487
195	20.42	9.8	0.479
233	24.39	11.9	0.487
265	27.75	13.85	0.499

Tabela 4.2: Velocidade e tensão gerada pela força contra-electromotriz.

A Figura 4.6 mostra os dados apresentados na Tabela anterior e a respectiva linha de tendência. Pode-se observar que o erro na origem para esta aproximação não é muito significativo, sendo apenas 0.0218 Volt. Apesar do pequeno erro apresentado é possível aumentar o rigor na estimação do parâmetro K_s obrigando a linha de tendência a passar na origem. O valor obtido para K_s é sensivelmente 0.4908 V.rad/s.

Para o cálculo do atrito procedeu-se à obtenção dos dados relativos à velocidade angular que o motor irá assumir para diferentes tensões aos seus terminais. Após regressão linear obteve-se a equação (4.10).

$$rot = 1.9248v - 0.2519 \tag{4.10}$$

Onde rot é a velocidade angular em rad/s e v é a tensão aos terminais do motor. Assumindo que o binário desenvolvido (T_D) é igual ao binário gerado pelo atrito estático (T_C) pode-se, a partir da regressão linear apresentada na equação (4.10), obter a tensão necessária para igualar o atrito estático (v_c), substituindo rot por zero e resolvendo a equação em ordem a v. Para esta situação não foi necessário incluir no modelo o atrito viscoso, dado este só

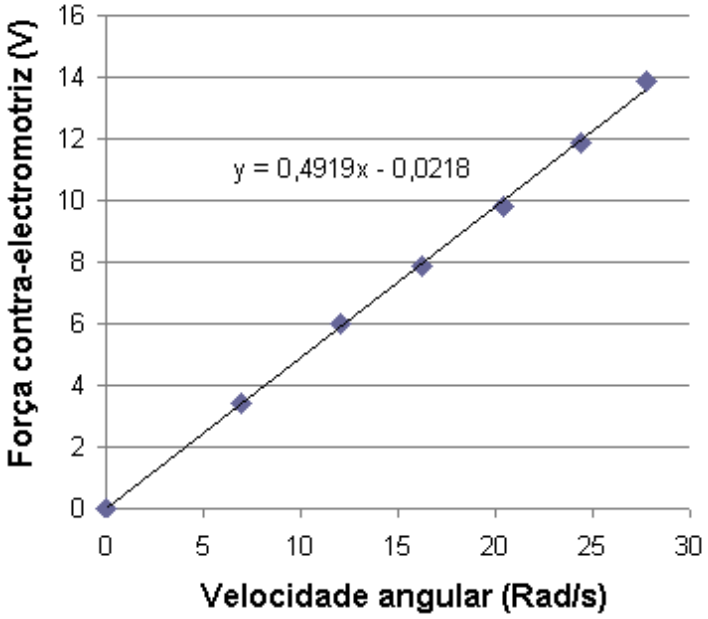

Figura 4.6: Valor de K_s do modelo do motor DC.

existir com movimento de rotação do motor. Como não existe movimento não existe tensão gerada pela força contra-electromotriz e como a corrente não tem variação é possível obter o binário devido ao atrito estático recorrendo às equações 4.6 e 4.7, estando descrito na equação (4.11). O binário devido ao atrito estático terá um valor numérico de 3.558E-4 Nm.

$$T_C = K_s \frac{v_c}{R_a} \tag{4.11}$$

Após a obtenção do atrito estático falta apenas, para completar o modelo do motor, a estimação do parâmetro que relaciona o atrito viscoso com a velocidade angular do motor, tal como mostrado na equação (4.12).

$$T_\omega = B\omega \tag{4.12}$$

Estando o motor a rodar sem carga o binário desenvolvido será expresso pela equação (4.13).

$$T_D = T_C + B\omega \tag{4.13}$$

Como o binário é proporcional à corrente tal como exemplificado na equação (4.7) pode-se deduzir, a partir da equação (4.13), que a corrente pode ser expressa pela equação (4.14).

$$i_a = \frac{T_C + B\omega}{K_s} \tag{4.14}$$

A tensão aos terminais do motor pode ser expressa pela equação (4.6), mas como para os vários pontos de funcionamento o motor encontra-se em regime permanente as variações da corrente podem ser desprezadas, obtendo-se a equação (4.15) para a tensão aos terminais do motor.

$$U_a = K_s\omega + R_a i_a \tag{4.15}$$

onde e foi substituído por $K_s\omega$ tal como exemplificado na equação (4.8). Das equações 4.14 e 4.15 obtem-se a equação (4.16).

$$\omega = \frac{U_a - \frac{R_a T_C}{K_s}}{\frac{R_a B}{K_s} + K_s} \tag{4.16}$$

Minimizando a soma do valor absoluto das diferenças entre a velocidade angular obtida para o modelo (equação (4.16)) e para os dados experimentais obteve-se o valor numérico para B, sendo de 1.92E−3.

Os parâmetros estimados para o motor estão mostrados na Tabela 4.3.

Parâmetros	Valor unidades SI
K_s	0.4908
T_C	3.558E-4
B	1.92E−3
R_a	7.6
L_a	4.88E−3

Tabela 4.3: Parâmetros do motor.

Os diferentes modelos obtidos para o motor e os seus respectivos erros podem ser observados nos gráficos apresentados na Figuras 4.7 e 4.8.

4.3.3 Não linearidades do motor DC

Com o objectivo de melhor modelar a realidade, na qual as variáveis não podem assumir todos valores, devem ser impostas ao modelo limitações. A primeira é a tensão que pode ser aplicada ao terminais do motor U_a. Esta tensão deve ser limitada à tensão das baterias. A corrente também deve ser limitada, estando esta relacionada com o binário pela equação (4.7). A Figura 4.9 mostra as limitações apresentadas pelo servomotor.

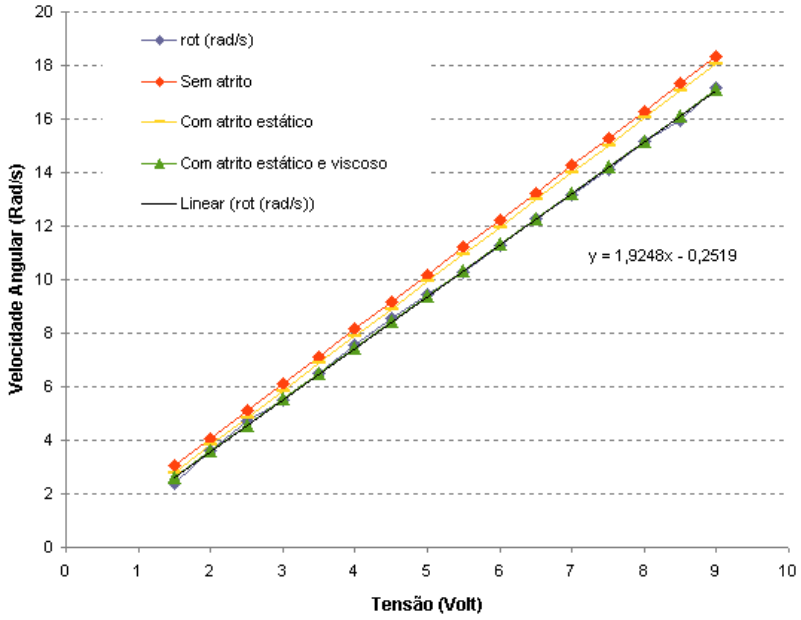

Figura 4.7: Modelos obtidos para o motor.

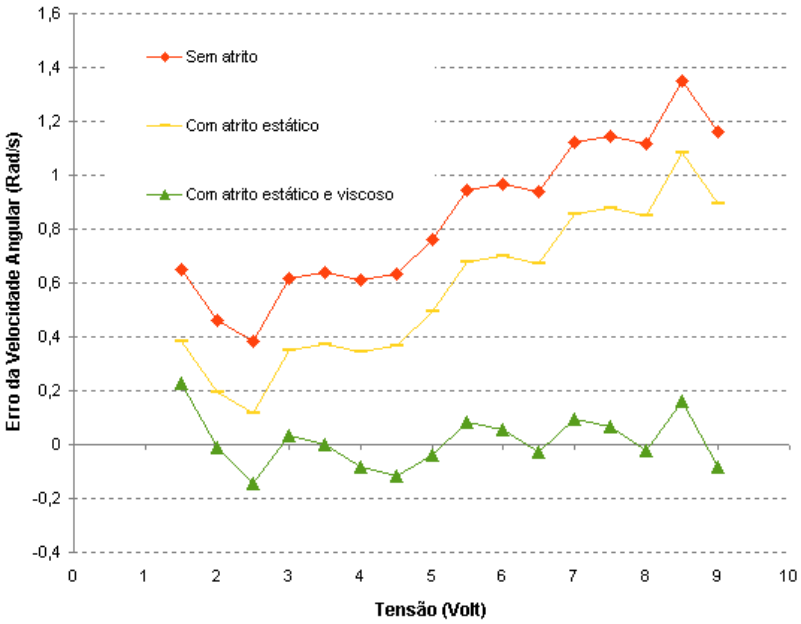

Figura 4.8: Erros dos modelos do motor.

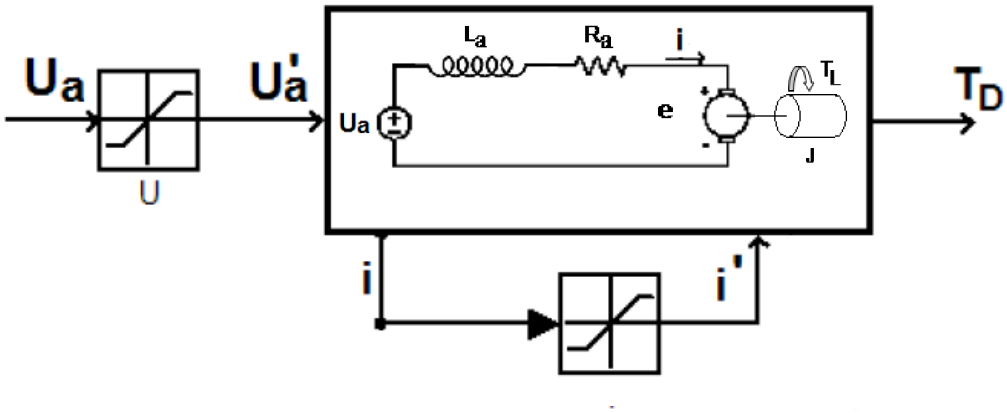

Figura 4.9: Não linearidades do servomotor.

4.3.4 Modelação do sensor de luz

O sensor de luz, representado na Figura 4.10, permite ao *robot* distinguir entre claro e escuro. Pode ler intensidade de luz num compartimento e medir intensidade de luz reflectida de uma superfície colorida. O sensor para medidas ao longo de um gradiente impresso numa página A4 dá valores de acordo com o gráfico mostrado na Figura 4.11, onde podem ser vistos os dados do sensor e a sua aproximação a uma recta com recurso a uma regressão linear. Observou-se que a taxa de variação do sensor em relação ao gradiente é constante, mas o valor lido pelo sensor depende da iluminação do compartimento, devendo-se adicionar um *offset* à medida retornada. O gradiente utilizado para obter dados do sensor de luz pode ser visto na Figura 4.14, onde está ilustrado o desafio utilizado para validar a modelação do *robot*.

Figura 4.10: Sensor de luz do Kit Lego Mindstorms.

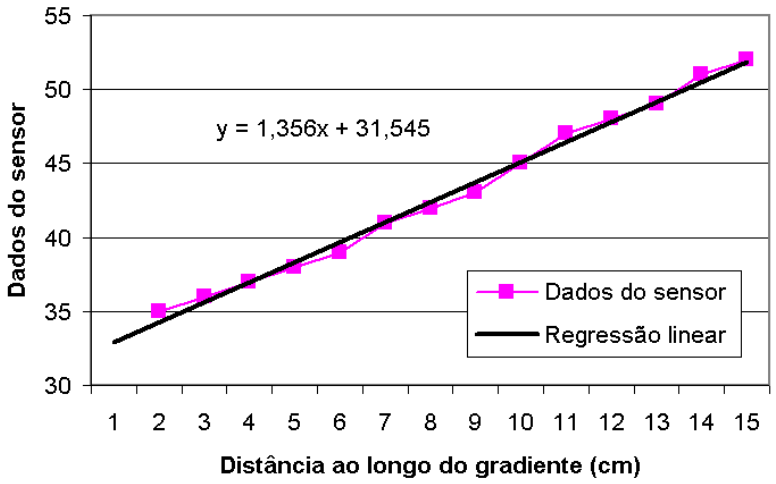

Figura 4.11: Característica do sensor de luz.

4.3.5 Simulação do *robot*

Para a simulação do protótipo desenvolvido com base no Kit Lego Mindstorms foi utilizado o simulador realista SimTwo, sendo uma ferramenta versátil para a simulação de *robots* e ambientes onde estes operam, permitindo modelar e testar *robots* diferenciais, omnidireccionais, manipuladores, humanóides, etc., sendo desenvolvido em Object Pascal. O SimTwo tem um conjunto de componentes predefinidos como por exemplo motores, sendo que os parâmetros usados para o motor do Kit Lego Mindstorms NXT estão especificados na Tabela 4.3. O Open Dynamics Engine (ODE) [ode09] é utilizado para a simulação da dinâmica de corpos rígidos. A aparência do *robot* e o seu comportamento são definidos em ficheiros XML (linguagem standard eXtensible Markup Language), estando o ficheiro incluído no Anexo D . O mundo virtual é representado utilizando componentes de GLScene [gl09], sendo estes uma implementação simples de OpenGL [OGL09]. O *robot* baseado no Kit NXT é definido utilizando paralelipípedos e cilindros como mostrado na Figura 4.12. A estes objectos são associadas propriedades físicas utilizando o ODE para determinar as colisões e atritos. O SimTwo permite a substituição dos sólidos por qualquer modelo 3DS dando um aspecto realista ao *robot*, estando este representado na Figura 4.13.

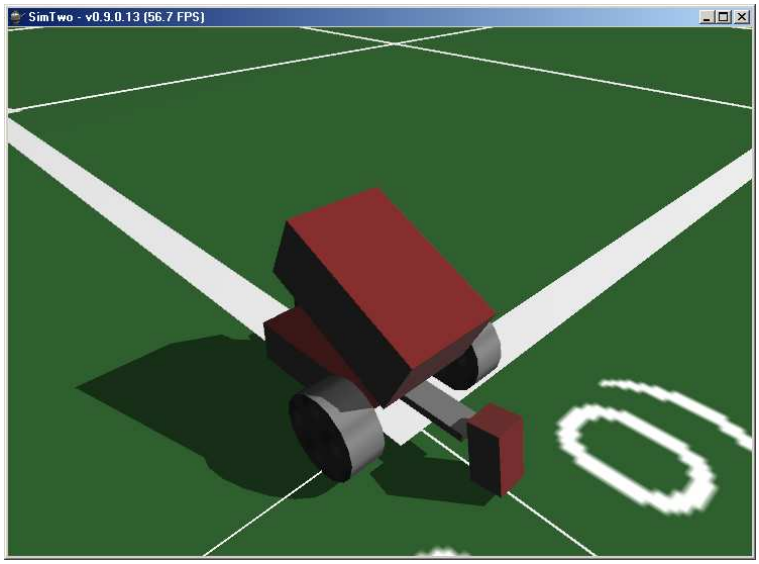

Figura 4.12: *Robot* modelado no SimTwo.

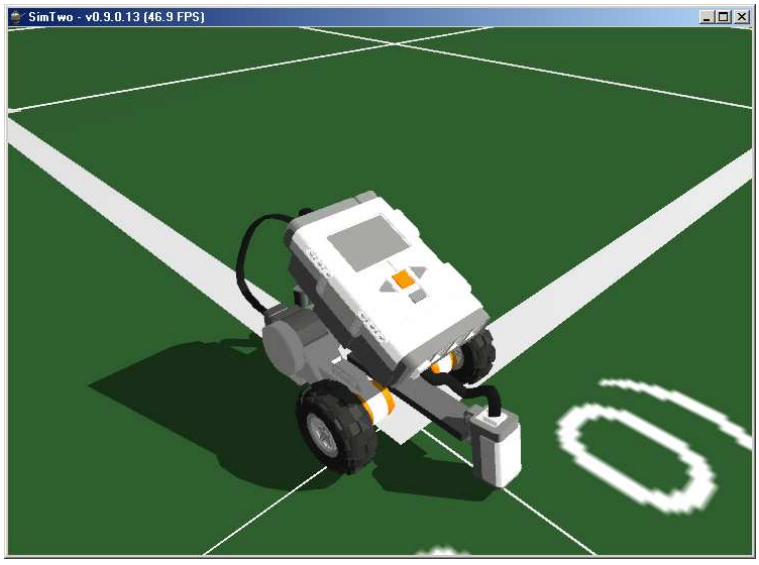

Figura 4.13: *Robot* modelado no SimTwo com modelos 3DS.

4.3.6 Desafio proposto

Por forma a validar a modelação e simulação do protótipo, o *robot* real e o simulado são comparados através de um desafio. O desafio proposto é que o *robot* siga um trajecto baseado num gradiente. O *robot* deve seguir pelo centro do gradiente de cada secção. O trajecto do *robot* real está representado na Figura 4.14.

Figura 4.14: Desafio do *robot*.

Foi aplicado o mesmo controlador para o *robot* simulado e para o real. Dadas as medidas do sensor é possível estimar, a cada amostragem, o erro da distância ao centro do trajecto. O controlador aplicado mantém a velocidade linear constante e a velocidade angular proporcional ao erro da distância. As velocidades escolhidas para cada roda estão descritas nas equações 4.17 e 4.18.

$$V_1(t) = k_v + k_w de \tag{4.17}$$

$$V_2(t) = k_v - k_w de \tag{4.18}$$

Onde k_v e k_w são constante que podem ser ajustadas e de o erro de distância ao centro do trajecto.

Dada a equação (4.3) a velocidade linear do *robot* resulta na equação (4.19) e dada a equação (4.4) a velocidade angular resulta na equação (4.20).

$$v(t) = k_v \tag{4.19}$$

$$w(t) = \frac{k_w 2de}{b} \tag{4.20}$$

Simulação

O controlador implementado no simulador foi realizado utilizando a funcionalidade de *script* embebida. A caracterização do sensor encontra-se embebida na *script* obtendo-se os dados do sensor em relação à posição actual do *sensor* em relação ao mundo. O procedimento que implementa o controlador do *robot* móvel encontra-se no Anexo A. É também possível controlar *robots* utilizando uma aplicação remota que comunica com o simulador através do protocolo UDP. Observou-se que a trajectória é realizada correctamente pela análise do ângulo do *robot*, apresentado na Figura 4.15, sendo este fornecido pelo simulador. O simulador disponibiliza várias variáveis de interesse mas a variável ângulo é a mais representativa para mostrar o resultado do desafio proposto. No gráfico apresentado o eixo dos x está em segundos sendo a contagem iniciada desde que o simulador arranca e no eixo dos y as unidades são radianos.

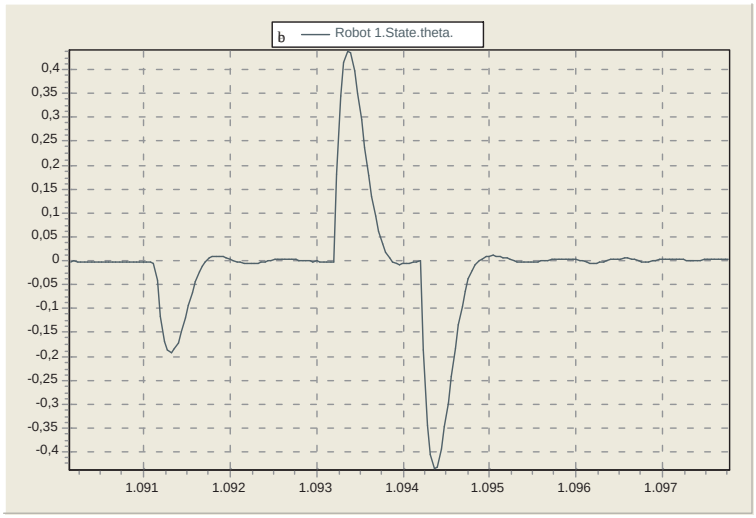

Figura 4.15: Ângulo do *robot*.

Sistema real

A linguagem de programação escolhida para o programa de controlo foi o Lejos. O programa de controlo consiste numa *thread*, sendo os eventos periódicos gerados colocando a *thread* em modo *Sleep* durante o tempo necessário até à próxima amostragem. Após passar este tempo a *thread* acorda realiza todas as operações, calcula o tempo que falta para a próxima amostragem e coloca a *thread* em modo *Sleep* outra vez. A *thread* tem um tempo de amostragem associado, escolhido de maneira a que o controlador seja actualizado o mais rapidamente possível. A escolha do tempo de amostragem foi de 40 ms, cumprindo os requisitos de tempo real propostos. Após vários testes foi observado que o *Brick* demora sempre menos que 40 ms a executar o controlador. A estimativa da posição do *robot* é baseada no cálculo da odometria, o seu registo é feito com base na classe *Datalogger* do Lejos, a qual permite guardar valores na forma de *float* e transmiti-los para um PC via Bluetooth ou USB. O erro da estimativa da localização é cumulativo, dado o cálculo da odometria ser uma medida relativa. Apesar do erro cumulativo é possível observar que a trajectória é realizada correctamente pela análise do gráfico apresentado na Figura 4.16. A arquitectura de controlo utilizada é genérica e pode ser aplicada a diversos desafios de controlo que não sejam robótica móvel. A arquitectura genérica de controlo em Java encontra-se no Anexo B.

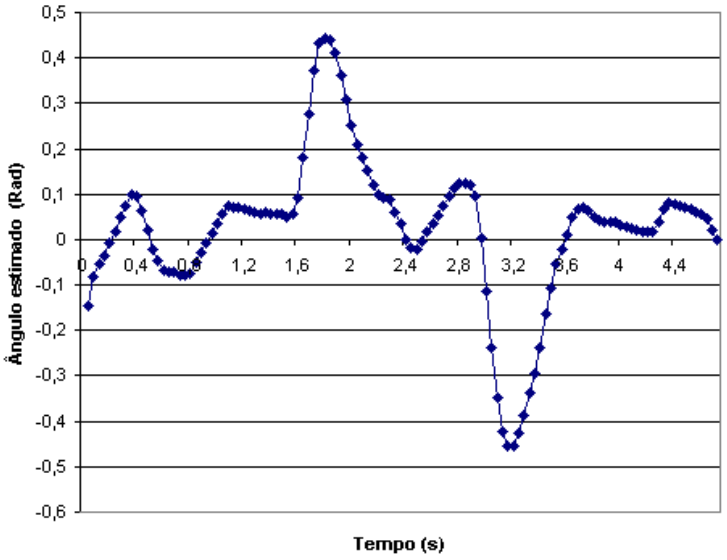

Figura 4.16: Ângulo estimado do *robot*.

4.4 Modelação do sensor de toque

Por forma a modelar o sensor de toque foi necessário registar uma considerável quantidade de dados, para esta tarefa foi utilizado um *robot* industrial para mover o sensor, estando este representado na Figura 4.17. A preparação experimental utilizada está ilustrada na Figura 4.18, o *robot* industrial utilizado foi o HP6 da Motoman com o controlador NX100.

Figura 4.17: Sensor de toque do Kit Lego Mindstorms NXT.

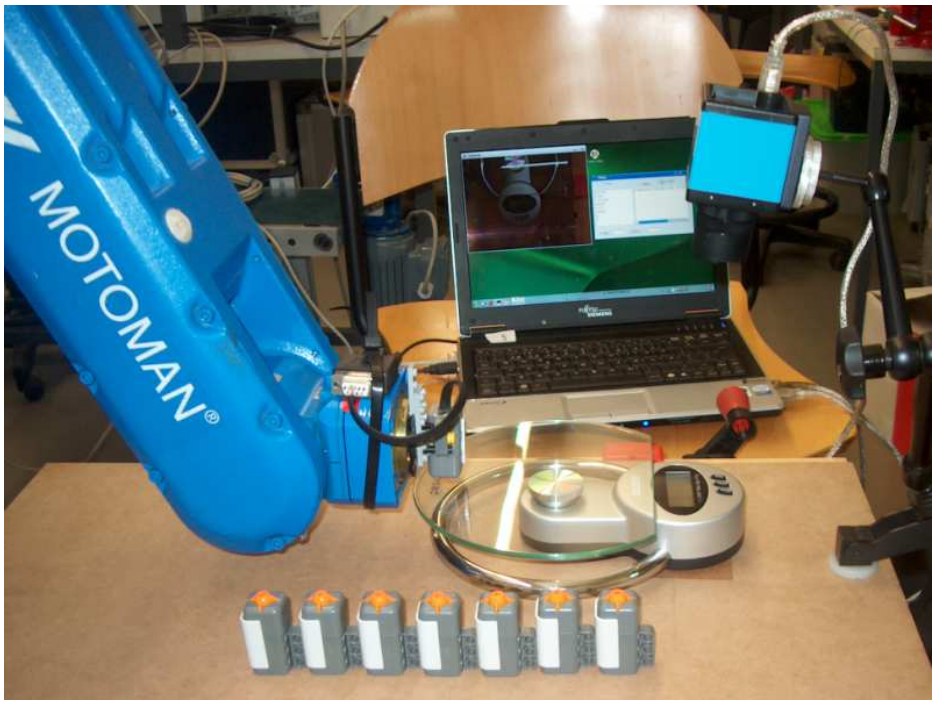

Figura 4.18: *Robot* industrial a mover o sensor de toque.

O controlador do *robot* industrial disponibiliza um modo remoto com a comunicação baseada no protocolo descrito em [Jap04], o qual utiliza uma

conexão *standard* TCP/IP. O sensor de toque é pressionado contra uma balança, por forma a modelar a força e a compressão da mola necessárias para forçar a comutação de estado. Foi observado que existem diferenças significativas de sensor para sensor, mas uma grande repetibilidade para amostras do mesmo sensor. A força aplicada para obter transições de estado está apresentada em gramas força nas Tabelas 4.4 e 4.5.

Amostras	1	2	3	4	5	6	7	8	μ	σ
Sensor 1	57	57	57	57	57	57	57	57	57	0
Sensor 2	62	62	63	63	63	63	63	63	62.6	0.38
Sensor 3	71	71	71	71	71	71	71	71	71	0
Sensor 4	66	67	67	67	67	67	67	67	66.9	0.33
Sensor 5	62	62	62	62	63	62	63	62	62.3	0.43
Sensor 6	71	71	71	71	71	71	71	71	71	0
Sensor 7	50	49	49	50	49	50	50	50	49.6	0.48
Sensor 8	52	52	52	52	52	53	52	52	52.1	0.33

Tabela 4.4: Força aplicada para realizar a transição de *off* para *on*.

Amostras	1	2	3	4	5	6	7	8	μ	σ
Sensor 1	53	53	53	53	53	53	53	53	53	0
Sensor 2	60	60	60	60	60	60	60	60	60	0
Sensor 3	67	67	67	67	67	67	67	68	67.1	0.11
Sensor 4	66	66	66	66	66	66	66	66	66	0
Sensor 5	58	58	58	58	58	58	57	58	57.9	0.11
Sensor 6	67	67	67	67	67	67	67	68	67.1	0.11
Sensor 7	48	48	46	48	48	48	48	48	48.8	0.44
Sensor 8	52	52	52	52	52	52	52	52	52	0

Tabela 4.5: Força aplicada para realizar a transição de *on* para *off*.

A compressão da mola de cada sensor necessária para forçar a transição de estado é apresentada em mm nas Tabelas 4.6 e 4.7. Para a compressão o desvio padrão é somente diferente de zero para a transição de estado de *off* para *on* para o sensor 8 e para a transição de estado de *on* para *off* para o sensor 6, sendo para ambos os casos 1.6E−3.

Observou-se que existe histerese nas transições de estado, como exemplo são mostradas as transições de estado para o sensor 1 na Figura 4.19.

Amostras	1	2	3	4	5	6	7	8
Sensor 1	2.11	2.11	2.11	2.11	2.11	2.11	2.11	2.11
Sensor 2	2.22	2.22	2.22	2.22	2.22	2.22	2.22	2.22
Sensor 3	2.22	2.22	2.22	2.22	2.22	2.22	2.22	2.22
Sensor 4	2.02	2.02	2.02	2.02	2.02	2.02	2.02	2.02
Sensor 5	2.11	2.11	2.11	2.11	2.11	2.11	2.11	2.11
Sensor 6	2.22	2.22	2.22	2.22	2.22	2.22	2.22	2.22
Sensor 7	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
Sensor 8	2.21	2.21	2.21	2.21	2.31	2.21	2.21	2.21

Tabela 4.6: Compressão da mola transição de estado de *off* para *on*.

Amostras	1	2	3	4	5	6	7	8
Sensor 1	1.80	1.80	1.80	1.80	1.80	1.80	1.80	1.80
Sensor 2	2.01	2.01	2.01	2.01	2.01	2.01	2.01	2.01
Sensor 3	1.91	1.91	1.91	1.91	1.91	1.91	1.91	1.91
Sensor 4	1.81	1.81	1.81	1.81	1.81	1.81	1.81	1.81
Sensor 5	1.80	1.80	1.80	1.80	1.80	1.80	1.80	1.80
Sensor 6	2.02	2.00	2.00	2.00	2.00	2.00	1.91	2.00
Sensor 7	1.70	1.70	1.70	1.70	1.70	1.70	1.70	1.70
Sensor 8	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00

Tabela 4.7: Compressão da mola transição de estado de *on* para *off*.

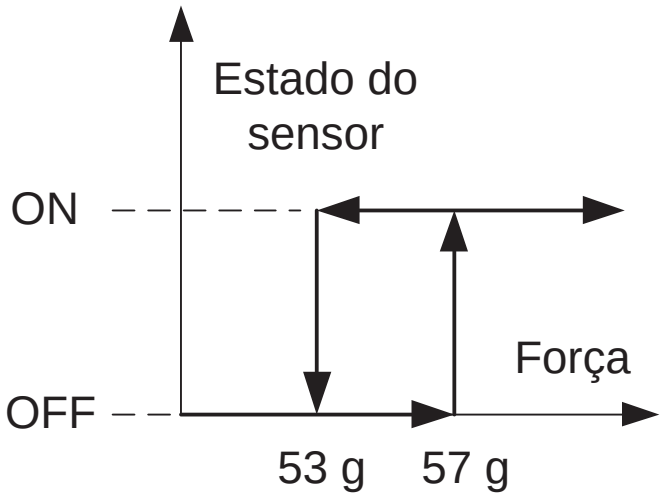

Figura 4.19: Transições de estado sensor 1.

4.5 Modelação do sensor de ultra-sons

O sensor de ultra-sons, mostrado na Figura 4.20, é um dos sensores mais usados na medida de distâncias em robótica móvel e para uma grande variedade de aplicações [BEF96] [Sil95] [ZGA98] [YBW00]. O princípio de funcionamento deste sensor consiste no cálculo de uma distância recorrendo ao tempo de voo de uma onda acústica que se propaga num meio.

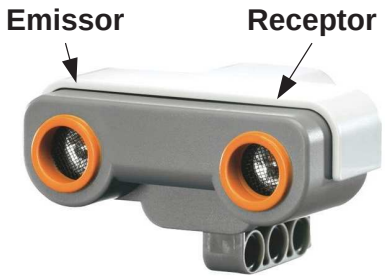

Figura 4.20: Sensor de ultra-sons do Kit Lego Mindstorms.

Por forma a obter os dados necessários para modelar o sensor de ultra-sons foi utilizado um *robot* industrial para posicionar um sensor em diferentes posições, tal como exemplificado na Figura 4.21. O *robot* industrial move o sensor para diferentes distâncias do chão, desde 10 mm até próximo de 1550 mm, com incrementos de 1 mm de passo, tal como mostrado na Figura 4.22.

Observou-se que entre 250 e 1500 mm o sensor é capaz de determinar correctamente as medidas, aumentando o ruído com a distância. A característica do sensor para este intervalo pode ser modelada como uma linha de tendência linear $y = mx + b$, com $m = 1.0055$ e $b = 0.013$. Para distâncias abaixo dos 40 mm o sensor é incapaz de fornecer medidas correctas. Entre os 40 e 250 mm a característica do sensor apresenta para a maior parte das medidas um *offset* de 20 mm. Todas estas diferentes zonas podem ser vistas em pormenor nos gráficos apresentados no Anexo C.

Figura 4.21: *Robot* industrial a posicionar o sensor.

O comportamento do sensor é afectado pelo seu ângulo em relação ao obstáculo. Para modelar esta característica foi feito um teste no qual o sensor é rodado para diferentes ângulos mantendo sempre a distância constante a um obstáculo. As medidas foram feitas para distâncias entre 138 e 1538 mm, sendo espaçadas de 100 mm. Para cada uma das distâncias o ângulo é incrementado de 5 graus até que se deixem de receber ecos válidos. Para cada posição do sensor foram registadas 64 amostras. Como resultado desta experiência foi obtido o padrão do feixe do sensor de ultra-sons es-

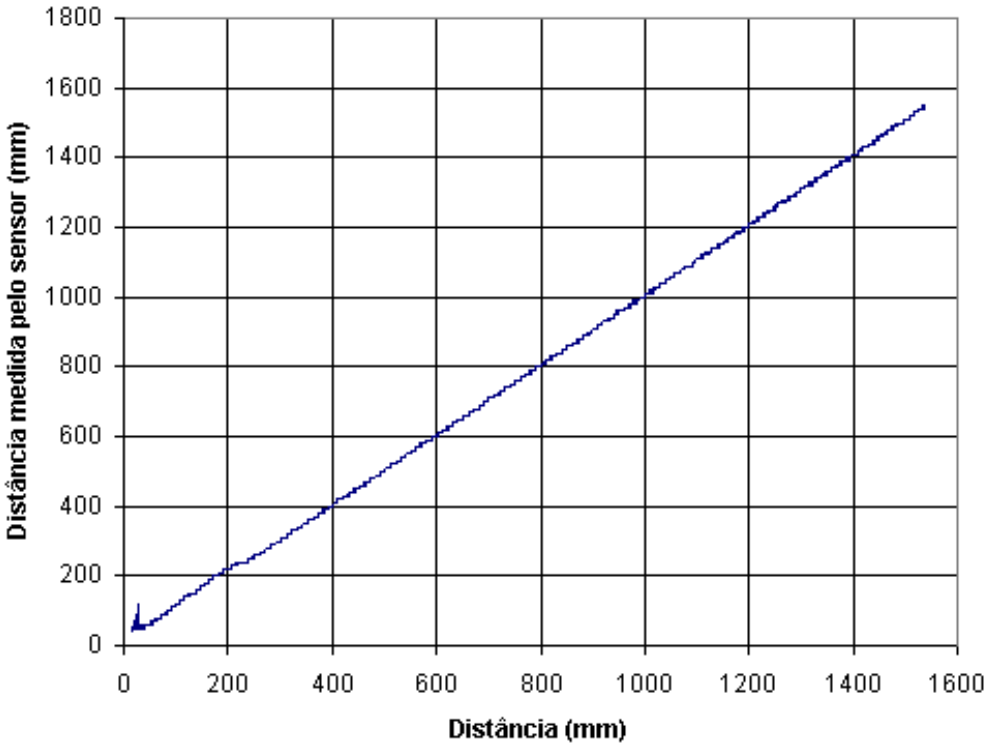

Figura 4.22: Característica do sensor de ultra-sons.

tando apresentado na Figura 4.23. O padrão do feixe do sensor aumenta para as distâncias de 138 a 438 mm, diminuindo a partir deste ponto até à última distância medida. O melhor desempenho do sensor foi registado para a distância de 438 mm, sendo obtidas medidas válidas para um ângulo máximo de 55 graus.

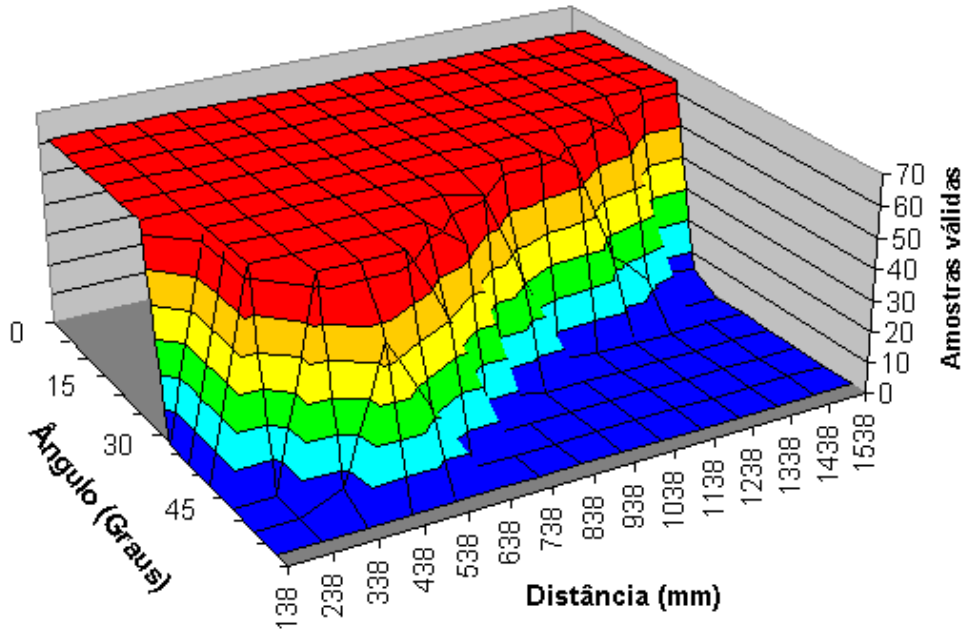

Figura 4.23: Padrão do sensor de ultra-sons.

Capítulo 5

Modelação e simulação da plataforma robotizada

5.1 Introdução

A plataforma utilizada é um *robot* omnidireccional de três rodas, equipado com motores *brushless* e sensores de distância de infravermelhos, sendo apresentado nas Figuras 5.1 e 5.2. Esta plataforma robotizada foi parcialmente prototipada anteriormente a este trabalho, com o objectivo da sua utilização na análise de desempenho de *robots* omnidireccionais de três e quatro rodas [OLi07], tendo sido obtido o seu modelo preciso [OSMC08] [GLOC08]. A opção de utilizar motores *brushless*, neste tipo de aplicações, prende-se com o facto de permitir melhores desempenhos que os tipicamente utilizados motores DC. O *robot* possui sensores de infravermelhos, sendo acrescentados ao protótipo original.

5.2 Sistema de locomoção do protótipo

A configuração omnidireccional surgiu para colmatar as limitações da configuração diferencial, permitindo deslocações em todas as direcções [KNDG02] [GCM06] [RMS+04]. Para garantir a característica de omnidireccionalidade é necessário que as rodas usadas tenham pouco atrito na direcção do veio do motor, sendo o contacto com o chão realizado por rodas passivas. As rodas omnidireccionais eram primariamente usadas apenas para mesas de transporte, estando algumas rodas exemplificadas na Figura 5.3, sendo estas de diferentes tamanhos e com diferentes requisitos de robustez [omn09]. As rodas utilizadas no protótipo desenvolvido não são comerciais (Figura 5.4), foram projectadas e produzidas para a sua utilização nas competições

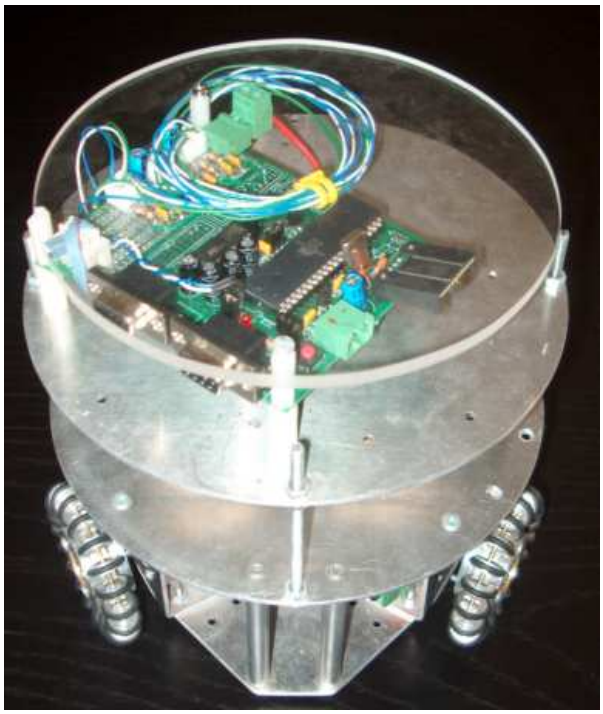

Figura 5.1: Protótipo do *robot* omnidireccional (1).

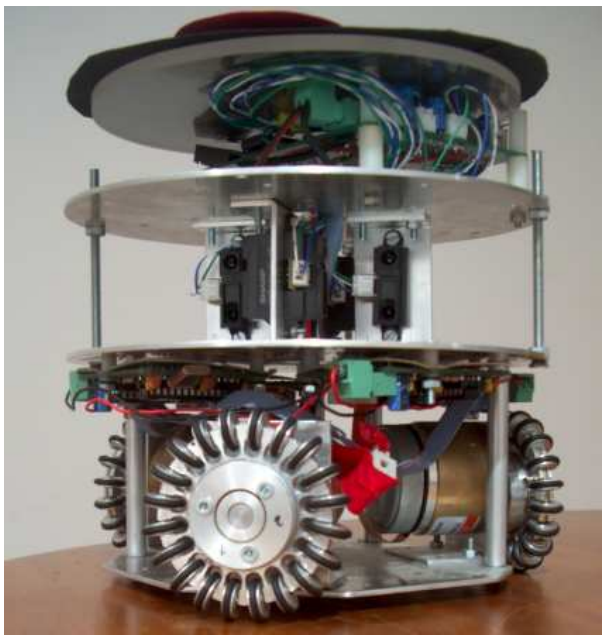

Figura 5.2: Protótipo do *robot* omnidireccional (2).

de futebol robótico na *Small Size League* do Robocup [rob09]. A descrição detalhada do protótipo encontra-se no ficheiro de descrição do *robot* omnidireccional incluído no Anexo D, estando todas as dimensões e massas em unidades SI.

Figura 5.3: Rodas omnidireccionais - Kornylak Corporation.

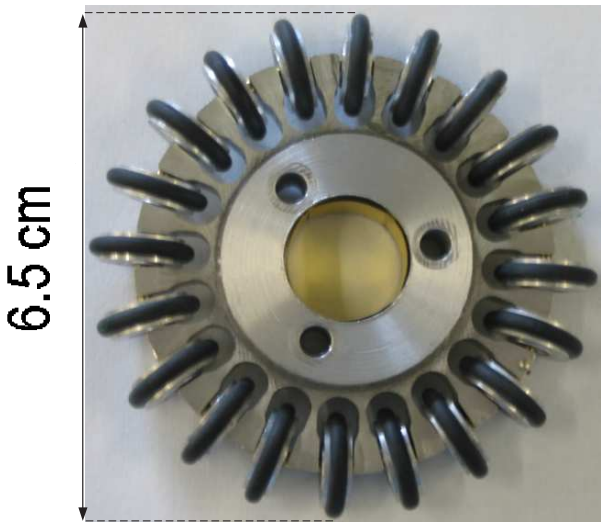

Figura 5.4: Roda omnidireccional da equipa 5DPO.

Como podemos comprovar da observação da geometria de um *robot* omnidireccional com três rodas, representada na Figura 5.5, as velocidades V_x, V_y e w variam com a velocidades lineares V_1, V_2 e V_3, baseando-se em 5.1 [KNDG02].

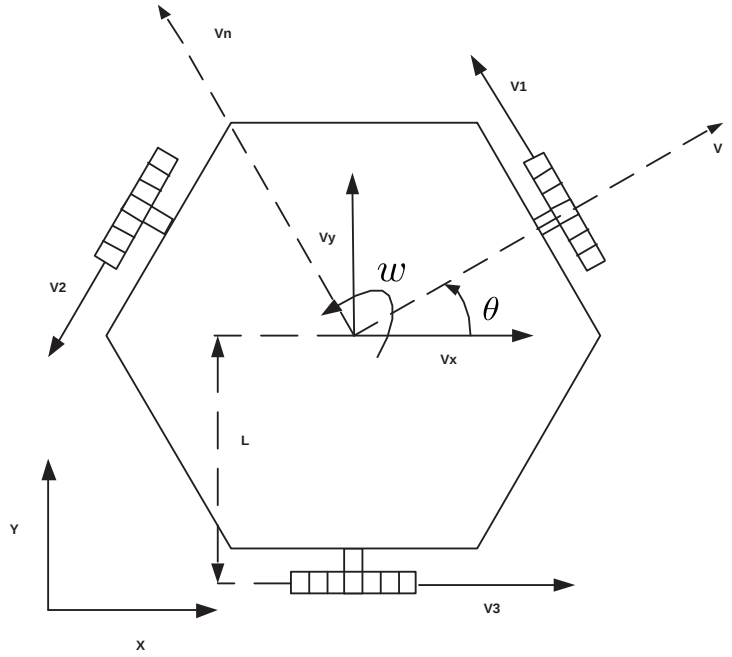

Figura 5.5: Geometria de um *robot* omnidireccional de três rodas.

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = \begin{pmatrix} -sin(\theta) & cos(\theta) & L \\ -sin(\frac{\pi}{3}-\theta) & -cos(\frac{\pi}{3}-\theta) & L \\ sin(\frac{\pi}{3}+\theta) & -cos(\frac{\pi}{3}+\theta) & L \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ w \end{pmatrix} \tag{5.1}$$

As equações da cinemática do *robot* podem ser representadas pelo sistema de equações (5.2) em alternativa ao sistema de equações (5.1).

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & L \\ -sin(\frac{\pi}{3}) & -cos(\frac{\pi}{3}) & L \\ sin(\frac{\pi}{3}) & -cos(\frac{\pi}{3}) & L \end{pmatrix} \begin{pmatrix} V \\ V_n \\ w \end{pmatrix} \tag{5.2}$$

5.3 Modelação e simulação do sensor de distância

Para uma eficiente estimação da localização do *robot* é necessário uma correcta modelação dos seus sensores permitindo ao *robot* uma maior taxa de sucesso na realização das suas missões [BEF96]. A família de sensores de distância de infravermelhos da Sharp é muito popular para aplicações de medida de distâncias no domínio da robótica móvel. Estes sensores apresentam algumas limitações tais como a sua resposta não linear e a inevitável distância mínima de medida. Nesta secção será apresentada a modelação e simulação

do sensor de infravermelhos da Sharp GP2Y0A21YK0F. Em primeiro lugar será apresentada a preparação experimental para a aquisição de dados, posteriormente a modelação do sensor com base na informação recolhida e finalmente a comparação de um sensor real com a sua simulação.

5.3.1 Preparação experimental para aquisição de dados

Para modelar o sensor de distância foi necessário recolher uma quantidade considerável de dados para diferentes distâncias, para esta tarefa foi utilizado o *robot* industrial IRB 1400 da ABB com o controlador S4, como exemplificado na Figura 5.6.

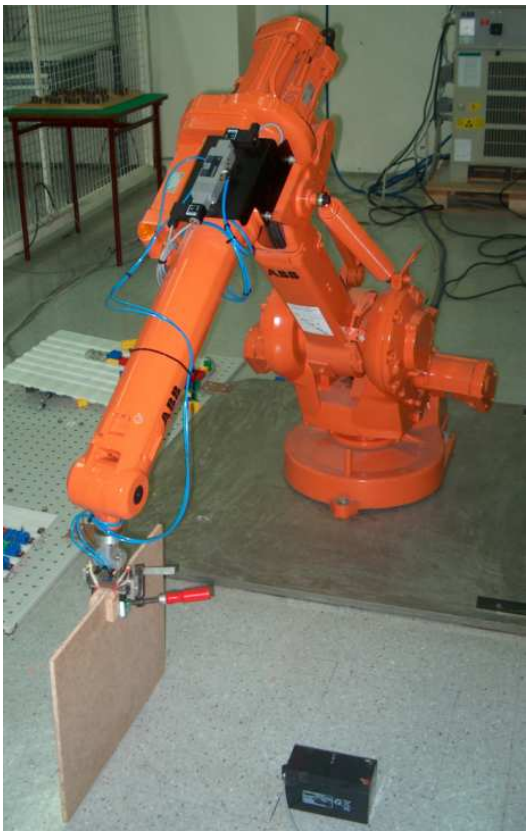

Figura 5.6: IRB 1400 colocando o obstáculo.

A interface com o *robot* é feita com base no ActiveX *RobComm* fornecido pela ABB [ABB96]. Esta interface permite a comunicação com o *robot* industrial com base na seguinte primitiva:

- `S4ProgramNumVarWrite`- permite escrever para um registo do programa do *robot* industrial.

Em baixo é mostrado um fragmento de código do programa do *robot* industrial (escrito em Rapid), o qual permite ao *robot* executar um movimento do obstáculo para a posição de destino.

```
IF move=1 THEN
MoveL Offs(p10,0,-dy,0),vmax,fine,tool0;
move:=0;
ENDIF
```

O programa do *robot* contém dois registos:

- `dy` - distância do sensor ao obstáculo.
- `move` - autorização para o *robot* realizar o movimento.

A indicação de um pedido para ser executado um movimento é dada pelo registo `move`. Se este registo tiver o valor 1 isso significa que foi realizado um pedido, sendo dada autorização para o *robot* realizar um movimento. O ponto p10 é a referência, sendo o ponto no qual o sensor se encontra encostado ao obstáculo. Todos os movimentos são realizados em relação a esta referência aplicando *offsets* segundo o eixo dos y.

Os dados do sensor são adquiridos utilizando o conversor de analógico para digital (ADC) interno do Atmel AVR ATMega8, com 8 bit de precisão. A cada amostragem do *robot* móvel o ADC regista 10 amostras do sensor, as quais são somadas e enviadas para um computador pessoal a fim de serem registadas. Por forma a estudar o ruído foram registados dados de 256 amostragens do *robot* móvel para cada distância.

5.3.2 Modelação e simulação do sensor

Para aquisição de sinal foi usado o conversor de analógico para digital que é disponibilizado internamente no ATMega8. Para aumentar a precisão foi utilizada uma referência externa de tensão de 4.096 Volt, sendo imune a variações da tensão de alimentação. Esta abordagem permite um aumento de precisão por várias razões:

- Caso haja uma pequena variação na tensão de alimentação não vai implicar erros nas leituras do conversor.

- A saída do sensor é imune a pequenas variações da sua tensão de alimentação sendo tipicamente alimentado a 5 Volt mas podendo funcionar entre valores de 4.5 a 5.5 Volt sem que existam variações na sua característica.

- Permite aproximar a tensão máxima de saída do sensor (sensivelmente 3.2 Volt) ao valor de fim de escala (V_{ref}).

A característica do sensor encontra-se representada na Figura 5.7. Foi calculada com base na equação (5.3), onde v é a tensão, si é a amostra ith, n é o número de amostras adquiridas e V_{ref} é a referência externa do ADC.

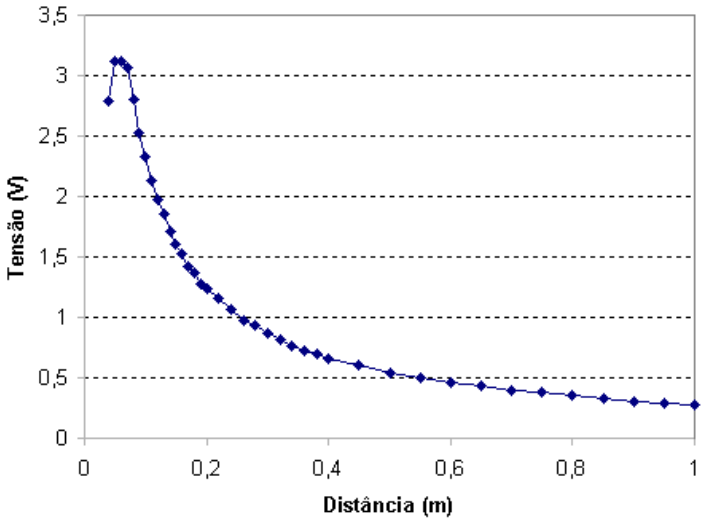

Figura 5.7: Característica do sensor de distância de infravermelhos.

$$v = V_{ref}(\frac{\sum si}{n.255}) \tag{5.3}$$

A relação entre o inverso da tensão e a distância pode ser aproximado a uma recta como exemplificado na Figura 5.8, onde a curva real e a aproximada são mostradas. Os valores utilizados para obter a curva apresentada são da gama de 7 a 100 cm, tendo em conta a imposição de distância mínima obrigatória.

Por forma a obter a distância no *robot*, tendo em conta a aproximação exemplificada na Figura 5.8, pode ser aplicada a equação (5.4), onde d é a distância em m e v e a tensão de saída do sensor, k_1 tem o valor 0.0879 e k_2 tem o valor 3.5204.

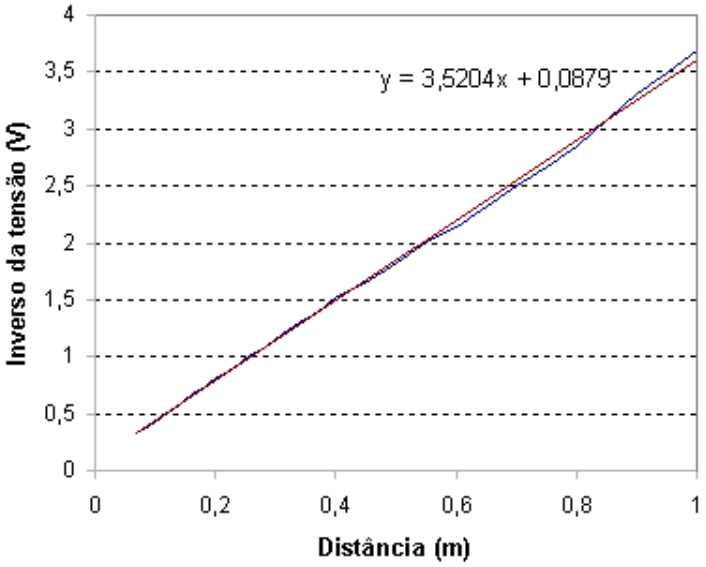

Figura 5.8: Inverso da tensão em função da distância.

$$d = \frac{\frac{1}{v} - k_1}{k_2} \tag{5.4}$$

Uma informação importante a extrair do sensor é a variância da sua resposta, a qual expressa a confiança nas medidas. A variância da tensão foi aproximada a uma recta recorrendo a uma regressão linear, sendo possível observar que aumenta com a distância, tal como exemplificado na Figura 5.9.

Para obter a variância da distância foi feita a aproximação mostrada na equação (5.5), com uma derivada diferente para cada distância, onde m é a derivada da tensão de saída do sensor, apresentada na equação (5.6) e na Figura 5.10. A equação da tensão foi obtida a partir da aproximação apresentada na equação (5.4).

$$v = m.d + b \tag{5.5}$$

$$m = \frac{-k_2}{(d.k_2 + k_1)^2} \tag{5.6}$$

Esta aproximação foi feita de forma a obter a variância da distância relacionada com a conhecida variância da tensão, estando apresentada na Figura 5.11 e na equação (5.7).

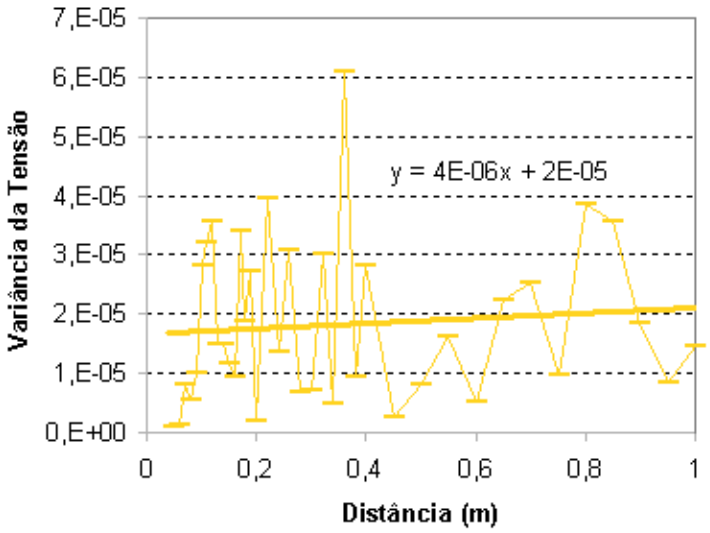

Figura 5.9: Variância da tensão.

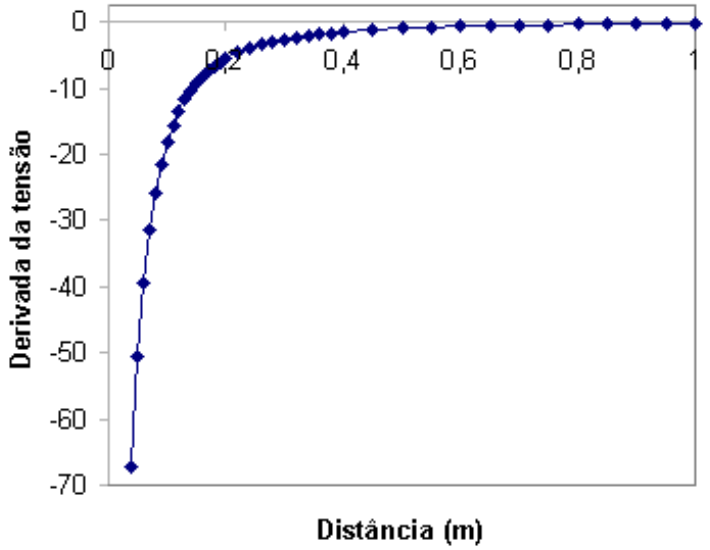

Figura 5.10: Derivada da tensão.

$$Var(d) = \frac{Var(v)}{m^2} \tag{5.7}$$

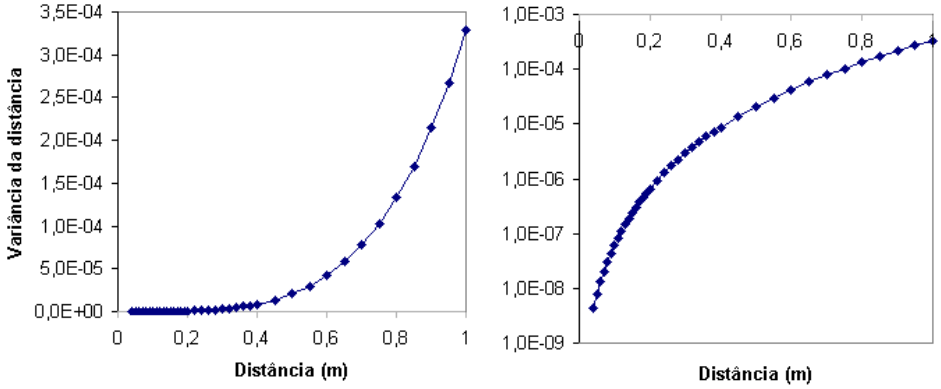

Figura 5.11: Variância da distância.

Um exemplo da leitura de um sensor de distância de infravermelhos para duas distâncias diferentes é apresentado na Figura 5.12. É apresentada a leitura do sensor real e a respectiva simulação com e sem ruído.

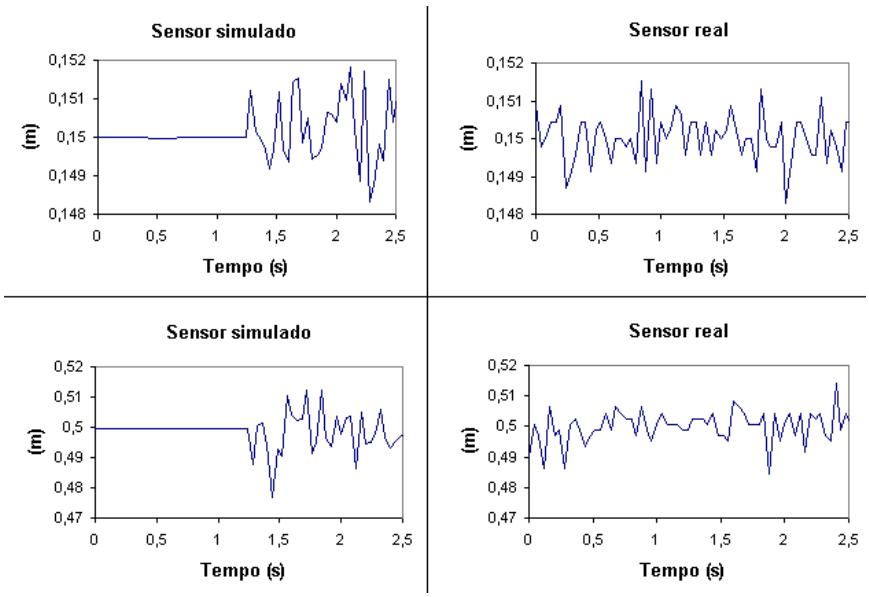

Figura 5.12: Sensores simulado e real.

5.4 Modelação e simulação do actuador

5.4.1 Modelação do actuador

Para a locomoção do *robot* foram usados motores *brushless* DC de três enrolamentos, com a referência "EC 45 flat 30" da Maxon Motor. Este tipo de motor foi escolhido devido à sua elevada performance face aos típicos motores DC. Por forma a aumentar o binário foi acoplado ao motor uma caixa redutora GS45 de 51:10 da Maxon Motor. Para a estimação da velocidade do motor são utilizados os sensores de efeito de Hall disponibilizados pelo próprio motor, não existindo necessidade de usar *encoders* adicionais [OLi07]. O *drive* de potência utilizado é o L6235 [Mar03], sendo este controlado por um micro-controlador Atmega8. A modelação do actuador foi feita conjuntamente com o *drive*, micro-controlador e caixa redutora. Foi realizado um ensaio em malha aberta colocando o motor com uma carga com momento de inércia conhecido. O modelo do actuador foi aproximado a um motor DC, sendo monitorizado no ensaio realizado a corrente consumida, velocidade e valor de PWM (modulação de largura de impulso) aplicado. A preparação experimental realizada para a modelação do actuador encontra-se na Figura 5.13.

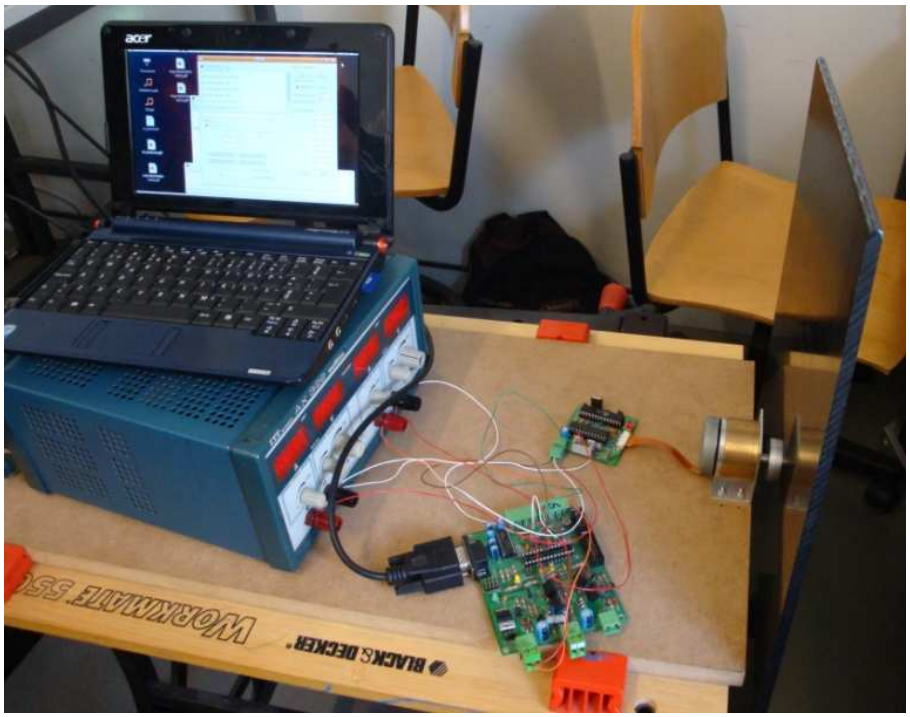

Figura 5.13: Preparação experimental para modelação do actuador.

O modelo utilizado para o motor *brushless* foi o mesmo utilizado para a modelação do motor do Kit Lego Mindstorms apresentado no Capítulo 4, com a diferença de que foi desprezada a indutância equivalente. O binário útil que será aplicado à carga (T_L) é o binário desenvolvido subtraído do binário gerado pelo atrito (T_F), tal como exemplificado na equação (5.8).

$$T_L = T_D - T_F \tag{5.8}$$

O valor de T_F será a soma de duas componentes o atrito estático T_C e o atrito viscoso T_ω. O atrito estático trata-se de uma constante e o atrito viscoso é directamente proporcional à velocidade angular do motor, tal como apresentado na equação (5.9).

$$T_L = T_D - (T_C sign(\omega) + B\omega) \tag{5.9}$$

Estando disponível o valor do binário útil pode ser obtida a aceleração angular a partir da equação (5.10).

$$\dot{\omega} = \frac{T_D - (T_C sign(\omega) + B\omega)}{J} \tag{5.10}$$

Onde J representa o momento de inércia da carga sendo o seu valor $0,0497$ Kg m^2. Após discretização da equação (5.10) obtém-se a equação (5.11), a qual sendo reescrita como equação (5.12) permite a simulação da velocidade angular. Nas equações (5.11) e (5.12) T representa o tempo de amostragem sendo o seu valor 40 ms.

$$\frac{\omega(K+1) - \omega(K)}{T} = \frac{T_D - (T_C sign(\omega(K)) + B\omega)}{J} \tag{5.11}$$

$$\omega(K+1) = \omega(K) + \frac{T(T_D - (T_C sign(\omega(K))) + B\omega)}{J} \tag{5.12}$$

A corrente consumida por um motor DC (i_a), pode ser relacionada com o binário desenvolvido T_D através da equação (5.13).

$$T_D = K_s i_a \tag{5.13}$$

Substituindo o valor do binário T_D na equação 5.12 obtém-se a equação (5.14). Esta equação permite, minimizando a soma do valor absoluto dos erros entre a velocidade estimada e a real, obter os parâmetros K_s, B e T_C.

$$\omega(K+1) = \omega(K) + \frac{T(K_s i_a - (T_C sign(\omega(K))) + B\omega)}{J} \tag{5.14}$$

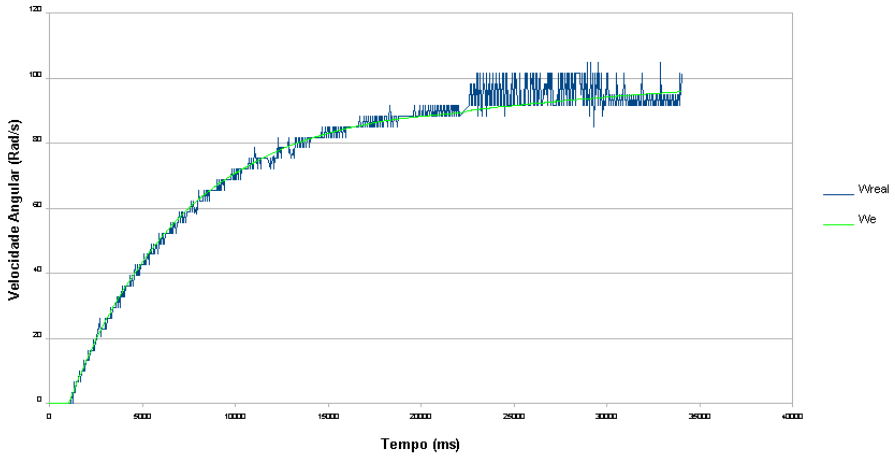

Figura 5.14: Velocidade angular estimada (We) e real ($Wreal$).

As velocidades estimada e real encontram-se no gráfico da Figura 5.14.

Para a obtenção da resistência (R_a) recorreu-se às equações (5.15) e (5.16), onde e representa a força contra-electromotriz e U_a representa a tensão aos terminais do motor, tendo para o ensaio em malha aberta um valor de 3 Volt. Como foi desprezada a indutância interna (L_a) obtém-se a equação (5.17).

$$e = K_s\omega \tag{5.15}$$

$$U_a = e + R_a i_a + L_a \frac{\partial i_a}{\partial t} \tag{5.16}$$

$$i_a = \frac{U_a - K_s\omega}{R_a} \tag{5.17}$$

Minimizando soma do valor absoluto dos erros entre a corrente lida e a corrente estimada obteve-se o parâmetro R_a, sendo apresentadas as correntes estimada e real no gráfico da Figura 5.15. Os parâmetros estimados para o actuador estão mostrados na Tabela 5.1, o valor para o atrito viscoso foi desprezado pois tem um valor muito pequeno face ao atrito estático.

5.4.2 Simulação do actuador

Para simular a experiência realizada foi utilizado o simulador realista SimTwo. O motor pode disponibilizar um binário T_L e a carga tem momento de inércia J o qual será simulado computacionalmente pelo modelo físico criado com base no ODE. Na Figura 5.16 está apresentada a simulação da experiência.

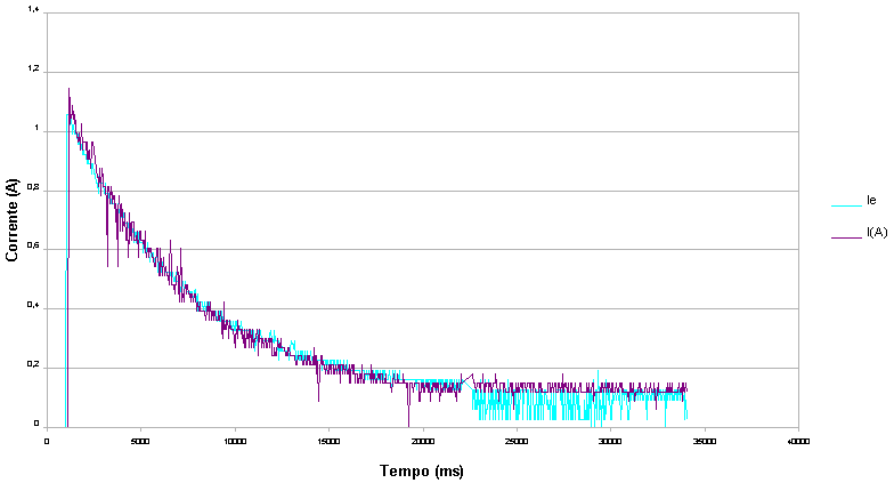

Figura 5.15: Corrente estimada (ie) e real (I).

Parâmetros	Valor unidades SI
K_s	0.0289
T_C	0.00283
B	0
R_a	$2,853$

Tabela 5.1: Parâmetros do actuador.

Com o objectivo de melhor modelar a realidade, na qual as variáveis não podem assumir todos valores, devem ser impostas ao modelo limitações. A primeira limitação é a tensão que pode ser aplicada ao terminais do motor U_a, esta deve ser limitada à tensão das baterias. A corrente também deve ser limitada, estando esta relacionada com o binário pela equação (5.13). A abordagem seguida foi a mesma que a apresentada para a simulação do motor do Kit Lego Mindstorms apresentada no Capítulo 4.

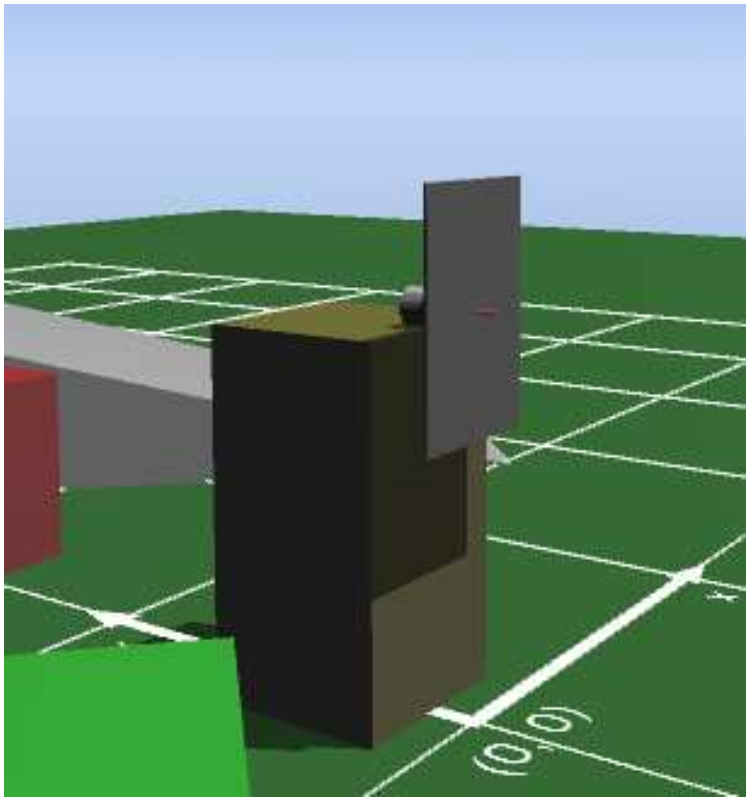

Figura 5.16: Simulação do actuador.

Os gráficos relativos à simulação da corrente e da velocidade angular são apresentados nas Figuras 5.17 e 5.18. Existindo repetibilidade quando comparados com os dados reais, em ambos os gráficos no eixo dos x o tempo é apresentado em segundos, a corrente é apresentada em amperes e a velocidade é apresentada em graus por segundo. Para obter os valores de velocidade apresentados no gráfico da Figura 5.14 os dados da Figura 5.17 devem ser convertidos para rad/s e multiplicados pelo valor 5.1 que é o valor da redução da caixa redutora.

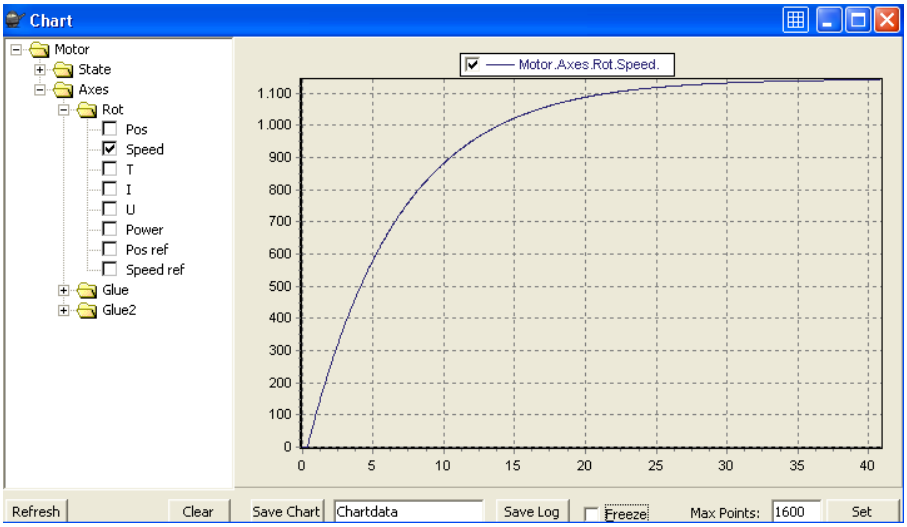

Figura 5.17: Velocidade angular do motor simulado.

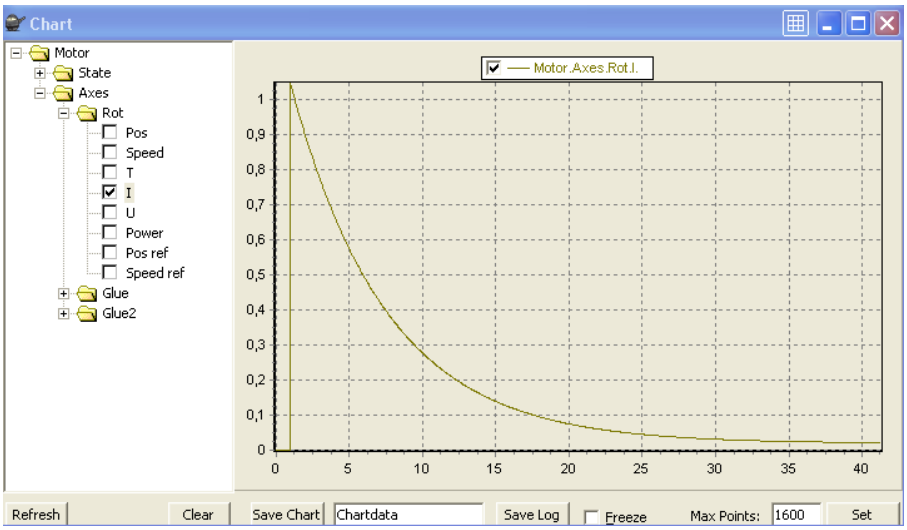

Figura 5.18: Corrente do motor simulado.

Capítulo 6

Sistema de localização e navegação

6.1 Introdução

O *robot*, os sensores e o ambiente são descritos no simulador realista em ficheiros XML. Para a descrição dos obstáculos são especificados o seu tamanho, posição, orientação e cor. A intersecção do feixe de um sensor com um obstáculo retorna uma distância com ruído. Os ficheiros relativos à construção do mundo e do *robot* encontram-se no Anexo D. O *robot* e o seu ambiente são mostrados na Figura 6.1. O ambiente onde navega o *robot* tem apenas paredes com ângulos de 90 graus, sendo uma representação possível do mundo, no qual os nossos edifícios são normalmente bastante artificiais [Col08]. O ambiente é construído com paralelipípedos com ângulos de 90 graus entre si e o *robot* é omnidireccional com três rodas estando equipado com três pares de sensores de infravermelhos de distância.

Para desenvolver programas de controlo, localização e navegação para o *robot* foi utilizada uma aplicação remota, tal como exemplificado na Figura 6.2. O simulador disponibiliza dados dos *encoders*, dados dos sensores de distância com ruído e a posição real do *robot*. Os dados dos sensores de distância e dos *encoders* serão utilizados para estimar a localização e a posição real vai permitir a validação dos algoritmos desenvolvidos. O facto de esta aplicação ser remota apresenta a vantagem de ser independente do simulador, o que torna possível que este código seja mais facilmente migrado para o *robot* real, tal como mostrado na secção 6.4.

Esta aplicação comunica por UDP, sendo o protocolo mais comum em aplicações de tempo real. A sua utilização deve-se, sobretudo, a factores de velocidade no envio dos pacotes de rede. Se tivermos em conta que num

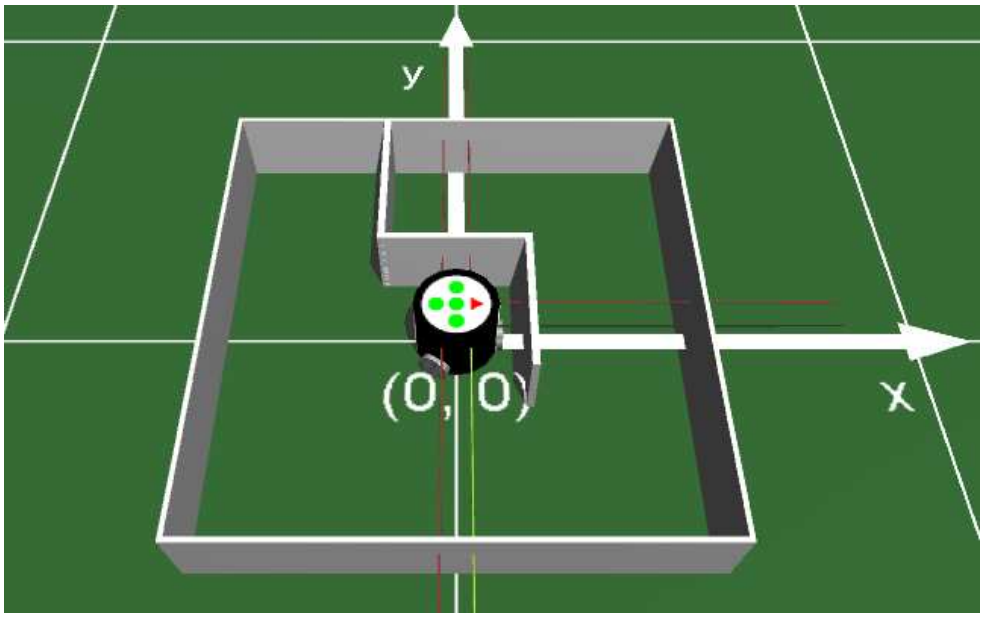

Figura 6.1: Simulação do *robot* no seu ambiente.

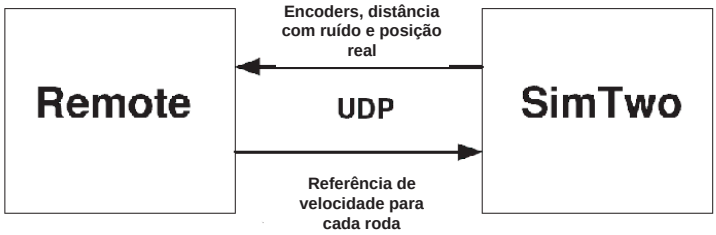

Figura 6.2: Simulador a comunicar com uma aplicação remota.

sistema de tempo real a 25 hz são enviados 25 pacotes de dados pela rede por segundo, não é obrigatoriamente necessário ter a certeza que o pacote é entregue. Como o envio/recepção é constante, na eventualidade de um pacote não ser entregue, o pacote seguinte é entregue num período de tempo reduzido fazendo com que o sistema se torne tolerante a eventuais falhas de comunicação, recuperando rapidamente.

O algoritmo de localização consiste na fusão sensorial da odometria (medidas relativas) e medidas de distância a obstáculos (medidas absolutas), com o objectivo de localizar um *robot* móvel omnidireccional num ambiente estruturado. O algoritmo de localização está dependente da navegação do *robot*, pois este tem que recolher informação do ambiente para se localizar enquanto executa uma missão. O facto de o *robot* ser omnidireccional permite obter mais facilmente, enquanto navega, melhores medidas absolutas quando comparado com outras configurações que têm mais restrições de locomoção.

6.2 Localização

A fusão dos dados de odometria e de sensores de distância foram alcançados aplicando um filtro de Kalman estendido. Este método foi escolhido porque o modelo de locomoção do *robot* é não linear e também porque as distribuições de probabilidade do erro das medidas podem ser aproximadas a distribuições gaussianas.

6.2.1 Medidas relativas

A medida relativa utilizada para a estimação do posicionamento do *robot* móvel foi o cálculo da odometria. O cálculo da odometria consiste na utilização da velocidade de cada roda de maneira a obter a cada instante de amostragem a nova posição do *robot*, tendo como desvantagem o erro de esta estimativa ser cumulativo.

As velocidades lineares e angular V, V_n e w podem ser obtidas reescrevendo o sistema de equações (5.2) como o sistema de equações (6.1),

$$\begin{pmatrix} V \\ V_n \\ w \end{pmatrix} = G \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} \tag{6.1}$$

Onde G é representado pela matriz 6.2.

$$\begin{pmatrix} 0 & \frac{-1}{2\sin(\frac{\pi}{3})} & \frac{1}{2\sin(\frac{\pi}{3})} \\ \frac{1}{1+\cos(\frac{\pi}{3})} & \frac{-1}{2(1+\cos(\frac{\pi}{3}))} & \frac{-1}{2(1+\cos(\frac{\pi}{3}))} \\ \frac{\cos\frac{\pi}{3}}{L(1+\cos(\frac{\pi}{3}))} & \frac{1}{2L(1+\cos(\frac{\pi}{3}))} & \frac{1}{2L(1+\cos(\frac{\pi}{3}))} \end{pmatrix} \tag{6.2}$$

Desta maneira θ pode ser calculado, aplicando-se uma aproximação de primeira ordem, tal como exemplificado na equação (6.3),

$$\theta(k) = \theta(k-1) + \omega T \tag{6.3}$$

onde T é o tempo de amostragem.

Após o cálculo de θ é aplicada uma matriz de rotação, representada em (6.4), de maneira a obter V_x e V_y, tal como exemplificado no sistema de equações (6.5),

$$B = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.4}$$

$$\begin{pmatrix} V_x \\ V_y \\ w \end{pmatrix} = BG \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} \tag{6.5}$$

onde:

$$BG = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \tag{6.6}$$

$$M_{11} = \frac{-\sin(\theta)}{1+\cos(\frac{\pi}{3})} \tag{6.7}$$

$$M_{12} = \frac{-\cos(\theta)}{2\sin(\frac{\pi}{3})} + \frac{\sin(\theta)}{2(1+\cos(\frac{\pi}{3}))} \tag{6.8}$$

$$M_{13} = \frac{\cos(\theta)}{2\sin(\frac{\pi}{3})} + \frac{\sin(\theta)}{2(1+\cos(\frac{\pi}{3}))} \tag{6.9}$$

$$M_{21} = \frac{\cos(\theta)}{1+\sin(\frac{\pi}{3})} \tag{6.10}$$

$$M_{22} = \frac{-\sin(\theta)}{2\sin(\frac{\pi}{3})} + \frac{\cos(\theta)}{2(1+\cos(\frac{\pi}{3}))} \tag{6.11}$$

$$M_{23} = \frac{\sin(\theta)}{2\sin(\frac{\pi}{3})} + \frac{-\cos(\theta)}{2(1+\cos(\frac{\pi}{3}))} \tag{6.12}$$

$$M_{31} = \frac{\cos\frac{\pi}{3}}{L(1+\cos(\frac{\pi}{3}))} \tag{6.13}$$

$$M_{32} = \frac{1}{2L(1+\cos(\frac{\pi}{3}))} \tag{6.14}$$

$$M_{33} = \frac{1}{2L(1+\cos(\frac{\pi}{3}))} \tag{6.15}$$

A estimativa de x e y é calculada aplicando uma aproximação de primeira ordem, tal como exemplificado nas equações (6.16) e (6.17),

$$x(k) = x(k-1) + V_x T \tag{6.16}$$

$$y(k) = y(k-1) + V_y T \tag{6.17}$$

onde T é o tempo de amostragem.

6.2.2 Medidas absolutas

Para obter a localização absoluta do *robot* é necessário realizar a estimativa de x, y e θ. O *robot* dispõe de três pares de sensores de distância de infravermelhos, cada par pode disponibilizar a estimativa de x, y e θ. Para o *robot* realizar a auto-localização é lhe inicialmente fornecida a sua posição correcta e o mapa do ambiente onde navega.

Mapa do ambiente

O *robot* conhece o seu ambiente conhecendo o mapa disponibilizado, sendo este composto por linhas com os seguintes parâmetros caso seja uma linha horizontal:

- posição em y
- x mínimo
- x máximo
- VFB2T - *visible from bottom to top (boolean)*

e os seguintes parâmetros caso seja uma linha vertical:

- posição em x
- y mínimo
- y máximo
- VFL2R - *visible from left to right (boolean)*

O ambiente é auscultado com os sensores de distância caracterizados com os seguintes parâmetros:

- ângulo
- posição em x
- posição em y

Os parâmetros dos sensores estão num referencial local do *robot*.

Informação extraída do mapa

Como é fornecida ao *robot* a cada amostragem (cada 40 ms) uma nova estimativa da sua posição, é possível prever, para cada sensor, a distância e linha do mapa na qual é esperado que esteja a incidir o feixe do sensor. Por forma a obter, a cada amostragem, os parâmetros do sensor num referencial do mundo é necessário em primeiro lugar, realizar um *offset* da posição do sensor com a posição estimada do *robot* e posteriormente realizar a rotação da posição obtida com o ângulo estimado para o *robot*. Finalmente calcula-se o novo ângulo do sensor que é o ângulo do sensor nas coordenadas locais do *robot* somado com o ângulo estimado para a posição do *robot*. Para obter a distância a uma linha e a linha que é esperado que o feixe do sensor esteja a incidir é necessário recorrer à equação (6.18).

$$(x_l, y_l) = (x_s, y_s) + d(u, v) \tag{6.18}$$

onde:

- x_l - posição em x na linha onde o feixe do sensor está a incidir
- y_l - posição em y na linha onde o feixe do sensor está a incidir
- x_s - posição em x do sensor no referencial do mundo

- y_s - posição em y do sensor no referencial do mundo
- d - distância medida
- ϕ - parâmetro do ângulo do sensor somado com o ângulo estimado para o *robot*
- u - $cos(\phi)$
- v - $sin(\phi)$

Os parâmetros enumerados acima encontram-se ilustrados na Figura 6.3.

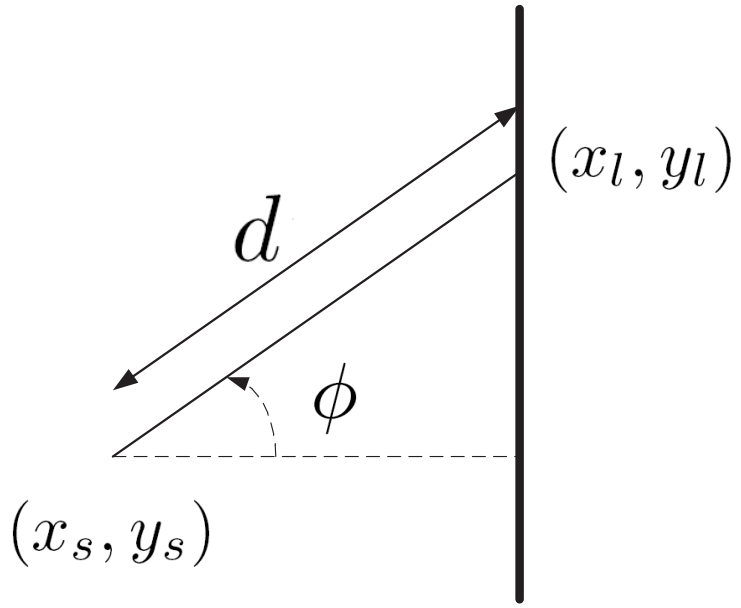

Figura 6.3: Feixe de um sensor a incidir numa parede.

Como exemplo, para uma linha vertical, como x_l é conhecida e é uma constante é possível assumir que:

$$d = \frac{x_l - x_s}{u} \tag{6.19}$$

Sabendo d, a partir da equação (6.19), é possível obter y_l a partir da seguinte equação:

$$y_l = y_s + dv \tag{6.20}$$

O valor y_l é válido apenas se estiver dentro dos limites da linha do mapa. Para uma linha horizontal o processo de cálculo é similar ao exemplo descrito. Repetindo este processo para todas as linhas do mapa (horizontais e verticais), é possível conhecer a distância e a linha que é esperado que o feixe do sensor esteja a incidir, escolhendo o valor positivo válido mais baixo.

Estimação da posição do *robot*

A abordagem proposta para estimar a posição absoluta do *robot* é apenas válida se os feixes do par de sensores estiverem a incidir na mesma parede. Se for esperado que os feixes dos sensores de um par estejam a incidir em paredes diferentes é considerado que a estimativa tem uma variância suficientemente alta para que a sua contribuição seja desprezável quando fundida a informação de várias fontes. Esta metodologia é válida para qualquer dos parâmetros a estimar.

Como exemplo é mostrado o cálculo de estimação da posição de um *robot* em que os feixes de ambos os sensores estão a incidir numa parede vertical, tal como exemplificado na Figura 6.4.

O par de sensores utilizado como exemplo tem os seguintes parâmetros no referencial local do *robot*:

- posição em x : 0 para ambos os sensores
- posição em y : L_s para o sensor 1 e $-L_s$ para o sensor 2
- ângulo : 0 para ambos os sensores

Cálculo do ângulo θ:

Como o feixe dos dois sensores está a incidir na mesma parede é possível estimar o ângulo recorrendo à equação (6.21).

$$\theta = \arctan(\frac{d_1 - d_2}{2L_s}) \tag{6.21}$$

Uma parede horizontal também disponibiliza informação relativamente ao ângulo. Toda a informação proveniente dos três pares de sensores será fundida de forma a obter uma estimação óptima de cada parâmetro.

Se a distância esperada de pelo menos um dos sensores do par diferir de um valor superior a 1.25 cm do valor lido considera-se que não existe confiança na medida e a sua variância será considerada alta.

Cálculo de x e y:

Como o feixe dos dois sensores está a incidir na mesma parede é possível estimar a posição absoluta do *robot* em x recorrendo à equação (6.22).

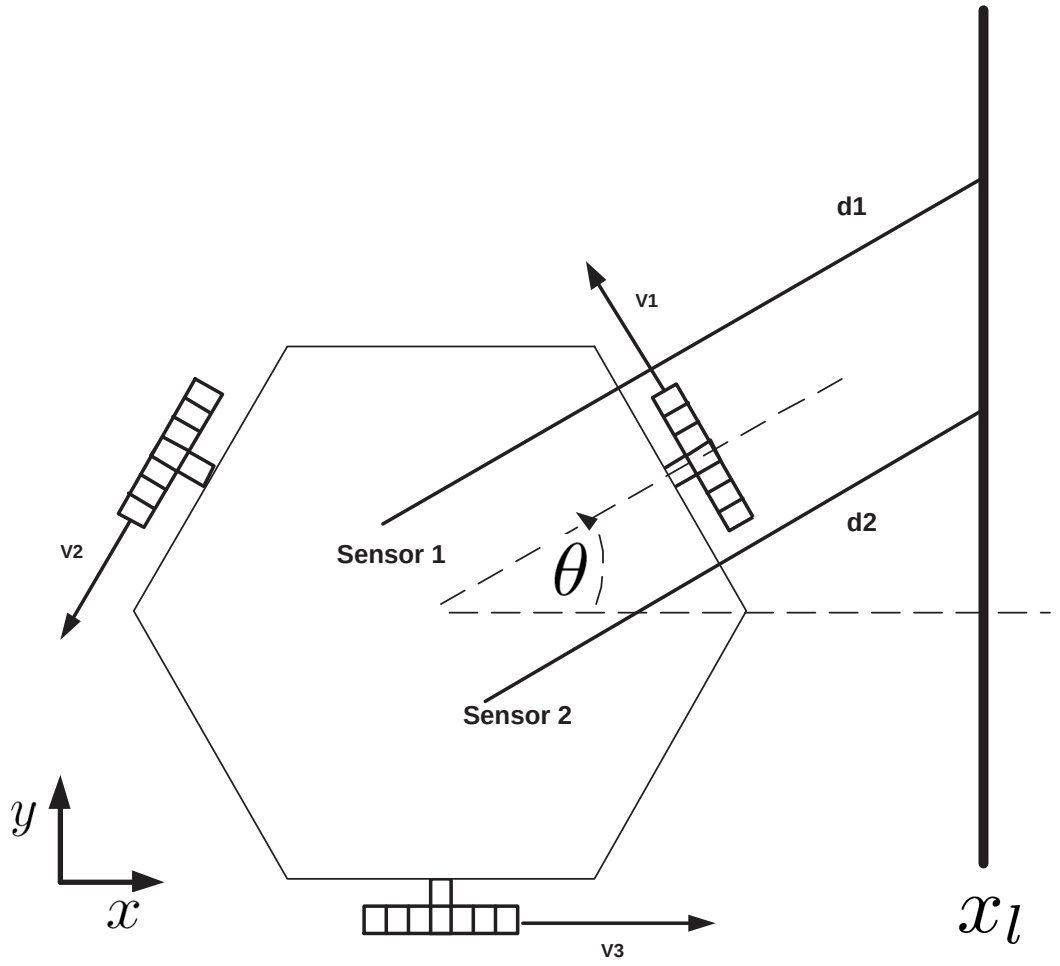

Figura 6.4: Feixes de um par de sensores a incidir numa parede vertical.

$$x = x_l - \frac{d_1 + d_2}{2}\cos(\theta) \tag{6.22}$$

Uma parede vertical não disponibiliza informação em relação ao parâmetro y, sendo considerado que para este parâmetro a variância é suficientemente alta que a sua contribuição, quando se está a fundir informação de várias fontes, se torna desprezável.

Se a distância esperada de pelo menos um dos sensores do par diferir de um valor superior a 5 cm do valor lido considera-se que não existe confiança na medida e a sua variância será considerada alta.

6.2.3 Fusão sensorial

A fusão sensorial dos dados de odometria e medidas de distâncias foi alcançada utilizando-se um filtro de Kalman estendido. Este método foi escolhido devido à não linearidade do modelo do *robot* omnidireccional e porque as distribuições de probabilidade do erro dos sensores usados podem ser aproximadas a distribuições gaussianas.

Com o modelo dinâmico apresentado pelo sistema de equações descrito em (6.5) e considerando que os sinais de controlo mudam somente nos instantes de amostragem, a equação de estado é dada por:

$$\frac{dX(t)}{dt} = f(X(t), u(t_k), t), t\epsilon[t_k, t_{k+1}] \tag{6.23}$$

Onde $u(t) = [V_1 V_2 V_3]^T$, isto é, os dados de odometria são usados como entrada do modelo cinemático. Este estado deve ser linearizado em torno de $t = t_k$, $X(t) = X(t_k)$ e $u(t) = u(t_k)$, resultando em:

$$A^*k = \begin{pmatrix} 0 & 0 & \frac{-\sin(\theta)}{2\sin(\frac{\pi}{3})} + \frac{\cos(\theta)}{2(1+\cos(\frac{\pi}{3}))} \\ 0 & 0 & \frac{\cos(\theta)}{2\sin(\frac{\pi}{3})} + \frac{\sin(\theta)}{2(1+\cos(\frac{\pi}{3}))} \\ 0 & 0 & 0 \end{pmatrix} \tag{6.24}$$

com a matriz de transição de estado:

$$\phi^*(k) = exp(A^*(k)(t_k - t_{k-1})) \tag{6.25}$$

Resultando em:

$$\phi^*k = \begin{pmatrix} 1 & 0 & (\frac{-\sin(\theta)}{2\sin(\frac{\pi}{3})} + \frac{\cos(\theta)}{2(1+\cos(\frac{\pi}{3}))})T \\ 0 & 1 & (\frac{\cos(\theta)}{2\sin(\frac{\pi}{3})} + \frac{\sin(\theta)}{2(1+\cos(\frac{\pi}{3}))})T \\ 0 & 0 & 1 \end{pmatrix} \tag{6.26}$$

Onde T é o tempo de amostragem $(t_k - t_{k-1})$.

Como as observações são obtidas directamente, H^* é a matriz de identidade.

Os passos do filtro de Kalman são os seguintes [WB01]:

1. Estimação do estado no instante $t = t_k, X(k^-)$, sabendo o anterior estado no instante $t = t_{k-1}$, $X(k-1)$ e entrada $u(t_k)$, calculado por integração numérica tal como mostrado na equações (6.3), (6.16) e (6.17).

2. Propagação da covariância do estado

$$P(k^-) = \phi^*(k)P(k-1)\phi^*(k)^T + Q(k) \tag{6.27}$$

   Onde $Q(k)$ é a covariância do ruído e também reflecte o rigor do modelo.

   Como existem medidas o seguinte também se aplica:

3. Ganho do filtro de Kalman

   Como estão disponíveis três pares de sensores, existem três estimativas para cada parâmetro, as quais devem ser fundidas de uma maneira óptima. Para fundir as várias estimativas é necessário aplicar a equação (6.28), por forma a minimizar a variância do parâmetro p pela equação (6.29) [KSO76].

$$p = \frac{p_1 \cdot var(p_1)^{-1} + p_2 \cdot var(p_2)^{-1} + p_3 \cdot var(p_3)^{-1}}{var(p_1)^{-1} + var(p_2)^{-1} + var(p_3)^{-1}} \tag{6.28}$$

$$var(p) = \frac{1}{\frac{1}{var(p_1)} + \frac{1}{var(p_2)} + \frac{1}{var(p_3)}} \tag{6.29}$$

   onde $var(p_i)$ é a variância de cada parâmetro dada por um par de sensores i e sendo p o resultado de uma estimação óptima.

   Todos os parâmetros da posição absoluta do *robot* dependem das distâncias d_1 e d_2. É possível aproximar a variância de cada parâmetro tal como mostrado na equação (6.30) [Rib04a].

$$var(p_i) = \frac{\partial p_i}{\partial d_1}^2 var(d_1) + \frac{\partial p_i}{\partial d_2}^2 var(d_2) \tag{6.30}$$

   As variâncias das distâncias $var(d_1)$ e $var(d_2)$ são dadas pela equação (5.7).

A variância de θ será:

$$var(\theta) = \frac{\partial\theta}{\partial d_1}^2 var(d_1) + \frac{\partial\theta}{\partial d_2}^2 var(d_2) \tag{6.31}$$

onde:

$$\frac{\partial\theta}{\partial d_1} = \frac{L_s}{L_s^2 + (d_1 - d_2)^2} \tag{6.32}$$

$$\frac{\partial\theta}{\partial d_2} = \frac{-L_s}{L_s^2 + (d_1 - d_2)^2} \tag{6.33}$$

A variância de x será:

$$var(x) = \frac{\partial x}{\partial d_1}^2 var(d_1) + \frac{\partial x}{\partial d_2}^2 var(d_2) \tag{6.34}$$

onde:

$$\frac{\partial x}{\partial d_1} = a + b \tag{6.35}$$

$$\frac{\partial x}{\partial d_2} = a + c \tag{6.36}$$

$$a = -0.5\cos(\arctan(\frac{d_1 - d_2}{L_s})) \tag{6.37}$$

$$b = (d_2 + d_1)\sin(\arctan(\frac{d_1 - d_2}{L_s}))\frac{-L_s}{L_s^2 + (d_1 - d_2)^2} \tag{6.38}$$

$$c = (d_2 + d_1)\sin(\arctan(\frac{d_1 - d_2}{L_s}))\frac{L_s}{L_s^2 + (d_1 - d_2)^2} \tag{6.39}$$

A equação da variância para o parâmetro y é igual à obtida para x, pois a equação para o seu cálculo é idêntica. Deste modo a variância será igual para as mesmas distâncias medidas pelos sensores.

Posteriormente procede-se ao cálculo do ganho do filtro de Kalman:

$$K(k) = P(k^-)H^*(k)^T(H^*(k)P(k^-)H^*(k)^T + R(k))^{-1} \tag{6.40}$$

Onde $R(k)$ é a matriz da covariância do ruído das medidas.

4. Actualização da covariância do estado

$$P(k) = (I - K(k)H^*(k))P(k^-) \tag{6.41}$$

5. Actualização do estado

$$X(k) = X(k^-) + K(k)(z(k) - h(X(k^-, 0))) \tag{6.42}$$

onde z(k) é vector das medidas e $h(X(k^-, 0))$ é $X(k^-)$.

6.2.4 Validação do algoritmo de localização

Por forma a validar o algoritmo de localização proposto foi feita uma trajectória do *robot*, o qual navega no ambiente mostrado na Figura 6.1. A trajectória realizada encontra-se na Figura 6.5. O erro e a variância da estimativa de localização podem ser visualizadas nos gráficos da Figura 6.6. Pode-se observar que o algoritmo de localização é robusto apesar da dependência de uma localização prévia que não se afaste muito da realidade. Também para este algoritmo foi corrigida uma possível fonte de erro na localização aquando os sensores realizam transições entre paredes, existindo diferenças significativas entre o valor esperado e a distância efectivamente lida. Este fenómeno pode observar-se nas Figuras 6.7, 6.8 e 6.9, as quais representam as paredes em que se espera que o sensor esteja incidir e a diferença entre o valor esperado e o medido por um sensor de distância. A correcção desta possível fonte de erro consistiu em não considerar válidas as medidas de um par de sensores sempre que seja esperado que pelo menos um deles esteja a incidir a menos de 5 cm da transição entre duas paredes. Na maior parte dos casos os dados da localização pioram, pois tem-se acesso a menos medidas, mas usando a correcção mencionada o *robot* nunca se perdeu para 100 corridas realizadas, enquanto que sem a correcção perdeu-se 12 vezes, devido a uma injecção de erro no ângulo demasiado grande, fazendo com que o filtro de Kalman nunca mais convergesse para a posição real.

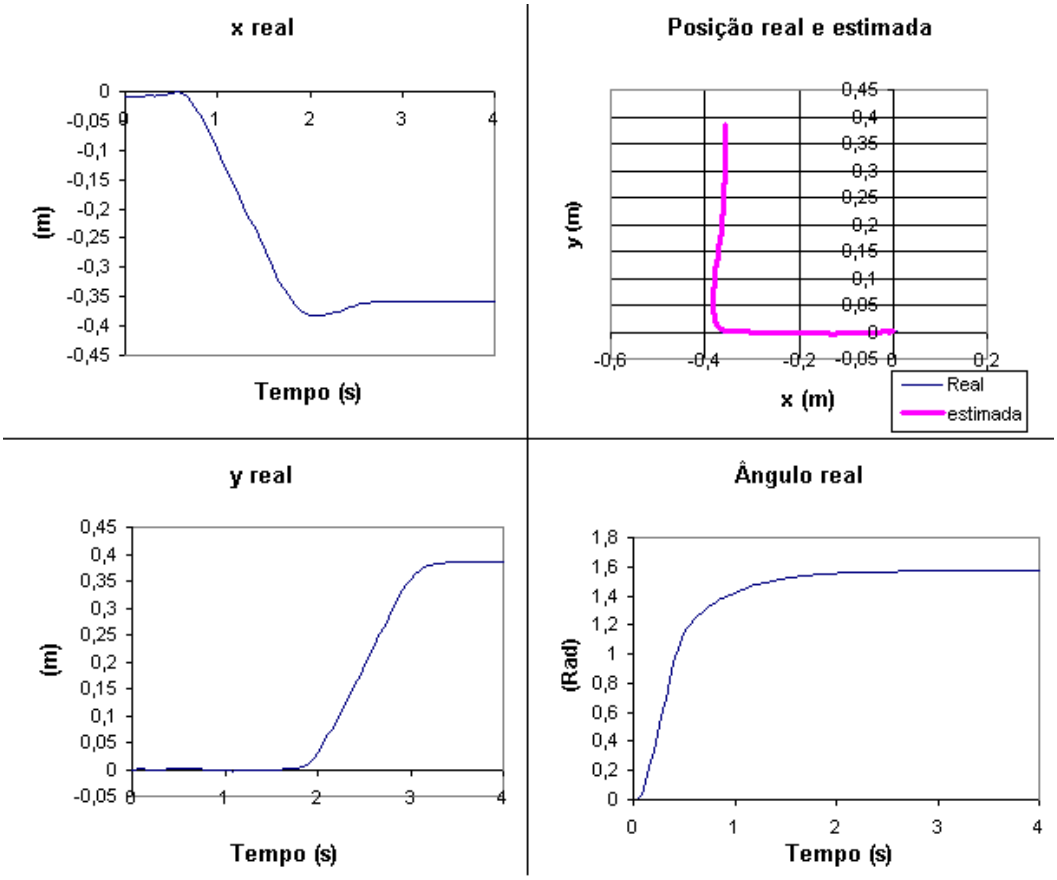

Figura 6.5: Trajectória realizada pelo *robot*.

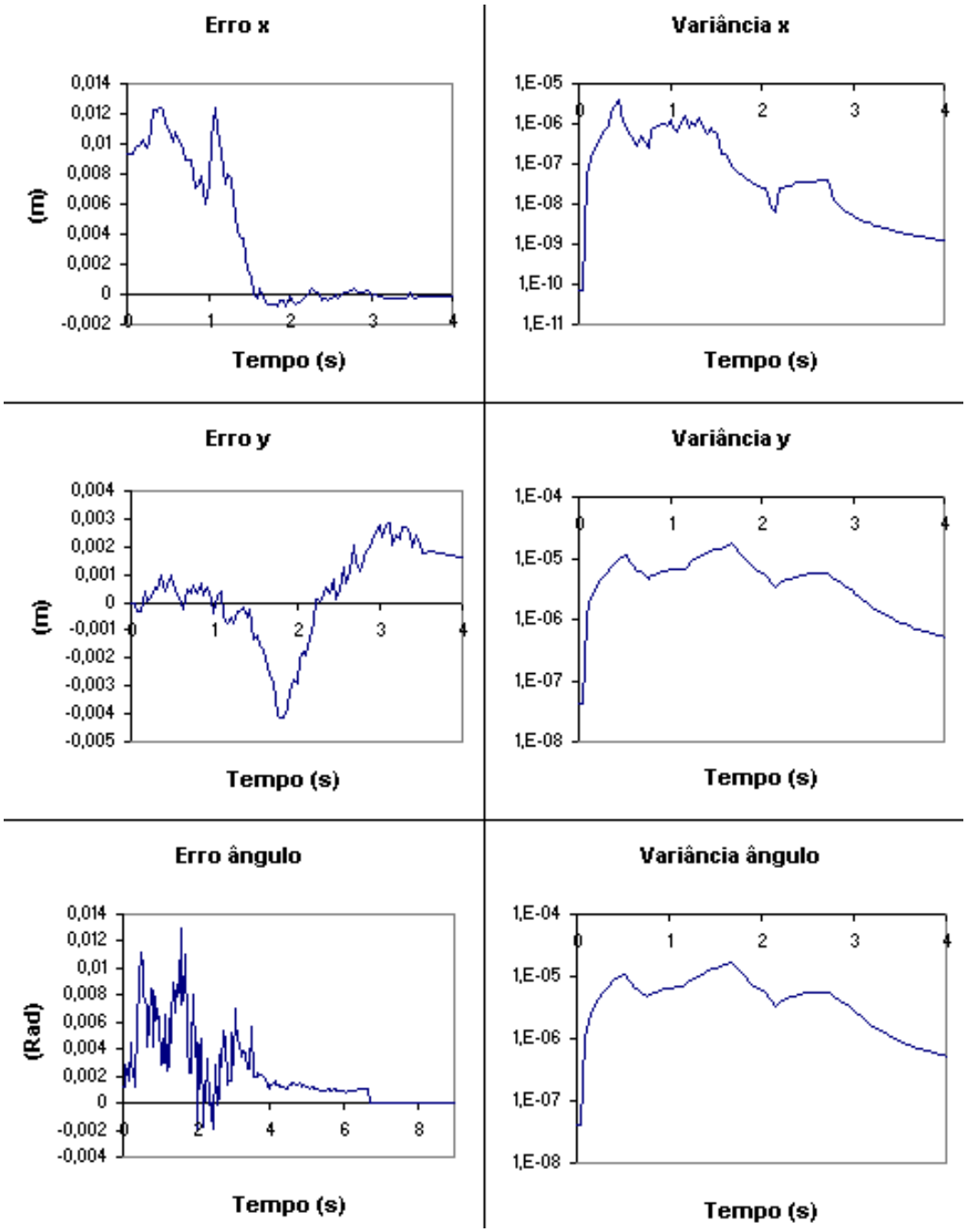

Figura 6.6: Erro e variância da estimativa de localização.

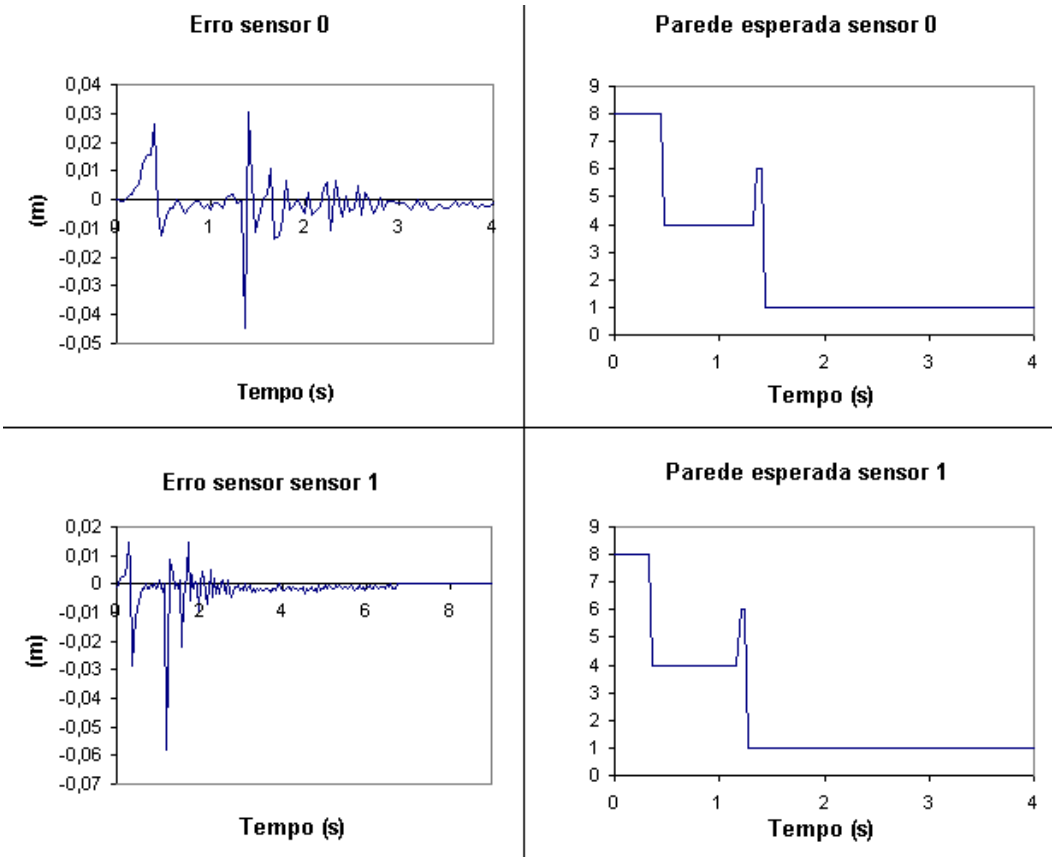

Figura 6.7: Erros do par de sensores 0 e respectivas paredes esperadas.

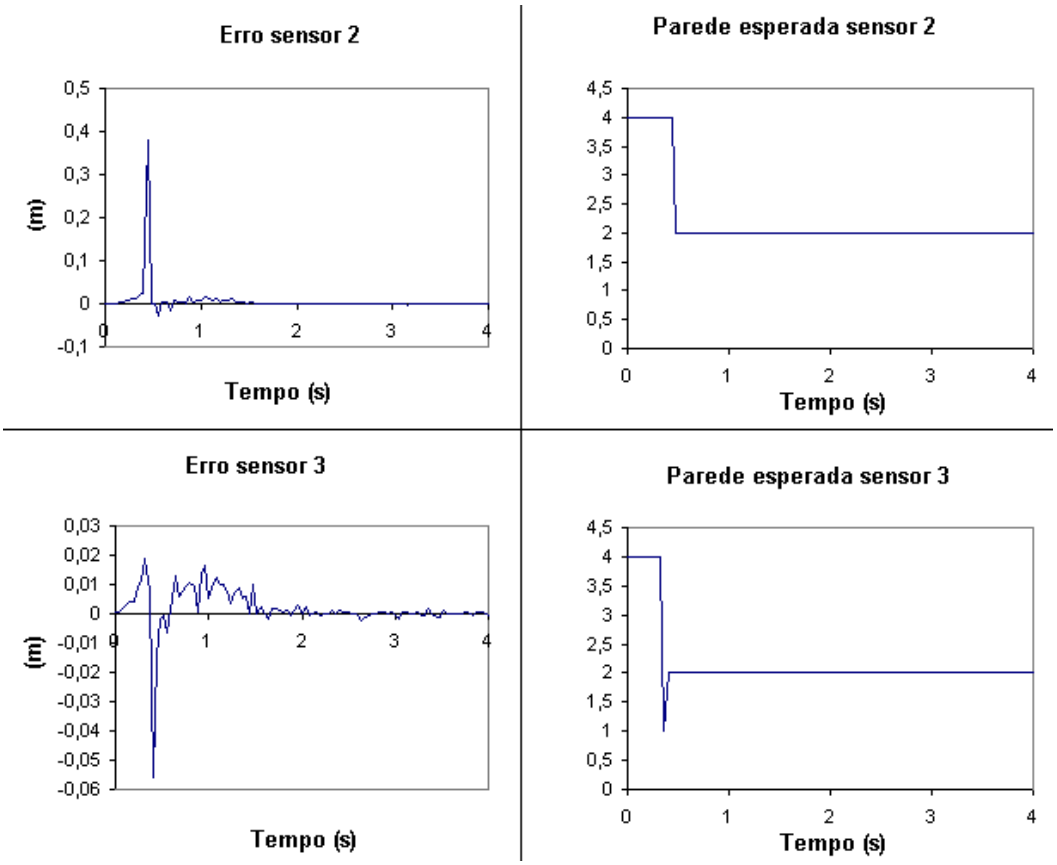

Figura 6.8: Erros do par de sensores 1 e respectivas paredes esperadas.

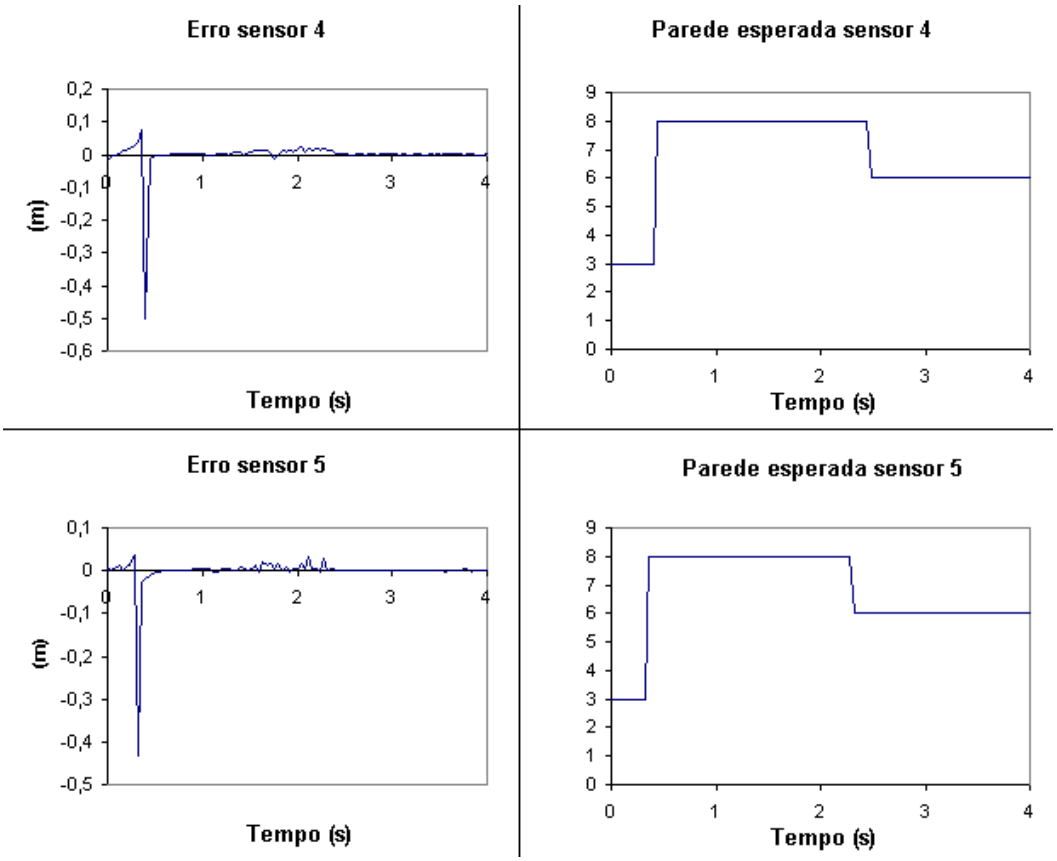

Figura 6.9: Erros do par de sensores 2 e respectivas paredes esperadas.

6.3 Navegação

O *robots* omnidireccionais são amplamente utilizados nas competições de futebol robótico [rob09], permitindo movimentos em todas as direcções, sendo que a mobilidade extra se torna uma vantagem importante. As características de locomoção dos *robots* omnidireccionais podem ser também muito úteis para a navegação em ambientes estruturados. Os *robots* omnidireccionais podem executar missões mais rápido porque as manobras podem ser simplificadas e também podem auscultar o ambiente com mais facilidade enquanto navegam. O erro e a variância da estimativa de cada parâmetro da localização podem ser consideravelmente reduzidos, quando comparado com os resultados obtidos para um *robot* diferencial.

A abordagem proposta tira partido do facto de o *robot* omnidireccional se poder mover com velocidades lineares e angular independentes, com o objectivo de os sensores de distância estarem sempre virados com a orientação que lhes permita uma melhor estimativa dos parâmetros de localização. É possível concluir com base na observação da trajectória apresentada na Figura 6.10 que o *robot* omnidireccional por forma a navegar e se localizar enquanto navega, tem a necessidade de executar apenas manobras simples, tendo que alterar o seu ângulo somente uma vez. As trajectórias são predefinidas, tornando-se o desempenho do *robot* dependente da localização, já que os objectivos do *robot* são localizações x, y e θ. O controlador do *robot* é um controlador de posição proporcional com saturação. A cada nova localização é calculado um vector de posição que aponta para o objectivo a alcançar, este vector é convertido em vector de velocidade, depois calcula-se a velocidade e a velocidade normal do *robot*, tendo em conta os seus valores máximos admissíveis. Finalmente tendo também em conta a velocidade angular que o *robot* deve assumir calcula-se a velocidade de referência de cada roda pela equação (5.1). O ambiente do *robot* é limitado a um quadrado com um metro de lado, a posição origem é o centro do quadrado, tal como mostrado na Figura 6.1. Foi testada para um *robot* diferencial a navegação no mesmo ambiente, sendo observado que para velocidades semelhantes, com o objectivo de executar a mesma missão, o *robot* gastou mais tempo, porque teve que executar mais manobras devido às suas restrições de locomoção.

Pode ser observado a partir da Figura 6.11 de *robot* diferencial tem que mudar várias vezes o seu ângulo por forma a executar o mesmo trajecto que o *robot* omnidireccional. Também pode ser observado nas Figuras 6.13 e 6.12 que para a configuração omnidireccional o erro das estimativas e as variâncias dos parâmetros de localização são menores devido ao facto de o *robot* poder seguir uma trajectória com qualquer ângulo, tendo sempre acesso às melhores medidas possíveis.

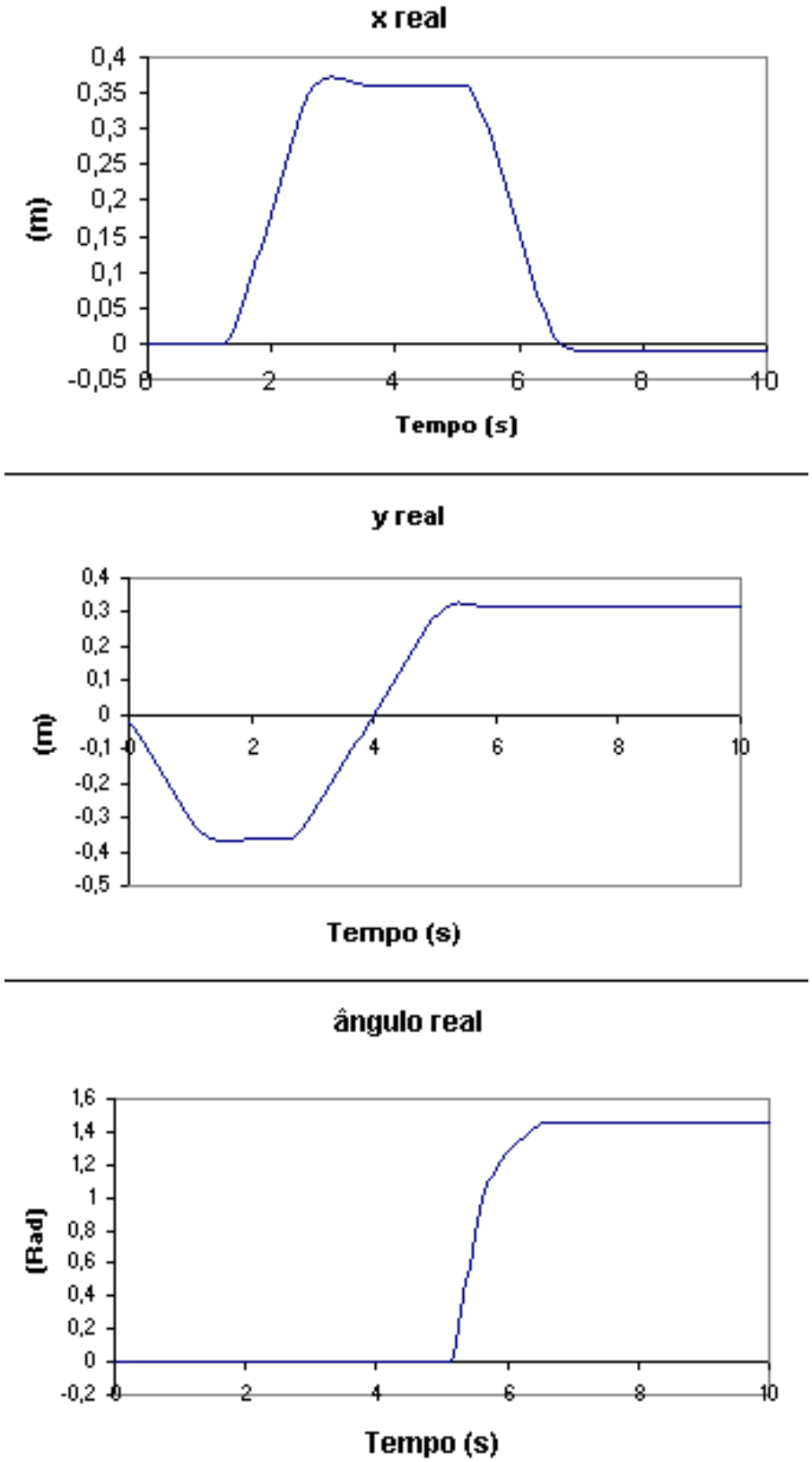

Figura 6.10: Trajectória do *robot* omnidireccional.

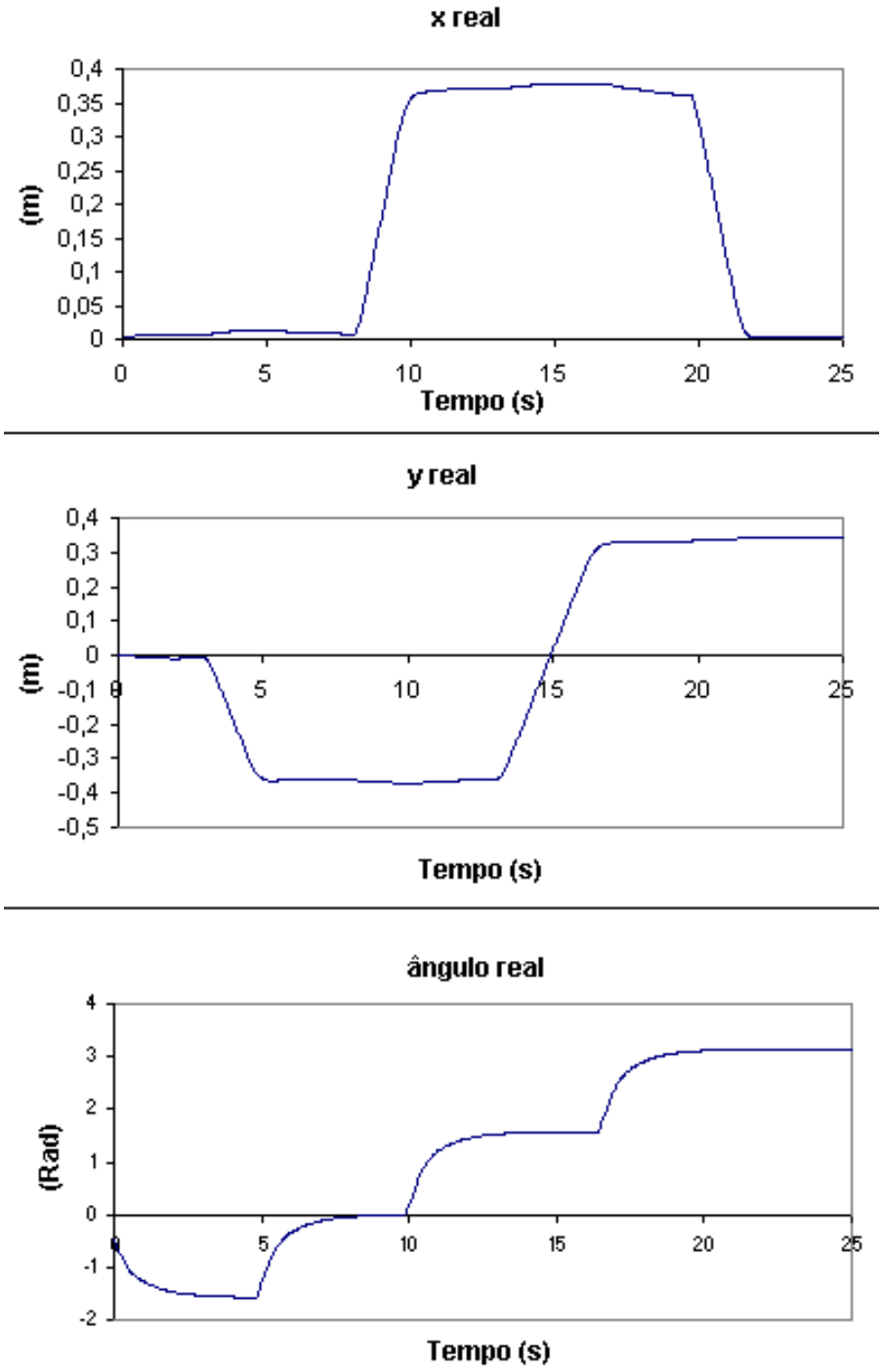

Figura 6.11: Trajectória do *robot* diferencial.

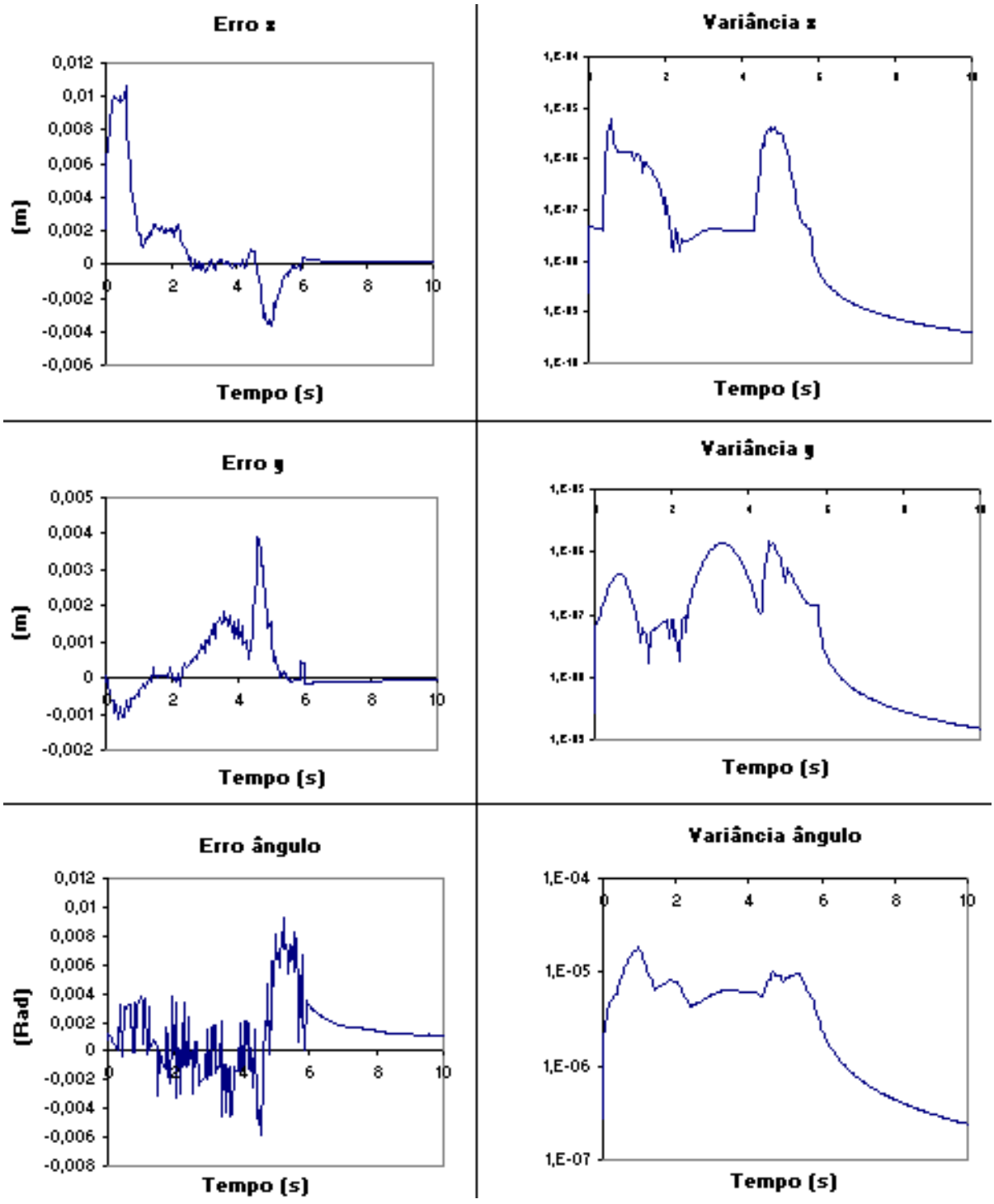

Figura 6.12: Variância e erro da estimativa relativa ao *robot* omnidireccional.

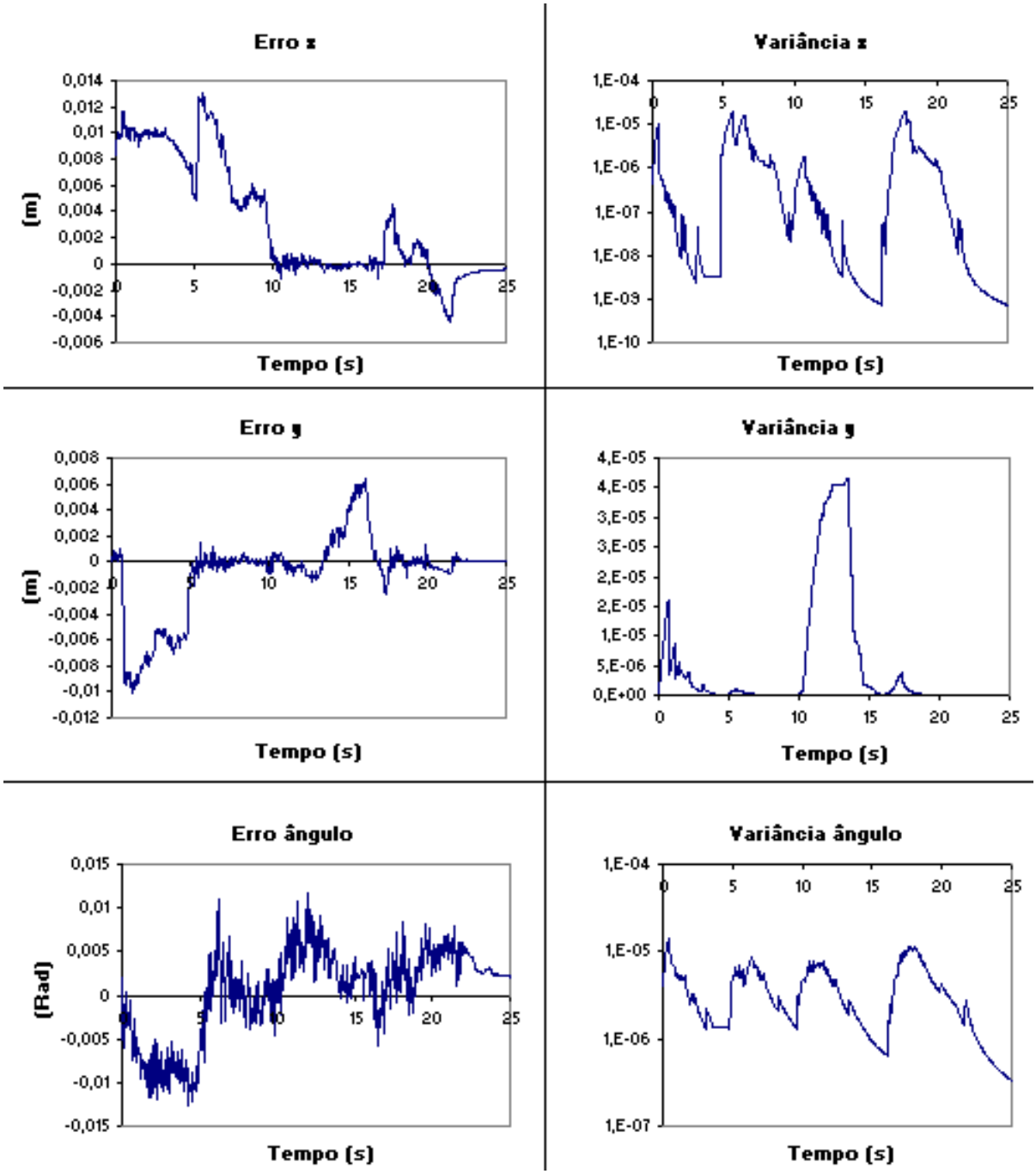

Figura 6.13: Variância e erro da estimativa relativa ao *robot* diferencial.

6.4 Migração de código para o *robot* real

A presente secção descreve a migração do *software* de localização e navegação do *robot* realizado com base no simulador realista para o *robot* real [GLMC09a]. O *software* desenvolvido consiste na localização e navegação de um *robot* móvel omnidireccional num ambiente estruturado (Figura 6.14). O *software* desenvolvido no simulador encontra-se na subsecção 6.3.

A arquitectura usada para migrar o código desenvolvido no simulador para o *robot* real está apresentada na Figura 6.15. A aplicação Remote é partilhada com a simulação para que o código gerado seja aplicado directamente ao *robot* real, sem que sejam realizadas quaisquer alterações. A posição real do *robot* é fornecida à aplicação Gate a uma frequência de 25 hz pelo sistema de visão global descrito no Capítulo 3. O *loop* de controlo é iniciado pelo *robot* quando envia para a aplicação Gate (via RS232) o valor dos *encoders* e os dados dos sensores de distância. Após a recepção dos dados do *robot* a aplicação Gate envia-os conjuntamente com a posição real do *robot* para a aplicação Remote via UDP. A aplicação Remote executa os algoritmos de localização e navegação e retorna para a aplicação Gate a velocidade de referência de cada roda do *robot*. Por fim a velocidade de referência de cada roda é enviada para o *robot* real via RS232. O *loop* de controlo é realizado a uma frequência de 25 hz. A trajectória executada pelo *robot* real, mostrada na Figura 6.16, é similar à trajectória simulada, fazendo com que a simulação se torne útil, pois permitiu reduzir consideravelmente o tempo de desenvolvimento de *software* para o *robot*. O erro da estimativa e a variância podem ser visualizados na Figura 6.17. Na Figura 6.15 o *robot* é apresentado com marcadores com disposições e em número diferente do sistema descrito no Capítulo 3. O *robot* tal como é apresentado está preparado para o sistema de localização em tempo real para múltiplos *robots* utilizado pela equipa 5DPO na *Small Size League* [MSC01]. O facto de a aplicação receber os dados por rede permite a utilização de diferentes sistemas externos de validação, apesar de o único sistema de validação utilizado estar descrito no Capítulo 3.

Figura 6.14: *Robot* no seu ambiente.

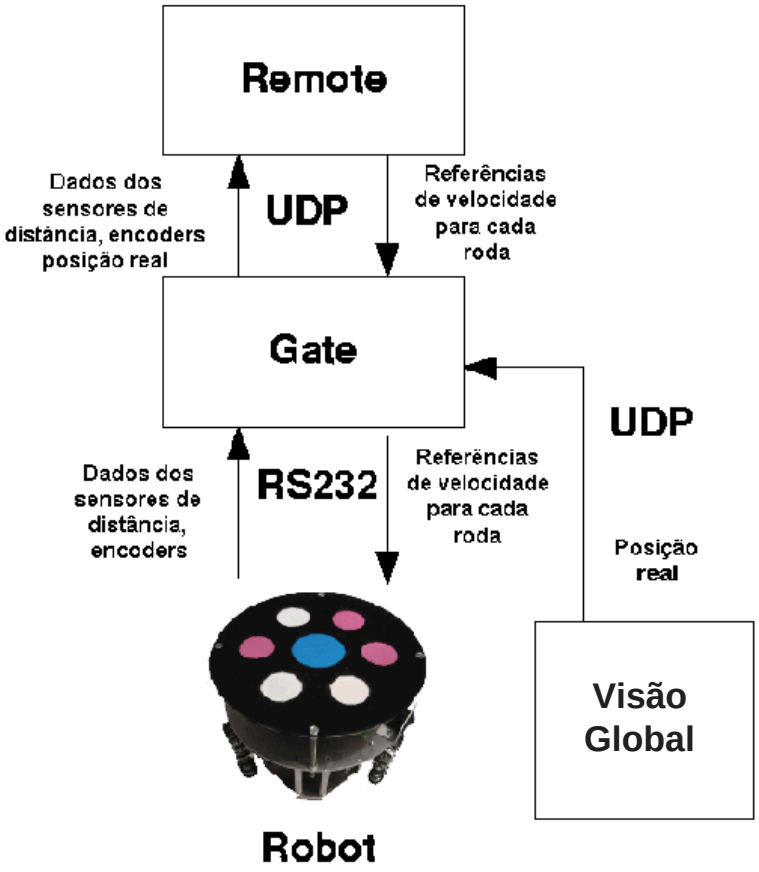

Figura 6.15: Arquitectura do sistema real.

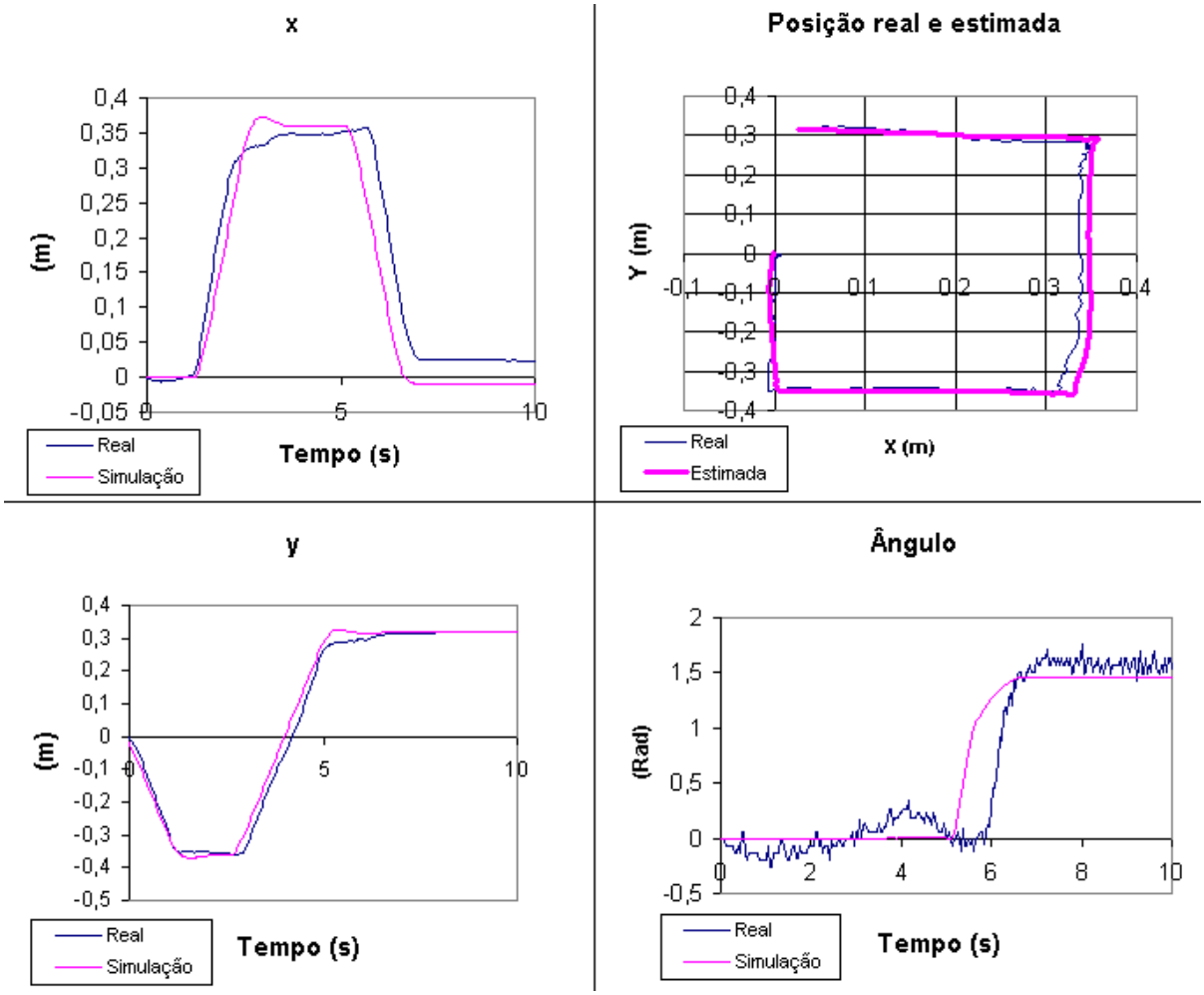

Figura 6.16: Trajectória do *robot*.

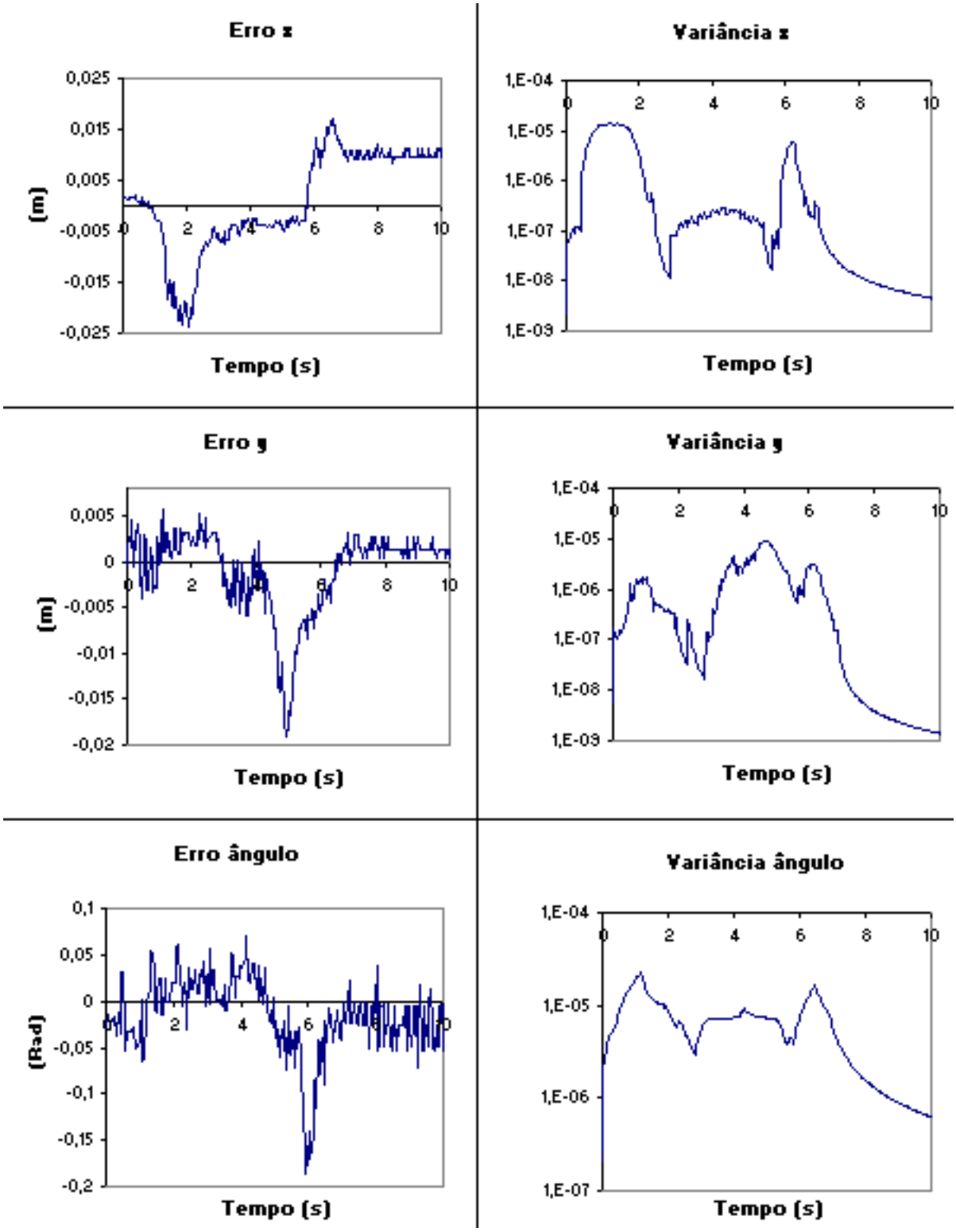

Figura 6.17: Erro e variância da estimativa da localização.

Capítulo 7

Conclusões e trabalho futuro

A migração de código de simuladores realistas para sistemas reais é a chave para reduzir o tempo de desenvolvimento *software* de localização, navegação e controlo de *robots* móveis. Com este propósito foi desenvolvido o simulador realista SimTwo, sendo uma ferramenta versátil para a simulação de *robots* e ambientes onde estes operam, permitindo modelar e testar *robots* diferenciais, omnidireccionais, manipuladores, humanóides, etc., sendo desenvolvido em Object Pascal. O Open Dynamics Engine (ODE) é utilizado para a simulação da dinâmica de corpos rígidos. A aparência do *robot* e o seu comportamento são definidos em ficheiros XML (linguagem standard eXtensible Markup Language). O mundo virtual é representado utilizando componentes de GLScene, dado que estes oferecem uma representação e visualização de uma cena 3D num ambiente OpenGL. A estes objectos são associadas propriedades físicas utilizando o ODE para determinar as colisões e atritos. O SimTwo permite a substituição dos sólidos por qualquer modelo 3DS dando um aspecto realista ao *robot*.

Para desenvolver simuladores realistas é necessário uma prévia modelação dos *robots* móveis, tendo que ser modelado os seus sensores e actuadores e a maneira como estes interagem com o meio. Neste trabalho foi realizada a modelação e simulação de duas plataformas robotizadas. Foi modelado e simulado o Kit Lego Mindstorms NXT, sendo uma plataforma frequentemente utilizada para o ensino de fundamentos de robótica e também foi modelado e simulado um *robot* omnidireccional de três rodas equipado com motores *brushless* e sensores de distância de infravermelhos destinado a operar em ambientes estruturados.

Para a plataforma Lego Mindstorms NXT foram modelados o servomotor e os sensores de toque, de ultra-sons e o sensor de luz, mas apenas foram simulados o sensor de luz e o servomotor. Por forma a validar a modelação e simulação do protótipo, um *robot* real e um simulado foram comparados

através de um desafio. O desafio proposto é que o *robot* siga um trajecto baseado num gradiente. O *robot* deve seguir pelo centro do gradiente de cada secção. Foi aplicado o mesmo controlador para o *robot* simulado e para o real. Este desafio permitiu validar a simulação do actuador e do sensor de luz. Como trabalho futuro é proposto realizar a simulação do sensor de ultra-sons e do sensor de toque.

Para a plataforma robotizada com base no sistema de locomoção omnidireccional foi desenvolvida uma arquitectura que permite a migração de *software* para o *robot* real, sendo desenvolvido como exemplo o *software* de localização e navegação do *robot* omnidireccional num ambiente estruturado. O algoritmo de localização é baseado na fusão sensorial de dados de distância de sensores de infravermelhos e de odometria, recorrendo a um filtro de Kalman estendido. Para realizar a validação física do sistema de localização e navegação do *robot* móvel foi também desenvolvido um sistema de localização em tempo real de baixo custo baseado em visão global. A ferramenta desenvolvida para localização em tempo real baseada em visão global permite a validação de sistemas de localização e navegação de *robots* móveis, monitorizando o seu desempenho em tempo real. Em robótica móvel os requisitos de tempo real são muito apertados, levando a que os algoritmos de localização sejam o mais possível optimizados, principalmente no que diz respeito ao processamento de imagem. Um algoritmo que acrescente muito atraso no processamento pode ter consequências graves, principalmente no que diz respeito ao tempo de reacção do *robot*. Para esta aplicação foram atingidos os requisitos de tempo real pretendidos, ou seja, 25 *frames* por segundo (máximo permitido pelo sistema PAL). A ferramenta desenvolvida é versátil para a sua utilização em várias etapas de aprendizagem no domínio da robótica móvel. Pode ser utilizada para validação de controladores e tácticas (monitorizando em tempo real jogos segundo as regras do Robocup Junior), permite a validação de sistemas de localização e navegação para ambientes estruturados (monitorizando o desempenho do sistema de localização e navegação de um *robot*), sendo também utilizado no estudo de controlo e técnicas de fusão sensorial no domínio do Robocup.

Apêndice A

Controlador do *robot* no modo *script*

```
procedure TrackControl(v, k: double);
var v1, v2, err, ys: double;
    P: TPoint3D;
begin
  P := GetSolidPos(irobot, isensor);
  if P.x > 0 then begin
    err := -P.y;
  end else if P.x > -0.25 then begin
    err := -P.y + 0.1;
  end else begin
    err := -P.y;
  end;

  v1 := v + k * err;
  v2 := v - k * err;

  SetAxisSpeedRef(irobot, 0, v1);
  SetAxisSpeedRef(irobot, 1, v2);
end;
```

Apêndice B

Arquitectura genérica de controlo em Java

```
import lejos.nxt.*;

public class main {
  //declare variables
  static boolean quit = false;
  static int end_time, start_time, lag_time;

  while (!quit) {
    start_time = (int)System.currentTimeMillis();
    //Sample
    //Execute
    end_time = (int)System.currentTimeMillis();
    lag_time = end_time - start_time;

    if (Button.ESCAPE.isPressed()) {
      quit = true;
    }

    if ((40 - lag_time) > 0) {
      Thread.sleep(40 - lag_time);
    }
  }
}
```

Apêndice C

Característica do sonar

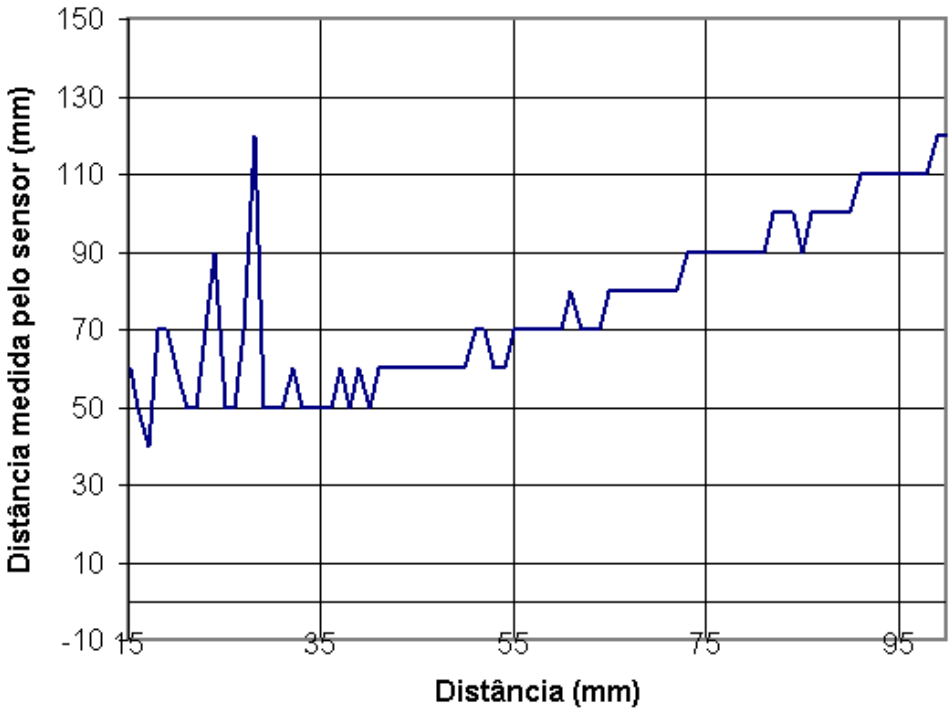

Figura C.1: Característica do sonar para leituras de 15 a 95 mm.

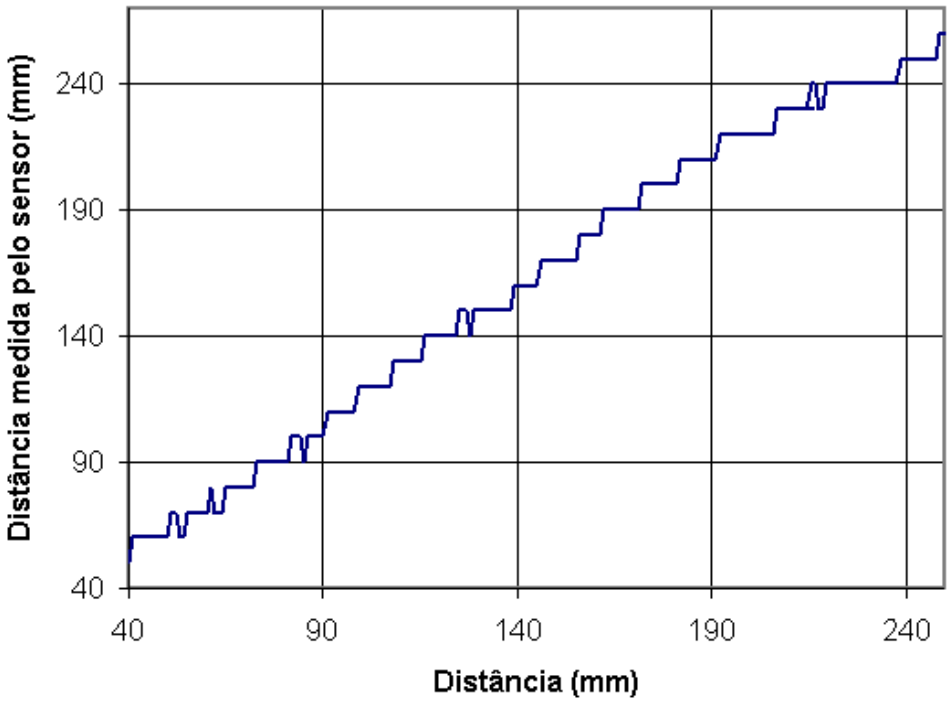

Figura C.2: Característica do sonar para leituras de 40 a 240 mm.

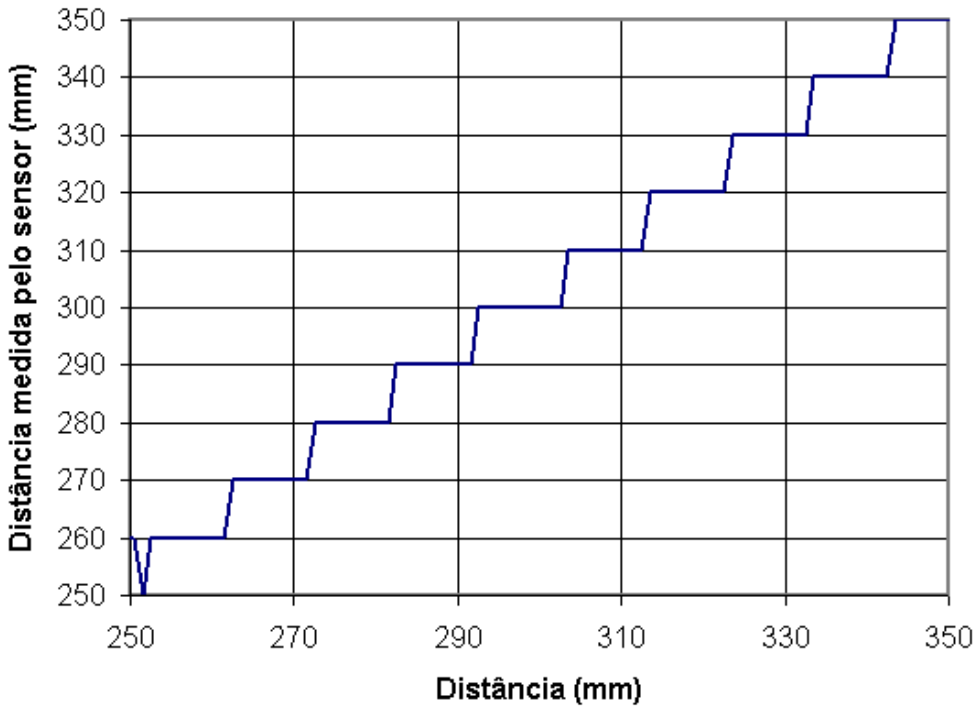

Figura C.3: Característica do sonar para leituras de 250 a 350 mm.

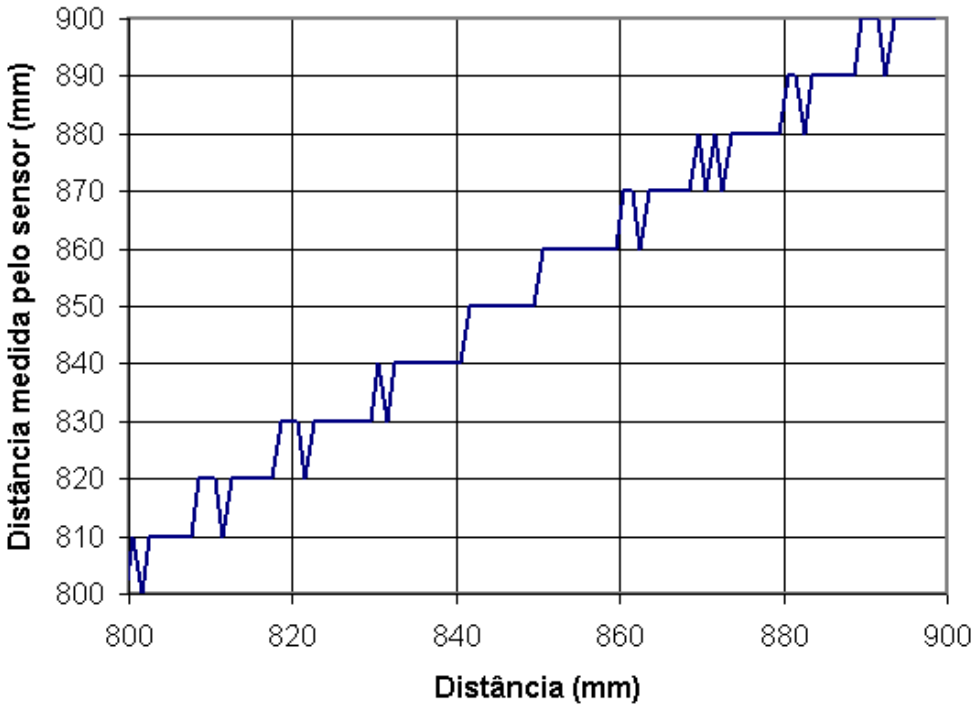

Figura C.4: Característica do sonar para leituras de 800 a 900 mm.

Apêndice D

Ficheiros XML do SimTwo

D.1 Ficheiro de descrição do *robot* da Lego Mindstorms

```
 <?xml version="1.0" ?>
<robot>
  <solids>

    <cuboid>
      <ID value='NXTBrick'/>
      <mass value='0.2'/>
    <mesh file='nxt_brick4.3DS' shadow='0'
     shadowfile='nxt_brick2_shadow.3DS' scale='0.001'/>
      <size x='0.1125' y='0.072' z='0.048'/>
      <pos x='0.0' y='0.02' z='0.09'/>
      <rot_deg x='0' y='26' z='-90'/>
      <color_rgb r='128' g='0' b='0'/>
    </cuboid>

    <cuboid>
      <ID value='NXTLightSensor'/>
      <mass value='0.05'/>
    <mesh file='nxt_sensor2.3DS'  shadow='0'
     shadowfile='nxt_sensor_shadow.3DS' scale='0.001'/>
      <size x='0.043' y='0.022' z='0.022'/>
      <pos x='0' y='-0.075' z='0.027'/>
      <rot_deg x='0' y='80' z='-90'/>
```

```
  <color_rgb r='128' g='0' b='0'/>
</cuboid>

<cuboid>
  <ID value='NXTMotorLeft'/>
  <mass value='0.1'/>
<mesh file='nxt_motor2.3DS' shadow='0'
 shadowfile='nxt_motor_shadow.3DS' scale='0.001'/>
  <size x='0.030' y='0.088' z='0.030'/>
  <pos x='0.030' y='0.042' z='0.04'/>
  <rot_deg x='0' y='180' z='0'/>
  <color_rgb r='128' g='0' b='0'/>
</cuboid>

<cuboid>
  <ID value='NXTMotorRight'/>
  <mass value='0.1'/>
<mesh file='nxt_motor2.3DS' shadow='0'
 shadowfile='nxt_motor_shadow.3DS' scale='0.001'/>
  <size x='0.030' y='0.088' z='0.030'/>
  <pos x='-0.030' y='0.042' z='0.04'/>
  <rot_deg x='0' y='180' z='0'/>
  <color_rgb r='128' g='0' b='0'/>
</cuboid>

<cylinder>
  <ID value='NXTWheelLeft'/>
  <mass value='0.05'/>
  <mesh file='nxt_wheel3.3DS'  shadow='0'
   shadowfile='nxt_wheel_shadow.3DS' scale='0.001'/>
  <size x='0.029' y='0.010' z='0.018'/>
  <pos x='0.056' y='0' z='0.029'/>
  <rot_deg x='0' y='90' z='0'/>
  <color_rgb r='128' g='0' b='0'/>
</cylinder>

<cylinder>
  <ID value='NXTWheelRight'/>
  <mass value='0.05'/>
  <mesh file='nxt_wheel3.3DS'  shadow='0'
  shadowfile='nxt_wheel_shadow.3DS' scale='0.001'/>
```

```
      <size x='0.029' y='0.010' z='0.018'/>
      <pos x='-0.056' y='0' z='0.029'/>
      <rot_deg x='0' y='90' z='0'/>
      <color_rgb r='128' g='0' b='0'/>
    </cylinder>

    <cylinder>
      <ID value='CasterWheel'/>
      <mass value='0.02'/>
      <size x='0.012' y='0' z='0.014'/>
      <pos x='0' y='0.121' z='0.012'/>
      <rot_deg x='0' y='90' z='0'/>
      <color_rgb r='128' g='0' b='0'/>
    </cylinder>

    <cylinder>
      <ID value='CasterVerticalSupport'/>
      <mass value='0.04'/>
      <size x='0.005' y='0' z='0.046'/>
      <pos x='0' y='0.105' z='0.030'/>
      <rot_deg x='0' y='0' z='0'/>
      <nodrag stokes='1e-4' roll='1e-4'/>
      <color_rgb r='128' g='0' b='0'/>
    </cylinder>


  </solids>

  <articulations>
    <default>
      <motor ri='7.6' li='4.88e-3' ki='0.4908/48' vmax='9' imax='1' active='1'/>
      <gear ratio='48'/>
      <friction bv='1.92e-3' fc='3.558e-4' coulomblimit='1e-2'/>
      <encoder ppr='360' mean='0' stdev='0'/>
      <controller mode='pidspeed' kp='1' ki='0' kd='0' kf='0.05'
       active='1' period='10'/>
    </default>

    <joint>
      <ID value='LeftAxis'/>
      <connect B1='NXTWheelLeft' B2='NXTMotorLeft'/>
```

```
    <pos x='0' y='0' z='0.029'/>
    <axis x='1' y='0' z='0'/>
    <type value='Hinge'/>
    <desc Eng='hinge'/>
  </joint>

  <joint>
    <ID value='RightAxis'/>
    <connect B1='NXTWheelRight' B2='NXTMotorRight'/>
    <pos x='0' y='0' z='0.029'/>
    <axis x='1' y='0' z='0'/>
    <type value='Hinge'/>
    <desc Eng='hinge'/>
  </joint>

  <joint>
    <ID value='CasterWheelAxis'/>
    <connect B1='CasterWheel' B2='CasterVerticalSupport'/>
    <pos x='0' y='0.121' z='0.012'/>
    <axis x='1' y='0' z='0'/>
    <type value='Hinge'/>
    <motor active='0'/>
    <controller active='0'/>
    <desc Eng='hinge'/>
  </joint>

  <joint>
    <ID value='CasterVertAxis'/>
    <connect B1='CasterVerticalSupport' B2='NXTBrick'/>
    <pos x='0' y='0.105' z='0'/>
    <axis x='0' y='0' z='1'/>
    <type value='Hinge'/>
    <motor active='0'/>
    <controller active='0'/>
    <desc Eng='hinge'/>
  </joint>

  <joint>
    <ID value='SensorSupport'/>
    <connect B1='NXTBrick' B2='NXTLightSensor'/>
    <type value='Fixed'/>
```

```
      <desc Eng='fixed'/>
    </joint>

    <joint>
      <ID value='MotorSupportLeft'/>
      <connect B1='NXTBrick' B2='NXTMotorLeft'/>
      <type value='Fixed'/>
      <desc Eng='fixed'/>
    </joint>

    <joint>
      <ID value='MotorSupportRight'/>
      <connect B1='NXTBrick' B2='NXTMotorRight'/>
      <type value='Fixed'/>
      <desc Eng='fixed'/>
    </joint>

  </articulations>

  <shells>
    <cuboid>
      <solid id='NXTMotorRight'/>
      <size x='0.02' y='0.10' z='0.006'/>
      <pos x='-0.030' y='-0.05' z='0.01'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='128' g='128' b='128'/>
    </cuboid>
  </shells>

</robot>
```

D.2 Ficheiro de descrição do *robot* omnidireccional

```
<?xml version="1.0" ?>
<robot>
  <kind value='omni3'/>

  <solids>
    <cylinder>
      <ID value='1'/>
      <mass value='1.8'/>
      <size x='0.09' y='0' z='0.135'/>
      <pos x='0' y='0' z='0.08'/>
      <rot_deg x='0' y='0' z='0'/>
      <color_rgb r='128' g='0' b='0'/>
      <texture name='MatFeup' scale='6'/>
    </cylinder>
  </solids>

    <wheels>
    <default>
      <omni/>
      <tyre mass='0.13' radius='0.0325' width='0.018' centerdist='0.09'/>
      <axis angle='0'/>
      <motor ri='2.853' ki='0.0289' vmax='12' imax='2' active='1'/>
      <gear ratio='51/10'/>
      <friction bv='0' fc='0.00283' coulomblimit='1e-2'/>
      <encoder ppr='216' mean='0' stdev='0'/>
<controller mode='pidspeed' kp='0.2' ki='0' kd='0.01'
  kf='0.05' active='1' period='10'/>
      <color_rgb r='128' g='0' b='128'/>
    </default>
    <wheel>
      <axis angle='0'/>
    </wheel>
    <wheel>
      <axis angle='120'/>
    </wheel>
    <wheel>
      <axis angle='240'/>
```

```
    </wheel>
  </wheels>


  <sensors>

   <IRSharp>
    <beam length='0.8' initial_width='0.002' final_width='0.002'/>
    <pos x='0' y='-0.03' z='0.03'/>
    <rot_deg x='0' y='0' z='0'/>
    <noise var_k='2e-5' var_d='4e-6' offset='0.0979' gain='3.5204'/>
    <color_rgb r='0' g='0' b='0'/>
  </IRSharp>

  <IRSharp>
    <beam length='0.8' initial_width='0.002' final_width='0.002'/>
    <pos x='0' y='0.03' z='0.03'/>
    <rot_deg x='0' y='0' z='0'/>
    <noise var_k='2e-5' var_d='4e-6' offset='0.0979' gain='3.5204'/>
    <color_rgb r='128' g='0' b='0'/>
  </IRSharp>

 <IRSharp>
    <beam length='0.8' initial_width='0.002' final_width='0.002'/>
    <pos x='0.03' y='0.0145' z='0.03'/>
    <rot_deg x='0' y='0' z='90'/>
    <noise var_k='2e-5' var_d='4e-6' offset='0.0979' gain='3.5204'/>
    <color_rgb r='128' g='0' b='0'/>
  </IRSharp>

   <IRSharp>
    <beam length='0.8' initial_width='0.002' final_width='0.002'/>
    <pos x='-0.03' y='0.0145' z='0.03'/>
    <rot_deg x='0' y='0' z='90'/>
    <noise var_k='2e-5' var_d='4e-6' offset='0.0979' gain='3.5204'/>
    <color_rgb r='128' g='0' b='0'/>
  </IRSharp>

  <IRSharp>
    <beam length='0.8' initial_width='0.002' final_width='0.002'/>
    <pos x='-0.03' y='-0.0145' z='0.03'/>
```

```
    <rot_deg x='0' y='0' z='270'/>
    <noise var_k='2e-5' var_d='4e-6' offset='0.0979' gain='3.5204'/>
    <color_rgb r='128' g='0' b='0'/>
  </IRSharp>

   <IRSharp>
    <beam length='0.8' initial_width='0.002' final_width='0.002'/>
    <pos x='0.03' y='-0.0145' z='0.03'/>
    <rot_deg x='0' y='0' z='270'/>
    <noise var_k='2e-5' var_d='4e-6' offset='0.0979' gain='3.5204'/>
    <color_rgb r='128' g='200' b='0'/>
  </IRSharp>
  </sensors>
</robot>
```

D.3 Ficheiro de descrição dos obstáculos

```
<?xml version="1.0" ?>
<obstacles>

  <cuboid>
    <imovable/>
  <size x='1.02' y='0.01' z='0.35'/>
  <pos x='0.0' y='0.505' z='0.0'/>
  <rot_deg x='0' y='0' z='0'/>
    <color_rgb r='255' g='255' b='255'/>
  </cuboid>

 <cuboid>
    <imovable/>
  <size x='1.02' y='0.01' z='0.35'/>
  <pos x='0.0' y='-0.505' z='0.0'/>
  <rot_deg x='0' y='0' z='0'/>
  <color_rgb r='255' g='255' b='255'/>
</cuboid>

 <cuboid>
    <imovable/>
  <size x='0.01' y='1' z='0.35'/>
  <pos x='0.505' y='0.0' z='0.0'/>
  <rot_deg x='0' y='0' z='0'/>
  <color_rgb r='255' g='255' b='255'/>
</cuboid>

 <cuboid>
    <imovable/>
  <size x='0.01' y='1.0' z='0.35'/>
  <pos x='-0.505' y='0.0' z='0.0'/>
  <rot_deg x='0' y='0' z='0'/>
<color_rgb r='255' g='255' b='255'/>
</cuboid>
```

```
    <cuboid>
    <imovable/>
  <size x='0.016' y='0.326' z='0.35'/>
  <pos x='0.155' y='0.003' z='0.0'/>
  <rot_deg x='0' y='0' z='0'/>
  <color_rgb r='255' g='255' b='255'/>
</cuboid>

    <cuboid>
    <imovable/>
  <size x='0.321' y='0.016' z='0.35'/>
  <pos x='0.0' y='0.158' z='0.0'/>
  <rot_deg x='0' y='0' z='0'/>
  <color_rgb r='255' g='255' b='255'/>
</cuboid>

    <cuboid>
    <imovable/>
  <size x='0.016' y='0.36' z='0.35'/>
  <pos x='-0.16' y='0.33' z='0.0'/>
  <rot_deg x='0' y='0' z='0'/>
  <color_rgb r='255' g='255' b='255'/>
</cuboid>

    </obstacles>
```

Bibliografia

[AAC+00] L. Almeida, J. Azevedo, C. Cardeira, P. Costa, P. Fonseca, P. Lima, F. Ribeiro, and V. Santos. Fostering advances in research, development and education in robotics. *Proceedings of the 4th Portuguese conference in Automatic Control*, 2000.

[AAC02] L. Almeida, J. Azevedo, and B. Cunha. Sete anos de concurso micro-rato na universidade de aveiro. *Revista Robótica, Nr^o 48, Publindústria*, 2002.

[ABB96] ABB. *RobComm User´s Guide*. ABB Flexible Automation inc, 1996.

[any09] Anykode. http://www.anykode.com/, 2009.

[BEF96] Borestein, Everett, and Feng. *where am I, Sensors and Methods for Mobile Robot Positioning*. Prepared by the University of Michigan, 1996.

[Bis02] R. Bishop. *The Mechatronics Handbook*. CRC Press, New York, 2002.

[BNMS05] L. Bento, U. Nunes, F. Moita, and A. Surrecio. Sensor fusion for precise autonomous vehicle navigation in outdoor semi-structured environments. *IEEE Conference on transportation Systems, Vienna, Austria*, pages 13–16, September 2005.

[Bra03] Brajovic. A model for reflectance perception in vision. In *SPIE, EMT03 - Microtechnologies for the New Millennium 2003*. SPIE, May 2003.

[Bre07] F. Bremond. Scene understanding: perception, multi-sensor fusion, spatio-temporal reasoning and activity recognition, July 2007. Keeneo.

[bul09] Bullet. http://www.bulletphysics.com/, 2009.

[BVR06] F. Bravo, A. Vale, and M. Ribeiro. Particle-filter approach and motion strategy for cooperative localization. *ICINCO*, 2006.

[CaMC06] A. Conceição, A. Moreira, and P. Costa. Model identification of a four wheeled omni-directional mobile robot. In *Controlo 2006, 7th Portuguese Conference on Automatic Control*, Instituto Superior Técnico, Lisboa, Portugal, 2006.

[CBDN96] G. Campion, G. Bastin, and B. Dandrea-Novel. Structural properties and classification of kinematic and dynamic models of wheeled mobile robots. *IEEE Transactions on Robotics and Automation*, 12(1):47–62, 1996. 1042-296X.

[CLB+04] H. Choset, K. Lynch, W. Burgard, L. Kavraki, and S. Thrun. *Principles of robot motion*. MIT Press, 2004.

[CMC06] A. Conceição, A. Moreira, and P. Costa. Dynamic parameters identification of an omni-directional mobile robot. *Proceedings of the International Conference on Informatics in Control, Automation and Robotics*, 2006.

[Col05] H. Colestock. *Industrial Robotics: Selection, design and maintenance*. McGraw Hill, 2005.

[Col08] Trinity College. Fire fighting robot competition faq - why does the arena have only 90 degree angles? In *http://www.trincoll.edu/events/robot/FAQ*, Hartford Connetticut, 2008.

[Cos95] P. Costa. Identificação, modelização e controlo de um veículo móvel, Master Thesis, FEUP, 1995.

[Cos99] P. Costa. Localização em tempo real de múltiplos robots num ambiente dinâmico, Tese de Doutoramento, FEUP, 1999.

[DJ00] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, 2000.

[Eve95] H. Everett. *Sensors for Mobile Robots: Theory and Application*. A K Peters Ltd, 1995.

[FBS+04] B. Figueiredo, D. Bento, E. Silva, J. Almeida, and A. Martins. Análise de veículos aéreos não tripulados e aplicações. *Encontro Científico do Festival nacional de Robótica*, Abril 2004.

[FGGE99] L. France, A. Girault, J. Gascuel, and B. Espiau. Sensor modeling for a walking robot simulation. In *Eurographics Workshop on Computer Animation and Simulation*, Sep 1999.

[gaz09] 3d multiple robot simulator with dynamics. http://playerstage.sourceforge.net/index.php?src=gazebo, 2009.

[GCM04] J. Gonçalves, P. Costa, and A. Moreira. Desenvolvimento de um robot omnidireccional para fins didácticos usando o kit lego mindstorms. *Proceedings of Cientific Meeting of Robótica 2004*, pages 25–30, 2004.

[GCM06] J. Gonçalves, P. Costa, and A. Moreira. Controlo e estimação do posicionamento absoluto de um robot omnidireccional de três rodas. *Revista Robótica*, 1º Trimestre 2006.

[gl09] Glscene. http://glscene.sourceforge.net/wikka/HomePage, 2009.

[GLC06] J. Gonçalves, J. Lima, and P. Costa. Rapid prototyping of mobile robots extending lego mindstorms platform. *in Proceedings of the 7th Ifac Symposium on Advances in control education*, 2006.

[GLC08a] J. Gonçalves, J. Lima, and P. Costa. Real-time localization of an omnidirectional mobile robot resorting to odometry and global vision data fusion: an ekf approach. *IEEE International Symposium on Industrial Electronics, Cambridge*, June 2008.

[GLC08b] J. Gonçalves, J. Lima, and P. Costa. Real-time tracking of an omnidirectional robot : An extended kalman filter approach. *International Conference on Informatics in Control, Automation and Robotics, Funchal*, May 2008.

[GLMC09a] J. Gonçalves, J. Lima, P. Malheiros, and P. Costa. Code migration from a realistic simulator to a real wheeled mobile robot. *9th Conference on Autonomous Robot Systems and Competitions, Castelo Branco, Portugal,*, Maio 2009.

[GLMC09b] J. Gonçalves, J. Lima, P. Malheiros, and P. Costa. Realistic simulation of a lego mindstorms nxt based robot. *Scheduled for presentation during the Invited CCA Session "LEGO based Control Education and Prototyping in Robotics, Mechatronics*

and Embedded Systems, IEEE Multi-conference on Systems and Control", Saint Petersburg, Russia,, July 2009.

[GLOC08] J. Gonçalves, J. Lima, H. Oliveira, and P. Costa. Sensor and actuator modeling of an realistic wheeled mobile robot simulator. *Proceedings of the 13th IEEE International Conference on Emerging Technologies and Factory Automation*, 2008.

[Gon05] J. Gonçalves. Controlo de robots omnidireccionais, Tese de Mestrado, FEUP, 2005.

[GPLC07] J. Gonçalves, P. Pinheiro, J. Lima, and P. Costa. Tutorial introdutório para as competições de futebol robótico. *IEEE RITA - Revista Iberoamericana de tecnologias da aprendizagem*, 2(2):63–72, Novembro 2007.

[Gro86] M. Grover. *Industrial Robotics: Technology, Programming, and Applications*. McGraw Hill, 1986.

[GWA01] M. Grewal, L. Weill, and A. Andrews. *Global positioning systems, inertial navigation and integration*. John Wiley and Sons Inc., 2001.

[hav09] Havok. http://www.havok.com, 2009.

[Jap04] Yaskawa Electric Corporation Japan. *NX 100 applications options - Instructions for data transmission function*. Motoman, August 2004.

[KC06] D. Kulic and E. Croft. Real-time safety for human-robot interaction. *Robotics and Autonomous Systems*, 54(1):1–12, 2006.

[Kho89] P. Khosla. Categorization of parameters in the dynamic robot model. *IEEE Transactions on Robotics and Automation*, 5(3):261–268, 1989. 1042-296X.

[KNDG02] T. Kalmár-Nagy, R. D'Andrea, and P. Ganguly. Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle. Sibley School of Mechanical and Aerospace Engineering Cornell University Ithaca, NY 14853, USA, April 8 2002.

[KSO76] M. Kendall, A. Stuart, and J. Ord. *The Advanced Theory of Statistics*. Griffin, 1976.

[LP00] H. Lund and L. Pagliarinis. Robocup jr. with lego mindstorms. *in Proceedings of the 2000 IEEE International Conference on Robotics and Automation, San Francisco, CA, IEEE*, April 2000.

[LSR+03] P. Lima, J. Sousa, M. Ribeiro, C. Cardeira, and L. Custódio. 3º festival nacional de robótica. *Artigo convidado Revista Robótica*, Julho 2003.

[Mar03] V. Marano. L6235 three phase brushless dc motor driver. *Application Note*, 2003.

[Mic04] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.

[mic09] Microsoft robotics developer center. http://msdn.microsoft.com/en-us/robotics/default.aspx, 2009.

[MMBV06] A. Moreira, M.Antunes, E. Bicho, and P. Ventura. A service robot for samples transportation and delivery. *Encontro Científico do Festival nacional de Robótica*, Abril 2006.

[MSC01] A. Moreira, A. Sousa, and P. Costa. Vision based real-time localization of multiple mobile robots. In *3rd Int. Conf. on Field and Service Robotics, Helsinki, Finland*, 2001.

[Neg03] R. Negenborn. Robot localization and kalman filters - on finding your position in a noisy world, Master Thesis, Utrecht University, 2003.

[new09] Newton game dynamics. http://www.newtondynamics.com/, 2009.

[ode09] Ode. http://www.ode.org/, 2009.

[OGL09] Opengl. http://www.opengl.org/, 2009.

[OLi07] H. OLiveira. Análise do desempenho e da dinâmica de robôs omnidireccionais de três e quatro rodas, Tese de Mestrado, FEUP, 2007.

[O.M09] O.Michel. Khepera simulator. http://diwww.epfl.ch/lami/team/michel/, 2009.

[omn09] Omni wheels - kornylak corporation. http://www.omniwheel.com/omniwheel/omniwheel.htm, 2009.

[ope09] A 3d simulator for autonomous robots. http://opensimulator.sourceforge.net/, 2009.

[OSMC08] H. Oliveira, A. Sousa, A. Moreira, and P. Costa. Dynamical models for omni-directional robots with 3 and 4 wheels. *ICINCO 2008 -International Conference on Informatics in Control, Automation and Robotics. ISBN:978-989-8111-31-9.Vol I. pp.189-196. Funchal, Madeira, Portugal.*, May 2008.

[Pir06] J. Pires. *Industrial Robots Programming: Building Applications for the Factories of the Future*. Springer, 2006.

[PMC03] L. Pereira, A. Matos, and N. Cruz. Post mission trajectory smoothing for the isurus auv. *Proceedings of the OCEANS03 Conference, Vol.2, San Diego, EUA*, Setembro 2003.

[Rek03] I. Rekleitis. A particle filter tutorial for mobile robot localization. *International conference on robotics and automation*, 2003.

[Rib04a] M. Ribeiro. Gaussian probability density functions: Properties and error characterization. *Technical Report, IST*, 2004.

[Rib04b] M. Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Technical Report, IST*, 2004.

[RMS+04] F. Ribeiro, I. Moutinho, P. Silva, C. Fraga, and N. Pereira. Controlling omni-directional wheels of a robocup msl autonomous mobile robot. In *Scientific Meeting of the Portuguese Robotics Open*, 2004.

[RN96] F. Ribeiro and J. Norrish. Case study of rapid prototyping using robot welding - "square to round" shape. *27th International Symposium on Industrial Robotics, Milan, Italy, pag. 275-280*, October 1996.

[rob09] Robocup. http://www.robocup.org/, 2009.

[SCM04] A. Sousa, P. Costa, and A. Moreira. Sistema de localização de robôs móveis baseado em filtro de kalman estendido. *Proceedings of Cientific Meeting of Robótica 2004*, pages 83–87, 2004.

[Sie03] Siegwart. Mobile robots facing the real world. In Yuta et al. [YAT+06], pages 21–30.

[Sil95] P. Silva. Navegação acústica em ambientes estruturados, Tese de Mestrado, FEUP, 1995.

[sim09] Simtwo. http://www.fe.up.pt/∼paco/wiki, 2009.

[Sou03] A. Sousa. Arquitecturas de sistemas robóticos e localização em tempo real através de visão, Tese de Doutoramento, FEUP, 2003.

[TBF05] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, 2005.

[WB01] G. Welch and G. Bishop. An introduction to the kalman filter. *Technical Report, University of North Carolina at Chapel Hill*, 2001.

[WCGR02] R. Williams, B. Carter, P. Gallina, and G. Rosati. Dynamic model with slip for wheeled omnidirectional robots. *IEEE Transactions on Robotics and Automation*, 18(3):285–293, 2002. 1042-296X.

[Web] Webots. http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

[yad09] Yade. http://yade.wikia.com/wiki/Yade, 2009.

[YAT+06] S. Yuta, H. Asama, S. Thrun, E. Prassler, and T. Tsubouchi, editors. *Field and Service Robotics, Recent Advances in research and Applications, FSR 2003, Lake Yamanaka, Japan, 14-16 July 2003*, volume 24 of *Springer Tracts in Advanced Robotics*. Springer, 2006.

[YBW00] H. Yang, J. Borenstein, and D. Wehe. Sonar-based obstacle avoidance for a large, non-point, omnidirectional mobile robot. *in Proc of the International Conference in Nuclear and Hazardous Waste Management*, 2000.

[ZGA98] F. Zhao, H. Guo, and K. Abe. Mobile robot localization using two sonar sensors and one natural landmark. *Proceedings of the 37th SICE Annual Conference*, 1998.