Universidade do Porto

Controlo de *robots* omnidireccionais

José Alexandre de Carvalho Gonçalves

Fevereiro - 2005

Controlo de *robots* omnidireccionais

José Alexandre de Carvalho Gonçalves

Licenciado em Engenharia Electrotécnica e Computadores no Ramo de Automação, Produção e Electrónica Industrial

Dissertação submetida para efeito de atribuição do grau de Mestre em Engenharia Electrotécnica.

Sob orientação científica do Doutor Paulo José Cerqueira Gomes da Costa e do Doutor António Paulo Gomes Mendes Moreira

Fevereiro - 2005

Aos meus pais e a todos aqueles que partilham comigo a alegria de viver

Resumo

Esta tese aborda o tema Controlo de *Robots* Omnidireccionais. Os *robots* abordados como exemplo utilizam motores DC com *encoders* e possuem 3 rodas especiais que permitem a sua omnidireccionalidade. O movimento destes *robots* não está sujeito às restrições dos *robots* mais usuais que utilizam apenas duas rodas normais, tendo como desvantagem o seu controlo ser mais complexo.

Para efectuar o estudo do controlo e estimação do posicionamento deste tipo de *robots* foram efectuados dois protótipos, sendo um deles feito com *Kit Lego Mindstorms* e o outro com uma estrutura mecânica de alumínio e micro-controladores AVR.

O principal objectivo deste trabalho foi o de desenvolver um controlador que permita o seguimento de trajectórias pré-definidas com velocidade controlada, estimando-se o posicionamento absoluto do *robot* baseado na odometria.

Abstract

The presented work is about Control of Omnidirectional robots. The presented robots as example use DC motors with encoders and they have 3 special wheels that allow the robot to move in every direction. The movement of these robots doesn't have the restraints of the most usual robots that have two wheels, having as a disadvantage a more complex control.

with the objective of studying the control and estimation of the positioning of this type of robots there have been made two prototypes, one of them with Lego Mindstorms Kit and the other with a much more solid mechanical structure made of alumina and using AVR microcontrollers.

The main objective of this work is to develop a controller that permits the following of predefined trajectories with controlled velocity, estimating the positioning of the robot based on the odometry.

Résumé

Ce travail aborde le thème Contrôle de *Robots* omnidirectionnelle. Ces *robots* utilisent des moteurs DC avec *encoders* et ont 3 roues spéciales qui permettent sa omnidirectionalité. Le mouvement de ces *robot* ne sont pas soumis aux restrictions des *robots* plus usuelles qui utilisent seulement deux roues normales, ayant l´inconvénient d´avoir un contrôle plus complexe.

Pour effectuer l´étude du contrôle et estimation de positionnement de ce type de *robots* il a fallut faire deux prototypes, l´un d´eux a été fait avec *Kit Lego Mindstorms* et l´autre avec une structure mécanique d´aluminium et microcontrôleurs AVR.

L´objectif essentielle de ce travail a été celui de développer un contrôleur qui permet le parcours de trajectoires prédéfinis avec une rapidité contrôlé, en s'estimant la position du *robot* a partir de l´odomètrie.

Agradecimentos

Aos meus orientadores pela sua inteira disponibilidade e pela quantidade de conhecimentos que me transmitiram quer a nível técnico quer a nível científico.

Ao Fernando Ribeiro da Minho Team da Universidade do Minho e ao Luís Seabra Lopes da equipa Cambada da Universidade de Aveiro, pela disponibilização de informação sobre *robots* omnidireccionais.

Ao Henrique Teixeira com quem trabalhei durante a parte curricular do Mestrado.

Ao pessoal do Laboratório de Sistemas e Tecnologia Subaquática (é sempre bom trocar ideias convosco).

Ao Laboratório de Automação e ao Laboratório de Electrónica e Instrumentação da ESTiG IPB pela total disponibilidade na utilização de equipamentos no decorrer de este trabalho.

À Celeste Morais pelas suas ajudas com o LaTeX.

E em especial à Elisabete que sempre me deu força para continuar.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A robótica móvel é um campo multi disciplinar envolvendo Ciência de computadores e Engenharia, dividindo-se várias áreas primordiais:

- Sensores - Permitem a um *robot* autónomo auscultar o meio em que se insere.
- Decisão e controlo - Concentra-se na navegação e exploração em ambiente dinâmico de um sistema autónomo com vista a alcançar um determinado objectivo.
- Locomoção e actuação - A locomoção concentra-se no estudo do movimento de *robots* com rodas, aquáticos, veículos aéreos e os denominados *Legged robots*.

A locomoção é área principal a ser abordada nesta dissertação sendo a plataforma estudada um *robot* omnidireccional com três rodas para omnidireccionais.

O desenvolvimento de *robots* levanta questões de segurança, existindo a nível da robótica industrial normas *Standard* que obrigam os operadores a ter uma determinada conduta na presença de um *robot*. Estas normas estão divididas em passivas (impedir que aconteça um acidente) e activas (caso aconteça um acidente

as acções que se devem tomar para minimizar os danos do acidente). Em robótica móvel a segurança está longe de ser *standard*, pois os *robots* não são *standard*, logo devem ser seguidas as três regras base sugeridas por Isaac Asimov [DJ00]:

- Um *robot* nunca deve causar ferimentos a um humano.
- Um *robot* deve obedecer às ordens de um humano desde que a primeira regra seja respeitada
- Um *robot* deve proteger a sua existência desde que isso respeite as regras anteriores.

Esta tese aborda o tema Controlo de *Robots* Omnidireccionais, inserindo-se na área de locomoção. É importante realçar o baixo custo do material utilizado no desenvolvimento do primeiro protótipo, sendo este realizado com peças *Lego*, tendo estas um papel muito importante no ensino para prototipagem rápida. Foi também projectado e implementado um outro *robot*, com estrutura de alumínio e micro-controladores da família AVR, tendo-se obtido melhores dados com este *robot*, com vista a tirar conclusões quanto ao desempenho do controlador projectado, o que não invalida a importância do primeiro protótipo.

Os problemas de controlo em robótica prendem-se com a inevitável não linearidade e com requisitos de tempo real muito duros [Cos95], sendo isto crítico dado o controlador e cálculo da odometria de um *robot* omnidireccional ser computacionalmente muito mais pesado que os tipicamente usados *robots* diferenciais. Apesar disso foi possível, para o primeiro protótipo, o cálculo do controlador e da odometria, sem que haja comunicação com o computador pessoal *(*PC), sendo isto alcançado á custa de baixar a frequência de amostragem. Para o segundo protótipo usando-se um micro-controlador da família AVR com arquitectura $RISC$ (acrónimo de *reduced instruction set computer*), o facto de as operações serem efectuadas com 8 *bits*, implica que o código gerado rapidamente esgota a memória. Nestas condições a solução usada foi o controlador e a odometria serem calculadas no *PC*, comunicando este com o micro-controlador.

1.1 Definição do problema e abordagem proposta

Esta tese aborda o tema Controlo de *Robots* Omnidireccionais. Os *robots* abordados como exemplo utilizam motores DC com *encoders* e possuem 3 rodas especiais que permitem a sua omnidireccionalidade. O movimento destes *robots* não está sujeito às restrições dos *robots* mais usuais que utilizam apenas duas rodas normais, tendo como desvantagem o seu controlo ser mais complexo. Para efectuar o estudo do controlo de este tipo de *robots* foram construídos dois protótipos. Dispondo-se da informação odométrica procurámos projectar um controlador que permitisse o seguimento de trajectórias pré-definidas.

1.2 Organização da Tese

No capítulo dois é apresentado o estado da arte da aplicabilidade de *robots* omnidireccionais, sendo apresentada como motivação para a utilização deste tipo de *robot* a comparação desta configuração face à comummente usada configuração diferencial. São também apresentadas as rodas omnidireccionais, as quais eram inicialmente usadas apenas para tapetes de transporte na indústria, tendo posteriormente aplicações na robótica móvel. As principais aplicações deste tipo de *robots* é primordialmente no futebol robótico, apesar de ter aparecido um projecto em que a aplicação final é uma cadeira de rodas. A nível do futebol robótico foram apresentados os *robots* das equipas 5DPO, da Minho Team e Cambada da *Middle Size League* (todas estas equipas têm *drives* omnidireccionais, apresentando a equipa 5DPO um sistema omnidireccional de quatro rodas e as restantes equipas um sistema de três rodas) e a equipa 5DPO da *Small Size League*. Além da aplicação no futebol robótico foi apresentada uma aplicação de uma configuração de um *robot* omnidireccional numa cadeiras de rodas, mostrando que a investigação em robótica traz benefícios directos na comunidade em geral e não apenas à comunidade científica.

O capítulo três descreve como pode ser desenvolvido rapidamente um *robot* omnidireccional, utilizando o *Kit Lego Mindstorms*, tendo como vantagens: baixo custo, modularidade, conectividade de peças e permitindo reutilização das mesmas. É apresentado como foi desenvolvido o protótipo a nível mecânico, os seus sensores e actuadores e a arquitectura de *Software*, dando-se ênfase ao facto de se ter feito uma prototipagem rápida, à linguagem utilizada e à optimização do controlador com vista a cumprir requisitos de tempo real.

No capítulo quatro é descrito o estudo do controlo e posicionamento de um *robot* omnidireccional com uma estrutura mecânica de alumínio e microcontroladores AVR. O objectivo essencial deste capítulo foi o de descrever o controlador desenvolvido, o qual permite ao *robot* o seguimento de trajectórias pré-definidas com velocidade controlada, estimando-se o posicionamento absoluto a partir do cálculo da odometria.

No capítulo cinco são apresentadas conclusões e trabalho futuro

Capítulo 2

Estado da arte

2.1 *Robot* omnidireccional vs *robot* diferencial

O *robot* diferencial é provavelmente o *robot* móvel mais usado, sendo representado na figura 2.1. 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.

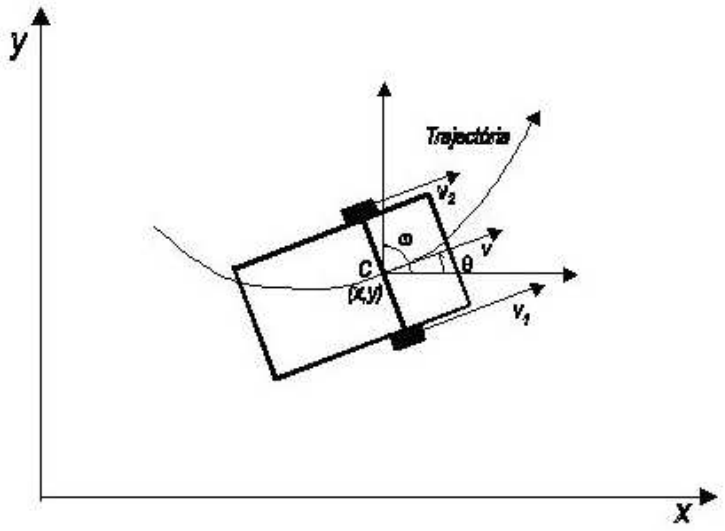


Figura 2.1: *Robot* diferencial

A estrutura do *robot* diferencial, representada na figura 2.1, impede que sejam feitos movimentos de translação segundo o eixo que passa pelos veios 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 equação 2.1.

$$X(t)^T = \begin{pmatrix} \mathrm{x(t)} & \mathrm{y(t)} & \theta(t) & \mathrm{v(t)} & w(\mathrm{t}) \end{pmatrix} \tag{2.1}$$

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 :

$$X(t)^T = \begin{pmatrix} \mathrm{x(t)} & \mathrm{y(t)} & \theta(t) & \mathrm{V}_1(t) & \mathrm{V}_2(t) \end{pmatrix} \tag{2.2}$$

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 2.3 e a equação 2.4. Na equação 2.4 b representa a distância entre os pontos de contacto das rodas com o chão.

$$V(t) = \frac{V_1(t) + V_2(t)}{2} \tag{2.3}$$

$$w(t) = \frac{V_1(t) - V_2(t)}{b} \tag{2.4}$$

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

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \mathrm{v(t)cos}(\theta(\mathrm{t})) \\ \mathrm{v(t)sin}(\theta(\mathrm{t})) \\ w(\mathrm{t}) \end{pmatrix} \tag{2.5}$$

Esta equação permite usando as equações 2.3 e 2.4 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].

Para colmatar a limitações do *robot* diferencial surgiu o *robot* omnidireccional, permitindo deslocações em todas as direcções [KNDG02]. Para garantir a característica da omnidireccionalidade é necessário que as rodas usadas tenham pouco atrito na direcção do veio do motor, o que impediria deslocações segundo esse eixo. As rodas especiais para omnidireccionais usadas no futebol robótico eram primariamente usadas apenas para mesas de transporte, estando representadas na figura 2.2, sendo estas de diferentes tamanhos e com diferentes requisitos de robustez. Muitas das equipas que competem no futebol robótico já produzem as suas próprias rodas.

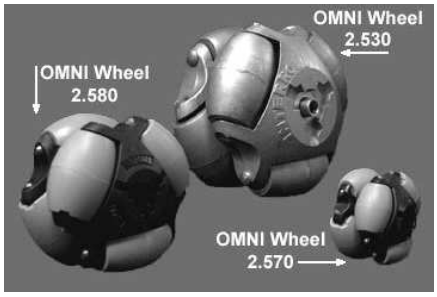


Figura 2.2: Rodas omnidireccionais

Como podemos comprovar da observação da geometria de um *robot* omnidireccional com três rodas, representada na figura 2.3, as velocidades V_x, V_y e w variam com a velocidades lineares V_1, V_2 e V_3, baseando-se na equação 2.6 [KNDG02].

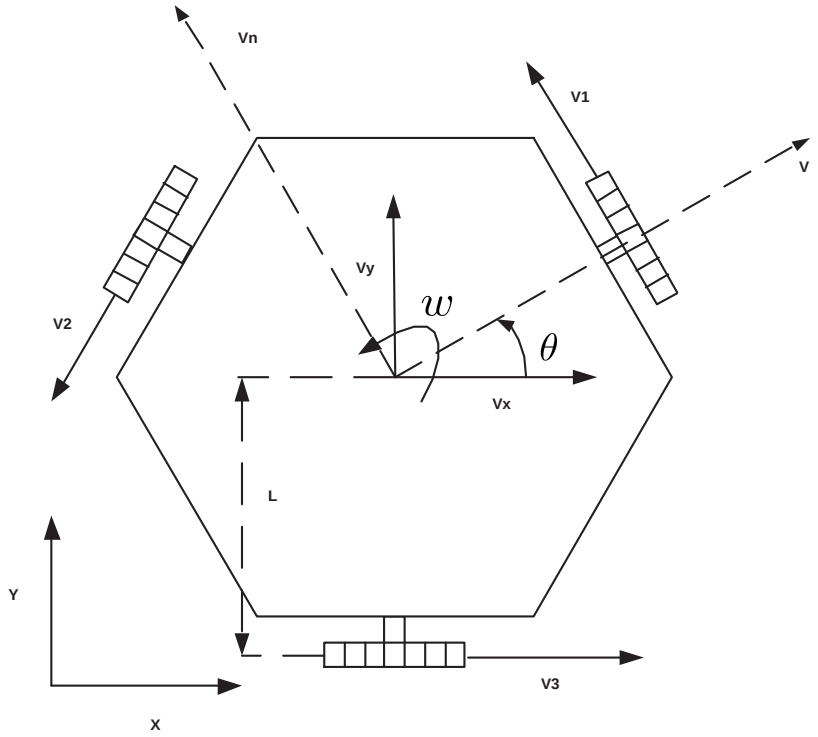


Figura 2.3: Geometria de um *robot* omnidireccional de três rodas.

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = \begin{pmatrix} -sin(\theta) & cos(\theta) & L \\ -sin(\frac{\pi}{3}-\theta) & -cos(\frac{\pi}{3}-\theta) & L \\ sin(\frac{\pi}{3}+\theta) & -cos(\frac{\pi}{3}+\theta) & L \end{pmatrix} \begin{pmatrix} \mathrm{V}_x \\ \mathrm{V}_y \\ w \end{pmatrix} \tag{2.6}$$

O cálculo do controlador é baseado no modelo apresentado na equação 2.6. As equações de cinemática do *robot*, poderiam ser representadas pelas equações 2.7 e 2.8 em alternativa à equação 2.6.

$$\begin{pmatrix} V \\ V_n \\ w \end{pmatrix} = \begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ w \end{pmatrix} \tag{2.7}$$

$$\begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 & L \\ -sin(\frac{\pi}{3}) & -cos(\frac{\pi}{3}) & L \\ sin(\frac{\pi}{3}) & -cos(\frac{\pi}{3}) & L \end{pmatrix} \begin{pmatrix} V \\ V_n \\ w \end{pmatrix} \tag{2.8}$$

A equação 2.7 representa a velocidade (V) e a velocidade normal (V_n) do *robot* em função das velocidades V_x e V_y. A equação 2.8 representa as velocidades V_1, V_2 e V_3 calculadas a partir da velocidades do *robot*.

As equações para o cálculo da odometria foram deduzidas através da inversão da matriz representada pelo sistema de equações 2.6. As equações obtidas são as seguintes:

$$\begin{aligned} V_x = {} & (-V_2 \sin(\frac{\pi}{3}) + \sin(\frac{\pi}{3})V_3 + \sin(2\theta)V_3 - \sin(2\theta)V_1 + 2V_2 \sin(\frac{\pi}{3}) \sin(\theta)^2 \\ & + \mathrm{V}_1 \sin(2\theta) \cos(\tfrac{\pi}{3}) - \cos(\tfrac{\pi}{3}) sin(2\theta)V_2 - 2V_1 \sin(\tfrac{\pi}{3}) \sin(\theta)^2) \\ & /2((\sin(\theta) + \sin(\tfrac{\pi}{3}) \cos(\theta) + \cos(\tfrac{\pi}{3}) \sin(\theta)) \sin(\tfrac{\pi}{3}) \end{aligned} \tag{2.9}$$

$$\begin{aligned} V_y = {} & -\frac{1}{2 \sin(\frac{\pi}{3})^2} (V_3 \cos(\theta) - 2V_1 \cos(\theta) + V_2 \cos(\theta) + V_2 \sin(\theta) \sin(\frac{\pi}{3}) \\ & \text{-}\mathrm{V}_3 \sin(\theta) \sin(\tfrac{\pi}{3}) - V_3 \cos(\theta) \cos(\tfrac{\pi}{3}) - V_2 \cos\theta cos(\tfrac{\pi}{3}) + 2V_1 \cos(\theta) \cos(\tfrac{\pi}{3})) \end{aligned} \tag{2.10}$$

$$w = (-2\cos(\theta)V_1 + \cos(\theta)V_3 + V_2\cos(\theta) - \cos(\theta)cos(\frac{\pi}{3})V_2) + \sin(\theta)V_2\sin(\frac{\pi}{3}) +$$

$$\sin(\theta)V_3\sin(\tfrac{\pi}{3}) + 2V_1\sin(\tfrac{\pi}{3})\cos(\tfrac{\pi}{3})\sin(\theta) + 2V_1\sin(\tfrac{\pi}{3})^2\cos(\theta) + 2\cos(\theta)\cos(\tfrac{\pi}{3})V_1 -$$

$$\cos(\theta)\cos(\tfrac{\pi}{3})V_3)/(2L\sin(\tfrac{\pi}{3})(\sin(\theta) + \sin(\tfrac{\pi}{3})\cos(\theta) + \cos(\tfrac{\pi}{3})\sin(\theta)))$$

(2.11)

O cálculo destas equações torna-se bastante dispendioso computacionalmente e como o objectivo é utilizá-las para efectuar o controlador estas devem ser o mais optimizadas possível, com vista a cumprir requisitos de tempo real.

O cálculo da odometria pode ser optimizado usando a equação 2.12.

$$2\sin(\theta)\cos(\theta) = \sin(2\theta) \tag{2.12}$$

O objectivo de usar esta função trigonométrica foi o de o controlador se tornar computacionalmente menos pesado, pois é muito menos dispendioso a nível de tempo de cálculo calcular apenas uma vez $sin(\theta)$ e $cos(\theta)$, que efectuar o cálculo de vários senos e cosenos.

Nestas condições torna-se mais eficiente para o cálculo da odometria usar a equação seguinte em detrimento apresentada anteriormente:

$$V_x = (-V_2\sin(\frac{\pi}{3}) + \sin(\frac{\pi}{3})V_3 + 2\sin(\theta)\cos(\theta)V_3 - 2\sin(\theta)\cos(\theta)V_1 + 2V_2\sin(\frac{\pi}{3})sin(\theta)^2$$

$$+\mathrm{V}_1 2\sin(\theta)\cos(\theta))\cos(\tfrac{\pi}{3}) - \cos(\tfrac{\pi}{3})2\sin(\theta)\cos(\theta)V_2 - 2V_1\sin(\tfrac{\pi}{3})\sin(\theta)^2)$$

$$/2((\sin(\theta) + \sin(\tfrac{\pi}{3})\cos(\theta) + \cos(\tfrac{\pi}{3})\sin(\theta))\sin(\tfrac{\pi}{3})$$

(2.13)

Um dos aspectos importantes a ter em conta, para validar as equações usadas para a odometria é descobrir no estado inicial do *robot* por onde passam os eixos dos X e dos Y. Um teste que permite tirar conclusões sobre este aspecto é simular as equações apresentadas, com velocidade V_1 nula, V_2 com velocidade negativa e V_3 com velocidade positiva e de igual módulo que V_2. Por análise empírica da figura 2.3, o que será de esperar é que o ângulo permaneça constante, podendo-se tirar a conclusão, caso isto se verifique, que os dados são coerentes. Quanto aos eixos somente se podem tirar conclusões após a análise dos gráficos abaixo apresentados.

Como podemos observar os dados representados nos gráficos da figuras 2.4 e 2.5, são coerentes pois como seria de esperar o ângulo permanece inalterado. Também se pode tirar a conclusão de que inicialmente o eixo dos X passa pelo veio da roda 1, pois a velocidade dessa roda é nula, o movimento impresso pelas outras duas rodas(com igual módulo e valores contrários), aumentando apenas o valor do posicionamento absoluto segundo o eixo do X linearmente, mantendo-se segundo o eixo dos Y inalterado.

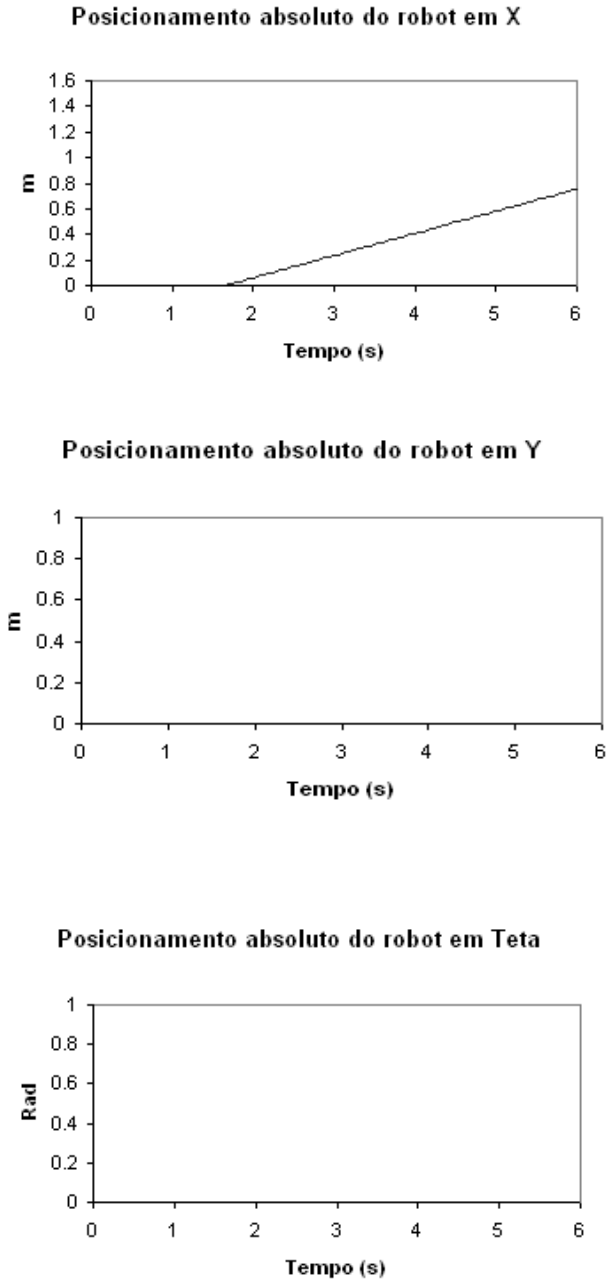


Figura 2.4: Posicionamento absoluto do *robot*

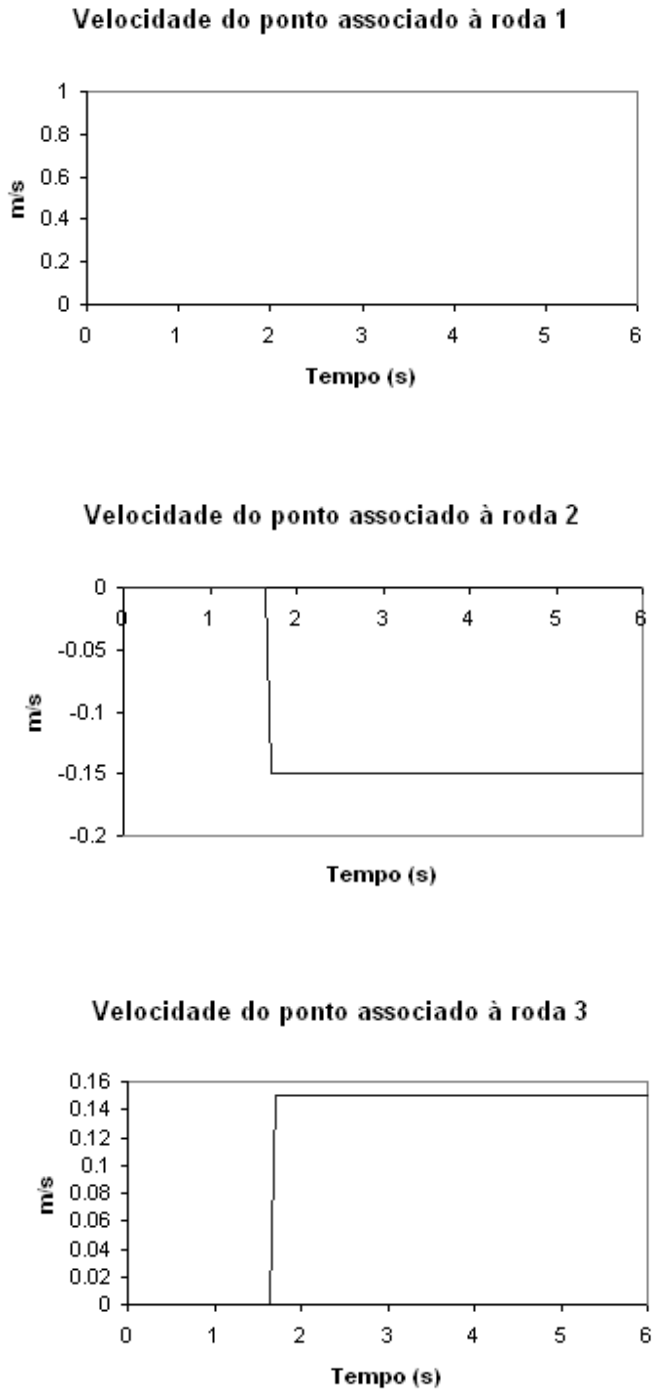


Figura 2.5: Velocidades dos pontos associados às rodas do *robot*

2.2 Aplicação de *robots* omnidireccionais no futebol robótico

O futebol serviu de inspiração para a definição do *Robocup*, pois além de ser um jogo bastante popular em todo o mundo, o que atrai visitantes a estes eventos, tem um conjunto muito significativo de desafios para os investigadores, pelo facto de ser um jogo cooperativo em ambiente dinâmico [rob04]. A maior parte das aplicações de *robots* omnidireccionais, neste momento, são para jogos de futebol Robótico, sendo exemplos a *Middle Size League* e a *Small Size League*. Como o tempo para chegar à bola é extremamente importante, conseguiu-se com esta configuração melhorar esse aspecto, diminuindo as manobras pois existe a possibilidade de os percursos poderem ser feitos em linha recta, sem que o *robot* tenha que rodar sobre si mesmo para se deslocar numa determinada direcção. Os motores são controlados independentemente de maneira a ser obtida uma determinada velocidade angular e linear. Os exemplos apresentados de *robots* omnidireccionais são os *robots* usados pela equipa 5DPO da Faculdade de Engenharia da Universidade do Porto (FEUP) para a *Small Size League* e os *robots* usados pelas Equipas da *Middle Size League* 5DPO da FEUP [CSM+03], Minho Team da Universidade do Minho [RBS+03] e Cambada da Universidade de Aveiro[AAB+04].

2.2.1 *Robots* omnidireccionais da equipa 5DPO (*Small size league*)

A equipa 5DPO evoluiu de *robots* diferenciais para omnidireccionais, obtendo melhores resultados a nível de tempo de aproximação à bola sendo o embate directo em jogo favorável para a equipa com omnidireccionais face aos diferenciais (Figura 2.6).

O posicionamento de estes *robots* é estimado com base em visão externa[Cos99], não tendo *encoders* para poder efectuar o cálculo da odometria. O cálculo da

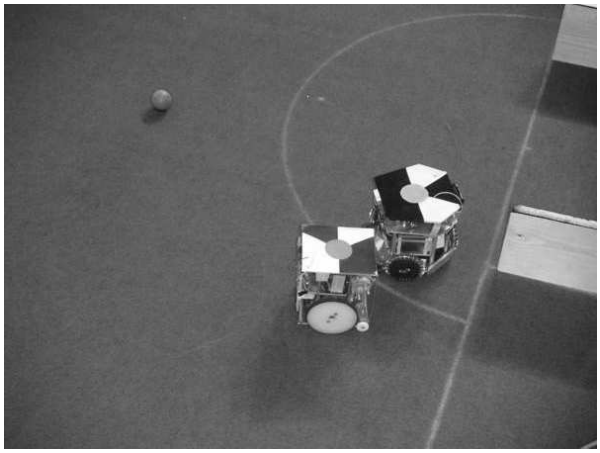


Figura 2.6: *Robot* omnidireccional vs *robot* diferencial (*Small size league*)

odometria apresenta-se como uma boa solução para estimar o posicionamento do *robot* quando este se torna temporariamente autónomo face a sensores externos, sendo este corrigido pela câmara posteriormente. Uma outra alternativa para estimar o posicionamento seria a utilização de sensores de ultra-sons [MdC02].

2.2.2 Exemplos de *Robots* omnidireccionais da *Middle size league*

Uma das grandes melhorias da *Minho Team* foi a substituição de *robots* diferenciais para *robots* omnidireccionais, com o objectivo de diminuir o tempo de aproximação à bola[RMS+04]. O *robot* desenvolvido, consiste num drive de três rodas omnidireccionais representado na figura 2.7, desfasadas igualmente de 120º.

Esta é a configuração típica usada quando se quer que o *robot* se mova em todas as direcções, mas não sendo condição necessária para garantir a omnidireccionalidade pois os ângulos de esfasamento entre rodas podem ser diferentes de 120º, no entanto isso trás algumas desvantagens pois gera um desiquilíbrio nos binários dos motores, o que pode levar a um esforço muito grande de um motor em relação aos outros dois. Também não é necessário que o *robot* seja feito com três rodas pois uma configuração com quatro rodas continua a garantir

Figura 2.7: Sistema omnidireccional Minho Team

a omnidireccionalidade, permitindo obter-se mais força e velocidade ao efectuar trajectórias rectilíneas com velocidade angular nula. Um exemplo desta configuração é apresentado na figura 2.8, sendo um dos *robots* usados pela equipa 5DPO na *middle size league*.

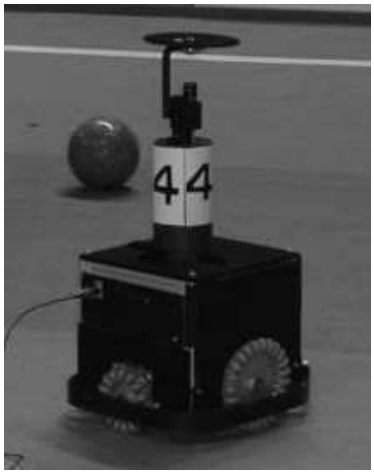


Figura 2.8: *Robot* omnidireccional da equipa 5DPO (*Vigilante*)

Também com uma configuração de três rodas, está representado na figura 2.9, um *robot* da equipa Cambada da Universidade de Aveiro [AAB+04].

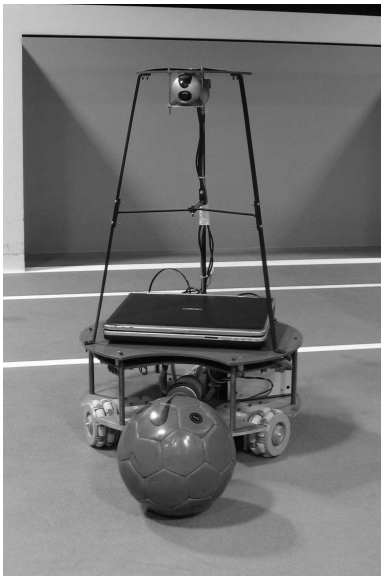


Figura 2.9: *Robot* omnidireccional da equipa Cambada

2.3 Cadeira de rodas omnidireccional (*Omni Project*)

Apesar de quase todas as aplicações apresentadas serem para utilização no futebol robótico, há uma preocupação em aplicar estas tecnologias em situações que melhorem a qualidade da vida das pessoas, tal foi alcançado no *Omni Project* [op04]. Este projecto contribuiu com uma cadeira de rodas omnidireccional (figura 2.10), possuindo um controlo intuitivo e sem conter algumas das limitações de locomoção das cadeiras de rodas clássicas.

2.3.1 Necessidades de mercado

É necessário compensar as deficiências das pessoas com ferramentas que lhes permitam uma maior manobrabilidade, para que se tornem o mais independentes possível, diminuindo a alocação excessiva de recursos humanos e materiais, contribuindo também para aumentar a sua auto-estima. O sistema inclui sensores para evitar colisões, uma interface amigável entre a cadeira e a pessoa, um sistema de controlo e um módulo de segurança adaptável a diferentes indíviduos.

Figura 2.10: Cadeira de rodas omnidireccional

2.3.2 Aspectos Inovadores

Os requisitos que impulsionam o desenvolvimento são a necessidade de intuitividade, mobilidade, segurança e baixo consumo. Também é necessário que estes protótipos não atinjam um preço proibitivo, tendo que ser robustos e de fácil manutenção. Outro objectivo essencial será a integração de diferentes módulos, estando estes harmonizados. Um dos aspectos mais inovadores deste projecto é o facto de o controlo ser muito intuitivo, tendo sensores de proximidade de ultra-sons e infra-vermelhos. A cadeira desvia-se de obstáculos em tempo real, existindo uma auscultação do meio através de vários sensores.

Capítulo 3

Protótipo realizado usando o *kit Lego Mindstorms*

3.1 Introdução

Esta secção descreve como pode ser desenvolvido rapidamente um *robot* omnidireccional para fins didácticos, utilizando o *Kit Lego Mindstorms*, o qual apresenta um papel importantíssimo na prototipagem [FFH02], tendo como vantagens modularidade, conectividade de peças e permitindo reutilização das mesmas.

A diversidade de áreas envolvidas (mecânica, electrónica, controlo...) ilustra bem a interdisciplinaridade da robótica, tornando-a cada vez mais importante a nível didáctico. As pessoas que estão a dar os primeiros passos na robótica, em especial na robótica móvel, devem usar ferramentas que lhes permitam aprender rapidamente alguns conceitos base com relativa facilidade e com grandes índices de motivação. Neste aspecto o *Kit Lego Mindstorms* apresenta um papel primordial.

3.2 Porquê usar *Lego Mindstorms*

Entende-se genericamente por *Lego* um conjunto de peças de plástico para construção de modelos mecânicos, sendo a palavra *Lego* uma marca registada [leg04]. Quase todos nós fizemos construções de *Lego*, o que os torna uma ferramenta com a qual estamos muito familiarizados, sendo possível com essas peças efectuar rapidamente diferentes configurações de *robots*. Esta facilidade apresenta-se como uma motivação extra para as pessoas que estão a dar os primeiros passos no mundo da robótica, em especial no mundo da robótica móvel. As peças *Lego* permitem conectividade, suprimindo a necessidade de usar parafusos ou cola, o que torna bastante limpo fazer construções. Para a situação particular abordada esta vantagem não foi totalmente aproveitada, dado não ser fácil obter conectividade entre peças *Lego* com ângulos diferentes de múltiplos de 90º, sendo condição necessária para a construção de um *robot* omnidireccional de três rodas. Também se torna uma ferramenta ecológica pois apesar de o plástico não ser um material fácil de reciclar, as peças *Lego* nunca são inutilizadas, consequentemente nunca constituem um desperdício, podendo ser usadas ano após ano. Em suma os *Lego Mindstorms* são uma boa ferramenta educacional para o estudo de alguns dos princípios da robótica. É uma ferramenta reutilizável, modular, podendo a mesma peça ser diferentes coisas consoante a aplicação que lhe queiramos dar, motivando muito as pessoas que a usam e sendo apresentada a um custo relativamente acessível.

3.3 Desenvolvimento de *Hardware*

O protótipo realizado consiste num *robot* omnidireccional realizado com o *Kit Lego MindStorms*, estando representado na figura 3.1. A inclusão de *encoders* foi um elemento muito importante, pois permite estimar o posicionamento absoluto do *robot* com base na odometria. Os sensores e actuadores usados são descritos nas subsecções seguintes.

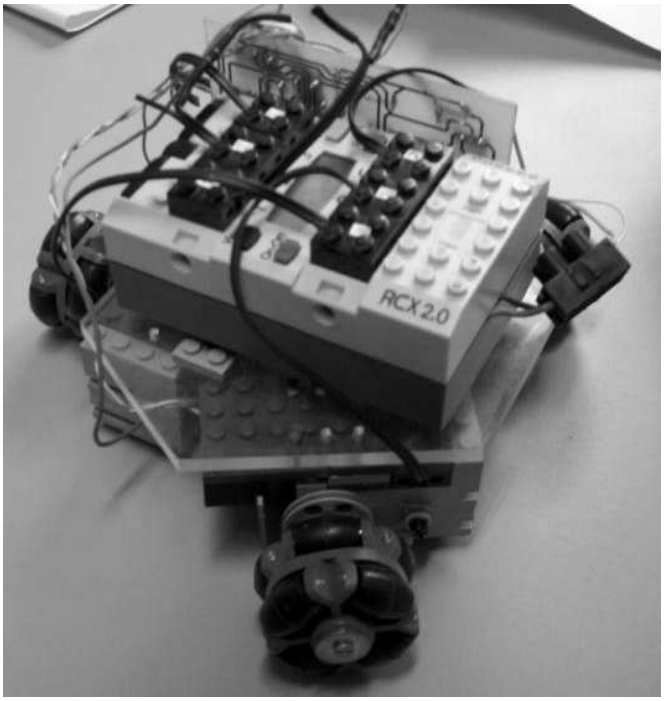


Figura 3.1: Protótipo realizado usando o *Kit Lego Mindstorms*

3.3.1 Sensores

O sensor usado para efectuar a actualização da odometria são *encoders* incrementais baseados no esquema eléctrico da figura 3.2, sendo realizados especificamente para este protótipo, pois não são disponibilizados com o *Kit Lego Mindstorms*.

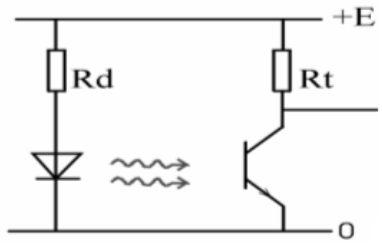


Figura 3.2: Circuito usado para o *encoder*

Sempre que interrompido o sinal de infravermelhos, o foto transístor deixa de estar polarizado, suprimindo-se a corrente na resistência Rt, consequentemente aparece uma tensão igual à da alimentação (E) na saída[PAW91], funcionando o

sensor como um interruptor electrónico. O sinal de saída no colector do foto transístor é apresentado na figura 3.3, passando posteriormente por um comparador com histerese, convertendo-o numa onda quadrada.

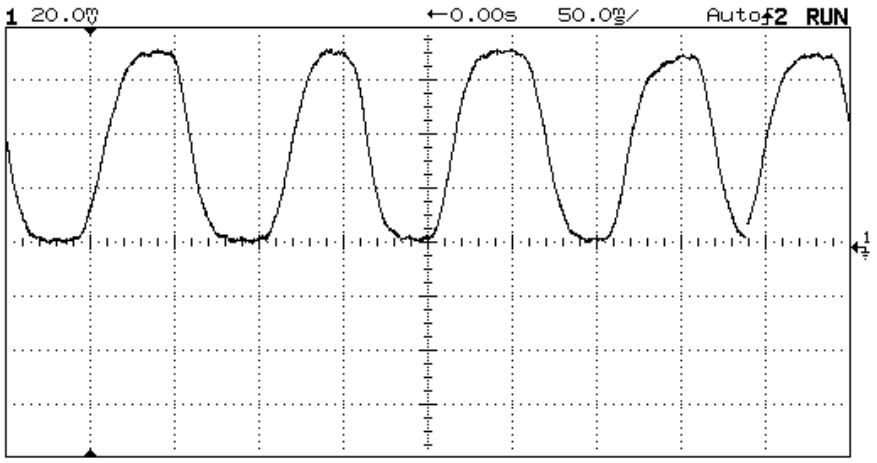


Figura 3.3: Sinal no colector do foto transístor

Associado ao veio do motor encontra-se uma folha de acetato representada na figura 3.4, a qual permite interromper o sinal de infravermelhos. A cada transição registada corresponde um deslocamento angular.

Figura 3.4: Folha de acetato associada ao veio do motor

As transições registadas, são convertidas em velocidade linear do ponto associado às rodas (D/T), estando representada a distância percorrida por D e o tempo de amostragem por T. Para o efeito multiplica-se as transições (C) por um factor (K) que permite saber a velocidade, usando-se uma aproximação de primeira ordem (assume-se que a velocidade é constante durante um período de amostragem).

Esta operação está representada na equação 3.1.

$$\frac{D}{T} = K.C \tag{3.1}$$

O "factor"converte transições em velocidade linear, seguindo várias etapas. Em primeiro lugar as transições são convertidas em deslocamento linear, correspondendo 12 a um deslocamento de *Pi*Diâmetro da Roda*. Após a obtenção do deslocamento este é divido pelo tempo durante o qual foram lidas as transições, obtendo-se velocidade linear, tal como é explicitado na equação 3.2.

$$factor = \frac{\pi.diam}{(CV.T)} \tag{3.2}$$

- CV-número de transições por volta
- *diam*-diâmetro da roda.

Motivação para usar um comparador com histerese

É muito comum em robótica móvel necessitar-mos de comparar um sinal analógico com uma tensão de *threshold*, para decidir se o sinal analógico é maior ou menor que esta tensão. Um dos problemas dos comparadores é o sinal a comparar ter normalmente ruído, consequentemente quando se aproxima muito do valor da tensão de *threshold* as flutuações do sinal podem tornar o sistema instável. Para prevenir esta situação pode ser usado um comparador com histerese. Um circuito típico com um comparador é mostrado na figura 3.5. A saída pode assumir dois valores distintos, dependendo da variável de Vin ser maior ou menor que o valor de *threshold* ($V+$). A realimentação positiva é usada para reforçar a escolha do estado da saída, pois nesta configuração chamada *Schmitt trigger* podemos ter dois *threshold*, dependendo do estado da saída. Desta maneira qualquer ruído menor que a diferença de Vh por Vl, não afectará a operação do comparador. O funcionamento do *Schmitt trigger* está ilustrado na figura 3.6.

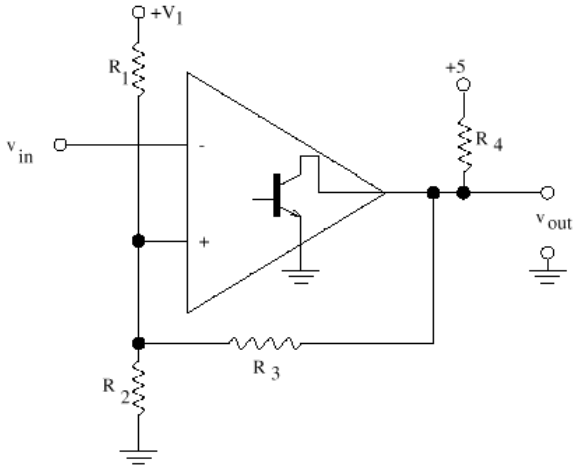


Figura 3.5: *Schmitt trigger*

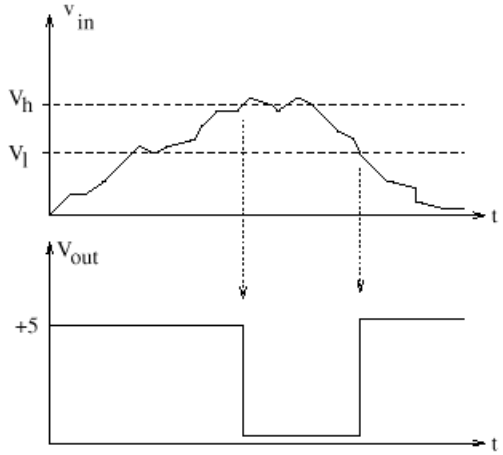


Figura 3.6: Exemplos dos sinais do Schmitt trigger versus tempo

3.3.2 Actuadores

Os actuadores usados para o Protótipo são motores DC de 9 Volt (figura 3.7), tendo caixa redutora interna. A caixa redutora (figura 3.8) permite reduzir a velocidade angular da roda, aumentando o binário disponível.

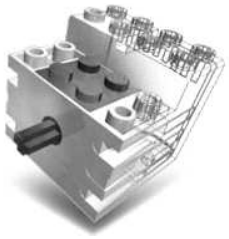


Figura 3.7: Motor DC *Lego Mindstorms*

A relação entre os binários disponíveis é dada pela equação 3.3, em que $T1$ representa o binário do primário, $T2$ o binário do secundário, $N1$ o número de dentes do primário e $N2$ o número de dentes do secundário.

$$\frac{T1}{T2} = \frac{N1}{N2} \tag{3.3}$$

Figura 3.8: Exemplo de caixa redutora

Aos motores DC estão acopladas umas rodas especiais para omnidireccionais, representadas na figura 3.9, cuja particularidade é terem pouco atrito na direcção do veio do Motor, permitindo deslizamento[LCGR02].

Figura 3.9: Roda para *robots* omnidireccionais

3.4 *Software* de controlo em *Java*

"It is amazing that they managed to squeeze a multithreading environment into what is effectively 14 kilobytes of memory. And that still leaves you with 16 kilobytes for your programs."

- Simon Ritter, Java Technology Evangelist, Sun Microsystems

3.4.1 Porquê *Java*

A escolha da linguagem de programação recaiu sobre a linguagem *Java* pelo facto de ter uma implementação da *API* bastante mais rápida que a alternativa *Not Quite C (NQC)*, tendo-se durante o projecto de desenvolvimento do *Software* de controlo do *robot* melhorado para sensivelmente 10 vezes mais rápido as funções $sin()$, $cos()$ e $sqrt()$ em relação à implementação existente, sendo o código desenvolvido por Paulo Costa [Cos03]. Também se tornou uma motivação extra usar *Java* para efectuar *Software* de Controlo, usando *Threads* para efectuar as temporizações.

O código deste programa consiste em duas *Threads* não sincronizadas (*Thread Omni e Thread Tmot*), sendo os eventos periódicos gerados colocando as *Threads* em modo *sleep* durante um tempo de amostragem subtraído do tempo gasto a efectuar os cálculos, após este tempo as *Threads* acordam e efectuam todos os cálculos e operações voltando ao modo de *sleep*, emulando-se deste modo o funcionamento de um temporarizador

3.4.2 *Thread* Omni

A *thread* principal tem como objectivo o cálculo do controlador e da odometria baseando-se na equação 2.6. Esta *Thread* tem como nome *Omni*, à qual está associado um tempo de amostragem, sendo escolhido de maneira a que as actualizações da odometria e do controlador sejam o mais rápidas possível. A escolha foi 200 ms pelo facto de após vários testes se verificar que processador demora entre 140 a 160 ms a efectuar estas duas funções. Para se ter alguma margem de manobra optou-se por um valor ligeiramente superior, majorando um pouco o tempo necessário para os cálculos.

O código base usado para esta *Thread* foi o seguinte:

```
{
while(1==1)
{
// efectuar actualização da odometria
odoUpdate();

//executar Controlador
Controlador();
}
LastTime=LastTime+deltaT;
SleepTime=LastTime-(int)System.currentTimeMillis();

if (SleepTime>0)
{
try
{ Thread.currentThread().sleep(SleepTime);}
catch (InterruptedException ie) {}
}
}
```

Controlador

O controlador tem como função definir qual a velocidade a que os motores devem rodar, de maneira a que o *robot* rode com uma determinada velocidade angular $\dot{\theta}$ e se desloque com uma velocidade linear $\dot{x}$ e $\dot{y}$

O controlador foi optimizado desenvolvendo as funções $sin(\theta + \frac{\pi}{3})$, $cos(\theta + \frac{\pi}{3})$, $cos(\theta - \frac{\pi}{3})$ e $sin(\theta - \frac{\pi}{3})$, usando as formulas:

$$\sin(a+b) = \sin(a)\cos(b) + \cos(a)\sin(b) \tag{3.4}$$

$$\sin(a-b) = \sin(a)\cos(b) - \cos(a)\sin(b) \tag{3.5}$$

$$\cos(a+b) = \cos(a)\cos(b) - \sin(a)\sin(b) \tag{3.6}$$

$$\cos(a-b) = \cos(a)\cos(b) + \sin(a)\sin(b) \tag{3.7}$$

O objectivo de desenvolver estas funções trigonométricas foi o de o controlador se tornar computacionalmente menos pesado, pois é muito menos dispendioso a nível de tempo de cálculo calcular apenas $sin(\theta)$ e $cos(\theta)$, que efectuar o cálculo de vários senos e cosenos, tal como é apresentado na equação 2.6. Consequentemente para o cálculo das velocidades V_2 e V_3 torna-se vantajoso efectuar as alterações acima referidas.

Assim para o cálculo de V_2, torna-se computacionalmente mais eficiente usar a equação 3.9 em vez da equação 3.8.

$$v2 = -sin(\frac{\pi}{3} - \theta)\dot{x} - cos(\frac{\pi}{3} - \theta)\dot{y} + L\dot{\theta} \tag{3.8}$$

$$\begin{aligned} v2 = &-(sin(\frac{\pi}{3})cos(\theta) - cos(\frac{\pi}{3})sin(\theta))\dot{x} - \\ &(cos(\frac{\pi}{3})cos(\theta) - sin(\frac{\pi}{3})sin(\theta))\dot{y} + L\dot{\theta} \end{aligned} \tag{3.9}$$

O mesmo se verifica para o calculo de V_3 tornando-se mais eficiente usar a equação 3.11 que a 3.10

$$v3 = -sin(\frac{\pi}{3} + \theta)\dot{x} - cos(\frac{\pi}{3} + \theta)\dot{y} + L\dot{\theta} \tag{3.10}$$

$$\begin{aligned} v3 = &(sin(\frac{\pi}{3})cos(\theta) + cos(\frac{\pi}{3})sin(\theta))\dot{x} - \\ &(cos(\frac{\pi}{3})cos(\theta) - sin(\frac{\pi}{3})sin(\theta))\dot{y} + L\dot{\theta} \end{aligned} \tag{3.11}$$

Deste modo é usado o seguinte código para o controlador, pré calculando-se apenas um coseno e um seno, sendo tudo o resto constantes.

Código usado:

```
//v_teta - velocidade angular do Robot
//v_x - velocidade linear do robot segundo o eixo dos X
//v_y - velocidade linear do Robot segundo o eixo dos Y
//cos_pi3 - constante

//Pré calcula-se o cos e sin de teta
cos_teta=Math.cos(teta);
sin_teta=Math.sin(teta);

v1=-sin_teta*v_x+cos_teta*v_y+L*v_teta;
v2=-(sin_pi3*cos_teta-cos_pi3*sin_teta)*v_x-(cos_pi3*cos_teta-sin_pi3*sin_teta)*v_y+L*v_teta;
v3=(sin_pi3*cos_teta+cos_pi3*sin_teta)*v_x-(cos_pi3*cos_teta-sin_pi3*sin_teta)*v_y+L*v_teta;
```

Odometria

O posicionamento do *robot* é calculado baseando-se na odometria. A odometria consiste em usar a medida da velocidade de cada roda para estimar a posição do *robot* [BEF96], apresentando como vantagem o facto de o *robot* se poder tornar completamente autónomo, estimando o posicionamento absoluto, mas apresentando como desvantagem o crescente acumular de erros, os quais não podem ser corrigidos a não ser que existam outros sensores que forneçam informação relativamente ao posicionamento absoluto do *robot*.

Para o cálculo da odometria foi usado o seguinte código:

```
  public static void odoUpdate()
  {
//ler as transições que cada roda andou
Odo1=FloydMotors.OdoCount1;
Odo2=FloydMotors.OdoCount2;
Odo3=FloydMotors.OdoCount3;

//saber quantas transições andou durante
//um tempo amostragem de 0.2 segundos (T)
do1=Odo1-lastOdo1;
do2=Odo2-lastOdo2;
do3=Odo3-lastOdo3;

//actualizar a leitura
lastOdo1=Odo1;
lastOdo2=Odo2;
lastOdo3=Odo3;

//converter transições em velocidade
va=factor_roda*do1;
vb=factor_roda*do2;
vc=factor_roda*do3;

//calcular as velocidade no sentido do eixo dos x,y e teta
v_x = F2 (va,vb,vc);
v_y = F1(va,vb,vc) ;
v_teta = F3(va,vb,vc);
// com a aproximação de primeira ordem efectuar o cálculo da nova posição do Robot
x=x+v_x*T;
y=y+v_y*T;
teta=teta+v_teta*T;
  }
```

3.4.3 *Thread* Tmot

A *Thread Tmot* tem como objectivo actualizar a contagem de transições recebidas dos *encoders* e actualizar a velocidade dos motores. O seu período de amostragem foi escolhido com base na frequência máxima a que podem ser recebidos as transições geradas pelos *encoders* e a restrição relativa ao tempo que é necessário para

processar todos os cálculos, tendo este que ser majorado. Os cálculos da *Thread* Tmot demoram sensivelmente 15 ms, logo o intervalo de tempo entre acordar esta *Thread* tem que ser superior a este valor. O tempo escolhido foi 20 ms, pois permite que sejam feitos os cálculos, não sendo perdidas transições. Para que não sejam perdidas transições o tempo deve ser inferior ao tempo mínimo entre transições, sendo este da ordem dos 40 ms, como pode ser observado na figura 3.3, na qual a roda se encontra à velocidade máxima o que implica um tempo de intervalo entre transições mínimo. O funcionamento da contagem de transições está representada pela rede de petri da (figura 3.10).

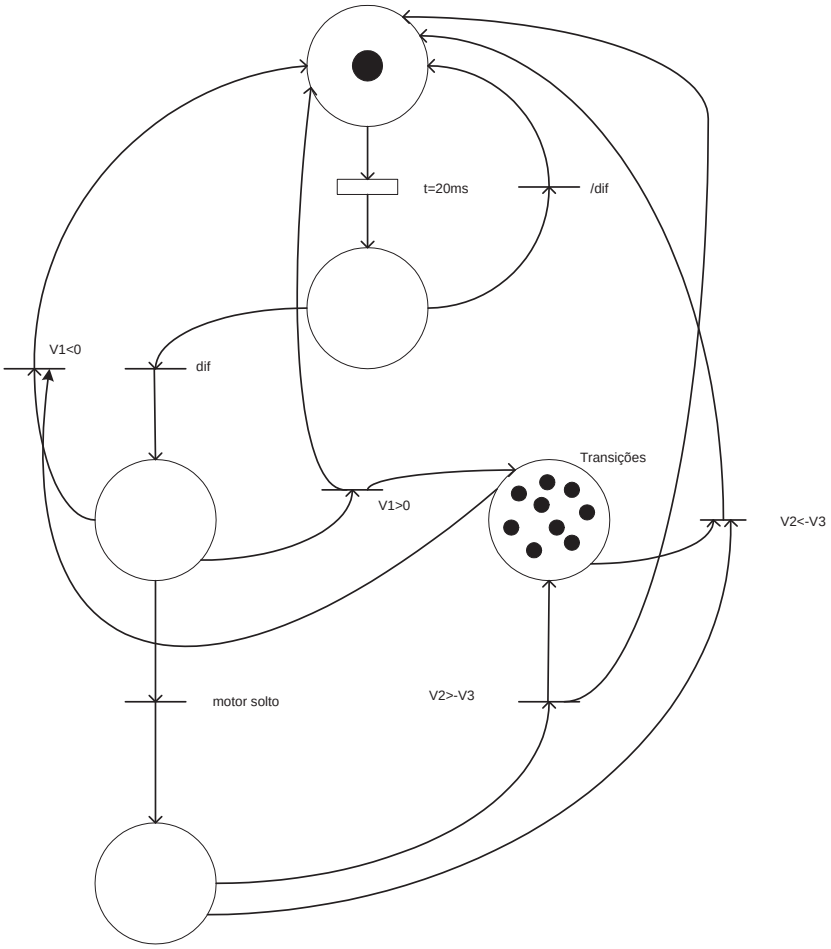


Figura 3.10: Rede de petri relativa à actualização das transições dos *encoders*

Periodicamente verifica-se se houve transição (dif). Caso haja transição e a referência de velocidade do motor seja maior que zero incrementamos o contador ou então decrementamos caso seja menor que zero. Na situação de o motor não estar travado a sua velocidade pode ser maior ou menor que zero, pode-se estimar em que sentido o motor roda baseando-nos na velocidade das outras duas rodas. Para além disso também é necessário que existam acelerações e desacelerações suaves pois o sinal das transições é registado com base na referência de velocidade e não na velocidade real. O motor pode por exemplo estar a frenar com velocidade positiva e com referência de velocidade negativa, sendo as transições registadas como negativas erróneamente. O código que faz esta operação é o seguinte:

```
public void run()
  {
  while (Active)
{//actualiza transições recebidas de um dos encoders
if(Sensor.S1.readBooleanValue()!= Odo1)
{
Odo1=!Odo1;
if(SpeedMotor1>0) OdoCount1++;
if(SpeedMotor1<0) OdoCount1++;
if(SpeedMotor1==0 & SpeedMotor2>-SpeedMotor3)
{
OdoCount2++;
}
if(SpeedMotor1==0 & SpeedMotor2<-SpeedMotor3)
{
OdoCount1--;}
}
LastTime=LastTime+20;
SleepTime=LastTime-(int)System.currentTimeMillis();
if (SleepTime>0)
try
{sleep(SleepTime);}
catch (InterruptedException ie) {}
}
}
}
```

3.5 Validação do controlador dos motores

O controlador apresenta uma falha que limita muito o desempenho do *robot*, com vista a alcançar um determinado objectivo, que é o facto de ser extremamente difícil efectuar o controlo de velocidade dos motores pois a função *Motor.A.setPower(x)* apresentava uma variação muito pequena de velocidade para diferentes parâmetros de entrada, tal como é explicitado na figura 3.11, não sendo linear a atribuição de um parâmetro vs velocidade alcançada. Devido a este facto optou-se pela utilização do *Floyd-Steinberg Algorithm* [FS04], para efectuar a modulação de largura de impulsos, sendo este algoritmo recursivo. Desta maneira consegue-se melhorar a linearidade da correspondência da velocidade aos parâmetros de entrada, tal como é apresentado na figura 3.12, sendo este algoritmo comummente usado em processamento de imagem.

O código usado para este algoritmo foi o seguinte:

```
ErrorMotorA=ErrorMotorA+SpeedMotorA;
if (ErrorMotorA>MOT_RANGE/2)
{
//coloca-se o motor a andar.
Motor.A.forward();
//actualiza-se o erro
ErrorMotorA=ErrorMotorA-MOT_RANGE;
}
else if (ErrorMotorA<-MOT_RANGE/2)
{
Motor.A.backward();
ErrorMotorA=ErrorMotorA+MOT_RANGE;
}
else
{
Motor.A.stop();
Motor.A.flt();
}
```

Este método não é o melhor para efectuar controlo de velocidade, dado o controlo não ser efectuado em malha fechada. Uma alternativa válida para o controlo em malha aberta é a utilização de motores passo a passo [pas04].

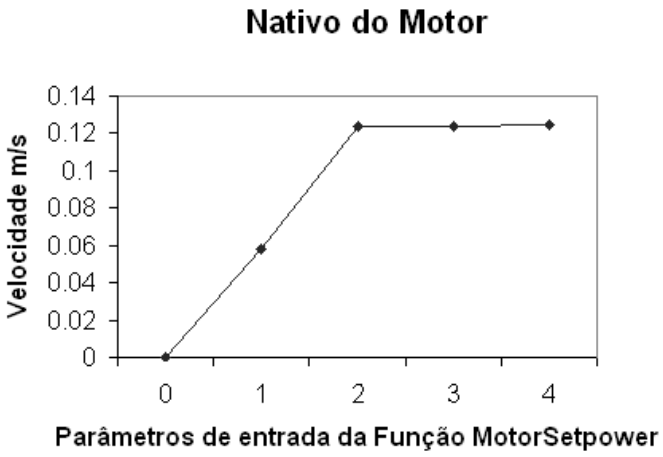


Figura 3.11: Resultados obtidos usando o método nativo da API do lejos

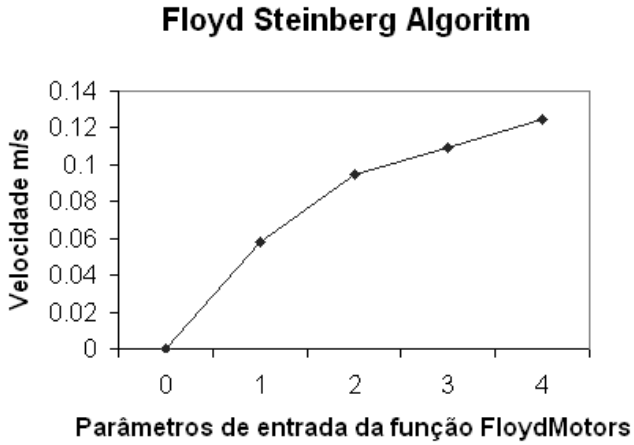


Figura 3.12: Resultados obtidos usando o algoritmo de Floyd-Steinberg

3.6 Calibração da odometria

Os testes e calibração da odometria foram efectuados usando-se um transferidor colado ao chão, representado na figura 3.13.

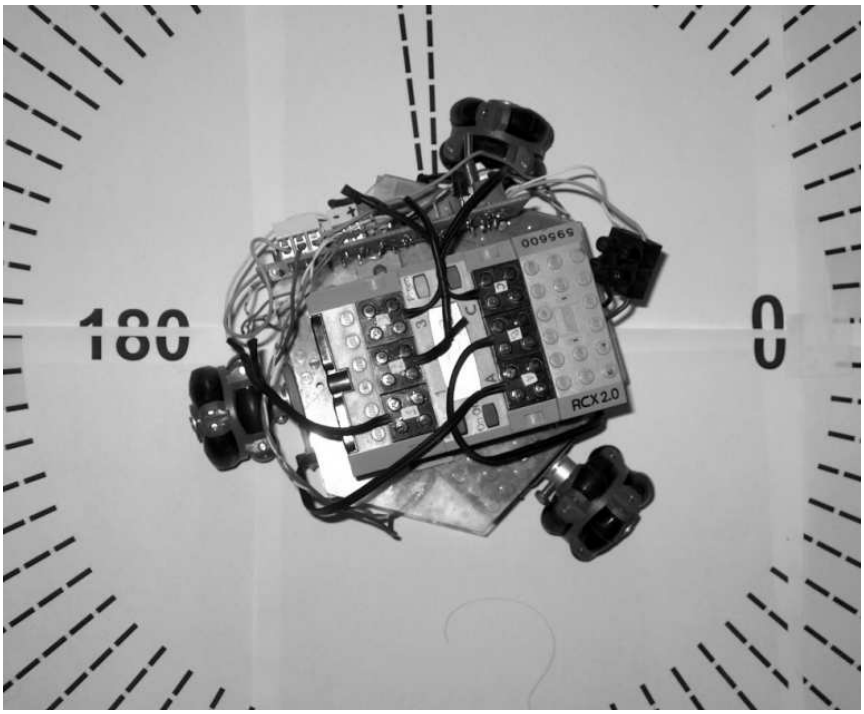


Figura 3.13: Transferidor colado ao chão para calibração e testes de odometria

Para se efectuar a calibração da odometria é necessário dar valores aos parâmetros que relacionam o número de transições que cada roda fornece com a distância percorrida pela roda. Sendo estes valores diferentes de uma roda para a seguinte, o procedimento para a calibração consiste em primeiro lugar realizar o cálculo da equação 3.2, com o objectivo de nos aproximar-mos do valor pretendido. Após termos efectuado o cálculo da equação 3.2, obtendo-se um factor que converte transições em distância percorrida, procede-se a uma afinação do mesmo, fazendo com que cada uma das rodas percorra distâncias, as quais são comparadas com as fornecidas pela fórmula. O factor é variado até se obterem valores que estejam o mais próximos possível. Para se efectuar a calibração do parâmetro L coloca-se o *robot* a rodar sobre si mesmo, comparando o ângulo

estimado pelos cálculos da odometria e o medido com um transferidor. Quando estes valores estiverem próximos podemos considerar que o parâmetro se encontra calibrado.

O erro dado pela odometria pode ser reduzido alterando-se o *factor roda* representado na equação 3.2, o qual converte transições em velocidade, ou ajustando de novo o parâmetro L representado na figura 2.3. Apesar deste erro poder ser minimizado nunca pode ser eliminado, pois existem várias fontes de erro tais como escorregamento da roda, o erro do *encoder*, a aproximação de primeira ordem para a velocidade, o facto de o parâmetro L variar dinamicamente e o facto de este erros serem cumulativos.

3.7 Conclusões

É de realçar que foi desprezada a dinâmica para o cálculo do controlador pois opera-se com velocidades baixas e com uma caixa com grande redução, tendo o motor caixa redutora interna.

Existem situações em que efectuar o cálculo da velocidade de cada roda não é directo pois quando um motor não está travado não existe informação relativa ao sentido de rotação, simplesmente são recebidas transições, as quais são convertidas em velocidade, sendo o sentido de rotação estimado a partir do conhecimento da referência de velocidade das outras duas rodas, tal como exemplificado na rede de petri da figura 3.10.

O estimação do posicionamento do *robot* a partir da odometria foi um dos objectivos finais deste trabalho, sendo um método muito usado sempre que um veículo é ou se tem que tornar temporariamente autónomo, sendo o posicionamento muito importante para as missões dos *robots*. O cálculo da odometria não deve ser o único método a usar para estimar a localização absoluta ou relativa de um *robot* pois os erros são cumulativos, tornando-se cada vez maiores. Torna-se um método muito útil quando existe fusão de dados de várias fontes, podendo-se

usar um filtro de *Kalman* [SCM04], não ficando o sistema dependente de apenas um sensor externo ou interno.

Apesar de o *hardware* ser muito limitado, principalmente a nível de robustez, torna-se muito aliciante fazer *robots* com os *kit Lego Mindstorms*, existindo um inúmero conjunto de conceitos que podem ser explorados usando este tipo de tecnologia.

Capítulo 4

Protótipo realizado com estrutura de alumínio

4.1 Introdução

Esta secção descreve como foi desenvolvido um protótipo de um *robot* omnidireccional com estrutura de alumínio, sendo esta bastante mais robusta que a estrutura usada para o protótipo realizado com o *kit Lego Mindstorms*, mas perdendo-se a vantagem relativa à prototipagem rápida, que é característica pelas peças *Lego*, o que torna o processo de aprendizagem mais demoroso [RMN00]. Mecanicamente o *robot* possui três rodas motrizes para omnidireccionais desfasadas de 120º (figura 4.1), permitindo movimentos em todas as direcção com velocidade controlada. O sistema de accionamento é constituído por três motores de corrente contínua com caixa redutora. Para se efectuar o controlo da velocidade de cada uma das rodas é usada modulação de largura de impulsos (PWM), controlando-se independentemente cada uma das rodas. O controlo dos motores é feito em malha fechada utilizando um micro-controlador *AVR* programado em *C* e um *PC* com uma aplicação em *Delphi*, comunicando entre si por *RS232*. O esquema eléctrico do *hardware* desenvolvido encontra-se em Anexo na figura 6.1.

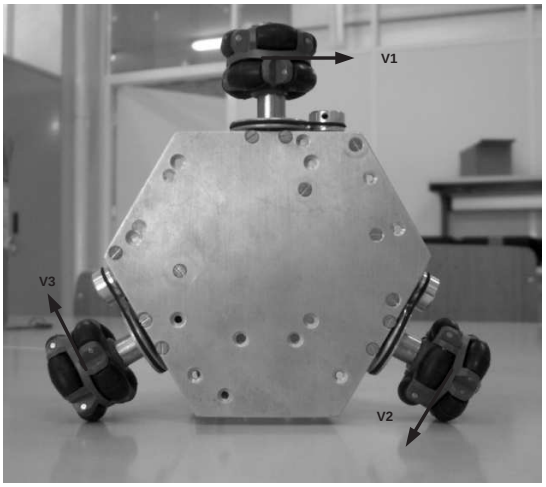


Figura 4.1: Vista de baixo do *robot* omnidireccional desenvolvido

O odometria e o controlador são calculados no *PC* comunicando este com o micro-controlador. O micro-controlador envia para o *PC* as transições dos *encoders* relativas a cada roda, com estes dados é estimada a posição absoluta do *robot*. Após ser calculado o posicionamento baseado na odometria é calculado o controlador, tendo como parâmetros de saída a velocidade a que devem rodar os motores para que o *robot* alcance um determinado objectivo. O diagrama de blocos do sistema está representado na figura 4.2

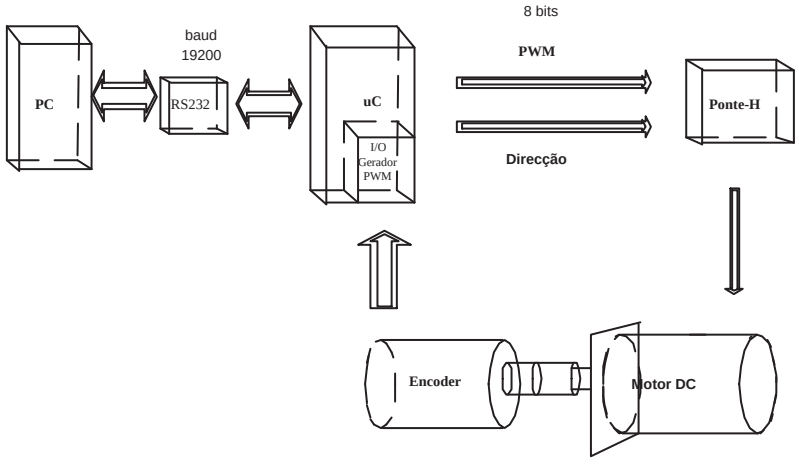


Figura 4.2: Diagrama de blocos do sistema

4.2 Descrição de *hardware*

4.2.1 *Encoder* Incremental

A função dos *encoders* incrementais é de obter a velocidade de rotação dos motores num dado instante, podendo-se com este dado estimar o posicionamento do *robot* a partir das equações de odometria e permitir também o controlo em malha fechada da velocidade do motores. Os *encoders* incrementais escolhidos são da OMRON com a referência E6A2-CW5C [Omr02], estando representados na figura 4.3. Estes *encoders* incrementais apresentam uma resolução de 100 transições por rotação, tendo saídas em colector aberto. A resposta em frequência destes *encoders* é de 20 KHz e a rotação máxima situa-se nos 5.000 rpm, sendo muito acima da velocidade de rotação máxima dos motores. O *encoder* possui duas fases as quais permitem saber em que direcção está a rodar o motor.

Figura 4.3: *Encoder* incremental

4.2.2 *Drives* dos Motores

Para comandar o motor, utilizou-se um módulo que converte o valor de PWM com níveis TTL obtidos a partir do micro-controlador (andar lógico) na tensão aos terminais dos motores (andar de potência). O drive escolhido foi o L6204[MIC94], estando representado na figura 4.4. O L6204 contém 2 canais que são utilizados como *drives* para aplicações de controlo de motores DC, passo a passo ou outras cargas indutivas. Embora a topologia da ponte em H seja igual, o seu controlo é efectuado de maneira diferente, consoante a aplicação.

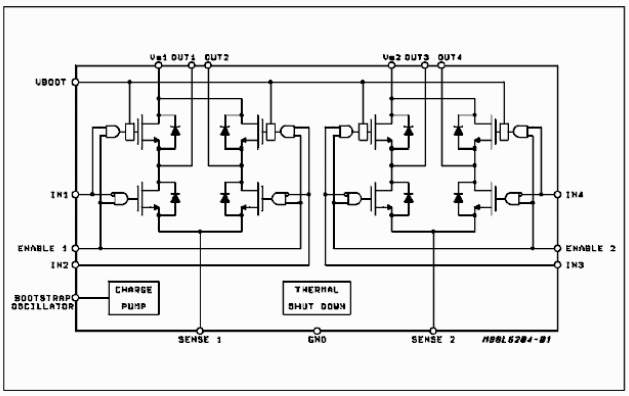


Figura 4.4: *Drives* dos motores

4.2.3 Micro-controlador

O controlo do *robot* é feito com um micro-controlador ATMEGA8 [Cor02] da *Atmel*. O ATMEGA8 é um micro-controlador de 8 bits com arquitectura *RISC*, logo é um micro-controlador que reconhece um número relativamente limitado de instruções, sendo estas executadas muitos rapidamente, dado serem funções muito simples. Este controlador tem 8 KBytes de memória FLASH, sendo este um tipo especial de memória, que tem a particularidade de poder ser apagada e reprogramada em blocos várias vezes sem a necessidade de ser programado um Byte de cada vez. O micro-controlador possui também três canais de PWM utilizados para controlar cada um dos motores e uma porta série programável, usada para efectuar a comunicação com o PC, onde é calculado o controlador.

A frequência de oscilação escolhida foi 14.745 Mhz, pois chega perfeitamente para efectuar as operações pretendidas, cumprindo requisitos de tempo real. Esta escolha da frequência deveu-se também ao facto de para frequências mais elevadas o valor que se coloca no registo *UBRR* (configuração do *Baud Rate*) ser muito pequeno, consequentemente ao ser aproximado às unidades, gera um erro relativo no *Baud Rate* da porta série muito significativo, o qual pode levar a perdas de sincronismo.

4.3 Dinâmica do sistema

Para controlar o sistema é necessário controlar independentemente cada uma das rodas do *robot*, as quais lhe vão imprimir movimento. O controlador tem diversas hierarquias, sendo a sua função a mais baixo nível controlar independentemente cada uma das rodas e a mais alto nível determinar a que velocidade deve rodar cada roda para que o *robot* se desloque com uma determinada velocidade angular e linear. O controlador a mais alto nível é efectuado baseando-se na equação 2.6 e num controlador de posição. A mais baixo nível cada motor é controlado em malha fechada usando-se um controlador do tipo PI. O controlo dos motores está representado na figura 4.5.

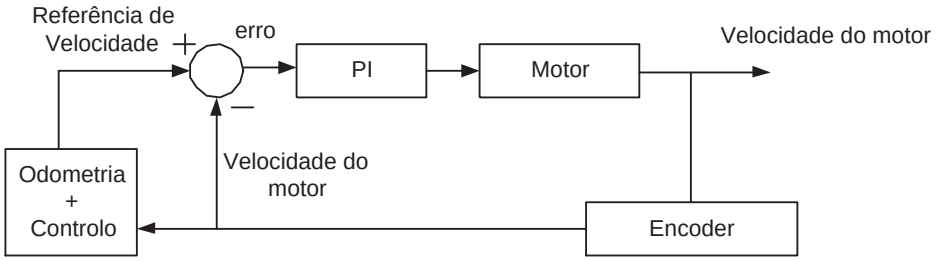


Figura 4.5: Controlo dos motores

4.3.1 Odometria

A cada roda está associado um *encoder*, este sensor permite saber em que sentido está a rodar cada roda e a que velocidade. Cada *encoder* possui dois sinais desfasados no tempo, com frequências proporcionais à velocidade angular de cada roda [LCM04], estando representados na figura 4.6. Os dois sinais desfasados permitem saber qual o sentido de rotação da roda e velocidade de uma roda, efectuando a máquina de estados representada na figura 4.7.

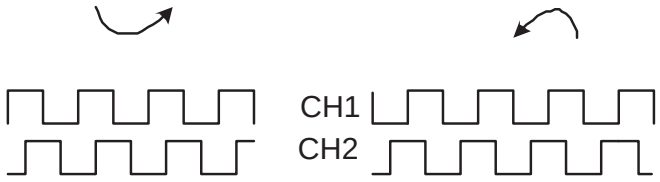


Figura 4.6: Sinais dos *encoders*

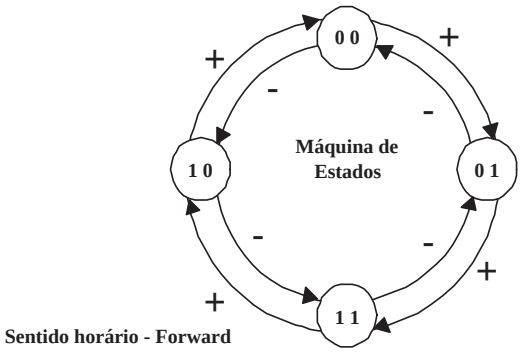


Figura 4.7: Máquina de estados

Para se efectuar este algoritmo utilizado o seguinte código:

```
void odometer()
{
//actual_state_encoder ..... estado actual dos encoders
//last_state_encoder   ..... estado anterior dos encoders
//speed_encoder++      ..... transições dos encoders
actual_state_encoder=readstate();
if (last_state_encoder==0x00)
{
if (actual_state_encoder==0x01)
{speed_encoder++;}
if (actual_state_encoder==0x02)
{speed_encoder--;}
}
if(last_state_encoder==0x01)
{
if (actual_state_encoder==0x03)
{speed_encoder++;}
if (actual_state_encoder==0x00)
{speed_encoder--;}
}
if(last_state_encoder==0x03)
```

```
{
if (actual_state_encoder==0x02)
{speed_encoder++;}
if (actual_state_encoder==0x01)
{speed_encoder--;}
}
if(last_state_encoder==0x02)
{
if (actual_state_encoder==0x00)
{speed_encoder++;}
if (actual_state_encoder==0x03)
{speed_encoder--;}
}
last_state_encoder=actual_state_encoder;
}
```

Calibração da odometria

Para se efectuar a calibração da odometria é necessário dar valores ao parâmetros que relacionam o número de transições que cada roda fornece com a distância percorrida pela roda. A maneira como é transmitido movimento da roda para o *encoder* está representado na figura 4.8. Sendo estes valores diferentes de uma roda para a seguinte, o procedimento para a calibração consiste em primeiro lugar em efectuar o cálculo da equação 4.1, com o objectivo de nos aproximar-mos do valor pretendido.

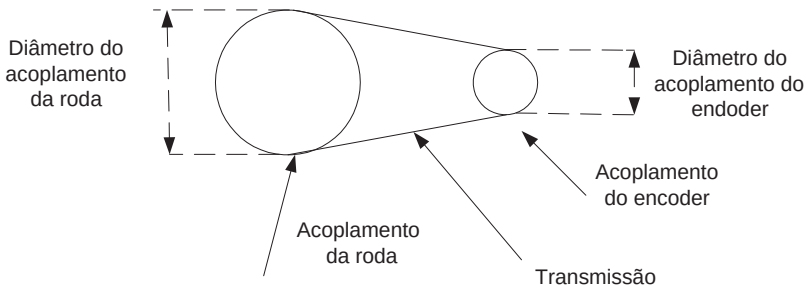


Figura 4.8: Transmissão entre a roda e o encoder

$$D = \frac{D1}{D2}.\frac{\pi.Diamr}{Impv}.Imp \tag{4.1}$$

- D Distância percorrida
- $D1$ perímetro do acoplamento do *encoder*
- $D2$ perímetro do acoplamento da roda
- $Diamr$ Diâmetro da roda
- Imp Transições registadas num Tempo de amostragem
- $Impv$ Transições registadas numa volta

Após termos efectuado o cálculo da equação 4.1 obtemos um factor que converte transições em distância percorrida, procede-se então a uma afinação deste factor para cada uma das rodas, fazendo com que percorram distâncias e comparando-as com as que a fórmula fornece.

Além de ser calibrado para cada uma das rodas um factor que converte transições em distância percorrida, é necessário calibrar também um outro factor que é o parâmetro L, estando este explicitado na figura 2.3. Para se efectuar esta calibração coloca-se o *robot* a rodar sobre si mesmo com vários valores para este parâmetro. O objectivo desta operação é comparar o ângulo estimado pelos cálculos da odometria e o medido com um transferidor, quando estes valores estiverem próximos podemos considerar que o parâmetro se encontra calibrado.

4.3.2 Controlador

O controlador tem como objectivo a deslocação do *robot* para um determinado ponto com velocidade controlada. Como parâmetros de entrada temos um objectivo que consiste em o *robot* se deslocar para um determinado ponto. Com este dado calcula-se um vector de posição que aponta para o sítio onde queremos

deslocar o *robot*, esse vector depois de normalizado é convertido num vector de velocidade, passando este a ser o objectivo a alcançar. O controlador determina que velocidade deve ter cada roda, com vista ao *robot* se deslocar com uma determinada velocidade em X, Y e em θ(caso seja necessário corrigir o ângulo). A cada tempo de amostragem a posição muda, consequentemente varia o vector de posição, o vector de velocidade e a referência da velocidade de rotação dos motores. O controlador está representado no fluxograma da figura 4.9

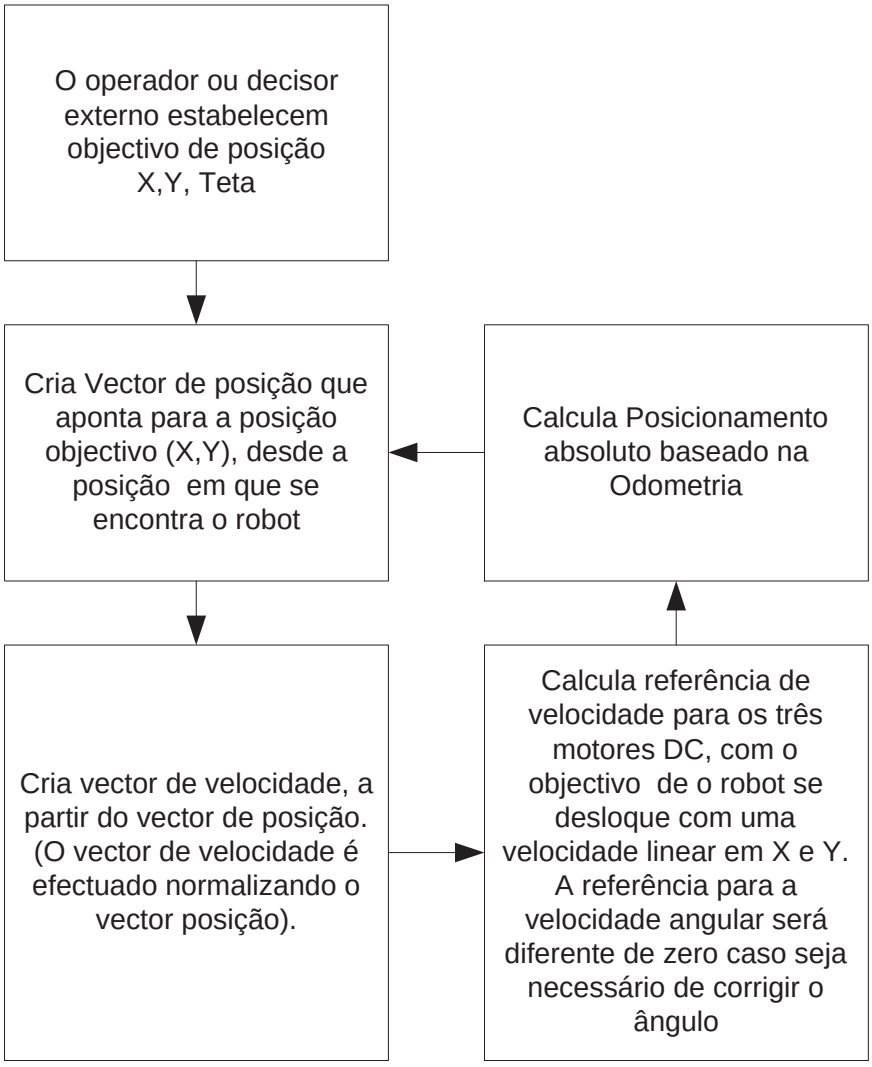


Figura 4.9: Controlador do *robot*

4.3.3 Controlo dos motores DC

Para se efectuar o controlo dos motores optou-se por se efectuar modulação de largura de impulsos com controlo bipolar. Neste tipo de controlo os interruptores (Ta+,Tb-) e (Tb+,Ta-) representados na figura 4.10, são tratados como dois pares de comutação, em que um dos pares está sempre a *on* [MUwPR95]. Apenas é

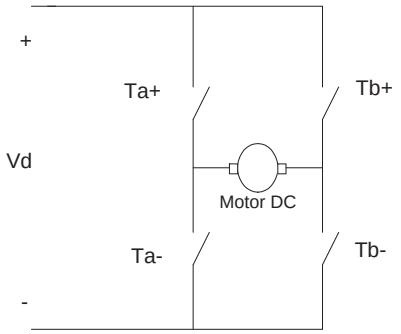


Figura 4.10: Ponte em H para controlo de um motor DC

necessário um sinal do micro-controlador para efectuar o controlo de um motor sendo este aplicado a um dos pares de comutação e a sua negação aplicada ao outro par de comutação, tal como é explicitado na figura 4.11.

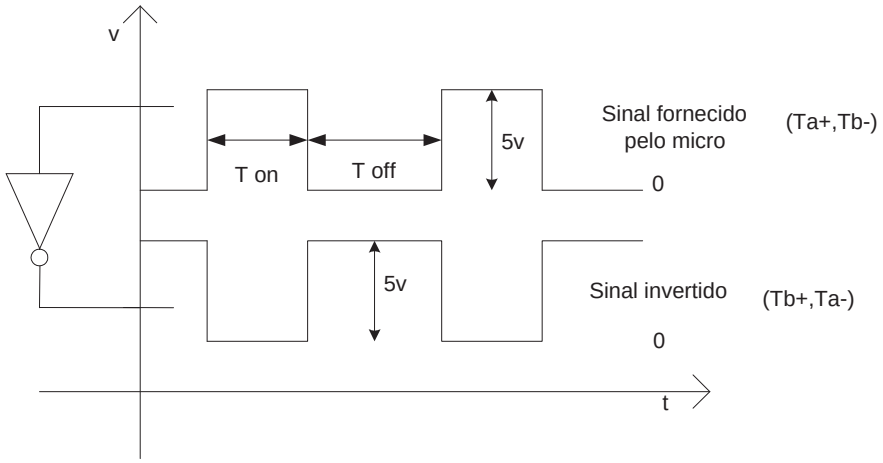


Figura 4.11: Sinais de controlo de um dos motores

Dado a carga ser indutiva e a frequência de comutação elevada face à inércia da carga, interessa-nos apenas controlar o valor médio da tensão aplicado

à carga, estando esta representada na figura 4.12.

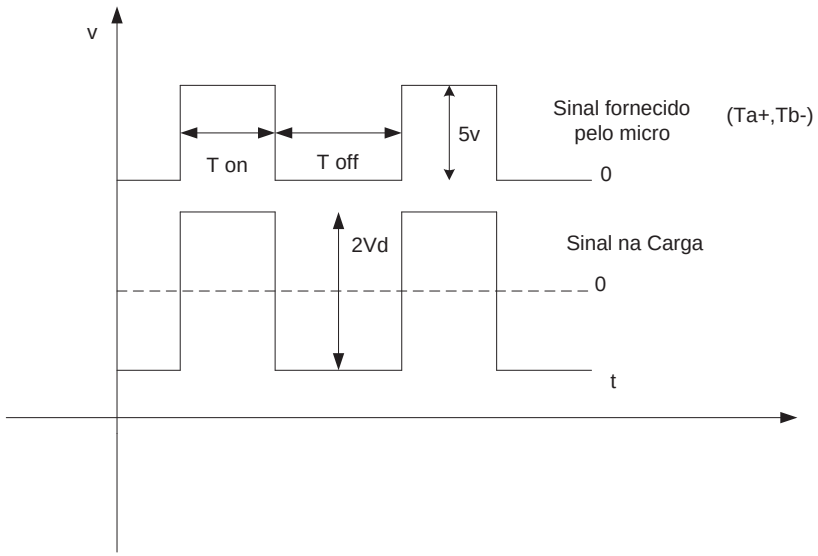


Figura 4.12: Valor de tensão aos terminais do motor Vs sinal de controlo

Se o valor médio da tensão é positivo o motor roda no sentido positivo, caso seja negativo o motor roda no sentido negativo e caso seja 0 o motor encontra-se travado.

Dimensionamento dos parâmetros dos controladores dos motores

O dimensionamento dos parâmetros dos controladores dos motores pode ser feito de diversas maneiras sendo apresentada uma solução possível, correspondendo a efectuar-se a identificação do pólo dominante do modelo do motor. Após a obtenção deste dado, efectua-se o cálculo de um controlador do tipo PI, colocando um zero ligeiramente maior que o pólo dominante e posteriormente variando o ganho até se obter a resposta requerida (isto é tornando o sistema o mais rápido possível sem que exista um *overshoot* significativo). O modelo de um motor de corrente contínua pode ser aproximado para um sistema de segunda ordem, tendo um pólo mecânico e um pólo eléctrico. O pólo mecânico é dominante, sendo o seu tempo de estabelecimento muito superior ao do pólo eléctrico. O tempo de estabelecimento da velocidade do motor em malha fechada é sensivelmente

5 vezes o valor da constante de tempo associada ao pólo dominante [Oga93]. Desconhecendo o valor dos dois pólos é possível estimar o pólo dominante por observação da resposta a um degrau unitário do motor (figura 4.13). Esta opção permite que não seja necessário saber todos os parâmetros do sistema para o dimensionamento do controlador.

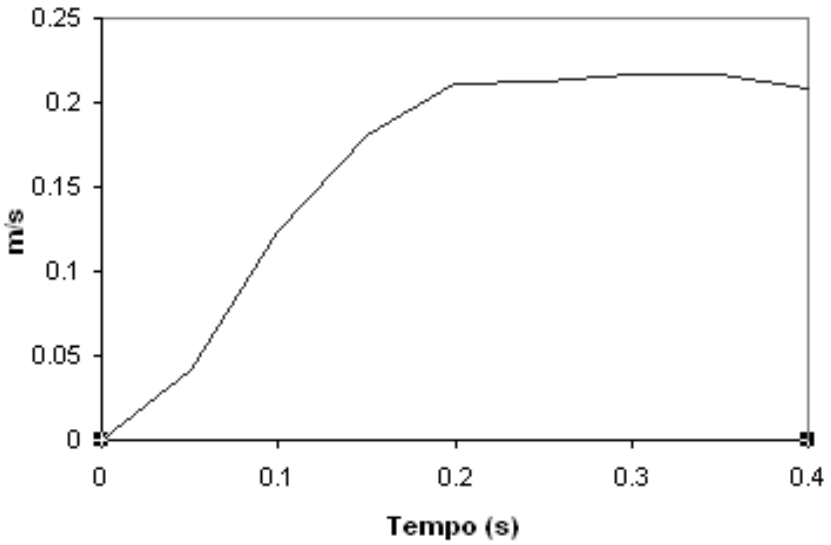


Figura 4.13: Resposta em malha aberta da velocidade da roda a um degrau unitário

Da observação dos dados da resposta ao degrau podemos constatar que o tempo de estabelecimento é sensivelmente $0,3$ segundos, sendo o pólo mecânico o inverso desta constante de tempo a dividir por cinco $(1/(0,3/5))$. Estando identificado o pólo mais lento deste sistema podemos dar início ao cálculo do controlador. O controlador usado foi um PI de maneira a ser garantido erro nulo em regime permanente, a sua função de transferência é mostrada na equação 4.2.

$$G(s) = \frac{Kp(s+a)}{s} \tag{4.2}$$

O parâmetro a tem um valor que é o dobro do valor do pólo mecânico estimado, e o parâmetro K_p será ajustado sucessivamente até se obter uma resposta

temporal que seja satisfatória, isto é com o melhor regime dinâmico possível e sem erro em regime permanente. Foram efectuados diferentes testes de maneira a se obter a resposta satisfatória em malha fechada, não sendo possível eliminar completamente o erro em regime permanente, devido às perturbações na carga. A resposta em malha fechada com uma referência de 20 cm/s, está representada na figura 4.14.

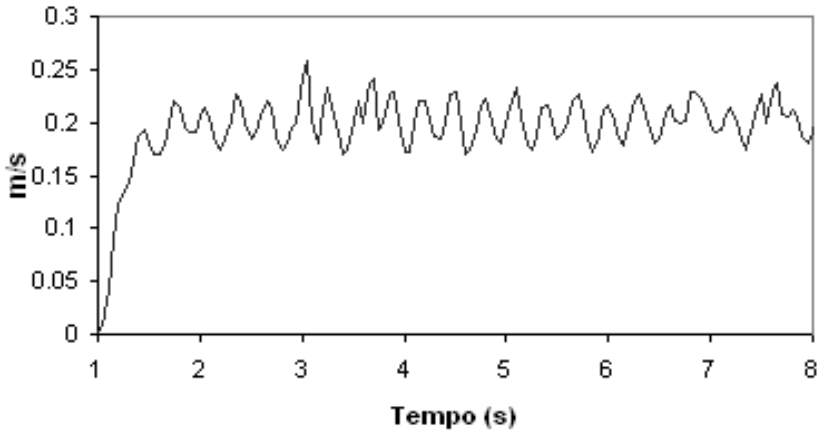


Figura 4.14: Resposta em malha fechada da velocidade com uma referência de 20 cm/s

Após obterem-se os parâmetros para um controlador analógico do tipo PI que cumpra os requisitos impostos, o passo seguinte foi o cálculo dos parâmetros para um controlador digital, que emule o controlador analógico. Sendo o controlador analógico dado pela equação 4.3.

$$PI(s) = Kp + \frac{a.Kp}{s} \tag{4.3}$$

Emula-se o Controlador analógico com um controlador digital, sendo o valor T o tempo de amostragem, o qual deve ser pelo menos 10 vezes menor que o tempo de estabelecimento do pólo mais rápido para que não exista *alliasing* na reconstrução do sinal erro, dado um amostrador ideal ter como propriedade replicar o espectro do sinal original centrado em $K * f$, com K pertencente a N

e f igual a $1/T$ [Phi95]. O controlador PID em controladores analógicos é dado pela expressão 4.4, mas porque o controlador usado será PI apenas nos vamos debruçar sobre a equação 4.5.

$$m(t) = Kp[e(t) + \frac{1}{Ti}\int_0^t e(t) + Td\frac{de(t)}{dt}] \tag{4.4}$$

$$m(t) = Kp[e(t) + \frac{1}{Ti}\int_0^t e(t)] \tag{4.5}$$

Em que $e(t)$ representa o erro, sendo este a entrada do controlador e $m(t)$ a saída do controlador, sendo Kp o ganho proporcional e Ti o ganho integral. Para obter resposta a um impulso de um PI digital podemos discretizar a função erro, aproximando o integral para uma soma de trapézios, obtendo-se a expressão 4.6, tal como é indicado na figura 4.15

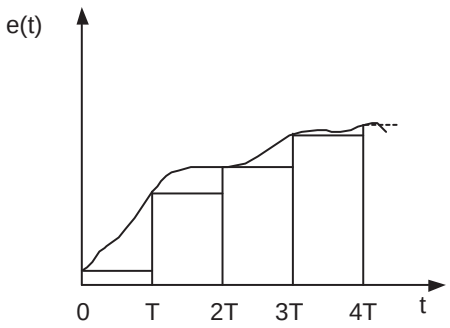


Figura 4.15: Discretização da função erro

A partir da equação 4.2, da equação 4.6 e da equação 4.7 obtem-se a equação 4.8 [Oga95], sendo este o controlador digital que emula o funcionamento do controlador analógico.

$$m(kT) = Kp[e(kT) + \frac{T}{Ti}(e(0) + e(T) + e(2T) + ... + e(K.T))] \tag{4.6}$$

$$Z[\sum_{h=1}^{k} e(hT)] = \frac{1}{(1-z^{-1})} E(z) \tag{4.7}$$

$$PI(z) = Kp + \frac{aKpT}{(1-z^{-1})} \tag{4.8}$$

4.4 Validação do controlador e da odometria

Com o objectivo de analisar a *performance* do controlador e avaliar os erros de odometria foi realizada uma corrida, estando descrita pelo fluxograma da figura 4.16. Durante a corrida foram registadas a posição estimada pelo cálculo da odometria e a posição absoluta real. Para se obter a posição absoluta real foi utilizado um sistema de visão para localização em tempo real de múltiplos *robots*[Cos99], utilizado pela equipa 5DPO na liga dos *small*.

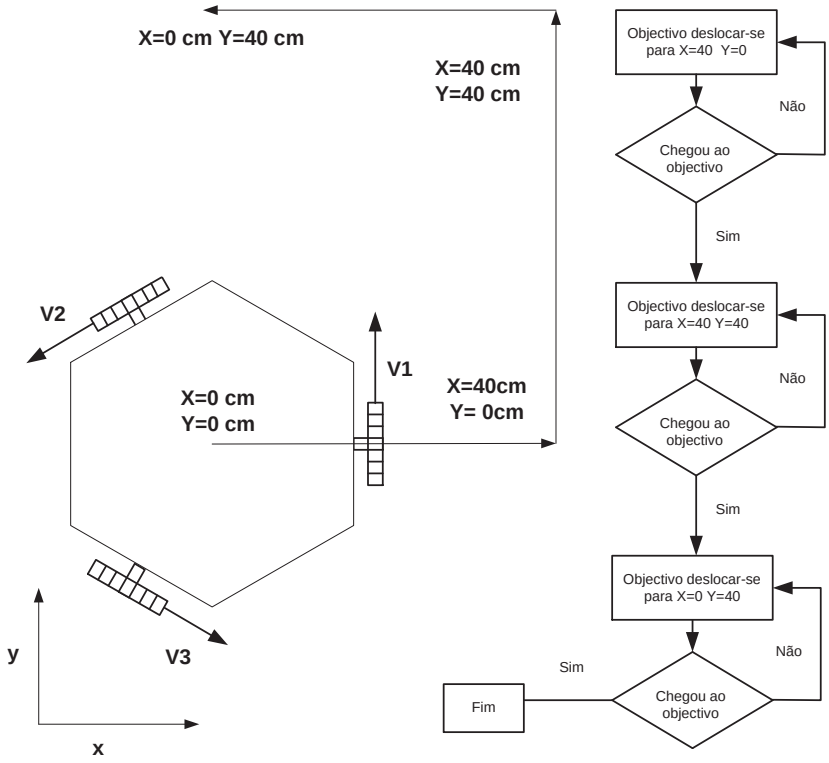


Figura 4.16: Fluxograma da corrida efectuada pelo *robot*

Como podemos observar na figura 4.16 o *robot* desloca-se passando por vários pontos efectuando deste modo uma trajectória. A estimação do posicionamento por visão externa também contem um erro absoluto não cumulativo inferior a 1 *cm*, permitindo saber com muito rigor o posicionamento absoluto do *robot* em qualquer instante. Por outro lado o posicionamento estimado pelo cálculo da odometria contem erros cumulativos, os quais passado algum tempo tornam a estimativa muito desfasada da realidade. Deste modo podemos tirar conclusões quanto às possíveis fontes de erro na odometria analisando-se a evolução do erro no cálculo da odometria, utilizando como referência o sistema de visão externo. A trajectória real (visão) e a estimada (odometria) estão representadas na figura 4.17.

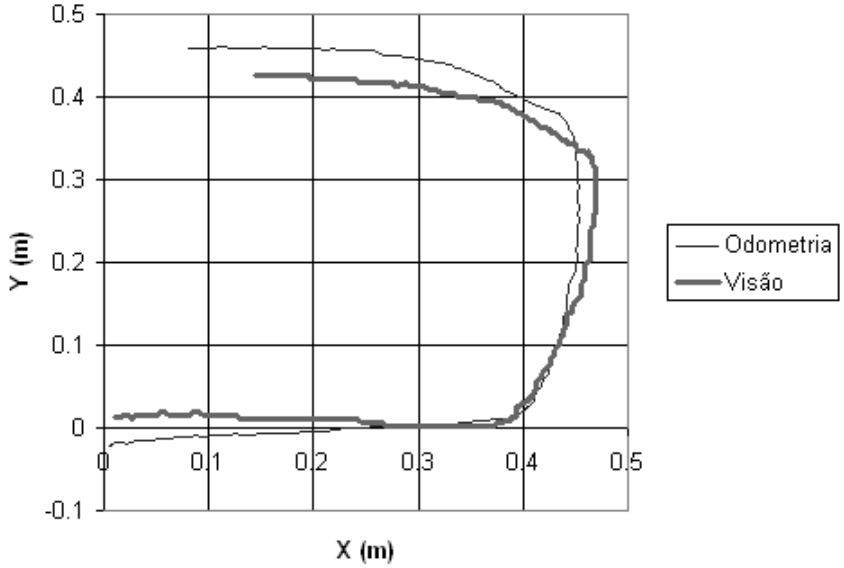


Figura 4.17: Localização do *robot* baseado em visão externa e na odometria

Nas figuras 4.18, 4.19 e 4.20, está apresentado o posicionamento absoluto do *robot*, registado com visão externa, sendo o *robot* colocado num valor de posicionamento absoluto indicado pela visão como sendo próximo da origem. Como podemos observar dos gráficos apresentadas que definem a trajectória o *robot* vai efectuar duas mudanças bruscas de trajectória, tendo este facto consequências

a nível da estimação do posicionamento do *robot* baseado no cálculo da odometria, pois os cálculos estão a ser efectuados utilizando parâmetros que variam. Numa situação em que existe uma variação brusca de direcção temos que ter em conta que as rodas podem plissar, que existem folgas nas rodas e o facto da polia que transmite movimento da roda para o *encoder* ter elasticidade. Além disso o parâmetro L varia dinamicamente, dada a estrutura física das rodas.

Um facto importante que deve ser realçado do gráfico da figura 4.17, é que quando se espera que o *robot* passe pelo ponto $X = 40\ cm$ e $Y = 0\ cm$ o *robot* começa a deslocar-se para o ponto seguinte antes de passar por este. Isto acontece porque o objectivo de chegar a um ponto é atingido se o erro em X e em Y forem menores que 2 cm, evoluindo a máquina de estados para o objectivo seguinte, que será o ponto X=40 cm e Y=40 cm.

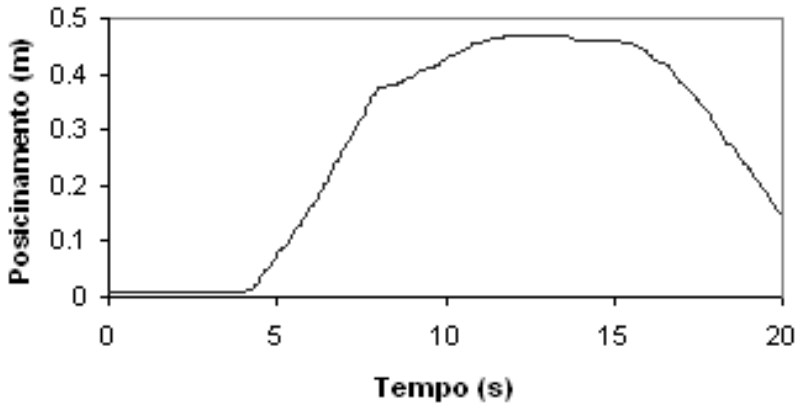


Figura 4.18: Posicionamento absoluto em X do *robot* baseado em visão externa

Da observação das figuras 4.21, 4.22 e 4.23 podemos afirmar que existe inicialmente um erro em x, y e $teta$ na posição inicial. Após o início da corrida observou-se que para uma trajectória rectilínea não existem variações significativas no erro do ângulo, mas existindo variações no erro em X e Y devido ao erro inicial do ângulo. O erro no ângulo sofre um incremento significativo quando existem mudanças bruscas de direcção, existindo incremento aos 11 e 16 segundos. A este incremento no erro em $\theta corresponde um consequente acumular do erro em X e Y, o que leva a que a estima$

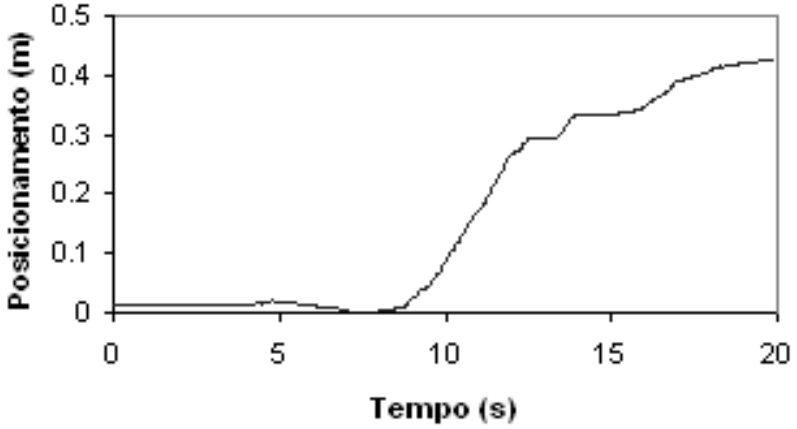
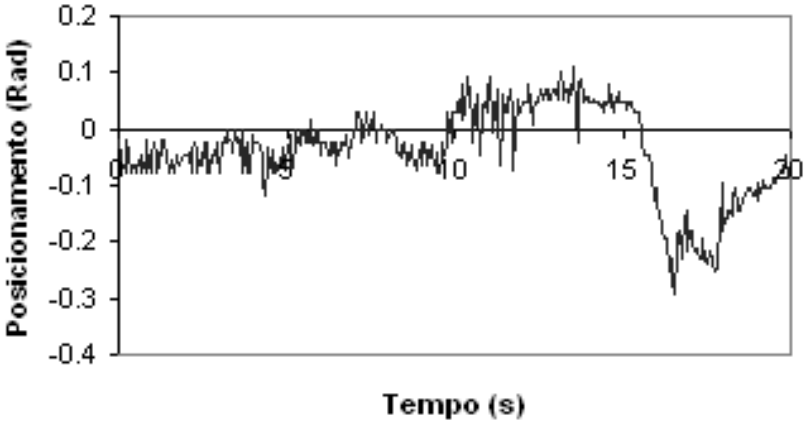


Figura 4.19: Posicionamento absoluto do *robot* em Y baseado em visão externa



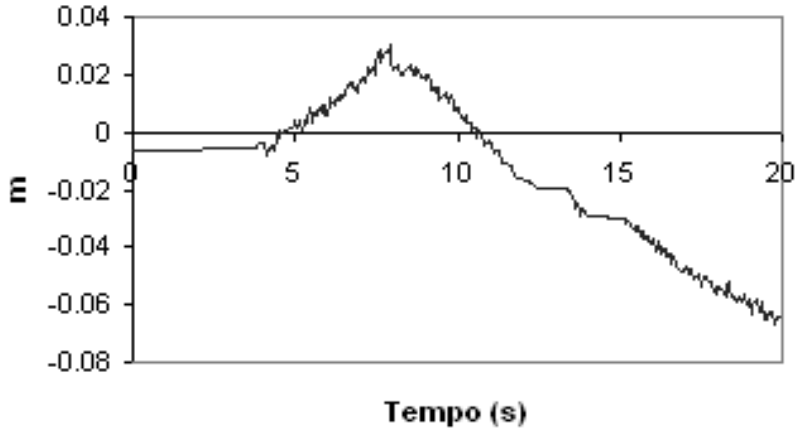


Figura 4.21: Erro absoluto de posicionamento em X do *robot*

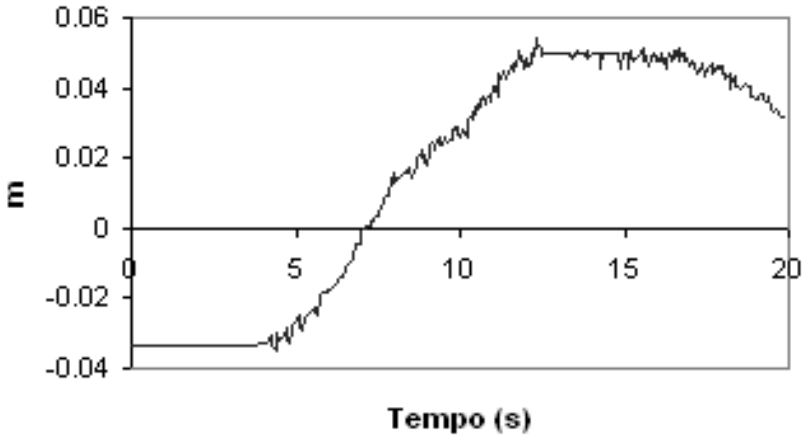


Figura 4.22: Erro absoluto de posicionamento *robot* em Y

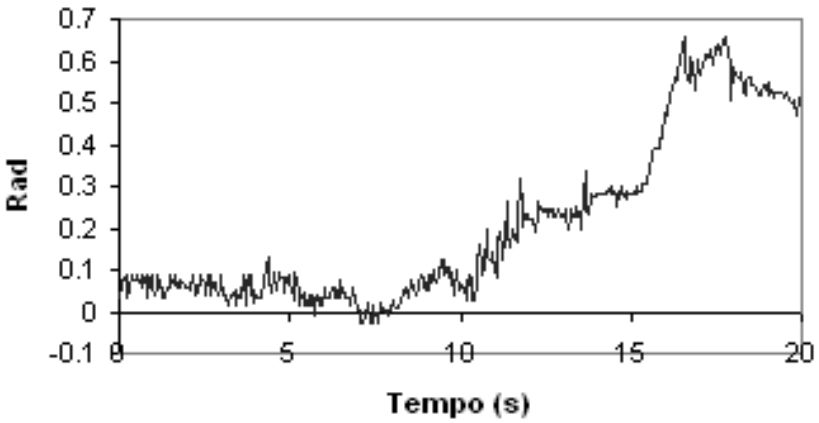


Figura 4.23: Erro absoluto de posicionamento do *robot* em Teta

4.5 Conclusões

A prototipagem do *robot* foi um processo demoroso, tendo como resultado final um protótipo robusto, conseguindo colmatar algumas das lacunas do primeiro protótipo realizado, tais como uma forte limitação nas portas de entrada e de saída, limitações a nível de poder de cálculo (dado o controlador neste protótipo ser efectuado no PC), *encoders* com menor erro e uma possibilidade de monitorização em tempo real da trajectória que o *robot* está a seguir e a que ele pensa que está a seguir.

Ao ser monitorizada a trajectória seguida pelo *robot* e comparando-se esta com a estimativa do posicionamento pelo cálculo da odometria podemos tirar algumas conclusões face às fontes de erro da odometria. Observou-se que numa situação em que existe uma variação brusca de direcção temos que ter em conta que as rodas podem plissar, que existem folgas nas rodas e o facto de da polia que transmite movimento da roda para o *encoder* ter elasticidade. Este facto implica que hajam nestas situações um erro acrescido na estimação da posição, acumulando-se essencialmente um maior erro no ângulo, o que vai fazer com que o erro posteriormente se acumule também em X e Y.

Conclui-se que se torna insuficiente o cálculo da odometria para o posi-

cionamento absoluto de um *robot* pois passado algum tempo o posicionamento estimado encontra-se completamente desfasado da realidade. Deste modo como alternativa o posicionamento pode ser estimado usando múltiplas fontes ou usando-se um sistema que não contenha um erro cumulativo, sendo o exemplo o sistema de visão usado para comparar a trajectória real com a estimada.

Capítulo 5

Conclusões

Pode-se concluir após a finalização de este trabalho que é insuficiente a definição de trajectórias baseando-se apenas no conhecimento do cálculo odómetrico, pois decorrido algum tempo o acumular do erro no posicionamento absoluto torna-se crítico, principalmente quando este é no ângulo, tornando-se cada vez mais afastado da realidade. Os erros na odometria surgem essencialmente devido a dois factores, sendo o mais óbvio o escorregamento das rodas e um outro que é a caracterização de um dos parâmetros do modelo para estimação do posicionamento do *robot*, que é o parâmetro L (distância do centro de rotação ao ponto em que assentam as rodas no chão), dado este variar dinamicamente. O parâmetro L varia pois a distância ao eixo vertical equidistante aos pontos onde assentam as rodas varia consoante a posição em que estas se encontram. Uma solução para minimizar este problema seria usar as rodas representadas na figura 5.1, uma vez estas permitirem que o cálculo do posicionamento absoluto baseado na odometria seja mais rigoroso, pois para estas rodas o parâmetro L não varia consoante a posição da roda. Esta solução apresenta como ponto negativo o facto de estes tipos de rodas não se adaptarem a pisos que não sejam similares a alcatifa.

O cálculo da odometria não deve ser o único método a usar para estimar a localização absoluta ou relativa de um *robot* pois os erros são cumulativos, tornando-se cada vez maiores. Torna-se um método muito útil quando existe

Figura 5.1: Rodas utilizadas pela equipa 5DPO

fusão de dados de várias fontes, podendo-se usar um filtro de *Kalman* [SCM04], não ficando o sistema dependente de apenas um sensor externo ou interno. Esta abordagem é imperativa dado a localização exacta dos *robots* ser imprescindível para a correcta execução das tarefas.

5.1 Conclusões relativas ao protótipo feito com o *Kit Lego Mindstorms*

É de realçar que foi desprezada a dinâmica para o cálculo do controlador pois opera-se com velocidades baixas e com uma caixa com grande redução, tendo o motor caixa redutora interna.

Existem situações em que efectuar o cálculo da velocidade de cada roda não é directo pois quando um motor não está travado não existe informação relativa ao sentido de rotação, simplesmente são recebidas transições, as quais são convertidas em velocidade, sendo o sentido de rotação estimado a partir do conhecimento da referência de velocidade dos outros dois motores, tal como exemplificado no fluxograma da figura 3.10.

Apesar de o *hardware* ser muito limitado, principalmente a nível de robustez, torna-se muito aliciante fazer *robots* com os *kit Lego Mindstorms*, existindo um inúmero conjunto de conceitos que podem ser explorados usando este tipo de

tecnologia.

5.2 Conclusões relativas ao protótipo com estrutura de alumínio

A prototipagem do *robot* foi um processo demoroso, tendo como resultado final um protótipo robusto, conseguindo colmatar algumas das lacunas do primeiro protótipo realizado, tais como uma forte limitação nas portas de entrada e de saída, limitações a nível de poder de cálculo (dado o controlador neste protótipo ser efectuado no PC), *encoders* com menor erro e uma possibilidade de monitorização em tempo real da trajectória que o *robot* está a seguir e a que ele pensa que está a seguir.

Ao ser monitorizada a trajectória seguida pelo *robot* e comparando-se esta com a estimativa do posicionamento pelo cálculo da odometria podemos tirar algumas conclusões face às fontes de erro da odometria. Observou-se que em situações em que existem variações bruscas de direcção é necessário ter em conta que as rodas podem plissar, que existem folgas nas rodas e o facto da polia que transmite movimento da roda para o *encoder* ter elasticidade. Este facto implica que hajam nestas situações um erro acrescido na estimação da posição acumulando-se essencialmente um maior erro no ângulo, o que vai fazer com que o erro posteriormente se acumule também no X e no Y.

Conclui-se também que se torna insuficiente o cálculo da odometria para o posicionamento absoluto de um *robot* pois passando algum tempo o posicionamento estimado encontra-se completamente desfasado da realidade. Deste modo como alternativa o posicionamento pode ser estimado usando múltiplas fontes ou usando-se um sistema que não contenha um erro cumulativo, sendo o exemplo o sistema de visão usado para comparar a trajectória real com a estimada.

5.3 Comparação dos dois protótipos

A nível de prototipagem tornou-se muito mais vantajoso iniciar este trabalho com um protótipo feito com *legos* pois aprendi alguns conceitos importantes rapidamente, coisa que seria impossível caso se demora-se algumas semanas ou até meses para desenvolver um outro protótipo, com muito mais robustez e com muito melhor hardware, apesar de que com o segundo protótipo os dados obtidos são muito melhores com vista a obter conclusões sobre a odometria e o controlador.

A arquitectura de *software* apresentada para o primeiro protótipo tem algumas particularidades interessantes dado o ser pouco comum efectuar-se *software* de controlo em tempo real em *Java*, efectuando-se os temporizadores usando *threads*. Tal como o *NQC (Not Quite C)* a Linguagem *Java* usada para programar os *Lego Minstorms* poder-se-ia chamar *Not Quite Java* em vez de *Lejos*, isto pelo facto de apenas trabalhar com um subconjunto das potencialidades desta linguagem, apresentando este facto algumas vantagens tais como não ter *Garbage Colector*, tratando-se de um processo que em *Java* condiciona o desempenho em tempo real de algumas operações, pois ocupar-se-ia o processador com uma operação que é executada ciclicamente, não podendo esta ser inibida. Tornou-se bastante aliciante efectuar o *software* de controlo em *Java*, dado ser possível tanto o controlador como o cálculo da odometria serem feitas dentro do *Brick* da *Lego* sem que existisse comunicação com um PC. Para isto aproveitou-se a particularidade de este fazer contas com 32 bits, obtendo-se um *overhead* muito menor que ao efectuar as mesmas contas com um micro-controlador *AVR*, que sendo *RISC*, ao efectuar cálculos com 8 *bits* rapidamente esgota a memória. A solução utilizada para contornar este problema foi o de efectuar o cálculo da odometria e do controlador para definir a velocidade a que devem rodar os motores no PC, sendo apenas o controlo dos motores e a leitura de sensores efectuada pelo micro-controlador. Esta solução apresenta também a vantagem de podermos registar em tempo real o que está a acontecer relativamente ao controlo do *robot*, coisa que no *Brick* da *Lego* apenas era possível através do seu *display* após a finalização

de uma corrida.

Apesar da maior complexidade no controlo a utilização de *robots* omni-direccionais torna-se vantajoso em grande parte das aplicações, pois permite deslocamentos em qualquer direcção com velocidade linear e angular controladas, sem estarem sujeitos às restrições dos tipicamente utilizados *robots* diferenciais.

5.4 Trabalho futuro

Como trabalho futuro proponho-me fazer-se fusão de dados usando-se várias fontes para o posicionamento dos *robots*, sendo o cálculo da odometria insuficiente para conhecer correctamente o posicionamento absoluto. Deste modo não dependendo apenas de uma fonte de informação pode-se melhorar a estimativa do posicionamento do *robot*, por exemplo evitando o acumular dos erros de odometria.

Também me proponho a trabalhar com a configuração de quatro rodas omnidireccional, pois tem algumas vantagens em relação à configuração com três rodas. Por exemplo para fazer trajectórias em linha recta apresenta mais força e velocidade, pois são quatro motores a fazer força em vez de apenas três. Além disso todas as vantagens que apresentadas na configuração de três rodas estão presentes na configuração de quatro rodas.

Capítulo 6

Anexos

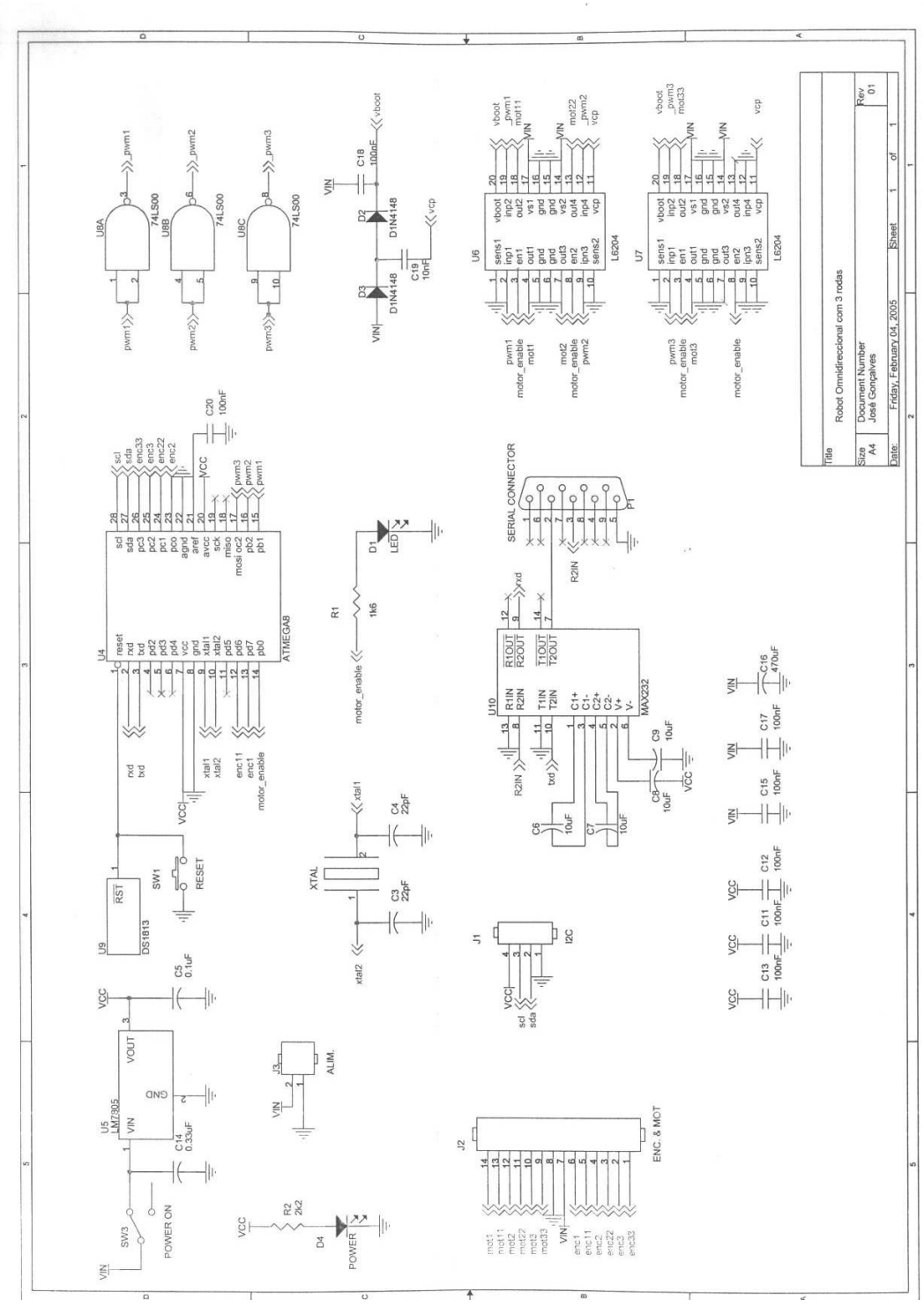


Figura 6.1: Esquemático do Hardware do protótipo de alumínio

Bibliografia

[AAB+04] L. Almeida, J.L. Azevedo, P. Bartolomeu, E. Brito, M.B. Cunha, J.P. Figueiredo, P. Fonseca, C. Lima, R. Marau, N. Lau, P. Pedreiras, A. Pinho A. Pereira, F. Santos, L. Seabra Lopes, and J. Vieira. Cambada team description paper. *Robocup 2004*, 2004.

[BCO03] Barnett, Cox, and O'Cull. *Embedded C Programming and the Atmel AVR.* Thomson Delmar Learning, 2003.

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

[Cor02] Atmel Corporation. Atmega8, 2002.

[Cos95] Paulo Costa. Identificação, modelização e controlo de um veículo móvel, Tese de Mestrado, Faculdade de Engenharia da Universidade do Porto, 1995.

[Cos99] Paulo Costa. Localização em tempo real de múltiplos robots num ambiente dinâmico, Tese de Doutoramento, Faculdade de Engenharia da Universidade do Porto, 1999.

[Cos03] Paulo Costa. Api docs. http://lejos.sourceforge.net/apidocs/, 2003.

[CSM+03] Paulo Costa, Armando Sousa, Paulo Marques, Pedro Costa, Susana Gaio, and António Moreira. Team descriptions of the robocup 2003

games and conferences. RoboCup2003, Padova, Italy, 02/07 - 11/07 2003.

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

[FFH02] Mario Ferrari, Giulio Ferrari, and Ralph Hempel. *Building Robots with lego Mindstorms The ultimate tool for mindstorm maniacs*. Syngress Publishing, Inc, 2002.

[FS04] Floyd-Steinberg. Floyd-steinberg motor control. http://home.earthlink.net/˜ mrob/pub/lego/fsmotor.html, 2004.

[KNDG02] Tamás Kalmár-Nagy, Raffaello D'Andrea, and Pritam 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.

[LCGR02] Robert L.Williams, Brian E. Carter, Paolo Gallina, and Giulio Rosati. Dinamic model with slip for wheeled omnidirectional robots. In *IEEE TRANSACTIONS on robotics and automation Vol 18*. IEEE Press, june 2002.

[LCM04] José Lima, Paulo Costa, and Paulo Moreira. Sistema de navegação e controlo de um agv por visão artificial. *Proceedings of Cientific Meeting of Robótica 2004*, pages 115–118, 2004.

[leg04] The free dictionary. http://www.thefreedictionary.com/Lego, 2004.

[LFS02] Dario Laverde, Giulio Ferrari, and Jurgen Stuber. *Programming Lego Mindstorms with Java*. Syngress Publishing, Inc, 2002.

[man02] Ron mancini. *Op amps for everyone*. Texas instruments, August 2002.

[MdC02] Rolando R. António Paulo Moreira and Paulo Gomes da Costa. Acoustic localization for mobile robots. *Revista do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro*, pages 540–452, 2002.

[MIC94] SGS-THOMSON MICROELECTRONICS. L6204. SGS-THOMSON MICROELECTRONICS, 1994.

[MUwPR95] Ned Mohan, Tore M. Undeland, and william P. Robbins. *Power electronics : Converters, applications and design 2nd Edition*. John Wiley and Sons, Inc, 1995.

[Oga93] Katsuhiko Ogata. *Engenharia de controle Moderno*. Prentice Hall do Brasil LDA, 1993.

[Oga95] Katsuhiko Ogata. *Discrete-time control systems*. Prentice-Hall, Inc, 1995.

[Omr02] Omron. E6a2-cw5c, 2002.

[op04] Omni project. http://prt.fernuni-hagen.de/pro/omni/omni.sum-eng.html, 2004.

[pas04] Lego® stepper motor. http://home.earthlink.net/~mrob/pub/lego/stepper.html, 2004.

[PAW91] Ramon Pallas-Areny and John G. Webster. *Sensors and Signal Conditioning*. John Wiley and Sons Inc, 1991.

[Phi95] Charles L. Phillips. *Digital Control Systems, analysis and design*. Prentice-Hall, Inc, 1995.

[PM04] Armando Peixoto and Pedro Moreira. Plataforma robótica genérica, Projecto de fim de curso, Faculdade de Engenharia da Universidade do Porto, 2004.

[RBS+03] Fernando Ribeiro, Paulo Braga, Pedro Silva, Ivo Martinho, and Jorge Monteiro. New improvements of minho team for robocup middle size league in 2003. RoboCup2003,Team Description Paper,Padova, Itália., April 8 2003.

[RMN00] Grigory B. Reshko, Matthew T. Mason, and Ill R. Nourbakhsh. Rapid prototyping of small robots. Carnegie Mellon University, 2000.

[RMS+04] Fernando Ribeiro, Ivo Moutinho, Pedro Silva, Carlos Fraga, and Nino Pereira. Controlling omni-directional wheels of a robocup msl autonomous mobile robot. *Proceedings of Cientific Meeting of Robótica 2004*, pages 11–15, 2004.

[rob04] Robocup. http://www.robocup.org/, 2004.

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