Ferramentas Computacionais para Robôs Móveis Autônomos

Douglas Machado Tavares

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Ferramentas Computacionais para Robôs Móveis Autônomos

Douglas Machado Tavares

Fevereiro de 2004

Banca Examinadora:

- Prof. Dr. Luiz Marcos Garcia Gonçalves
  Departamento de Engenharia de Computação e Automação, UFRN (Orientador)
- Prof. Dr. Siome Klein Goldstein
  Instituto de Computação, UNICAMP
- Prof. Dr. João Vilhete Viegas D'Abreu
  Núcleo de Informática Aplicada à Educação, UNICAMP
- Profa. Dra. Silvia Silva da Costa Botelho
  Departamento de Física, FURG

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Tavares, Douglas Machado

T197f Ferramentas computacionais para robôs móveis autônomos / Douglas Machado Tavares -- Campinas, [S.P. :s.n], 2004.

Orientador: Luiz Marcos Garcia Gonçalves

Co-orientador: Jacques Wainer

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Robôs móveis. 2. Robótica. 3. Robôs - Programação. I. Gonçalves, Luiz Marcos Garcia. II. Wainer, Jacques. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Ferramentas Computacionais para Robôs Móveis Autônomos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Douglas Machado Tavares e aprovada pela Banca Examinadora.

Campinas, 22 de Março de 2004.

Prof. Dr. Luiz Marcos Garcia Conçalves
Departamento de Engenharia de Computação e Automação, UFRN (Orientador)

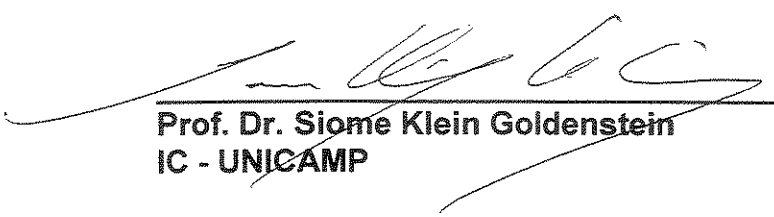
Prof. Dr. Jacques Wainer
Instituto de Computação, UNICAMP
(Co-orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 27 de fevereiro de 2004, pela Banca examinadora composta pelos Professores Doutores:

Profa. Dra. Silvia Silva da Costa Botelho
FURG - RS



Prof. Dr. Siome Klein Goldenstein
IC - UNICAMP

Prof. Dr. João Vilhete Viegas D'Abreu
NIED– UNICAMP

Prof. Dr. Luiz Marcos Garcia Gonçalves
UFRN

A
Guilherme e Eloina,
meus pais,
fonte de
amor e
dedicação.

Agradecimentos

A Deus por iluminar meus passos.
Aos meus pais, Guilherme e Eloina e aos meus irmãos Dalson e Michelle pelo carinho e amor, que foram essenciais nesta jornada.
À Caroline Antonieta pelo carinho, amor e dedicação.
Aos familiares que sempre torceram por mim.
Ao Professor Luiz Marcos, meu orientador e amigo, pela orientação, amizade, colaboração e incentivo.
Aos Professores Adelardo e Pablo pelas importantes sugestões.
Aos amigos e colegas de projetos George, Clauber e Meika pela amizade e companheirismo.
A Marcelo, Anfranserai, Toinho, Adriano, Viviane e a todos os alunos do LabSis (Laboratório de Sistemas Inteligentes) da UFRN, que ora contribuiram com críticas e sugestões.
A todos os colegas de mestrado da UNICAMP e da UFRN, em especial, à Daniele, Vanessa, Adriane Belle, Schubert, Edson Borin, Tatiana e Stephany.
E, por fim, agradeço à CAPES, pelo suporte financeiro.

"A robot may be worth a thousand simulations"
Rodney A. Brooks

Resumo

Robôs móveis autônomos são plataformas com poder de processamento embarcado, capazes de tomar decisões de forma independente frente a situações diversas impostas pelo ambiente sobre o qual operam. Têm sido muito usados em pesquisa e ensino, permitindo o desenvolvimento de aplicativos, experimentos e aprimoramento de conhecimentos. Programar um robô para que ele haja de forma autônoma, geralmente envolve a definição de estados de predicado, a partir dos quais ele possa inferir alguma ação dentro das suas possibilidades e limitações, que responda de melhor forma à situação que o ambiente lhe proporcionou. O desenvolvimento de ferramentas para comunicação, controle e navegação (esta, incluindo posicionamento, orientação e deslocamentos no ambiente), torna-se essencial para que um robô possa realizar, de forma autônoma, missões em ambientes adversos, geralmente hostís, onde mobilidade seja uma necessidade inerente.

Neste contexto, o presente trabalho discute e propõe ferramentas computacionais para robôs móveis autônomos, que podem ser utilizadas, principalmente, no contexto multi-robôs, incluindo detalhes das implementações e uma série de experimentos e testes. São apresentados os conceitos e a arquitetura da plataforma LEGO, ressaltando suas potencialidades e problemas, bem como uma análise detalhada de alguns de seus compiladores e outros *softwares*. Foi desenvolvido um protocolo de controle (ou de comandos), ou seja, uma ferramenta que permita que programas escritos em linguagem 'C' (executando em um PC), possam escrever nas saídas e ler as entradas de uma unidade de controle localizada no robô. É apresentado um estudo e análise do tempo de comunicação do referido protocolo de comandos. Foi desenvolvida uma ferramenta para determinação da localização atual do robô, composta por odometria e por um sistema de localização visual. Esta ferramenta usa medidas esparsas da localização absoluta dadas pelo sistema de localização visual, para corrigir o sistema de odometria do robô. Finalmente, foi implementada e testada uma segunda ferramenta de localização, a qual utiliza-se somente de um sistema de localização visual baseado em marcos.

Como principais contribuições deste trabalho, podem ser citadas essas ferramentas desenvolvidas, os conhecimentos adquiridos e disponibilizados à comunidade, a partir das pesquisas realizadas no intuito de formalizar conceitos e metodologias para a linha LEGO de mini-robôs. Os conhecimentos, bem como as ferramentas estão sendo essenciais à realização de vários tra-

balhos e aplicações, tendo gerado publicações em eventos nacionais e internacionais de qualidade, além de uma publicação em revista nacional sobre o potencial e limitações da plataforma usada para os desenvolvimentos.

Abstract

Autonomous mobile robots are hardware platforms with embedded processing power, which are able to take decisions, independently, in front of several situations imposed by the environment where they operate. They have been most used in research and education, allowing the development of applications, experiments and improvement of knowledge. Programming a robot to act in autonomous way generally involves the definition of predicate states, from which it can infer some action, regarding its possibilities and limitations, in order to give the best answer to the situation imposed by its environment. The development of tools for communication, control and navigation, the last including positioning, orientation and movements in the environment, becomes essential for a robot to realize missions in autonomous way in adverse environments, generally hostile, where mobility is an inherent necessity.

In this context, the present work discusses and proposes computational tools for autonomous mobile robots, that can be used mainly in the context of multi-robots, including details of the implementations, experiments and tests. The concepts and the architecture of LEGO platform are presented standing out its potentialities and problems as well as a detailed analysis of some of its compilers and others softwares. A control (or commands) protocol was developed. That is, a tool that allows programs written in 'C' language running in a PC to write in the outputs and to read from the inputs of a control unit located in the robot. It is presented a study and analysis of the communication time of this commands protocol. A tool for determination of the actual localization of the robot was also developed, composed of odometry and of a visual localization system. This tool uses sparse measurements of the absolute localization given by the visual system to correct the odometry system of the robot. Finally, it was implemented and tested a second tool for localization, which uses a visual localization system based on landmarks.

As main contributions of this work, we cite these developed tools and the knowledge acquired and made available to the community through the research carried with intention to formalize the concepts and methodologies of LEGO line mini-robots. The knowledge as well as the tools are being essential in the development of some works and applications, having generated publications in national and international events, besides a publication in a national magazine about the potential and limitations of the platform used for the developments.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O advento da tecnologia digital modificou vários segmentos da vida moderna, refletindo principalmente em processos de controle industrial, tornando-os cada vez mais complexos e exigindo alta flexibilidade. Surgiram diversos tipos e modelos de plataformas de prototipagem, visando desenvolvimento de aplicativos, experimentação, treinamento e aprimoramento do conhecimento principalmente no contexto industrial. Estas plataformas passaram a ser muito utilizadas por pesquisadores, cientistas, professores e alunos em diversas áreas de pesquisa e ensino, por darem um enfoque prático às questões que antes eram meramente virtuais [11]. Atualmente, tornam-se imprescindíveis para o ensino e pesquisa, em diversas áreas de tecnologia de ponta, como por exemplo em robótica, a área de trabalho a ser explorada neste trabalho.

A palavra robótica, derivada do Tcheco 'robota', foi primeiramente empregada por Isaac Asimov, para denominar a ciência que trata dos robôs. A robótica proporciona uma ampla integração de metodologias ou técnicas de diversas áreas do conhecimento, como: percepção (sensores, fusão sensorial, visão computacional, dentre outros), meta-heurísticas (Algoritmos Genéticos, Computação Evolutiva, Inteligência Social, Redes Neurais, Lógica Difusa, dentre outros), navegação, raciocínio e ação [33, 21, 6, 40, 16]. Podem ser assim citados os robôs móveis como exemplos atuais de plataformas viáveis, que permitem o desenvolvimento de aplicativos, experimentos, aprimoramento de conhecimentos e a verificação da aplicabilidade dessas integrações. Além de estarem vinculados às necessidades do meio industrial, por serem sistemas mais adaptativos, flexíveis, com maior espaço de trabalho e proporcionarem um maior grau de autonomia.

É dentro deste contexto que é apresentado este trabalho. Discutindo sobre ferramentas computacionais atuais, para robôs móveis autônomos e apresentando algumas implementações e testes de ferramentas, as quais podem ser utilizadas principalmente no contexto multi-robôs.

1.1 Motivação e Justifi cativas

Dentre alguns dos robôs móveis comerciais utilizados em pesquisas pode-se citar a linha de robôs da *K-Team*, como: o *Khepera*, o *K-alice* e o *Koala* [24], a linha de robôs da *Activ Media Robotics*, como: o *Pioneer* (2-DXe e 2-AT) e o *AmigoBot* [1] e os andarilhos da *Lynxmotion* [32].

Recentemente, os *kits* da linha LEGO *MINDSTORMS* [25], começaram a ser utilizados como uma boa opção de mini-robôs móveis para pesquisa e ensino, destacando-se pelo baixo custo, flexibilidade e pela grande velocidade de prototipagem. Podem ser citadas algumas disciplinas ministradas recentemente como: "Robótica: Sistema Sensorial e Motor" na UNICAMP e UFMS [19], "Percepção Robótica" na UFRN [20], "Sistemas Robóticos Autônomos" na UFRN [2]" e "Introdução à Engenharia de Controle e Automação" na UNICAMP [10].

Nota-se que os alunos usuários desta plataforma, vêem-se necessitados de ferramentas de programação, que lhes permitam explorar ao máximo o potencial da mesma e que lhes sejam familiares [12]. Estes fatos foram observados principalmente em disciplinas ministradas em cursos de Engenharia e de Ciência de Computação [9, 2, 34, 20, 19], tornando-se assim, necessário e cabível estudar esta plataforma, bem como seus compiladores e seus sistemas operacionais, executar testes e comparações e analisar e apontar ferramentas que atendam aos requisitos acima. E, principalmente, desenvolver novas ferramentas que possibilitem transformar um brinquedo educativo, simples, de adolescentes em um robô adequado à pesquisa.

Convém ressaltar ainda que, geralmente, as ferramentas desenvolvidas nesta plataforma, poderão ser usadas em outras plataformas robóticas móveis, dotadas de câmeras (simples) e que sejam capazes de processar esses dados sensoriais em tarefas de navegação e percepção do ambiente.

Para justificar a realização deste trabalho, podem ser citados inúmeros tópicos de pesquisas, que podem ser explorados com a utilização dessas ferramentas aliadas à plataforma da LEGO, como por exemplo: autonomia robótica, navegação, inteligência computacional, aprendizado de máquina, sistemas multi-robôs, sistemas embarcados e uma série de outros.

1.2 Estado da Arte

Muitos brinquedos têm atraído a atenção de pesquisadores. Projetados para serem brinquedos, essas máquinas estão, atualmente, sendo mais utilizadas para o ensino de conceitos básicos de programação e robótica do que simplesmente para a diversão. Como exemplo desses brinquedos pode-se citar o cachorro robô AIBO da Sony [44], que inclui algumas ferramentas de programação e acessórios para diversas funções, o pequeno robô pessoal R100 da NEC [36], o qual pode reconhecer pessoas, entender comandos de voz e conversar com os donos, por fim, os *kits* da linha LEGO *MINDSTORMS* [25], destinados à construção de pequenos robôs. Esses

brinquedos têm se mostrado verdadeiras plataformas de ensino sem, no entanto, apresentar-se como tal. Essa transformação de brinquedo em plataforma deve-se, principalmente, ao desenvolvimento de diversas ferramentas, pela comunidade científica, através de re-engenharias (ou engenharia reversa) do *hardware* e *software*.

Existem outros trabalhos, que como a presente proposta, disponibilizam algumas ferramentas ou utilizam-se de plataformas como as mencionadas anteriormente. A seguir, apresenta-se uma breve análise de algumas das mais significativas delas:

- os times de futebol de cachorros robôs AIBO, que participam da liga *Quadruped Robot-soccer Tournament* (*QuadroSot*), categoria de robôs quadrúpedes criada pela *Federation of International Robot-soccer* (FIRA) [18];

- os robôs desenvolvidos, com peças dos *kits* da linha LEGO, pelo laboratório *LEGO-Lab*, do departamento de Ciência da Universidade de Aarhus [26];

- os compiladores, as linguagens e os sistemas operacionais para os *kits* da linha LEGO, como o NQC [5], BRICKOS [8], LEJOS [28], emuladores como emulegOS, conversores musicais e muitas outras ferramentas essenciais, para criação de programas de alta performance;

- os mini-robôs móveis Manuelzão e Miguilim do Laboratório de Visão Computacional e Robótica - VERLab [48] do Departamento de Ciência da Computação da UFMG, controlados por uma *HandyBoard* [1] e construídos com peças da linha LEGO.

- o projeto SIROS (Sistemas Robóticos com SuperLogo), um ambiente em desenvolvimento no NIED/UNICAMP, que tem como objetivo implementar no *software* SuperLogo recursos que possibilitem o controle de dispositivos mecânicos automatizados, estes dispositivos podem ser construídos com *kits* da linha LEGO, o controle pode ocorrer no modo presencial ou no modo à distância, utilizando a rede *Internet*, dessa forma, o SIROS comporta-se como um Ambiente de Telerobótica [43];

- uma ferramenta de comunicação e envio de comandos (semelhante a que foi desenvolvida neste trabalho), biblioteca desenvolvida e disponibilizada pela LEGO, a qual segue a tecnologia *Activex* [2], porém, ela se restringe a compiladores proprietários que disponibilizem esta tecnologia (como o *Visual Basic* da *Microsoft*) e possui um tempo de transmissão muito alto, fator este prejudicial à construção de controladores.

[1] *HandyBoard* é um sistema de controle para robôs desenvolvido em uma pequena placa. Este sistema é baseado no processador Motorola MC68HC11, constando de $32k$ de memória RAM, 4 saídas PWM, 7 entradas analógicas e 9 entradas digitais.

[2] Tecnologia *Activex* é um recurso, que permite criar programas os quais funcionam como *plug-in* em outros programas.

Podem se destacar as disciplinas que utilizam ou utilizaram a plataforma LEGO, como por exemplo: "COM120" da Universidade de *Lancaster*, "6.270" do MIT [34], "Robótica: Sistema Sensorial e Motor" da UNICAMP e UFMS [19], "Percepção Robótica" da UFRN [20], "Sistemas Robóticos Autônomos" da UFRN [2]", "Introdução à Engenharia de Controle e Automação" da UNICAMP [10], "Introdução à Robótica" da UFMG [9] e "Introdução à Engenharia de Controle e Automação - DAS-5411" da UFSC [17].

Há outros trabalhos, que apesar de não serem desenvolvidos em plataformas semelhantes (brinquedos), apresentam algumas características ou idéias comuns com às ferramentas propostas, como:

- o trabalho de Lora, Hemerly e Lages [30, 31] desenvolve um sistema de localização, utilizando-se de odometria, implementado em *threads* (tarefas), semelhante à implementação do sistema de localização proposto neste trabalho;
- os trabalhos que abordam o problema de localização de rótulos em imagens, como por exemplo os sistemas de localização de robôs utilizados em partidas de futebol de robôs, promovidos por organizações internacionais com a *Robot World Cup Initiative* (RoboCup) e a *Federation of International Robot-soccer* (FIRA) [41, 18], as quais estão divididas em várias ligas/categorias. Em algumas ligas, a localização de rótulos é tratada com visão local, ou seja, câmera acoplada ao robô, utilizando-a para localização da bola por exemplo, semelhante ao sistema de localização visual proposto.

1.3 Objetivos

O objetivo geral deste trabalho é estudar e propor algumas ferramentas computacionais para robôs móveis autônomos, para uso acadêmico tanto em pesquisa quanto em ensino de graduação.

Como objetivos específicos podem ser citados:

- revisão bibliográfica, estudo, teste, comparações e análise das ferramentas disponíveis para a plataforma da LEGO;
- desenvolvimento de um protocolo de controle, que permita a um computador controlar remotamente robôs móveis (plataforma da LEGO);
- estudo e implementação de um sistema de localização, que corrija os erros de odometria, baseando-se em um sistema de localização visual;
- implementação de um sistema de localização visual, móvel e na horizontal.

1.4 Estrutura do Trabalho

Este trabalho está organizado da seguinte forma:

no Capítulo 2, discute-se a plataforma da LEGO, apresentando conceitos e arquitetura, elucidando seus problemas e potenciais, comparando dois compiladores e dois sistemas operacionais, levando em consideração fatores, como: a potencialidade, velocidade, tamanho em memória, complexidade de aprendizado, dentre outros;

no Capítulo 3, foi proposta uma ferramenta, que permite controlar remotamente robôs "embarcados" com a unidade de controle da plataforma da LEGO, isto é, um protocolo de comunicação, apresentando estudos e análises do tempo de comunicação deste protocolo;

no Capítulo 4, foi proposta uma ferramenta de localização constituída de um sistema de localização visual e de odometria, apresentando estudos e análises da correção do erro deste sistema de localização;

no Capítulo 5, foi proposta uma ferramenta de localização visual, isto é, uma ferramenta que identifica rótulos (esferas) em imagens, determinando a posição e o tamanho (diâmetro) dos rótulos, os quais poderão ser usados por um controlador;

no Capítulo 6, foram realizados vários experimentos e testes para validar as ferramentas disponíveis e as ferramentas desenvolvidas neste trabalho;

no Capítulo 7, foram apresentadas as principais contribuições, pontos e extensões, propondo melhoras e novos trabalhos, que venham contribuir para os resultados obtidos.

Capítulo 2

Plataforma Robótica

Neste trabalho adotou-se a plataforma LEGO (*kits* robóticos). Estes *kits* são compostos por pequenos blocos de plástico (peças tradicionais LEGO) e por outras peças como: motores, sensores de toque, de luz, de temperatura, de rotação, câmera de vídeo e engrenagens. Os *kits* possuem ainda uma unidade de controle programável chamada de RCX, a partir da qual pode-se fazer a leitura dos dados sensoriais ou gerar corrente para mover os motores (de forma independente) e uma torre de transmissão, que permite comunicação entre o computador e o RCX. Como exemplo destes *kits* pode ser citado o *kit* 9790 (ver Figura 2.1).

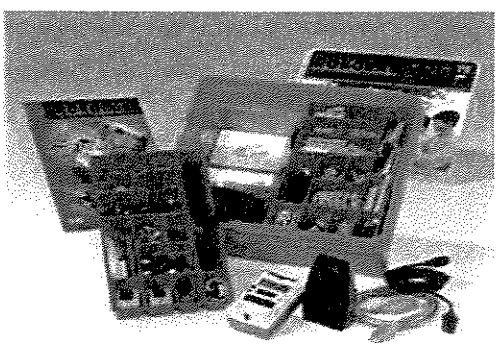


Figura 2.1: *Kit* 9790 da linha LEGO *MINDSTORMS*.

Neste Capítulo serão apresentados os conceitos e a arquitetura da unidade de controle RCX, procurando ressaltar seus potenciais e seus problemas e uma análise dos dois compiladores de programas para RCX, mais usados pela comunidade de pesquisa, o NQC e o BRICKOS (outros existentes são apenas citados), bem como os respectivos sistemas operacionais alvos. Também serão apresentadas análises comparativas entre eles, levando em consideração vários aspectos, como: potencialidade, velocidade, tamanho em memória, complexidade de aprendizado, dentre outros. Esta análise se baseia em inúmeros testes e implementações (incluindo mais de

200 horas de execução de programas), além de compilação de experiências coletadas, a partir de usuários dos compiladores e seus sistemas operacionais, atuais estudantes de Engenharia de Computação e de Ciência de Computação, à nível de pós-graduação (doutorado e mestrado) e de graduação, em disciplinas ministradas, usando os *kits* da LEGO, levadas a efeito na Universidade Estadual de Campinas (UNICAMP), Universidade Federal do Mato Grosso do Sul (UFMS) e Universidade Federal do Rio Grande do Norte (UFRN).

2.1 A Arquitetura do RCX

O *Robotic Control eXplorer* (RCX), é um computador embarcado de propósito específico (uma unidade de controle), sua forma aproxima-se a de um paralelepípedo de dimensões $95mm$ x $63mm$ x $40mm$ (ver Figura 2.2). O RCX é um produto da linha LEGO *MINDSTORMS* [25], com finalidade de controlar os dispositivos elétricos (motores e sensores) de mini-robôs construídos com blocos de plástico da LEGO. A característica "embarcado" permite criar programas em um computador, compilá-lo e transferi-lo para o RCX, usando uma interface de comunicação (torre de transmissão). Assim, o robô pode executar o programa de forma autônoma, a partir do acionamento de um botão no RCX. Nesse caso, o robô opera sem a intervenção humana, característica essencial a um robô móvel autônomo.

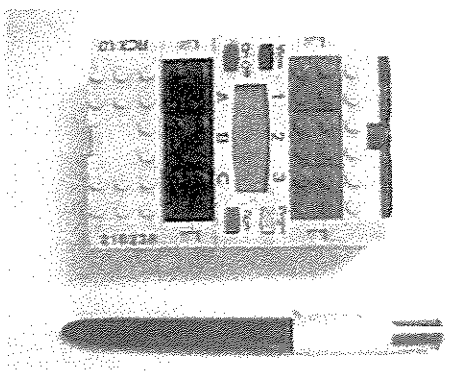


Figura 2.2: Unidade de Controle RCX.

O RCX usa a arquitetura RISC (conjunto reduzido de instruções), possuindo 32 registradores de propósito geral, um microcontrolador *Hitachi*, mais especificamente o $H8/3292$, de $16MHz$ e palavra com 8 *bits* de tamanho. O microcontrolador é responsável por tarefas como o controle lógico, incluindo entrada e saída serial (serial I/O), conversão de analógico para digital e vice-versa, relógios internos dentre outros. A memória disponível no RCX é constituída por $16K$ de memória ROM e $32K$ de memória RAM, sendo esta última destinada a parte do sistema operacional e a programas do usuário. A quantidade de memória RAM total ($32K$)

menos a quantidade ocupada pelo sistema operacional é toda a memória disponível, que pode ser manipulada pelos programas. A alimentação do RCX ocorre através de 6 pilhas do tipo AA (pequenas) de $1.5v$, ligadas em série ou através de uma entrada para conversores de energia.

Externamente, o RCX apresenta 3 saídas de tensões (A, B e C), onde podem ser conectados dispositivos eletrônicos (motores, buzinas, lâmpadas, dentre outros), 3 entradas (1, 2 e 3), onde podem ser conectados sensores (temperatura, toque, luz, dentre outros), uma tela de cristal líquido, onde serão exibidas pequenas mensagens e quatro botões (*On-Off*, *View*, *Prgm* e *Run*) (ver diagrama da Figura 2.3).

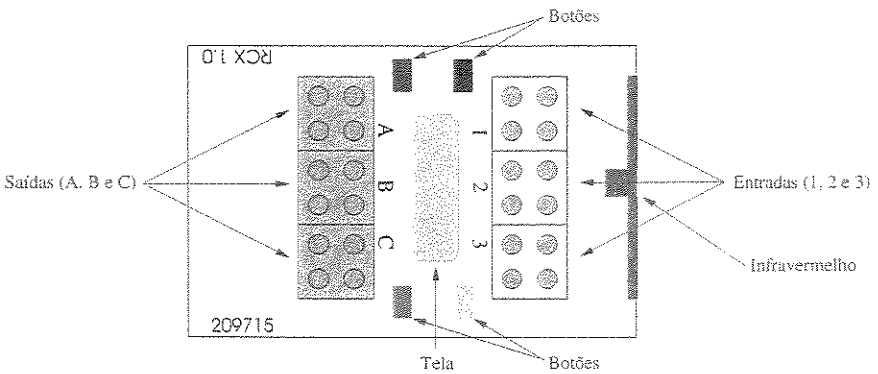


Figura 2.3: Diagrama Externo do RCX.

2.1.1 Torre de Transmissão IR (Infravermelho)

A torre de transmissão é uma ferramenta em *hardware* disponibilizada junto ao RCX com a função de transferência de programas implementados e compilados no computador, para serem executados pelo sistema operacional presente na memória do RCX. A torre é responsável por converter dados digitais em analógicos, codificando-os em um sinal a ser transmitido ao RCX em forma de luz infravermelha. A torre é também responsável pelo oposto, ou seja, receber dados analógicos codificados em forma de luz infravermelha e transformá-los para a forma digital, em seguida, repassá-los à memória do computador. Uma outra função que pode ser destacada apesar de não fornecida pelo fabricante é a viabilidade de comunicação entre o computador e o RCX, através da utilização de protocolos especiais discutido adiante.

2.2 Histórico dos *Softwares*

A linguagem de programação fornecida pela LEGO é muito simples, porém limitada. Motivados por essa limitação, usuários com conhecimentos avançados em programação de baixo

nível, como *assembly*, resolveram dedicar-se ao desenvolvimento de ferramentas mais poderosas e eficientes, de modo que o potencial oferecido pelo *hardware* da LEGO possa ser melhor explorado.

Um manual minucioso, detalhando os componentes internos do RCX foi disponibilizado por um aluno de graduação da Universidade de *Stanford, Califórnia, Kekoa Proudfoot*. A partir deste acesso ao interior do RCX, foi possível conhecer quais comandos o sistema operacional da LEGO pode suportar.

David Baum, um engenheiro de *software* da Motorola S/A, descobriu como driblar o protocolo de comunicação do transmissor/receptor de infravermelho. Assim, David Baum e Markus Noga, em alguns dias de trabalho coletivo, desenvolveram um protocolo para carregar (baixar) programas para o RCX, com velocidade de transmissão quatro vezes superior ao original. Não demorou muito para que David Baum implementasse um novo sistema de programação (um compilador), para o desenvolvimento de código para o RCX, em um dialeto da linguagem 'C', chamada NQC (Not Quite C) [5]. Ao mesmo tempo, Markus Noga decidiu abandonar o sistema operacional da LEGO e escrever o seu próprio, com um conjunto de rotinas mais eficientes para o controle do RCX, o qual chamou LEGOS [27].

Desde então, outros sistemas operacionais e compiladores para linguagens de programação foram desenvolvidos, principalmente, a partir de utilização de técnicas de engenharia reversa do *hardware*, feitas pela comunidade científica, usando ambientes e códigos, principalmente *software* livre (*open-source*) [37, 45].

2.3 Sistemas Operacionais

O sistema operacional (SO) é uma das camadas de *software* mais importantes em sistemas computacionais. Ele não apenas gerência recursos como CPU, memória e periféricos, mas também estende a funcionalidade do *hardware*, para suportar *softwares* aplicativos. Ou seja, na prática, o SO é a interface entre os programas de usuário e o *hardware*, que é controlado pela parte do SO executado em baixo nível. O SO do RCX é dividido em duas partes. Uma destas fica armazenada na memória ROM, sempre presente, chamada de micro-programa e outra armazenada na memória RAM chamada de *firmware*.

A primeira parte, o micro-programa, é responsável por controlar as saídas, fazer leitura de sensores, controlar os relógios internos, controlar um pequeno *display* (tela de cristal líquido), controlar quatro botões para interação direta com o usuário, fazer transferência de dados, controlar o infravermelho para comunicação com um computador ou com outro RCX, controlar um alto falante, dentre outras. Tais funções são implementadas em um nível mais baixo. Essa parte do SO apresenta algumas semelhanças com o programa armazenado no *BIOS* de um computador pessoal (PC).

Já a segunda parte, o *firmware*, "pode" prover uma interface mais fácil com a primeira parte,

permitir gerenciamento de memória (como paginação), implementar o controle de processos, implementar semáforos, implementar escalonador de processos, fornecer suporte ao fluxo de controle (como controle condicional e laços), permitir uso de operações básicas (uso da ULA), dentre outras funcionalidades. O *firmware* deve ser transferido para o RCX na primeira vez em que ele é ligado, após a troca de pilhas e quando a memória RAM for inicializada ("zerada").

Como mencionado anteriormente, alguns exímios programadores e grupos de usuários criaram novos *firmwares*, como o BRICKOS (nova versão do LEGOS), o LEJOS, dentre outros.

2.3.1 O *Firmware* da LEGO

O *firmware* da LEGO, ou *firmware* padrão, é um SO simples, o qual basicamente é constituído por uma máquina virtual. Uma das vantagens da máquina virtual é a abstração da arquitetura, com isso, caso a arquitetura seja modificada, o *firmware* ainda poderá manter compatibilidade com programas escritos antes das modificações, perante nova implementação da máquina virtual. Outra vantagem é que a máquina virtual apresenta alta estabilidade.

Uma das principais desvantagens da máquina virtual é a redução da velocidade de execução (apesar da utilização de diversas técnicas de otimização) e a ocupação de uma área de memória a ela destinada, que poderia ser utilizada por outros programas. A tarefa básica da máquina virtual consiste em interpretar os *bytecodes* (códigos de programas), isto é, os *bytecodes* são reduzidos a uma série de instruções e dados apropriados para a máquina alvo, no caso o microcontrolador *Hitachi H*8/3292 do RCX. Os *bytecodes* são constituídos pelo *opcode* (código da operação) seguido por zero ou mais operandos. O *opcode* indica a ação a ser tomada. Se mais informações forem requeridas, estarão codificadas em um ou mais operandos os quais seguem imediatamente o *opcode*.

Atualmente, existem duas versões do *firmware* da LEGO para o RCX chamadas de RCX1 e RCX2. Na versão RCX2 foram feitas diversas melhorias na máquina virtual. Tais melhorias permitem funcionalidades extras aos programas, isso sem perder a compatibilidade com programas escritos anteriormente para o RCX1. Dentre os compiladores que geram *bytecode* para a máquina virtual do *firmware* da LEGO destacam-se o Robolab, o RIS e o NQC.

2.3.2 O *Firmware* BRICKOS

O BRICKOS [8] (nova versão do LEGOS [27]) é um projeto – código fonte aberto. Ele é um SO preemptivo, padrão *POSIX*, multi-tarefa escrito para o RCX. Ele implementa gerenciamento de memória (como paginação), controle de processos, comunicação inter-processos, semáforos, escalonador de processos com prioridade, suporte ao fluxo do controle e controle total sobre o *display*, dentre outras funcionalidades. Além disso, o BRICKOS permite amplo uso da memória, sendo que o limite para o número de variáveis, de funções, de processos e de sub-rotinas é o próprio tamanho da memória.

Atualmente existem compiladores para 'C'/'C++' (compilador BRICKOS) e para Pascal (em fase de teste), que geram programas executáveis para o SO BRICKOS. Notam-se que esses compiladores, na realidade, suportam um 'dialeto' de 'C'/'C++' e de Pascal, isto é, eles suportam um conjunto restrito de funções. Além dos compiladores, há também um simulador (emulegOS) e um protocolo de troca de mensagens chamado de LNP (ver detalhes do LNP no Capítulo 3).

emulegOS

O emulegOS é uma ferramenta desenvolvida para simular parcialmente o *hardware* do RCX, executando o SO BRICKOS. Seu principal objetivo é propiciar um ambiente mais confortável para testar e eliminar erros dos programas em desenvolvimento. O emulegOS é um *software* inicialmente escrito para o SO *Linux*, o qual interpreta códigos 'C'/'C++', escritos para o BRICKOS, emulando o comportamento destes códigos, em uma janela (ver Figura 2.4), tal qual ocorreria em um sistema real.

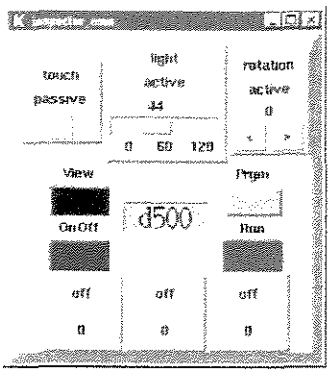


Figura 2.4: Simulador para RCX (emulegOS).

Algumas características principais:

- a interface visual é bem simples, permitindo que o usuário configure e interaja com os sensores, com o programa em funcionamento, para simular eventos externos, além de mostrar os estados atuais dos motores (virtualmente) conectados às saídas A, B, e C;
- existência de uma camada (API) para emular as rotinas do SO BRICKOS, desta forma, a maioria do código é executado numa máquina virtual, simulando o BRICKOS no *Linux*, incluindo a sustentação multi-tarefa e troca de mensagens;
- "emulação do mundo real", o emulegOS simula algumas características mecânicas de um robô, como um sensor de rotação girar, quando um motor funcionar ou um motor ligar 'x' segundos após um sensor de toque ser tocado, dentre outras;

- ganho de tempo e facilidade na eliminação de erros.

2.4 Compiladores

O nome compilador, criado nos anos 50, significa um programa, cuja função principal é o processo de tradução de uma linguagem fonte para uma linguagem objeto. Pode-se dizer que o compilador traduz um programa descrito em uma linguagem de alto nível, para um programa equivalente em código de máquina de um processador (baixo nível). Os *cross-compilers* são compiladores, que são executados em uma determinada arquitetura, mas que geram códigos para serem executados em uma outra arquitetura alvo. Os compiladores existentes para o RCX são na verdade todos *cross-compilers*, mas para simplificação serão denominados simplesmente de compiladores. A seguir serão discutidos alguns desses compiladores.

2.4.1 O Compilador Robolab

O Robolab [42] é um compilador integrado a um ambiente de programação visual fornecido pela LEGO e desenvolvido para o SO *Windows*. Ele gera código para ser interpretado pela máquina virtual do *firmware* da LEGO.

Utilizar o Robolab é relativamente fácil, nele a programação é baseada em ícones ou componentes (ver Figura 2.5), ou seja, a programação é feita dispondo esses componentes em uma seqüência lógica e conectando-os através de fios (ver Figura 2.6).

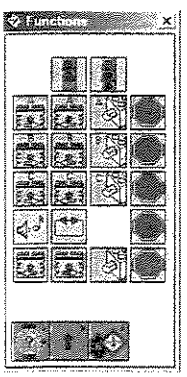


Figura 2.5: Alguns Componentes Disponíveis no Robolab.

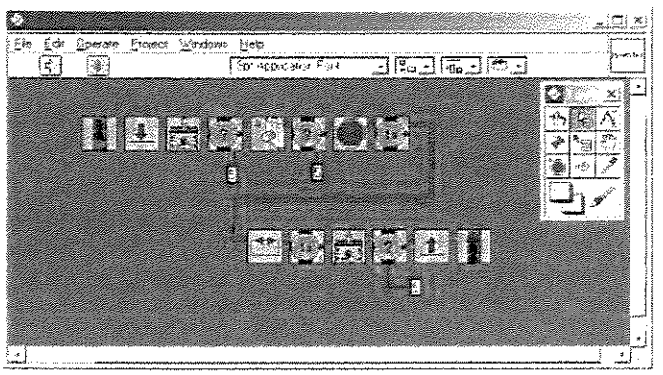


Figura 2.6: Um Programa Desenvolvido no Robolab.

O Robolab foi inspirado no *LabVIEW* (um produto da *National Instrument*), ele é bem didático e possibilita a usuários sem conhecimento profundo de programação desenvolverem programas para controlar o RCX com facilidade e rapidez. Porém, essa simplicidade acaba limitando todo o potencial oferecido pelo RCX que pode ser explorado. As linguagens gráficas

são freqüentemente mais fáceis de aprender (nenhum erro de sintaxe), mas são geralmente mais tediosas de usar do que uma linguagem textual. Ainda, as metáforas gráficas de codificação dessas linguagens podem limitar significativamente os tipos de programas, que podem ser escritos, bem como a utilização completa de todo o potencial do *hardware*.

2.4.2 O Compilador NQC

O "Not Quite C" (NQC) [5, 4, 3, 38] é um compilador para uma linguagem simples com sintaxe semelhante à linguagem 'C'. Pode-se dizer que sua linguagem é um dialeto do 'C'. O compilador NQC gera códigos para as máquinas virtuais presentes nos *firmwares* da LEGO para RCX, *Scout* e *CyberMaster*, produtos da linha LEGO *MINDSTORMS*.

O compilador NQC é desenvolvido e mantido sem qualquer suporte da LEGO. Tendo Dave Baum como idealizador e principal mantenedor. O NQC é um *software* livre licenciado pela *Mozilla Public License* (MPL) [35]. Possui versões oficiais até o momento, para os sistemas operacionais *Windows* (95/98/ME e NT 4,0) e *Mac* X, bem como versões não oficiais para *Linux*, *Solaris*, *FreeBSD*, *WinCE* e agendas *PDA*. Devido ao fato da sintaxe do NQC ser muito similar à linguagem de programação 'C', a tarefa de programar torna-se fácil para programadores com pequena experiência em 'C'. Como o NQC gera código para os *firmwares* da LEGO, significa que é possível armazenar no RCX programas, que foram gerados tanto pelo Robolab quanto pelo NQC.

A vantagem do NQC gerar código para os *firmwares* da LEGO é que ele se beneficia da estabilidade oferecida por esses *firmwares*. No entanto, os programas gerados pelo NQC estarão sujeitos a restrições impostas pelos *firmwares* da LEGO. Por exemplo, se o *firmware* não fornece suporte a ponto flutuante, o NQC também não poderá fornecê-lo. Outras alternativas de compiladores sem tais limitações são as que geram código para outros *firmwares*, como os: BRICKOS, LEJOS e o pbForth.

A interface do NQC permite compilação apenas através de linha de comando. Mas existem programas que fornecem ambientes integrados para construir, compilar, carregar (baixar) programas e receber dados, utilizando o NQC, sendo que o mais conhecido deles é o RCXCC.

RCXCC

O RCX *Comand Center* (RCXCC) é um ambiente de programação, desenvolvido para o SO *Windows* (95/98 e NT) [39]. A janela principal do RCXCC (ver Figura 2.7) é um *front-end* para o NQC. O editor possui a ferramenta *color-code*, que destaca palavras reservadas da linguagem, além de possibilitar a edição de vários arquivos de uma vez. Desse editor, pode-se invocar o compilador, bem como invocar o programa que transmite o código para execução ao RCX.

Dentre as propriedades mencionadas, o RCXCC possui ainda uma janela que permite monitorar o que está ocorrendo com recursos do RCX, como memória e sensores em tempo real.

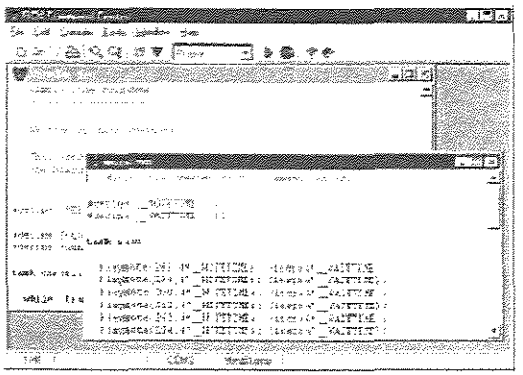


Figura 2.7: Janela Principal do RCXCC.

Nesta janela é possível controlar o robô diretamente ou através de *joystick*. Janelas de diagnóstico e de composição de músicas para o RCX também estão disponíveis (ver Figura 2.8). A versão mais recente possui uma documentação *on-line* completa tanto do RCXCC como da linguagem do NQC.

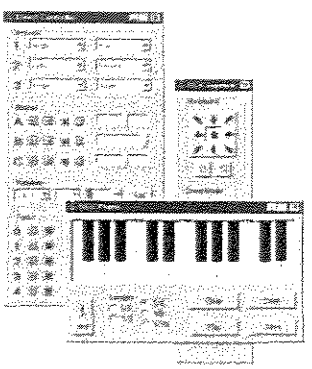


Figura 2.8: Outras Ferramentas do RCXCC.

2.4.3 O Compilador BrickOS

O BrickOS [8] é um compilador para linguagem 'C'/'C++', que contém uma parte das bibliotecas padrão do 'C' implementadas e suporta programação orientada a objetos, utilizando 'C++'. Com isso, basicamente qualquer programa em 'C'/'C++', que seja pouco menor que $32K$ e com algumas restrições de sintaxe devido ao *hardware* dedicado podem ser escritos, utilizando tais bibliotecas e compilados pelo BrickOS. Desta forma, o BrickOS torna-se uma boa opção para programadores de 'C'/'C++' que desejam ingressar na programação do RCX

sem a necessidade de aprendizado de uma nova linguagem.

O compilador BRICKOS gera códigos nativos para serem executados no *firmware* BRICKOS. Assim como o compilador NQC, o compilador BRICKOS e o *firmware* BRICKOS são desenvolvidos sem qualquer suporte da LEGO. Eles são *softwares* livres licenciados pela *GNU General License Public* (GPL) [22]. No momento, possui versão oficial somente para *Linux*, sendo possível recompilá-lo para executar sob o *Windows*.

Como os programas compilados pelo compilador BRICKOS serão executados no *firmware* BRICKOS, tais programas se beneficiarão de todas as facilidades oferecidas pelo *firmware* BRICKOS, como: ponto flutuante emulado, semáforos, utilização de *array* (vetores) com capacidade limitada somente pela memória, número de variáveis globais e locais limitado somente pela memória, dentre outras. Essas características tanto do *firmware* quanto do compilador BRICKOS fazem deste pacote de *software*, um dos mais poderosos, dentre os pacotes alternativos destinado ao RCX.

2.5 Considerações sobre *Hardware* e *Software*

O *hardware* LEGO permite prototipagem muito rápida de veículos robóticos, robôs e pequenas máquinas. No contexto deste trabalho, foram utilizados principalmente veículos robóticos autônomos, com processamento embarcado ou não. A realização de pesquisas em altos níveis torna-se muito mais rápida, mudando-se o *hardware*, se necessário, em tempo de desenvolvimento.

Os compiladores e *firmwares* foram testados exaustivamente, como será visto no Capítulo 6 que elucida os testes e experimentos, principalmente através de tabelas comparativas. Pode se adiantar que o compilador NQC foi, inicialmente, o preferido dos alunos, pela sua facilidade. Mas, posteriormente, o compilador BRICKOS mostrou-se mais plausível para pesquisas, fatores também discutidos no Capítulo 6.

Capítulo 3

Protocolo RemoteRCX

Apesar do ótimo aproveitamento do potencial do RCX, pelo SO BRICKOS (ver Capítulos 2), este pode não ser suficiente para implementar objetivos que, por exemplo, exigem uma gama de cálculos ou uma capacidade enorme de armazenamento de dados ou análise e processamento de imagens. Sendo assim, torna-se viável a implementação desses objetivos em máquinas com maior poder de processamento e armazenamento, como por exemplo, em computadores pessoais (PC).

Neste Capítulo, foi proposta uma ferramenta que permite que programas escritos em linguagem 'C' executados em um PC, possam escrever nas saídas e ler as entradas de um determinado RCX. Isso de forma transparente, sem que os desenvolvedores dos mesmos tenham que deter grande conhecimento do processo de comunicação entre um PC e um RCX. Basicamente, foi desenvolvido e implementado um protocolo de controle, que permite que um PC controle um RCX (ou vários), utilizando-se da torre de infravermelho. Também serão apresentados os conceitos, formato e funcionamento desse protocolo.

3.1 O Protocolo de Troca de Mensagens

O *LegOS Networking Protocol* (LNP) [29] é um protocolo de troca de mensagens presente no *kernel* do SO BRICKOS. Ele permite que dois ou mais RCX troquem mensagens de texto empacotadas. Os pacotes são transmitidos através de uma codificação, usando luz infravermelha. O protocolo LNP possui dois tipos de pacotes: um chamado de *LNP-integrity* e outro chamado de *LNP-addressing*.

O pacote *LNP-integrity*, garante que os dados foram transferidos sem alteração de seus valores, mantendo a integridade da mensagem. Isso é possível, porque o protocolo utiliza-se de um *byte* chamado de *checksum* (soma de controle) para verificar erros. Assim, quando é constatado um erro, o pacote é descartado sem nenhum aviso. As especificações de seus campos são (ver Figura 3.1):

Figura 3.1: Pacote *LNP-integrity*.

- ID: é o identificador do tipo de pacote, seu tamanho é de 1 *byte* e o valor do campo é 0xF0 para indicar que o pacote é do tipo *LNP-integrity*;
- LEN: é o comprimento do campo IDATA, seu tamanho é de 1 *byte*;
- IDATA: esse campo é a carga útil, o campo de dados, ou seja, é o campo destinado à mensagem propriamente dita, seu tamanho varia entre 1 a 255 *bytes*;
- CHK: *checksum* campo responsável pela integridade do pacote, seu tamanho é de 1 *byte*.

O pacote *LNP-addressing*, simplesmente adiciona endereçamento ao primeiro, de forma que o pacote passa a ser direcionado. Na verdade, ele encapsula os dados e os endereços no campo IDATA do pacote *LNP-integrity*. Assim, o pacote *LNP-addressing* tem funcionalidade similar ao UDP, o qual não garante que os pacotes cheguem, mas caso cheguem, garante a inexistência de erros. As especificações de seus campos são (ver Figura 3.2):

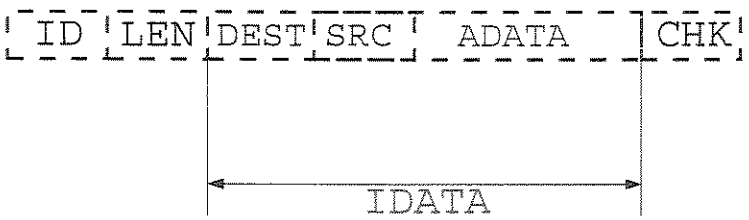


Figura 3.2: Pacote *LNP-addressing*.

- ID: é o identificador do tipo de pacote, seu tamanho é de 1 *byte*, o valor do campo é 0xF1 para indicar que o pacote é do tipo *LNP-addressing*;
- LEN: é o comprimento do campo IDATA, seu tamanho é de 1 *byte*;
- DEST: é o endereço do destinatário (destino), seu tamanho é de 1 *byte*;
- SRC: é o endereço do remetente (fonte), seu tamanho é de 1 *byte*;
- ADATA: esse campo é a carga útil, o campo de dados, ou seja, é o campo destinado à mensagem propriamente dita, seu tamanho varia entre 1 a 253 *bytes*;

- CHK: *checksum* campo responsável pela integridade do pacote, seu tamanho é de 1 *byte*.

O LNP também foi implementado para computadores pessoais. A implementação desse protocolo é chamado de LNPd e consta basicamente de um *daemon* (um tipo de *software*) e uma biblioteca, que possui a mesma interface da biblioteca lnp do compilador BRICKOS, ou seja, com os mesmas funções, porém ela pode ser utilizada por um compilador como por exemplo o 'gcc' no SO *Linux*, possibilitando assim, que *softwares* escritos na linguagem 'C', executados em um computador, troquem pacotes com um ou mais RCX. O *daemon*, também de nome LNPd, controla a torre de infravermelho conectada à porta PS2 ou serial do computador, tratando colisões e gerenciando a fila de pacotes.

3.2 Protocolo RemoteRCX

RemoteRCX foi o nome dado ao protocolo proposto e implementado. Ele permite que programas executados em um PC, possam escrever nas saídas e ler as entradas de um determinado RCX. Dessa forma é possível controlar um determinado RCX (ou mais de um), utilizando-se da torre de infravermelho, pode-se dizer que o RemoteRCX exerce basicamente a função de um protocolo de controle.

Ele utiliza-se do protocolo LNP, na verdade implementa uma camada de rede, sobre o LNP. O protocolo RemoteRCX possui dois tipos de pacotes: o rr-PC/RCX e o rr-RCX/PC. Os pacotes rr-PC/RCX e rr-RCX/PC, na verdade, encapsulam todos os seus campos no campo ADATA do pacote *LNP-addressing*.

3.2.1 Pacotes

O pacote rr-PC/RCX, como seu nome sugere, armazena os dados, código/controle a serem transmitidos e executados por um determinado RCX. As especificações dos campos de controle são (ver Figura 3.3):

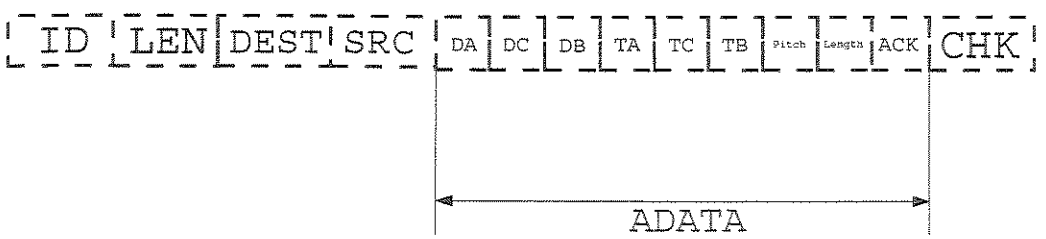


Figura 3.3: Pacote rr-PC/RCX.

- `DA, DC e DB`: são os sinais do valor das tensões a serem aplicados nas saídas A, C e B, respectivamente, do RCX, o tamanho de cada campo é de 1 *byte* e os valores dos campos podem ser 0x01 para positivo ou 0x02 para negativo;

- `TA, TC e TB`: são os valores das tensões em módulo a serem aplicados nas saídas A, C e B, respectivamente, do RCX, o tamanho de cada campo é de 1 *byte* e os valores dos campos podem assumir valores de 0x00 a 0xFF;

- `Pitch`: é nota musical (cifra européia), a ser aplicada ao alto falante do RCX, o tamanho do campo é de 1 *byte*;

- `Length`: é a duração da nota musical (o tempo da nota), o tamanho do campo é de 1 *byte*;

- `ACK`: (*acknowledgement*), tem por objetivo confirmar que o pacote de dados foi recebido. Nesse caso especifico, o campo `ACK` é um contador, um identificador do pacote, o tamanho do campo é de 1 *byte*.

O pacote rr-RCX/PC, também, como seu nome sugere, armazena os dados, valores dos sensores, a serem transmitidos e disponibilizados para um programa em execução em um PC. As especificações dos campos de sensores são (ver Figura 3.4):

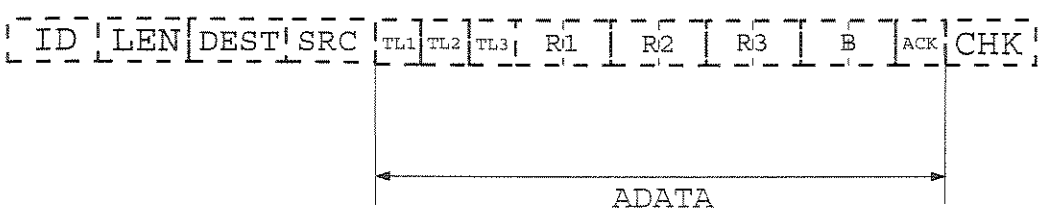


Figura 3.4: Pacote rr-RCX/PC.

- `TL1, TL2 e TL3`: são os valores lidos nas entradas 1, 2 e 3, respectivamente, do RCX. Analisados, como se nessas entradas estivessem conectados sensores de toque ou sensores de luz (infravermelho), o tamanho de cada campo é de 1 *byte*;

- `R1, R2 e R3`: são os valores lidos nas entradas 1, 2 e 3, respectivamente, do RCX. Analisados, como se nessas entradas estivessem conectados *encoders* (sensores de rotação), o tamanho de cada campo é de 2 *bytes*;

- `B`: é o valor da tensão da bateria em milivolts (*mv*), o tamanho do campo é de 2 *bytes*;

- ACK: (*acknowledgement*), tem por objetivo confirmar que o pacote de dados foi recebido. Nesse caso especifico, o campo ACK é um contador, um identificador do pacote, o tamanho do campo é de 1 *byte*.

3.2.2 Implementação

A implementação do protocolo RemoteRCX consta de uma biblioteca em 'C', para ser compilada pelo 'gcc' do SO Linux e um interpretador de comandos escrito em 'C' para ser compilado pelo BRICKOS e executado em um RCX com SO BRICKOS.

A biblioteca tem como função permitir que os programas desenvolvidos possam escrever nas saídas e ler as entradas de um determinado RCX. Isto de forma transparente, sem necessidade que os programadores detenham grande conhecimento do processo de comunicação entre um PC e o RCX. Ela apresenta uma interface simples, com quatro funções (ver Figura 3.5), as quais:

```
void StartRemote(unsigned char portRCX,
                 unsigned char portTower,
                 char *hostName);

int Connect();

void ClosedConnection();

rcxResult SetOutputs(int outA, int outB, int outC,
                     unsigned char pitch,
                     unsigned char length);
```

Figura 3.5: Funções da Biblioteca RemoteRCX.

- `StartRemote`: inicializa o valor da porta do RCX o qual será controlado, o valor da porta da torre e o nome da máquina a qual a torre está conectada;
- `Connect`: abre uma conexão com o *daemon* LNPd;
- `ClosedConnection`: fecha a conexão com o *daemon* LNPd;
- `SetOutputs`: incrementa o sinal ACK, empacota os comandos de escrita (saída do RCX) junto com um sinal de ACK no pacote rr-PC/RCX. Envia o pacote ao RCX especificado no comando `StartRemote`. Aguarda em espera ocupada, até que uma resposta do RCX (pacote rr-RCX/PC) chegue ou até que um determinado tempo expire – chamado de Tempo de Espera (TE). Caso a espera tenha terminado, porque o pacote rr-RCX/PC chegou, ela compara o ACK deste pacote com o ultimo ACK enviado e se forem iguais,

ela desempacota o pacote rr-RCX/PC e disponibiliza os valores de leitura dos sensores desempacotados em variáveis globais (ver Figura 3.6).

```
// Sensor touch. (0 or 1)
extern unsigned char touch_1;
extern unsigned char touch_2;
extern unsigned char touch_3;

// Sensor light. (0 - 100)
extern unsigned char light_1;
extern unsigned char light_2;
extern unsigned char light_3;

// Sensor rotation. (position)
extern short int rotation_1;
extern short int rotation_2;
extern short int rotation_3;

// Battery. (In mv)
extern int battery;
```

Figura 3.6: Variáveis Globais da Biblioteca RemoteRCX.

Denominou-se o interpretador de comandos simplesmente de *Shell*. Este é um programa executado em um RCX com SO BRICKOS. Basicamente, sua função é verificar continuamente se chegou algum pacote rr-PC/RCX para o RCX. Caso seja afirmativo, a *Shell* desempacota o pacote rr-PC/RCX e interpreta os comandos, executando-os logo em seguida. Depois, ela lê os sensores, empacota os valores lidos juntamente com o sinal de ACK do pacote rr-PC/RCX no pacote rr-RCX/PC e envia este pacote ao PC. A biblioteca da *Shell* apresenta uma interface simples, com três funções (ver Figura 3.7) as quais:

```
void StartShell(unsigned char portRCX,
                unsigned char portTower);

inline void RefreshSensors();

void RunShell();
```

Figura 3.7: Funções do Interpretador *Shell*.

- `StartShell`: inicializa o valor da porta do RCX e o valor da porta da torre da qual receberá os comandos;

- `RunShell`: verifica constantemente a chegada de um pacote rr-PC/RCX, caso afirmativo, desempacota o pacote rr-PC/RCX, interpreta os comandos e chama a função `RefreshSensors`;

- `RefreshSensors`: lê os sensores, empacota os valores lidos juntamente com o sinal de `ACK` do pacote rr-PC/RCX no pacote rr-RCX/PC e envia este pacote ao PC.

3.2.3 Perda de Pacotes

Apesar do LNP possuir boas implementações tanto para PC como para o RCX, elas estão sujeitas à perda de pacotes devido às sombras (obstáculos que impedem a transmissão do sinal). Como o RemoteRCX utiliza dessas implementações, ele também está sujeito à perda de pacotes. Porém, ao contrário o LNP, o RemoteRCX pode detectar se um determinado pacote foi entregue, uma vez que o mesmo utiliza-se do campo `ACK`.

Analisando a implementação do RemoteRCX, na qual o PC envia um comando a um RCX e fica aguardando as leituras dos sensores como confirmação, observam-se dois casos, quando esta confirmação não chegou e o Tempo de Espera expirou:

1. o pacote rr-PC/RCX não chegou ao RCX, assim o RCX não enviou o pacote de confirmação;

2. o pacote rr-PC/RCX chegou ao RCX, os comandos foram executados, o RCX enviou o pacote de confirmação (rr-RCX/PC), mas este, por algum motivo não chegou ao PC.

Para não ocorrer perda de pacotes, uma alternativa seria reenviar o pacote rr-PC/RCX, quantas vezes necessário, até receber o pacote rr-RCX/PC, o problema desta solução é que aumentaria o tempo médio de envio de pacotes e se caso ocorrer o Caso 2, o RCX poderá executar várias vezes o mesmo comando, o que para algumas aplicações seria inconveniente. Uma outra alternativa, seria fixar o número máximo de vezes de reenvio de um pacote da primeira alternativa, isso diminuiria a incidência de perda de pacotes. Dessa forma, o tempo médio de envio de pacotes não teria aumento significativo, mas caso ocorra o Caso 2, da mesma forma que na primeira, o RCX poderá executar várias vezes o mesmo comando. Por fim, também tem-se como alternativa não tomar nenhuma providência. Esta alternativa, foi a adotada uma vez que não aumenta o tempo médio de envio e como a torre tem maior poder de transmissão do que o RCX, os pacotes perdidos são em maior proporção os do Caso 2, assim, o número de comandos não executados é baixo em relação aos pacotes perdidos.

3.3 Considerações

No Capítulo 6, será apresentado um estudo e análise do tempo de comunicação do protocolo proposto, uma vez que se pretende que sua utilização seja em aplicações que exigem respostas em tempo hábil (aplicações de tempo real). O protocolo RemoteRCX mostrou-se eficiente em vários projetos discutidos no Capítulo 6. Sua implementação foi essencial para que vários trabalhos pudessem ser realizados e alguns deles publicados em eventos na área de Robótica [46, 47]. Assim, considera-se que seu desenvolvimento seja uma das contribuições mais importantes deste trabalho.

Capítulo 4

Sistema de Localização

A navegação permite aos robôs móveis se movimentarem livremente no seu ambiente de trabalho ora alcançando metas ora desviando de obstáculos. Isso é obtido através de um sistema de navegação. Tal sistema de navegação pode ser dividido em duas tarefas básicas: a de se localizar e a de evitar obstáculos [14]. Este capítulo dá ênfase à tarefa de se localizar (localização) devido ao fato de poder ser usada como retro-alimentação para controladores de trajetória.

Localizar um robô móvel consiste em determinar a sua posição e orientação no espaço em um determinado instante de tempo. Os métodos de localização de robôs móveis podem ser classificados em duas grandes categorias: métodos de localização relativa e métodos de localização absoluta [7]. Os métodos de localização relativa utilizam a localização do robô obtida no instante anterior, para estimar a atual, sendo a odometria e a navegação inercial, por exemplo, dois métodos baseados nesse princípio. Já os métodos de localização absoluta, utilizam apenas as informações atuais dos seus sensores, para determinar a localização do robô em relação a um referencial fixo absoluto. Exemplos desse tipo de localização são: o uso de balizas ativas, *map-matching* (casamento de mapas), marcos artificiais ou naturais e sistema de localização visual.

Neste Capítulo, foi proposta uma ferramenta de localização que utiliza-se de odometria e de um sistema de localização visual. Basicamente, pretende-se usar medidas esparsas de localização absoluta, dadas pelo sistema de localização visual, para corrigirem o sistema de odometria do robô. Porém, foi considerado que a câmera do sistema de localização visual deve estar livre a maior parte do tempo, para realizar outras operações, tais como em sistemas de percepção robótica. O propósito principal da ferramenta é requerer o uso da câmera, para corrigir os erros de odometria apenas em momentos no qual a mesma esteja ociosa. Uma característica essencial da ferramenta é permitir que o robô eventualmente saia do campo de visão da câmera ou passe por áreas de sombra (túneis, por baixo de cadeiras, de mesas, dentre outros), isso por um curto espaço de tempo sem perder sua localização. Este tempo sem correção será determinado a partir de uma análise do erro de odometria do sistema.

4.1 Odometria

Devido ao baixo custo da odometria, esse método de localização é bastante utilizado em robôs móveis com rodas. Ele utiliza *encoders* (sensores de rotação), os quais medem a rotação das rodas, permitindo calcular a localização do robô a partir de seu modelo cinemático. Esse método permite determinar a localização do robô, integrando os seus movimentos a partir de um referencial fixo.

Como já mencionado, a odometria utiliza a localização do robô obtida em instantes anteriores, para estimar a atual, isto é, ela é um método de integração. Nesse método, podem ocorrer erros devido ao escorregamento das rodas do robô, a arredondamento dos valores lidos pelos *encoders* e as medidas erradas das dimensões físicas do robô. Como esses erros na maioria das vezes ocorrem em uma mesma direção, isto é, os erros são polarizados, possuindo média diferente de zero, eles são propagados e acumulados com o tempo. Com isso, o uso de odometria pode-se tornar proibitivo, quando o robô percorre grandes distâncias, necessitando de um método para a correção desses erros.

Os erros de odometria podem ser classificados como erros sistemáticos ou erros não sistemáticos [13, 7]. Erros sistemáticos são causados por imperfeições no modelo cinemático do robô. Suas conseqüências podem ser desastrosas, uma vez que eles ocorrem durante a navegação do robô e se acumulam, gerando distorções na determinação da localização atual. Já os erros não sistemáticos são imprevisíveis, causados por situações que surgem inesperadamente. Ao contrário dos erros sistemáticos, esse tipo de erro não ocorre constantemente durante a navegação. Abaixo alguns exemplos de erros sistemáticos e não sistemáticos:

- erros sistemáticos:
  - desalinhamento das rodas;
  - diâmetros das rodas esquerda e direita diferentes;
  - erro das medidas dos diâmetros das rodas;
  - erro do comprimento do eixo;
  - resolução limitada dos *encoders*;
  - taxa de amostragem dos *encoders* limitada;
  - atrito.
- erros não sistemáticos:
  - terrenos irregulares;
  - derrapagem das rodas causada por:
    - terrenos escorregadios;

* aceleração linear acentuada;
* forças externas e internas;
* falta de contato entre a roda e o chão.

4.2 Sistema de Localização Visual

Um sistema de localização visual geralmente fornece posição absoluta global baseado em análise de imagens. O sistema usado é composto por uma câmera colorida localizada a $255cm$ do solo (ver Figura 4.1) e um *software* de processamento de imagens, responsável por calibrar tonalidades de cores e identificar e localizar rótulos. Este *software* foi desenvolvido no LabSis (Laboratório de Sistemas Inteligentes da Universidade Federal do Rio Grande do Norte - UFRN) por Anfranserai, semelhante ao *software* utilizado nos experimentos de sua dissertação [15].

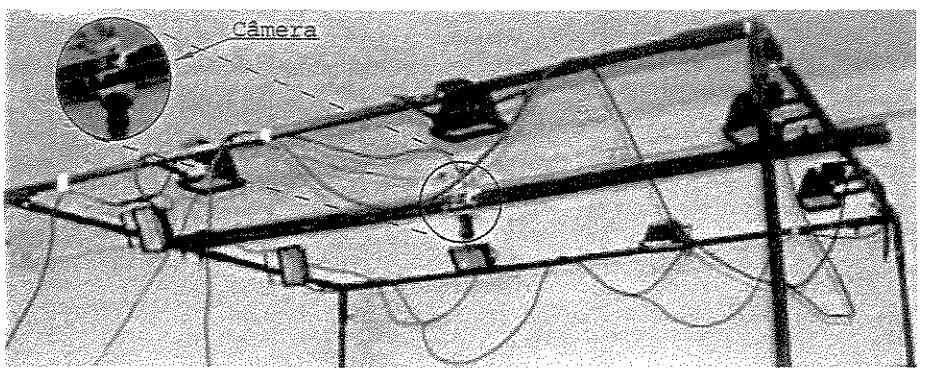


Figura 4.1: Câmera do Sistema de Localização Visual.

Este *software* analisa a seqüência de imagens quadro a quadro e em cada quadro procura dois rótulos, pequenas circunferências (ver Figura 4.2). Logo depois ele determina o centro de cada circunferência e através desses centros é calculado a posição X e Y, em *pixel*, na imagem e a orientação em graus. A captura de um quadro (imagem) junto à determinação da localização demora em media $33ms$, isto é, o *software* fornece aproximadamente 30 localizações por segundo.

Por ser um sistema de localização absoluta, este sistema fornece a posição calculada apenas por informações atuais, não ocorrendo erros acumulativos, porém uma desvantagem é o ambiente de trabalho ser limitado ao campo de visão da câmera.

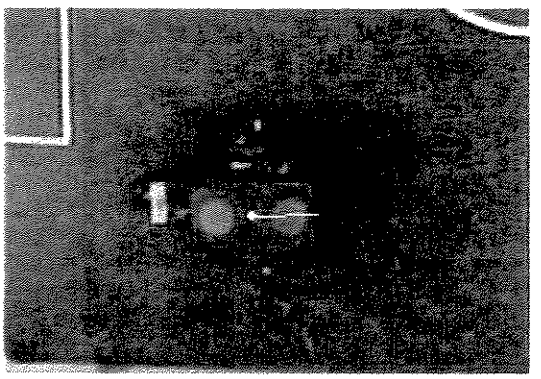


Figura 4.2: Rótulos para Localização do Robô.

4.3 Modelo Cinemático

A partir das informações medidas pelos *encoders* e conhecendo alguns parâmetros cinemáticos do robô, é possível deduzir as equações que irão determinar a sua posição e orientação em um dado instante. Isto é geralmente feito, incrementalmente, a partir de uma localização inicial conhecida. Veja o modelo cinemático ilustrado na Figura 4.3.

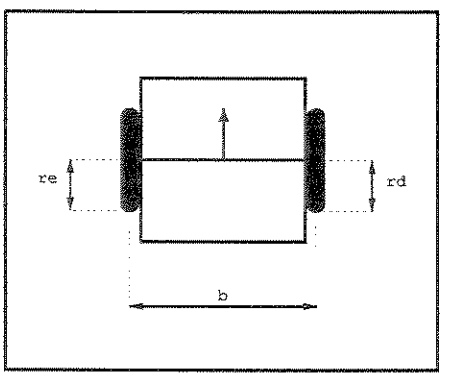


Figura 4.3: Modelo Cinemático.

Para descrever o modelo cinemático do robô, foi adotada a seguinte nomenclatura para as suas variáveis cinemáticas:

- b é o comprimento do eixo, ou seja, a distância entre as duas rodas;
- r_e é o raio da roda esquerda;
- r_d é o raio da roda direita;

- N_e e N_d são as variações do números de pulsos lidos pelos sensores esquerdo e direito respectivamente;
- N_r é a resolução dos sensores de rotação vezes eventuais reduções, isto é, N_r é o número de pulsos lidos pelo sensor de rotação, para o eixo da roda executar exatamente uma volta.

Como dito antes, a localização será calculada integrativamente. Para isso, basta calcular a variação do deslocamento linear, decompô-la em componentes, somá-las aos valores anteriores e depois calcular a variação da orientação e somá-la à orientação anterior.

Para calcular a variação do deslocamento linear do robô, aproxima-se o seu movimento (em um intervalo de tempo) a um arco de circunferência ilustrado na Figura 4.4, onde:

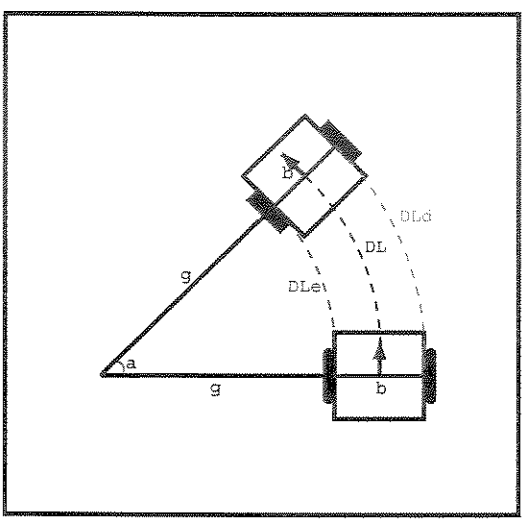


Figura 4.4: Variação do Deslocamento Linear do Robô.

- DL_e é a variação do deslocamento linear da roda esquerda;
- DL_d é a variação do deslocamento linear da roda direita;
- DL é a variação do deslocamento linear do centro do eixo;
- a é o ângulo de giro.

Calculando DL:

$$DL_e = 2\pi r_e \frac{N_e}{N_r} \tag{4.1}$$

$$DL_d = 2\pi r_d \frac{N_d}{N_r} \tag{4.2}$$

Analisando o arco menor:

$$2\pi \longrightarrow 2\pi g \tag{4.3}$$
$$a \longrightarrow DL_e \tag{4.4}$$

$$ag = DL_e \tag{4.5}$$

Analisando o arco maior:

$$2\pi \longrightarrow 2\pi(g+b) \tag{4.6}$$
$$a \longrightarrow DL_d \tag{4.7}$$

$$a(g+b) = DL_d \tag{4.8}$$
$$ag + ab = DL_d \tag{4.9}$$

Substituindo a Equação 4.5 na Equação 4.9:

$$DL_e + ab = DL_d \tag{4.10}$$
$$ab = DL_d - DL_e \tag{4.11}$$

Analisando o arco médio:

$$2\pi \longrightarrow 2\pi\left(g + \frac{b}{2}\right) \tag{4.12}$$
$$a \longrightarrow DL \tag{4.13}$$

$$DL = a\left(g + \frac{b}{2}\right) \tag{4.14}$$
$$DL = ag + \frac{ab}{2} \tag{4.15}$$

Substituindo as Equações 4.5 e 4.11 na Equação 4.15:

$$DL = DL_e + \frac{DL_d - DL_e}{2} \tag{4.16}$$
$$DL = \frac{DL_e + DL_d}{2} \tag{4.17}$$

Para calcular a variação da orientação do robô, aproxima-se o seu movimento (em um in-

tervalo de tempo) a um arco de circunferência ilustrado na Figura 4.5, onde $\Delta\theta$ é a variação da orientação.

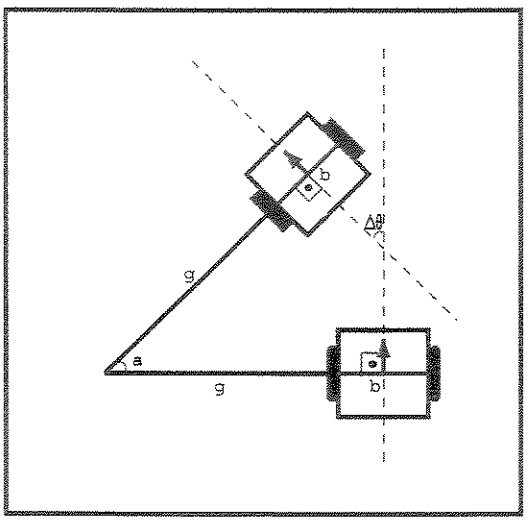


Figura 4.5: Variação da Orientação do Robô.

Analisando o quadrilátero cinza:

$$a = \Delta\theta \tag{4.18}$$

Substituindo a Equação 4.18 na Equação 4.11 e isolando $\Delta\theta$:

$$\Delta\theta b = DL_d - DL_e \tag{4.19}$$

$$\Delta\theta = \frac{DL_d - DL_e}{b} \tag{4.20}$$

Assim, as equações finais de odometria são:

$$X_{t+1} = X_t + DL_t \cos(\Delta\theta_t) \tag{4.21}$$

$$Y_{t+1} = Y_t + DL_t \sin(\Delta\theta_t) \tag{4.22}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{4.23}$$

4.4 O Protótipo

Dentre algumas plataformas com rodas, optou-se pela plataforma de Acionamento Diferencial, que com informações obtidas pelos *encoders* e pelo modelo cinemático, permite o cálculo

de sua posição e orientação relativas, utilizando-se de simples equações. Nessa plataforma existem duas rodas dispostas em cada lado do robô, acionadas por motores independentes. Velocidades distintas nas rodas proporcionam uma rotação no veículo de certo ângulo e velocidades iguais movem o robô em linha reta para frente ou para traz. Tal configuração provê a tal plataforma três graus de liberdade.

Após cinco protótipos, o protótipo final (ver Figura 4.6) consta de: duas rodas, de raio $40.8mm$, acionadas cada uma por um motor (da LEGO); redução entre os motores e essas rodas de $1/45$ (através de jogos de engrenagens); dois *encoders* (da LEGO) de 16 pulsos de resolução e duas rodas pequenas unidas por um mesmo eixo (rodas de apoio) desprezadas pelo modelo cinemático descrito na Seção 4.3.

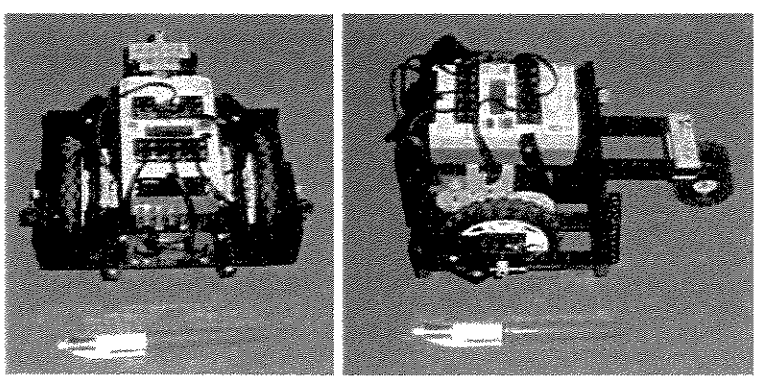


Figura 4.6: Protótipo com Acionamento Diferencial.

As variáveis cinemáticas deste protótipo são:

$$\begin{aligned} b &= 135mm \\ r_e &= 40.8mm \\ r_d &= 40.8mm \\ N_r &= 16 \cdot 45 = 720 \end{aligned}$$

4.5 Descrição do Sistema

Foi desenvolvida uma parte da ferramenta para o SO *Linux* (cálculo de odometria e RemoteRCX) e a outra parte para o SO *Windows* (sistema de localização visual). Desta forma, foi desenvolvido e utilizado um sistema de comunicação inter-sistemas. Este sistema de comunicação constitui-se de uma arquitetura mestre-escravo (servidor-cliente) implementada via *sockets*. Desenvolveu-se o servidor para o SO *Linux*, responsável pela odometria, controle do

robô (utilizando-se do RemoteRCX), correção da odometria e recepção de localizações calculadas pelo sistema de localização visual. Já o cliente foi desenvolvido para o SO *Windows*, responsável pelo sistema de localização visual e responsável por enviar as localizações para o servidor. Um esboço desse sistema pode ser visto na Figura 4.7.

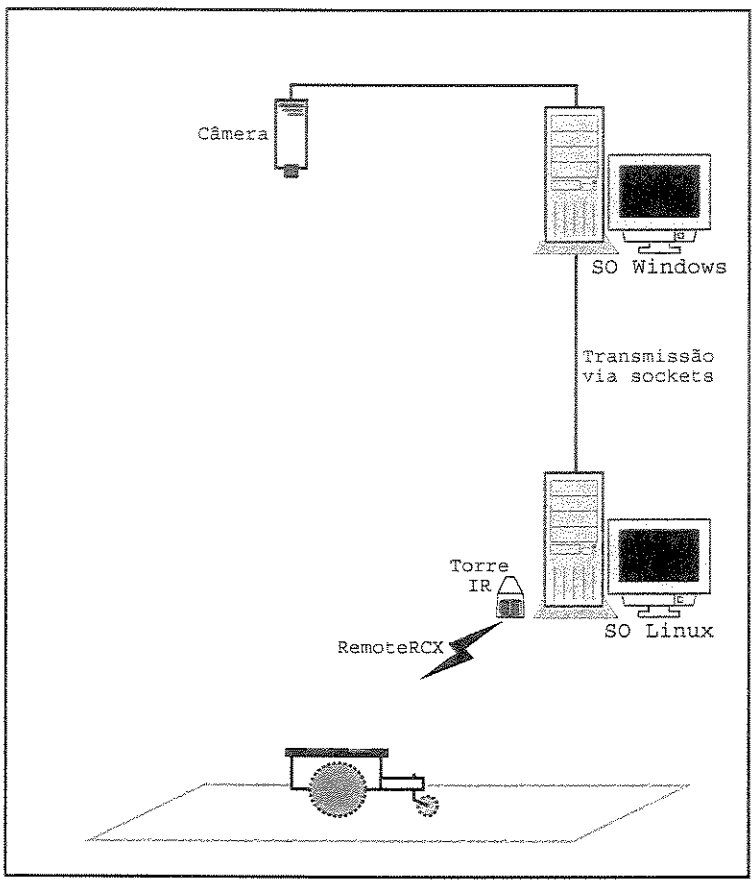


Figura 4.7: Sistema de Comunicação da Ferramenta de Localização.

No cliente, quando a câmera esta ociosa, isto é, não esta sendo utilizada para outros fins, o sistema de localização visual captura um quadro (imagem) calcula a localização como descrito na Seção 4.2 e envia esses dados ao servidor. São eles: o identificador do robô, a posição (X, Y) e a o orientação. Para evitar o *overhead* na transmissão, o cliente empacota todos esses dados em uma *string* e os envia de uma só vez ao servidor, que desempacota os mesmos, corrigindo a odometria.

O servidor foi implementado em três *threads* (tarefas): a primeira, responsável pela odometria; a segunda, responsável em receber os dados do cliente e a terceira, responsável pelo controlador, semelhante ao trabalho de Lora, Hemerly e Lages [30, 31].

A *thread* responsável pela odometria (a primeira) utiliza o protocolo *RemoteRCX* (ver Capítulo 3), para ler os dados dos *encoders*, realiza a odometria como elucidado na Seção 4.1, disponibiliza a localização calculada, além de gravá-la em arquivo para posterior análise; a *thread* responsável em receber os dados do cliente (a segunda), quando os recebe, sobrepõem os valores atuais da odometria e a *thread* responsável pelo controle (a terceira) possui um controlador proporcional simples, com o intuito de descrever trajetórias simples, poligonais, como: quadrados, triângulos, hexágonos, dentre outros. Esse controlador é alimentado pelas localizações disponibilizadas pela *thread*, responsável pela odometria e controla o robô, utilizando o RemoteRCX.

4.6 Correção de Erros Sistemáticos – UMBmark

Antes de realizar testes com o sistemas de localização proposto, foi realizado um procedimento para correção de erros sistemáticos do sistema de odometria, chamado de *University of Michigan Benchmark* (*UMBmark*) desenvolvido por Borenstein [7] e sucintamente detalhado a seguir.

O UMBmark destaca dois tipos de erros, os quais são passíveis de correção: o primeiro é chamado de erro tipo b, é a incerteza da distância entre as rodas, isso pode ocorrer, porque as rodas não tocam o chão em somente um ponto, mas em uma superfície que pode variar durante o deslocamento; o segundo é chamado de erro tipo d, é o erro devido a uma diferença entre os diâmetros das rodas, isso pode ocorrer, porque uma roda pode desgastar mais que a outra.

Conhecendo o valor desses erros é possível modificar o valor das variáveis cinemáticas, segundo as Equações 4.24 e 4.25, para correção dos mesmos, onde E_b é o erro do tipo b e E_d é o erro do tipo d, $b_{nominal}$ é o valor do tamanho do eixo medido e b_{atual} será o novo valor do tamanho do eixo.

$$E_b = \frac{r_d}{r_e} \tag{4.24}$$

$$E_d = \frac{b_{atual}}{b_{nominal}} \tag{4.25}$$

Nesse procedimento, o robô deve descrever $2n$ trajetórias quadradas, no mínimo 10 trajetórias, sendo uma metade (n) em sentido horário e a outra metade (n) em sentido anti-horário. Das trajetórias realizadas são retiradas as localizações finais calculadas ($calc$) e as localizações finais reais (abs). Com essas localizações são calculados os erros ex, ey e as médias dos erros (centro de gravidade horário (cgh) e anti-horário ($cgah$)) conforme as Equações 4.26, 4.27, 4.28 e a 4.29:

$$ex_{cgh/cgah} = X_{abs} - X_{calc} \quad (4.26)$$

$$ey_{cgh/cgah} = Y_{abs} - Y_{calc} \quad (4.27)$$

$$X_{cgh/cgah} = \frac{1}{n}\sum_{i=1}^{n}(ex)_{i,cgh/cgah} \quad (4.28)$$

$$Y_{cgh/cgah} = \frac{1}{n}\sum_{i=1}^{n}(ey)_{i,cgh/cgah} \quad (4.29)$$

Posteriormente são calculados os dois raios dos centros de gravidade:

$$r_{cgh/cgah} = \sqrt{X_{cgh/cgah}{}^2 + Y_{cgh/cgah}{}^2} \quad (4.30)$$

As trajetórias feitas pelo robô ficam inclinadas ou desajustadas em relação a um quadrado nominal. E, também, os lados das mesmas não são retas e sim aproximações de arcos. Veja Figura 4.8, onde: α e β são os ângulos de inclinação, R é o raio de curvatura do arco uv e L é o lado do quadrado nominal, os quais são dados pelas Equações 4.31, 4.32 e 4.33:

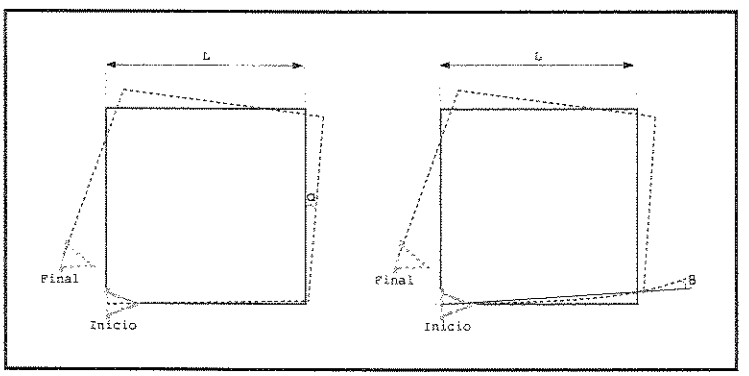


Figura 4.8: Ângulos α, β e o Raio de Curvatura.

$$\alpha = \frac{X_{cgh} + X_{cgah}}{-4L}\frac{180^o}{\pi} \tag{4.31}$$

$$\beta = \frac{X_{cgh} - X_{cgah}}{-4L}\frac{180^o}{\pi} \tag{4.32}$$

$$R = \frac{\frac{L}{2}}{\sin\left(\frac{\beta}{2}\right)} \tag{4.33}$$

Com o raio R é calculado a razão entre os raios das rodas, ou seja, é calculado E_d:

$$E_d = \frac{r_d}{r_e} = \frac{R + \frac{b}{2}}{R - \frac{b}{2}} \tag{4.34}$$

Por fim, é calculado E_b:

$$b_{atual} \longrightarrow 90^o \tag{4.35}$$

$$b_{nominal} \longrightarrow 90^o - \alpha \tag{4.36}$$

$$\frac{b_{atual}}{b_{nominal}} = \frac{90^o}{90^o - \alpha} \tag{4.37}$$

$$E_b = \frac{90^o}{90^o - \alpha} \tag{4.38}$$

4.6.1 Correção da Odometria

Apesar de todas as tentativas de correções realizadas, ainda existem erros consideráveis nas localizações fornecidas pelo sistema de odometria. Como proposto, um sistema de localização visual fornecerá uma correção para o sistema de odometria. A questão é definir se esta correção deve ser contínua no tempo ou não. No caso em questão, deseja-se que a câmera esteja livre para executar outras tarefas de percepção robótica, tais como atenção visual e reconhecimento de objetos. Portanto, a correção deve ser realizada esporadicamente.

4.7 Considerações

O sistema de localização foi exaustivamente testado e esses testes serão mostrados no Capítulo 6. A utilização da técnica UMBmark não produziu resultados significativos, devido a erros

não sistemáticos. A determinação empírica de parâmetros proporcionou uma melhora razoável para o sistema de odometria. Porém, foi somente com a inclusão da correção de erro, usando o sistema de localização visual, que minimizaram de forma substancial os erros, como será visto no Capítulo 6.

Capítulo 5

Sistema de Localização Visual (Horizontal)

Como descrito anteriormente (ver Capítulo 4) um Sistema de Localização Visual é um método de localização absoluta, que se utiliza apenas das informações atuais dos seus sensores para determinar a localização. Sua principal característica é a necessidade de extrair informações visuais relevantes da cena (imagem), em tempo hábil, determinando com estas informações orientação e posição.

Como o Sistema de Localização Visual apresentado no Capítulo 4 possuía a desvantagem da limitação do ambiente de trabalho, pelo campo de visão da câmera, neste Capítulo, é proposta uma ferramenta (um Sistema de Localização Visual), que pode amenizar essa desvantagem. Basicamente, esse sistema terá mobilidade e a visão será na horizontal e não mais na vertical como a do anterior. Assim, nesse sistema, a câmera estará acoplada a um robô móvel, na posição horizontal, permitindo a localização de demais robôs móveis em relação ao primeiro.

Neste Capítulo, foi proposta uma ferramenta de localização visual. Basicamente, essa ferramenta identificará rótulos/marcos (esferas), em quadros (imagem) capturados por uma câmera e determinará a posição e o tamanho (diâmetro) dos rótulos, disponibilizando estes dados os quais poderão ser usados por um controlador.

5.1 A Arquitetura de Controle Proposta

O sistema inicialmente proposto consistia de um robô dotado de sensores como câmeras e odômetros, chamado de robô Mestre (ver Figura 5.1) e um ou mais robôs sem sensores, chamados de robôs Escravos (ver Figura 5.2), sendo estes, controlados/guiados pelo robô Mestre, a partir do protocolo RemoteRCX, descrito no Capítulo 3. Assim, o sistema transmitiria comandos do computador para os vários robôs presentes no ambiente. Esses comandos também podem ser enviados do robô Mestre aos Escravos diretamente, mas isso não foi testado no presente trabalho.

A arquitetura de controle proposta é basicamente composta por três módulos de baixo nível

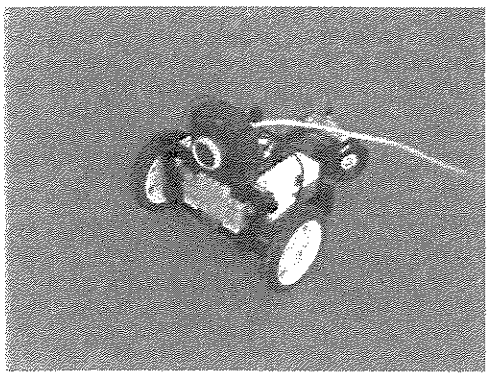


Figura 5.1: Robô Mestre.

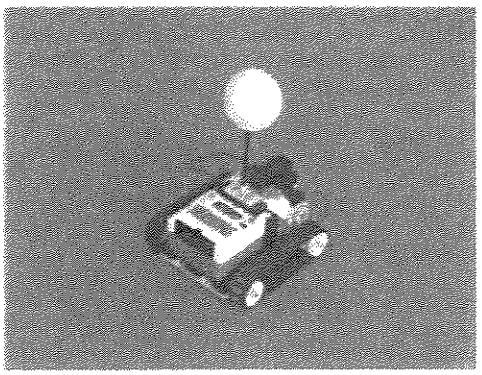


Figura 5.2: Robô Escravo.

e um módulo de alto nível, como pode ser visto na Figura 5.3. O primeiro módulo, o módulo "Aquisição", é responsável pela aquisição de imagens de dispositivos (câmeras) e a disponibilização dessas imagens. Esse módulo é executado no computador central e sua estrutura depende do tipo de câmera usada para aquisição das imagens. Sua saída é um *array* de *pixels* que codifica a imagem. Isto é transparente ao módulo "Visão", segundo módulo, o qual processa uma imagem codificada em um *array* de *pixels* e calcula a posição e orientação de cada robô dentro do campo de visão da câmera. O terceiro módulo, o módulo "Comunicação" é responsável por receber comandos de alto nível e por enviá-los a todos os robôs, isto é, ele provê comunicação entre o computador e os robôs.

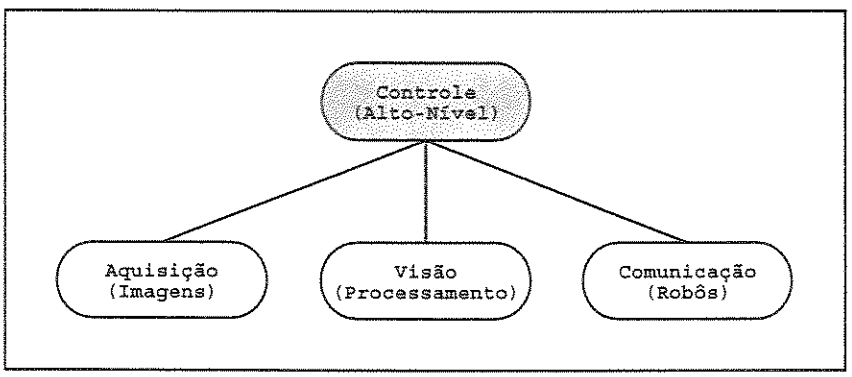


Figura 5.3: Arquitetura do Sistema.

Já o módulo de alto nível, o módulo "Controle" é responsável na prática por mapear o mundo real, em uma representação geométrica interna, gerando trajetórias e passando comandos de movimentos aos robôs. Isto é, dada uma localização (posição e orientação) atual e uma localização alvo, ele gera todos os comandos necessários, para que um robô específico se mova do atual para o alvo. Fatores como desvio de obstáculos e planejamento de trajetórias não serão

explanados aqui, uma vez que foge do escopo do presente trabalho.

5.2 Visão Robótica

Dentre os sentidos disponíveis em seres biológicos, a visão é um dos mais poderosos. Através dela é possível captar uma grande quantidade de informações do meio, possibilitando interagir com este meio de forma inteligente, tudo sem nenhum contato físico direto (pelo menos na fase adulta), aprendendo as posições e formas dos objetos e as relações entre eles. Tentativas de dar às máquinas o sentido da visão, a partir de imagens fornecidas por câmeras, levaram à definição de técnicas, que se aglomeraram sob as áreas de visão robótica ou visão computacional.

5.2.1 Câmera

Neste trabalho, a câmera utilizada é do tipo *webcam*, fornecida pela LEGO (ver Figura 5.4). Câmeras do tipo *webcam* apresentam algumas vantagens, como: o preço, não necessitam de *hardware* adicionais para aquisição de imagens, peso reduzido, simplicidade, baixo consumo de energia e alguns modelos podem atingir valores de 30 quadros por segundo. Porém, a qualidade de imagem é baixa, mas é suficiente para aplicações de controle e posicionamento.

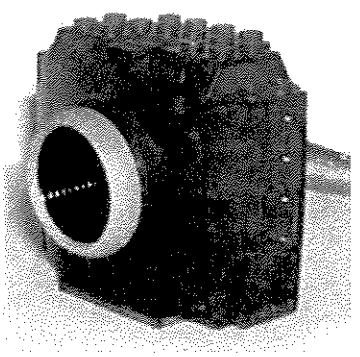


Figura 5.4: Câmera da LEGO.

O uso de câmeras como sensores possibilita a extração de sinais e características do ambiente, sem a necessidade de haver contato físico [23]. Uma das tarefas abordadas em robótica, utilizando-se de uma câmera como sensor, que interessa a este trabalho, é o controle e posicionamento em relação ao ambiente, para navegação de robôs móveis.

5.3 O Sistema de Localização por Marcos

Como já mencionado a ferramenta de localização visual é responsável por identificar rótulos/marcos nos quadros capturados pela câmera. Estes marcos (*landmarks*) presentes no espaço de trabalho do robô podem ser naturais ou artificiais.

5.3.1 Definindo Marcos Visuais

Definiram-se como marcos quaisquer objetos que possam ser identificados em uma imagem e que possuam coordenadas de mundo conhecidas. Após a aquisição da imagem e da localização do marco na imagem, uma transformação de coordenadas de imagem em coordenadas de mundo ($Cord._{Imagem} \rightarrow Cord._{Mundo}$) permite localizar a posição do robô no mundo real.

Existem basicamente dois tipos de marcos: os marcos naturais e os marcos artificiais. Os naturais são marcos que se encontram no espaço de trabalho do robô (como por exemplo: as linhas formadas pelas lajotas no chão, as luzes no teto, dentre outros), enquanto os artificiais foram inseridos. Como os marcos naturais se encontram no espaço de trabalho do robô, não há necessidade de alterações do espaço de trabalho. Já, com o uso dos marcos artificiais há necessidade de alterações do espaço de trabalho do robô, porém, esses marcos apresentam vantagens da fácil localização dos mesmos na imagem, já que são projetados para esse propósito.

Inicialmente, foram escolhidos marcos artificiais, esferas brancas com $6cm$ de diâmetro (ver Figura 5.5). Essas esferas permitem determinar apenas a posição de cada um dos robôs Escravos. Os centros das esferas são colocados sobre os robôs Escravos, na mesma altura que se encontra o centro de projeção da câmera do robô Mestre, por isso, a altura do marco na imagem pode sempre ser recuperada a menos que hajam distorções causadas por deslocamentos rápidos do robô Mestre.

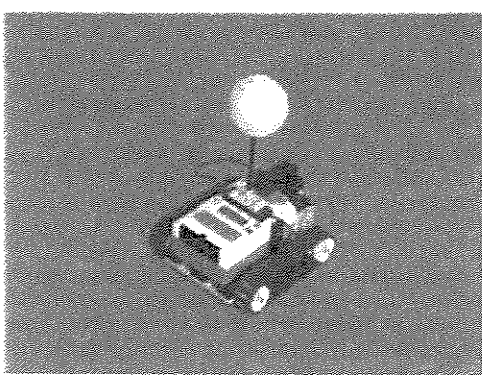


Figura 5.5: Esfera com Uma Cor.

Para se obter também a orientação, pode-se melhorar o marco, substituindo a esfera de uma

cor, por uma esfera com duas cores, também com $6cm$ de diâmetro (ver Figura 5.6). Essa esfera é dividida ao meio na posição vertical, em dois hemisférios, sendo um de cada cor. Com isso, a orientação de um robô Escravo pode ser determinada, respeitando alguma precisão. Mas nesse trabalho não foram recuperadas orientações.

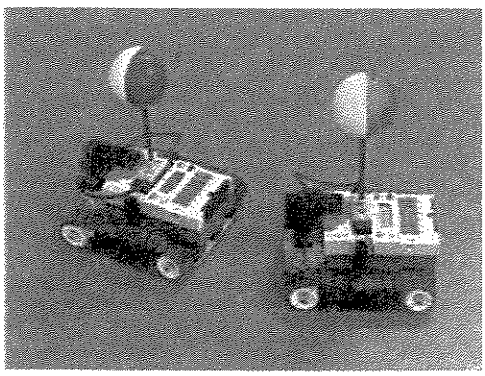


Figura 5.6: Esferas com Duas Cores.

Inicialmente, foi utilizado o robô Mestre, porém, o robô Escravo foi simulado por um pedestal com uma esfera tal como na Figura 5.7, mantendo o diâmetro e a cor da esfera bem como a altura de seu centro ao solo, igual a altura do centro de projeção da câmera do robô Mestre ao solo. Essa esfera é o marco que ficaria sobre os robôs Escravos, conforme visto na Figura 5.5.

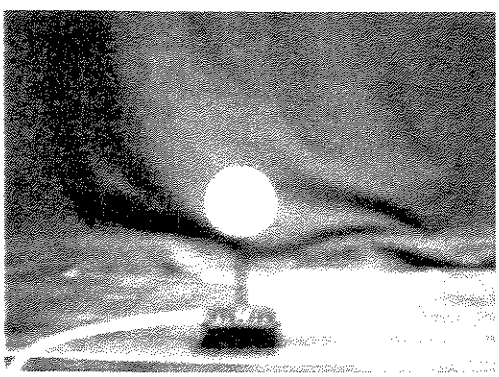


Figura 5.7: Imagem Inicial.

5.3.2 Localizando Marcos Visuais na Imagem

Na primeira implementação, considerou-se uma esfera com apenas uma cor. Como já mencionado, este marco possibilita determinar apenas a posição e não a orientação dos robôs. Ainda

se trabalhou em um ambiente controlado, tendo como plano de fundo um tecido da cor preta.

Após aquisição da imagem pelo módulo de "Aquisição", o módulo "Visão" basicamente segmenta a imagem, determinando o centro e o diâmetro da circunferência (projeção 2D da esfera) em coordenadas de imagem. Ressalta-se que na verdade a projeção da esfera na imagem é uma elipse, mas devido a grande proximidade de uma circunferência, será tratada como tal. Para isso, o histograma é calculado e uma segmentação é feita da seguinte forma: *pixels*, que possuam um valor acima da média do histograma em r, g e b mais uma constante, ficam com o valor 255 e *pixels*, que possuam valores abaixo ficam com valor 0. Assim, é possível com certa facilidade isolar a circunferência como pode ser visto na Figura 5.8.

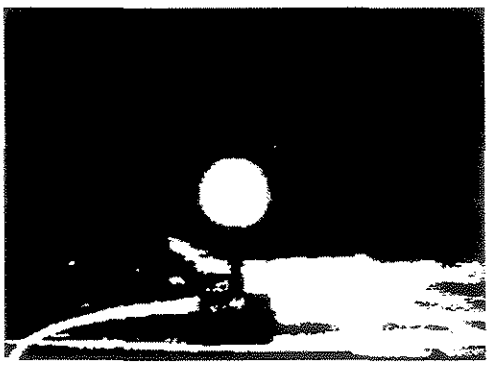


Figura 5.8: Imagem Filtrada.

Posteriormente, começa a determinação do centro e do diâmetro da circunferência, os quais são determinados conforme o processo ilustrado na Figura 5.9. Como o centro da esfera esta à mesma altura que o centro de projeção da câmera, espera-se encontrar a circunferência cortada por uma linha central e horizontal da imagem, que será chamada de "Equador". Assim, uma busca por um *pixel* branco é feita nas proximidades do Equador. Na verdade, essa busca ocorre sobre o Equador de forma intervalar, ou seja, de 6 em 6 *pixels*, uma vez que o limite estipulado para a menor projeção da esfera (circunferência) é de 11 *pixels*, garantindo que pelo menos um *pixel* dentro da circunferência seja analisado. Quando esse *pixel* é encontrado obtém-se o ponto **a**, cujo valor de sua ordenada é incrementado e decrementado até serem encontrados *pixels* que não sejam de cor branca, determinado assim os pontos **b** e **c**, que estão na borda da circunferência. Fazendo uma média das ordenadas de **b** e **c** obtém-se o ponto **d**, que possui a mesma ordenada do centro da circunferência.

Da mesma forma, o valor da abscissa de **d** é incrementado e decrementado até encontrarem-se os pontos **e** e **f**, também sobre a borda da circunferência. E com uma média das abscissas de **e** e **f** obtém-se o ponto **g**, o ponto central da circunferência.

A subtração dos pontos **e** e **f** fornece o diâmetro (D) da circunferência. Essa abordagem encontra os valores do diâmetro e do centro para a circunferência. Através do diâmetro da

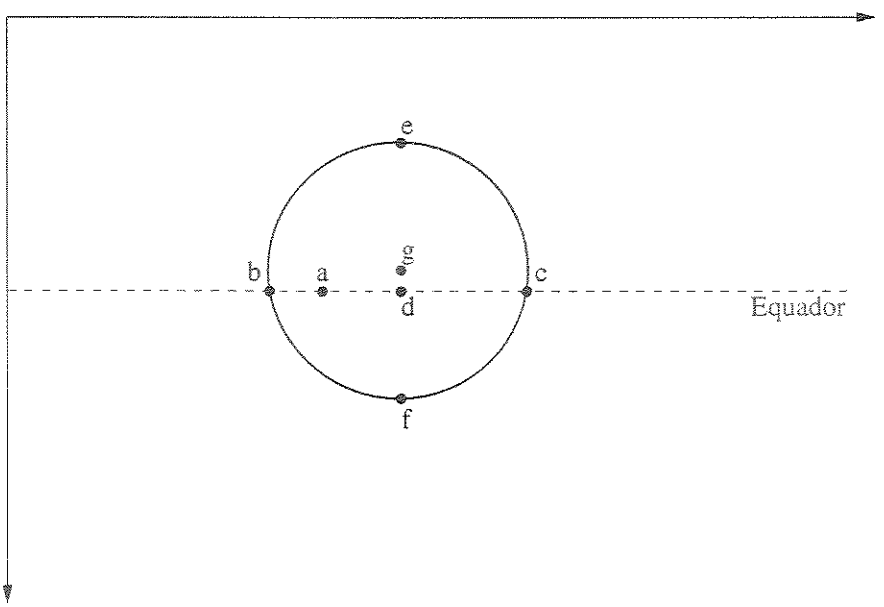


Figura 5.9: Procura da Circunferência.

circunferência é possível obter-se uma noção de distância (no eixo z) entre os robôs e através do valor do centro da circunferência é possível obter-se a distância em relação ao eixo y, pelo uso de uma equação de inversão de projeção simples. Para calcular a posição no mundo real, uma melhor calibração de câmera se faz necessária, porém, para os propósitos deste trabalho, não é necessário que seja de forma precisa. Foram desenvolvidos dois programas, que realizavam esse cálculo da imagem. Um deles com interface gráfica e outro com interface modo texto, a fim de diminuir a latência causada pela exibição da imagem.

Com a abordagem de colocar uma esfera branca em cima do robô Escravo para localização surgiram alguns problemas descritos a seguir:

- apenas um identificador sobre o alvo, não é possível estimar orientação;
- ambiente deve ser controlado para o algoritmo funcionar, ou seja, o chão bem como o fundo do ambiente tem de ser na cor preta, impedindo que o robô tenha uma liberdade maior;
- o controle torna-se mais complexo computacionalmente, sendo necessário um mapeamento de coordenadas para realizá-lo de melhor forma.

5.3.3 Determinando Efetivamente a Posição 3D da Esfera

Como descrito anteriormente, é possível determinar o diâmetro (D) e o centro da circunferência na imagem (**g**), para os diferentes marcos (esferas com duas cores ou com apenas uma cor). Como a câmera é calibrada, o tamanho do diâmetro fornece uma estimativa da distância entre a esfera e o robô Mestre. E a coordenada x do ponto **g** na imagem, permite calcular uma boa aproximação das coordenadas da esfera em relação ao robô Mestre. Na Figura 5.10, o X é a ordenada de **g** da circunferência na imagem e D é o diâmetro da circunferência, x é a ordenada do centro da esfera e d é o diâmetro da esfera. Os valores X e D são os valores fornecidos pelo Sistema de Visão.

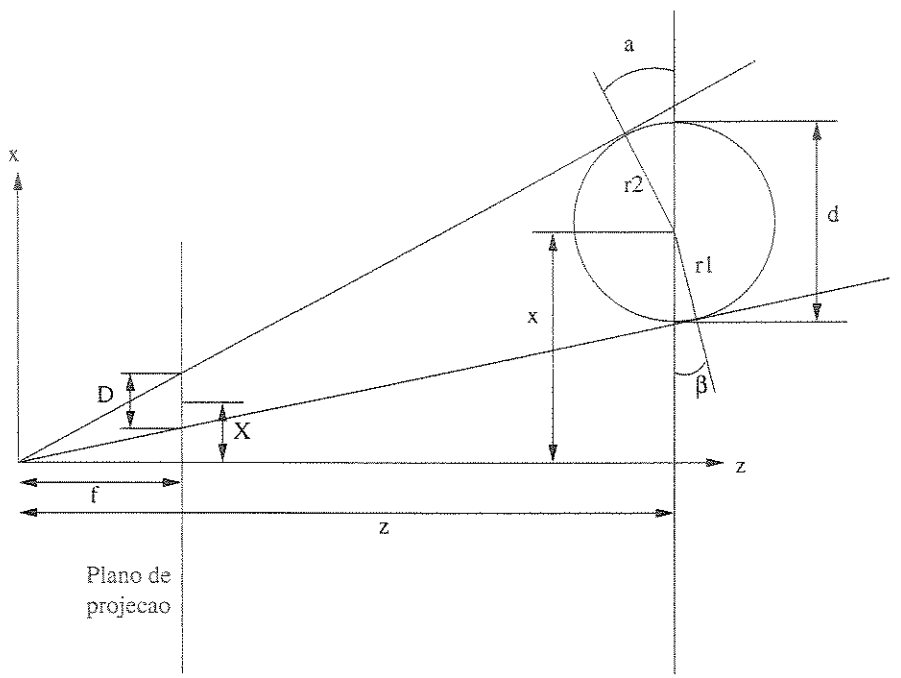


Figura 5.10: Projeção da Esfera.

Como f é conhecido com uma certa precisão através da execução de um procedimento grosseiro de calibração, a seguinte equação pode ser conseguida para se obter uma aproximação da coordenada z:

$$z = \frac{f}{D}d; \tag{5.1}$$

E a equação seguinte pode ser usada para determinação de x:

$$x = \frac{X}{f}z. \tag{5.2}$$

Note que essas equações são aproximações um pouco pobres dos valores reais de x e z, mas são suficientes para as necessidades em questão, supondo-se que os robôs podem ser controlados baseado em erro visual.

5.4 Tronco ou Cone de Visibilidade

O cálculo do tronco ou cone de visibilidade é muito importante para a parte de controle, pois através deste podem se definir os limites onde os robôs Escravos podem andar, para esquerda, para direita, para trás e para frente. Na verdade, o cálculo foi realizado de maneira bem empírica. Com a câmera ligada foi colado ao chão duas fitas adesivas pretas de forma que elas ficassem no limite esquerdo e direito da imagem. Logo depois foi definida e marcada com fita uma distância máxima do robô Escravo para o robô Mestre, para que a circunferência não ficasse muito pequena, e definida e marcada com fita uma distância mínima do robô Escravo para o robô Mestre, para que a imagem não contivesse uma esfera inteira, acabando com a visibilidade dos outros robôs. Estas fitas adesivas delimitaram assim, um trapézio o qual foi chamado de tronco ou cone de visibilidade.

5.5 Considerações

No Capítulo 6, será discutida a precisão na determinação do diâmetro da circunferência, bem como outros resultados sobre esse esquema, usando câmera horizontal para posicionamento.

Convém ressaltar que o esquema descrito resolve apenas o problema de determinação de posição dos robôs Escravos. Porém, caso sejam conhecidas as posições de dois robôs Escravos, pode ser determinada a posição e orientação do robô Mestre. Um sistema, usando esferas com duas cores, está sendo planejado e discutido no Capítulo 7.

Capítulo 6

Testes e Resultados

Vários experimentos foram realizados a fim de testar as ferramentas disponíveis e as desenvolvidas. Inicialmente, os compiladores e *firmwares* usados foram testados. Os experimentos seguintes foram feitos, visando validar as ferramentas implementadas. Foram feitos testes com o protocolo de comandos e com os sistemas de localização propostos. Convém ressaltar que esses últimos foram desenvolvidos através de implementações realizadas sobre o protocolo RemoteRCX.

6.1 Testes e Comparações de *Softwares* para LEGO

Para efetivar a análise comparativa entre os compiladores NQC e BRICKOS e seus respectivos SO, realizou-se alguns testes de instalação, verificando potencialidades das linguagens e o desempenho dos programas por eles gerados.

Para os testes de instalações do NQC foram utilizados os SO *Windows* (95 e 98) e *Linux*. No *Windows*, foi instalada a versão nqc-2.1r1, juntamente com o RCXCC, versão rcxcc-3.1, os quais mostram-se relativamente fáceis, bastando desempacotar e utilizar. A versão nqc-2.2.r2 para *Linux*, também mostrou-se muito fácil de instalar, bastando utilizar comandos como: `./configure` e `./make` e modificar permissões de escrita na porta de conexão da torre de infravermelho.

Quanto aos testes de instalação do BRICKOS, foram feitos apenas no SO *Linux*, nas versões brickos-0.2.6.10, legos-0.2.6 e legOS-0.2.5. Exigindo mais tempo para instalar o *cross-compiler* do microcontrolador *Hitachi* H8/3292, essencial ao BRICKOS.

Além dos testes de instalação, foram escritos um total de 40 programas para ambos os compiladores, com o objetivo de descobrir as potencialidades de cada uma das linguagens, tais como: comandos para controle de motores, leitura dos sensores, gerência de processos, semáforos e comunicação.

Para finalizarem os testes, foram idealizados quatro objetivos, com diferentes níveis de dificuldade, a serem cumpridos por um robô com o RCX "embarcado". Cada objetivo foi implementado em ambos os compiladores com a finalidade de comparar o desempenho dos programas gerados. Os objetivos propostos são que um robô:

1. pare assim que detectar qualquer obstáculo;
2. navegue sobre uma mesa, detectando e desviando-se das bordas da mesma;
3. tenha dois comportamentos básicos: desviar de obstáculos e seguir duas linhas no chão;
4. resgate objetos de isopor, de uma cor previamente determinada, localizados em uma pequena arena retangular.

Nos Objetivos 1 e 2, os quatro programas gerados, para cumprí-los, tiveram aproximadamente o mesmo tamanho e seus respectivos códigos tiveram o mesmo número de linhas. Constataram, ainda, que os mesmos foram de fácil implementação e atingiram os objetivos da forma esperada.

Quanto ao Objetivo 3 (ver Figura 6.1), devido ao BRICKOS utilizar semáforos e tratar multitarefa de forma muito elegante, a implementação ficou extremamente fácil e trivial. O contrário ocorreu com o NQC.

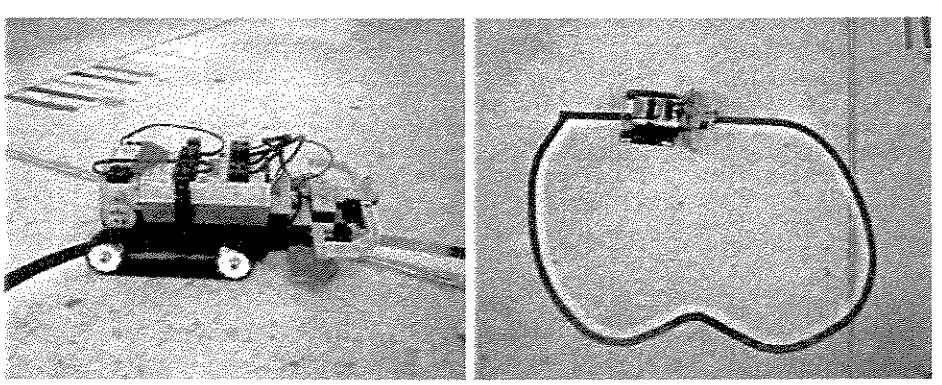


Figura 6.1: Robô Seguidor (Objetivo 3).

O Objetivo 4 foi pensado a partir de um exercício proposto para as disciplinas: "Robótica: Sistema Sensorial e Motor", na UNICAMP e UFMS [19], e "Percepção Robótica", na UFRN [20]. Tal exercício é uma competição entre dois robôs, onde vence o robô que resgatar mais objetos de isopor, de uma cor previamente determinada, localizados em uma pequena arena retangular. De posse dos códigos fonte, escrito pelos alunos das referidas disciplinas, foram escritos dois novos códigos: um para o NQC e outro para BRICKOS, e construídos dois robôs

idênticos. Constatou-se que o código gerado pelo BRICKOS para esse Objetivo ficou menor do que o gerado pelo NQC, como esperado, devido ao NQC gerar código para uma máquina virtual, o que não ocorre com o BRICKOS. Também observou-se que o programa escrito para o BRICKOS privilegiou-se da capacidade do mesmo em suportar maior número de variáveis, bem como vetores de maiores dimensões. Após realizadas as competições os dois robôs obtiveram números de vitórias parecidas (12 para o robô com código gerado pelo BRICKOS e 8 para o robô com código gerado pelo NQC). No ano de 2003, essa competição foi incorporada ao *Latin American IEEE Student Robottics Competition* e chamada de *Lego Rescue*.

6.1.1 Comparando os *Firmwares*

Algumas comparações entre os *firmwares* da LEGO e BRICKOS são apresentadas na Tabela 6.1, que elucida características e limitações dos mesmos.

Tabela 6.1: Comparando os *Firmwares*.

	RCX1	RCX2	BRICKOS
Programas	interpretado	interpretado	nativo
Paginação de memória	não	não	sim
Emulação de ponto flutuante	não	não	sim
Multi-tarefa	sim	sim	sim
Nº de tarefas	10	10	a memória é o limite
Nº de sub-rotinas	8	8	a memória é o limite
Nº de variáveis globais	32	32	a memória é o limite
Nº de variáveis locais/tarefa	0	16	a memória é o limite
Nº de *loop* aninhados	4	4	a memória é o limite
Nº de programas armazenados	5	5	8
Variáveis compartilhadas	não	sim	sim
Monitoramento de eventos	não	sim	sim
Suporte nativo a semáforos	não	não	sim
Licença	proprietária	proprietária	GPL

Da Tabela 6.1, podem ser observadas inúmeras vantagens do BRICKOS sobre o RCX1 e RCX2, tais como: emulação de ponto flutuante, maior velocidade de execução (programa executado em código nativo), possibilidade de maior número de sub-rotinas, de variáveis globais, de tarefas, de *loops* (laços) aninhados, de programas. As duas principais vantagens que podem ser destacadas, são: o suporte do BRICKOS aos semáforos e paginação de memória, sem a qual não seria possível aumentar a quantidade de variáveis, de tarefas, dentre outras. Em implementações que exijam bastante memória para armazenar tabelas de estado, como, por exemplo, em algoritmos de aprendizado por reforço, o BRICKOS se torna a opção viável e plausível, já que

o mesmo implementa paginação de memória. Algumas deficiências do RCX1, já superadas no RCX2, são o compartilhamento de memória entre processos e o monitoramento de eventos. O fato do RCX1 e RCX2 interpretarem o código fonte pode significar alguma vantagem sobre o BRICKOS, ou seja, o mesmo programa pode executar em várias arquiteturas diferentes. Isso pode ser útil por exemplo, no caso em que o *hardware* seja modificado pelo fabricante. A limitação de memória no RCX1 e RCX2 deve-se ao fato de terem sido desenvolvidos para um público alvo nada exigente ou que não precisa de tantos recursos de programação.

Com base em testes, em experimentos realizados e em análises obtidas no decorrer das disciplinas citadas na Seção 6.1, constatou-se que os *firmwares* da LEGO apresentaram alta estabilidade, enquanto o *firmware* BRICKOS ficou inoperante por dezesseis vezes. Isso pode ser significante na escolha da plataforma se estabilidade for uma condição básica. Verificou-se que tanto o RCX2 quanto o BRICKOS conseguem imprimir o valor de variáveis no *display*, o que é essencial na depuração de programas.

6.1.2 Comparando os Compiladores

Comparações entre os compiladores são apresentadas na Tabela 6.2 e na Tabela 6.3.

Tabela 6.2: Comparando os Compiladores.

	NQC	BRICKOS
Orientação a Objetos	não	sim
Bibliotecas padrão 'C'	não	sim
Velocidade de compilação	boa	boa
Arquitetura alvo	4	1
Licença	Mozilla	GPL

A Tabela 6.2 mostra uma comparação entre os compiladores NQC e BRICKOS, apontando algumas características e limitações importantes. Pode-se ver que algumas das vantagens do BRICKOS são a implementação de algumas das bibliotecas padrões da linguagem 'C' (`stdio, conio, math, etc.`) e o suporte à orientação a objetos. O BRICKOS não gera código executável para outras plataformas que não o RCX com o SO BRICKOS. Isso não vem ao caso, uma vez que estão sendo discutidas ferramentas para o RCX. Quanto às licenças, GPL provê mais liberdade de mudanças, o que pode ser uma vantagem.

Quanto a compilação, no BRICKOS é um pouco mais lenta, o que não acarreta em problemas, uma vez que isso é feito *off-line* (antes do robô entrar em execução). Pode-se considerar ambas compatíveis. Em compensação, a transferência de programas do computador para o RCX é mais rápida com o BRICKOS.

Tabela 6.3: Pesquisa.

	NQC	BRICKOS
Instalação	★★★★	★★★
Usabilidade	★★★	★★★★
Disponibilidade Documentação	★★★★	★★
Aprendizado	★★★★	★★★★★
Estabilidade	★★★★	★★

A Tabela 6.3 compara os compiladores com base em informações coletadas a partir de amostragem feita com os alunos das disciplinas citadas na Seção 6.1. Os resultados mostrados (estrelas) foram obtidos a partir da média ponderada de respostas aos questionários por mais de 50 alunos: categorizadas como fácil, normal e difícil.

Os ítens analisados foram:

- instalação: grau de dificuldade encontrado pelo usuário para instalação dos compiladores;
- usabilidade: esse ítem se refere a facilidade de uso dos compiladores, bem como outros *software* de apoio, como: emuladores, editores, ambiente de programação dentre outros;
- disponibilidade de documentação: número e facilidade de documentações para as linguagens e para os compiladores;
- aprendizado: nível de facilidade para o aprendizado das linguagens suportadas pelos compiladores;
- estabilidade: estabilidade dos *softwares* gerados pelos compiladores.

6.1.3 Considerações sobre os Compiladores e *Firmwares* Testados

Constatou-se que o NQC foi o preferido pelos alunos das disciplinas citadas na Seção 6.1, quanto aos critérios instalação, usabilidade, disponibilidade de documentação e estabilidade. Fato justificado pelo grau de dificuldade dos exercícios propostos, que nessas disciplinas não exigia alto desempenho tanto do *hardware* quanto do *software*. Por outro lado, o aprendizado do BRICKOS foi o quesito melhor apontado, devido à sua proximidade da linguagem 'C'/'C++'.

A partir das análises realizadas, não se pode generalizar, dizendo que o BRICKOS seja o melhor em todos os aspectos que o NQC. Porém, analisadas as tabelas comparativas, a pesquisa de campo e os testes realizados, pode-se, "*hoje*", destacar o BRICKOS como opção mais apropriada para trabalhar com o RCX em pesquisas. Seu uso por programadores experientes pode compensar sua falta de estabilidade. Além disso, falta de estabilidade é um preço que se paga pelas características essenciais à pesquisa (paginação, mais memória, dentre outras).

Finalmente, pode-se destacar o LNP como um fator diferencial na escolha do BRICKOS, o qual permitiu implementar o protocolo de controle RemoteRCX (ver Capitulo 3).

6.2 Testes com o Remote-RCX

Uma característica importante para o Protocolo RemoteRCX, já que este visa o controle de robôs móveis, é o tempo. Quanto menor o tempo de envio dos comandos, menos erros terá o controle. Assim, faz-se necessário o estudo do tempo de comunicação.

Fisicamente, no RCX e na torre, um *bit* 0 é codificado por um pulso com duração de $417us$ e com freqüência de $38kHz$, já um *bit* 1 é codificado por uma ausência com duração de $417us$. Assim, o tempo para a codificação de um *byte* é $3.336ms$. Porém, a transmissão envolve outros fatores, como: conversão de sinal digital para analógico e analógico para digital, cálculos para manter a integridade dos dados, tratamento de colisões, interrupção de processos, dentre outros. Tais aspectos influenciam o tempo de transmissão.

6.2.1 Estimativa do Tempo de Transmissão do LNP

Para mensurar o tempo de transmissão de *bytes* do LNP realizou-se um teste, utilizando o pacote *LNP-addressing*. Neste teste foram transmitidos pacotes com carga útil u, onde $u = 1, 2, 3, \ldots, 253$ *bytes*. Para cada valor de u, foram transmitidos 10 pacotes com carga útil u, mensurados os tempos de ida e volta e calculado a média. Os resultados do teste podem ser vistos no gráfico da Figura 6.2.

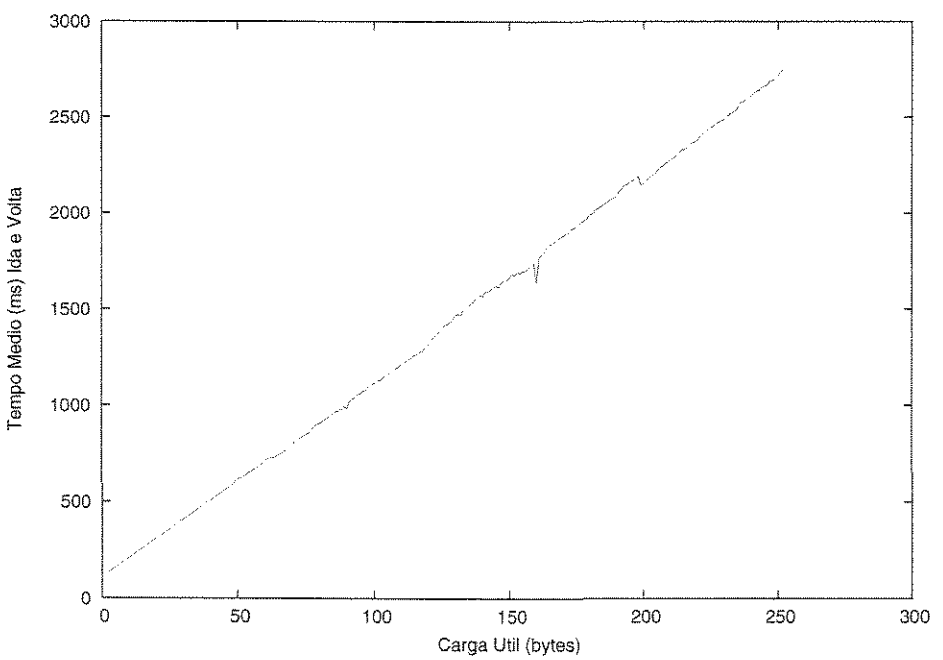


Figura 6.2: Tempo de Envio.

Analisando o gráfico da Figura 6.2, observou-se que o tempo é linear em função da carga útil. Assim, foi aplicado o método dos mínimos quadrados, nos dados do gráfico da Figura 6.2, para obter-se uma função com boa representatividade desses dados. Os resultados podem ser vistos na função da Equação 6.1 e no gráfico da Figura 6.3.

$$f(x) = 10.50x + 83.24 \tag{6.1}$$

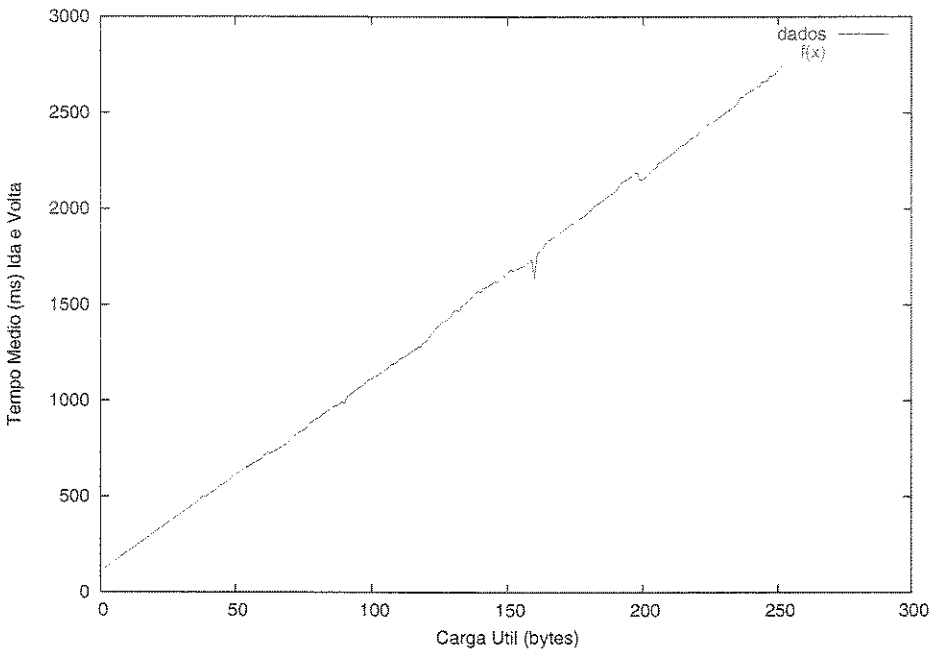


Figura 6.3: Tempo de Envio.

A função obtida é linear, do tipo $f(x) = ax + b$, onde: $f(x)$ é o tempo de transmissão (ida e volta) do pacote *LNP-addressing*, x é a carga útil em *bytes*, a é o tempo de transmissão (ida e volta) de um *byte* de carga útil e b é o tempo de transmissão (ida e volta) do cabeçalho do pacote *LNP-addressing*, da verificação de integridade, da verificação do endereço, das interrupções de sistema, dentre outras funções necessárias.

Como a é o tempo de transmissão (ida e volta) de um *byte*, o tempo para transmitir um *byte* (somente ida ou volta) é de $5.25ms$ valor próximo ao de $3.336ms$, calculado anteriormente. Essa diferença se deve, como já citado antes, a fatores como a conversão de sinais, cálculos para manter a integridade, tratamento de colisões, interrupção de processos, dentre outros.

6.2.2 Estimativa do Tempo de Transmissão do RemoteRCX

Como os dois pacotes do protocolo RemoteRCX encapsulam todos seus dados no pacote *LNP-addressing* e os pacotes são de tamanhos diferentes (rr-PC/RCX encapsula 9 *bytes* e rr-RCX/PC encapsula 12 *bytes*), pode-se estimar que o tempo de transmissão médio, pertença ao intervalo $[f(9), f(12)]$, ou pode-se estimar que o tempo de transmissão médio seja a média de $f(9)$ e $f(12)$, onde f é a função da Equação 6.1, obtendo-se assim:

$$\begin{aligned} f(9) &= 177.74ms \\ f(12) &= 209.24ms \\ \frac{f(9)+f(12)}{2} &= 193.49ms \end{aligned}$$

Assim, o tempo médio estimado de transmissão (ida e volta) do RemoteRCX, é de $193.49ms$. Mas como nesses cálculos não foram inclusos os tempos que o PC gasta para empacotar o rr-PC/RCX e desempacotar o rr-RCX/PC e os tempos que o RCX gasta para desempacotar o rr-PC/RCX, interpretar e executar os comandos, ler os sensores e empacotar essas leituras no rr-RCX/PC, realizou-se um teste, no qual, foi estabelecido um Tempo de Espera muito alto e feito e cronometrado n transmissões, para ser estimado o tempo médio de transmissão do RemoteRCX, que leve em conta esses fatores.

Ressalta-se que nesse teste foram descartados os pacotes perdidos ou dados como perdidos. Os resultados desse teste podem ser vistos no Histograma da Figura 6.4 e na Tabela 6.4, onde a unidade de tempo é dada em milisegundos, $\bar{x}$ é a média, mo é a moda, $fp(mo)$ é a freqüência porcentual da moda, md é a mediana, x_{min} é o menor tempo de transmissão e x_{max} é o maior tempo de transmissão.

Tabela 6.4: Medidas de Tendência Central.

TE	9999
n	10000
$\bar{x}$	204.75
mo	189
$fp(mo)$	38%
md	190
x_{min}	165
x_{max}	650

De posse dos dados da Tabela 6.4 é possível escolher um Tempo de Espera (TE) para o RemoteRCX, ressaltando que um TE alto implica em um x_{max} alto, o que compromete

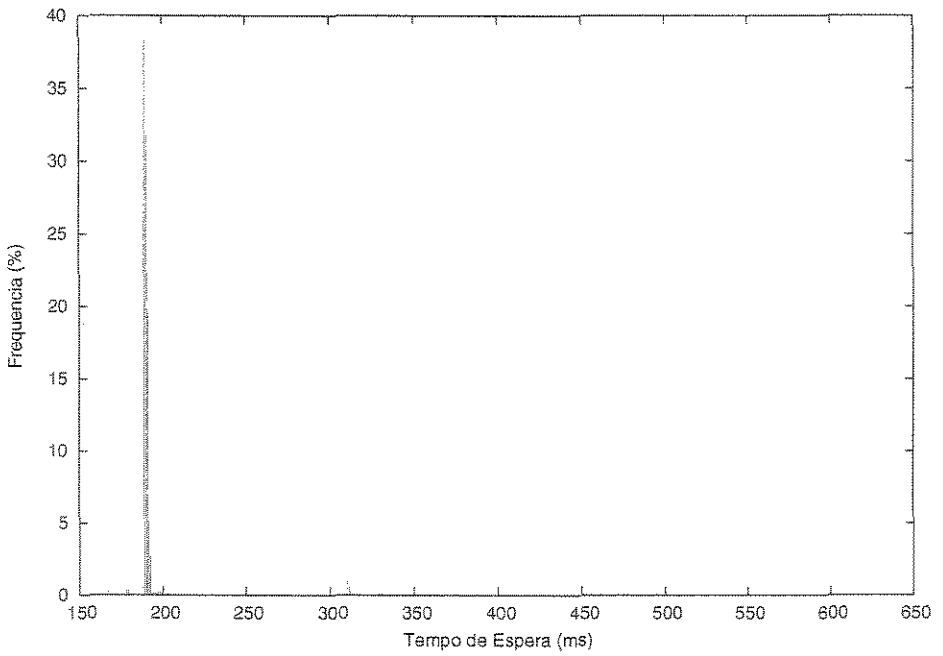


Figura 6.4: Histograma.

aplicações de controle, porém, um TE baixo pode aumentar o número de pacotes perdidos ou dado como perdidos, o que também compromete aplicações de controle. Assim, torna-se necessário estimar um bom TE.

6.2.3 Estimativa do Tempo de Espera

O Tempo de Espera (TE) é o tempo máximo que a implementação do RemoteRCX deve esperar por uma resposta de confirmação, ou seja, a implementação deve transmitir o pacote rr-PC/RCX e receber o pacote rr-RCX/PC como confirmação, em tempo menor ou igual ao TE, para não dar o pacote rr-PC/RCX como perdido.

Como mencionado anteriormente, a aplicação do RemoteRCX necessita de um bom TE. Para melhorar a visualização de candidatos a TE integrou-se o gráfico da Figura 6.4, obtendo-se assim, o gráfico da Figura 6.5. Neste gráfico, observa-se uma forte inclinação, devido a alta freqüência da moda e de vizinhos da moda, como o TE escolhido precisa minimizar o número de pacotes perdidos (ou dado como perdidos), este TE, deve se encontrar fora desta inclinação, ou melhor, deve se encontrar em um dos patamares desse gráfico. Assim, foram escolhidos alguns candidatos a TE, dispostos nos patamares, os quais foram 200, 250, 300, 350, 450, 550, 650.

De posse dos candidatos a TE foi realizado um teste, no qual para cada candidato a TE foram transmitidos n pacotes, cronometrados os tempos de transmissão (ida e volta) desses

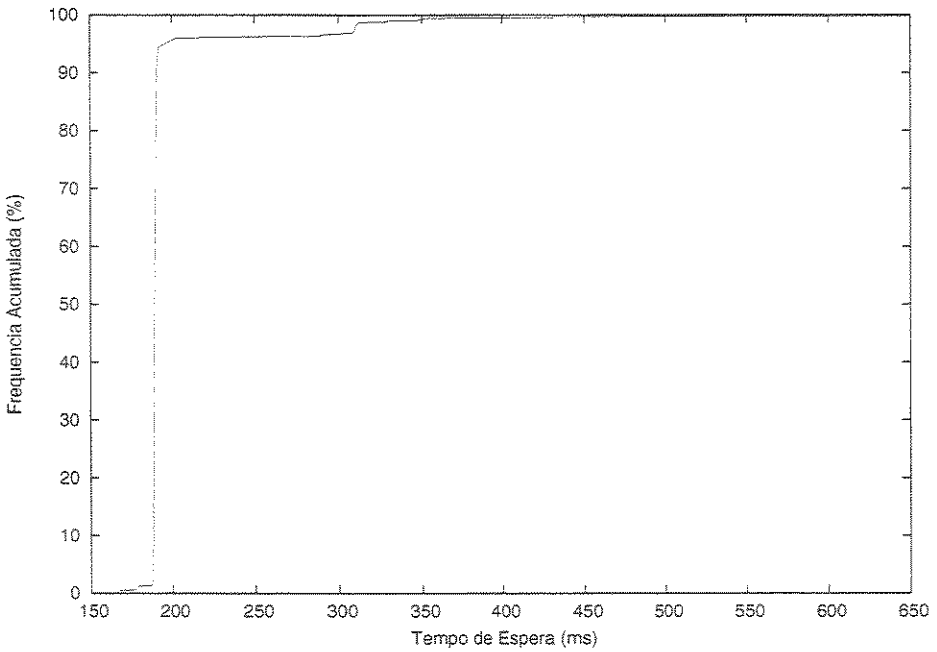


Figura 6.5: Freqüência Acumulada.

pacotes, a fim de se estimar o tempo médio de transmissão, calculando a média $\bar{x}$, a moda mo, a freqüência porcentual da moda $fp(mo)$, a mediana md, o menor tempo de transmissão x_{min}, o maior tempo de transmissão x_{max} e o número de pacotes perdidos (ou dado como perdidos) pp, tais resultados podem ser vistos na Tabela 6.5, onde a unidade de tempo é em milisegundos.

Tabela 6.5: Tempo de Espera.

TE	200	250	300	350	450	550	650
n	10000	10000	10000	10000	10000	10000	10000
$\bar{x}$	190.59	194.11	193.43	194.26	199.69	199.61	196.73
mo	191	191	191	191	191	190	190
$fp(mo)$	39%	39%	41%	41%	32%	42%	44%
md	190	190	190	190	191	190	190
x_{min}	165	165	166	166	165	165	165
x_{max}	200	250	300	350	450	550	650
pp	740	650	264	118	94	60	30
$pp(\%)$	7.40%	6.50%	2.64%	1.18%	0.94%	0.60%	0.30%

Analisando a Tabela 6.5, observa-se que a $\bar{x}$, a mo, a $fp(mo)$, a md e o x_{min} tiveram valores próximos para todos os testes. Essas variações de TE não influenciaram o tempo médio

de transmissão, mas influenciaram o número de pacotes perdidos e o x_{max}, como já mencionado anteriormente.

Para facilitar a escolha de um TE adequado, foram relacionados os candidatos a TE com os respectivos *pp* da Tabela 6.5, obtendo-se assim o gráfico da Figura 6.6.

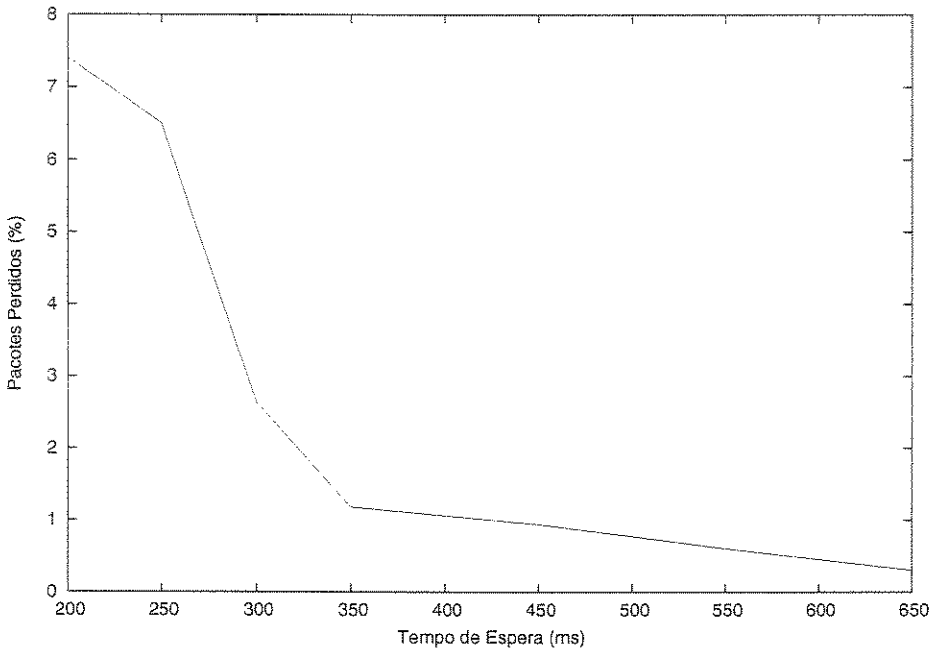


Figura 6.6: Pacotes Perdidos.

Analisando o gráfico da Figura 6.6 e a Tabela 6.5 foi escolhido para a implementação do RemoteRCX um TE padrão de $350ms$, devido a esse TE proporcionar um baixo número de pacotes perdidos (ou dado como perdidos), próximo a 1%, e um tempo de transmissão máximo de $350ms$. Porém, ressalta-se que esse valor pode ser facilmente alterado, dependendo das necessidades das aplicações que se utilizarem do RemoteRCX.

6.2.4 Considerações sobre o RemoteRCX

Constatou-se que o RemoteRCX, obteve um bom desempenho. Aplicações, utilizando-se do RemoteRCX, obtiveram êxito, e ficaram viáveis. Como exemplo de aplicações e projetos que utilizaram do RemoteRCX pode-se citar:

- construção de controladores cinemáticos para um mini-robô móvel (plataforma LEGO), por grupos de alunos da disciplina 'Sistemas Robóticos Autônomos', do curso de Engenharia de Computação - UFRN [2]. Esses controladores mostraram-se eficientes e aten-

deram as expectativas. Dessa forma, o protocolo mostrou-se uma ferramenta educacional, que permitiu implementar de forma concreta os conceitos elucidados na disciplina;

- construção de um sistema de localização, constituído por odometria e alimentação visual (Capítulo 4);
- projeto integrando, robôs, pessoas, mundo real, mundo virtual e realidade mista, o qual culminou no artigo "Hyperpresence - An Application Environment for Control of Multi-User Agents in Mixed Reality Space" [47] e no artigo "A Real Time Platform for Playing with Robots and Avatars in Mixed Reality Space" [46];
- projeto "Protocolos Sociais de Comunicação em Ambientes Virtuais Multiusuário", tese de doutorado em andamento, pela aluna Tatiana Aires Tavares do programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Rio Grande do Norte - UFRN.

As aplicações acima foram todas feitas para computadores com SO *Linux*, a maioria dos robôs foram imersos em um mundo real (uma sala de $20m^2$, com obstáculos reais, como cadeiras, mesas, cesto de lixo, pessoas, dentre outros) e, mesmo assim, as perdas de pacotes foram baixas. O alcance da comunicação foi alto e semelhante ao radial (apesar da comunicação ocorrer por infravermelho), uma vez que a luz reflete nas paredes e obstáculos do ambiente, atingindo quase todos os pontos do mesmo. O tempo de transmissão do RemoteRCX foi um fator limitante nesses projetos, mas solucionado com redução da velocidade dos robôs, utilizando-se de jogos de engrenagens. Outras versões do protocolo com otimizações para diminuir o tempo de transmissão estão sendo implementadas e testadas.

Todos os testes de comunicação desse capítulo foram feitos com baterias em máxima carga. Assim, os resultados desses testes podem variar com baterias com cargas mais baixas.

6.3 Testes com o Sistema de Localização

6.3.1 Aplicando o UMBmark

Nos testes, consideram-se os valores do sistema de localização visual (câmera vertical) como "valores reais", já que este sistema possui um erro pequeno e não acumulativo como descrito na Seção 4.2 do Capítulo 4. Nesse teste o robô (protótipo da Figura 4.6 do Capítulo 4) realizou dez trajetórias quadradas de lado $600mm$ aproximadamente. Os dados gerados pelo sistema de localização visual e pela odometria foram armazenados em arquivos e a partir dos mesmos foram construídos 10 gráficos de trajetória apresentados nas Figuras 6.7 e 6.8. Nesses gráficos podem-se fazer uma analogia na qual, a trajetória `'Odometro'` é a localização calculado pela odometria, ou seja, *é onde o robô pensa que está* e a trajetória `'Camera'` *é onde ele "realmente" está.*

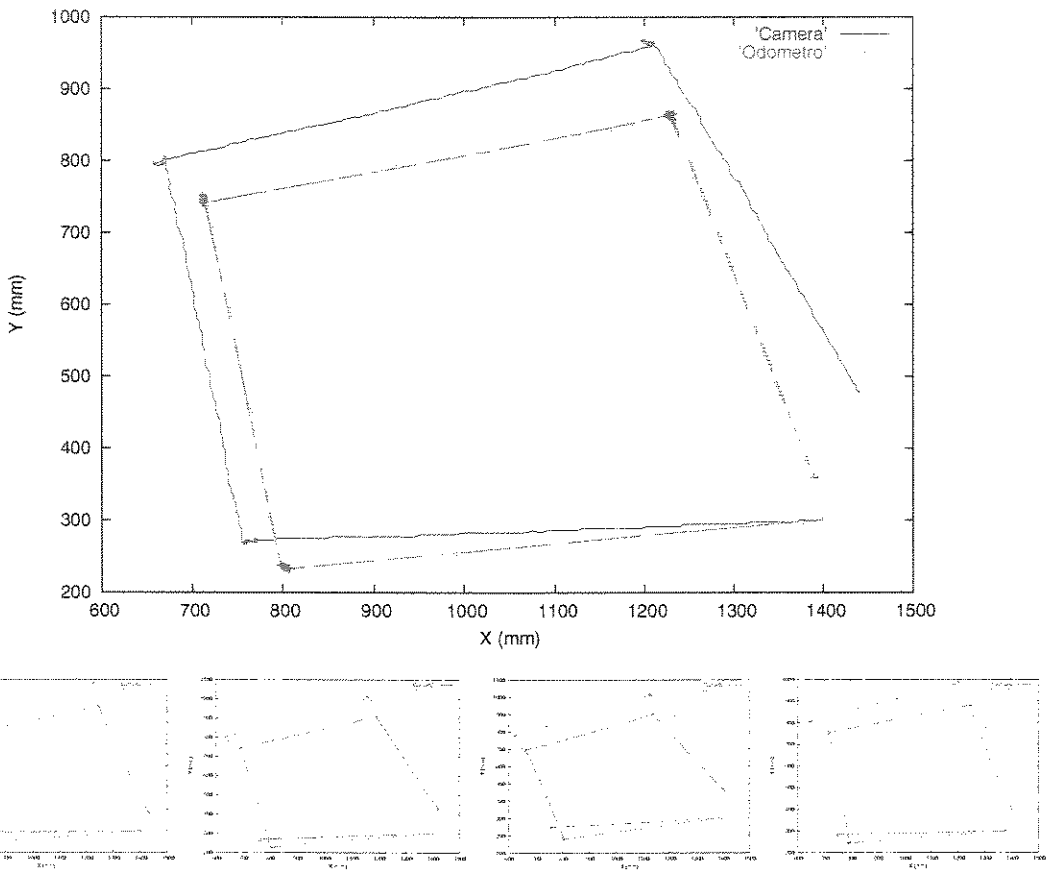


Figura 6.7: Trajetórias de Sentido Horário.

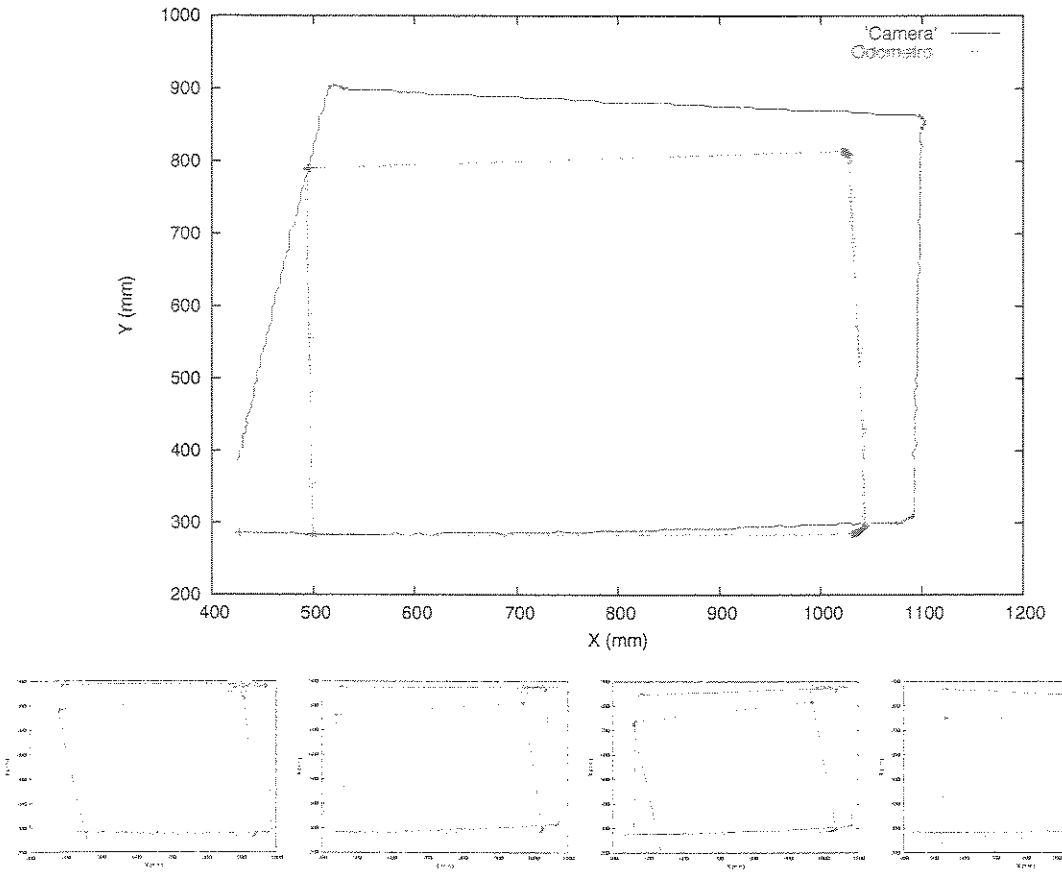


Figura 6.8: Trajetórias de Sentido Anti-horário.

Gerados os dez gráficos, calcularam-se os erros de acordo com a Equação 4.26 em relação ao eixo X e com a Equação 4.27 em relação ao eixo Y. Tais erros são apresentados nas Tabelas 6.6 e 6.7.

Tabela 6.6: Erros nas Trajetórias de Sentido Horário.

Trajetória	erro Xmm	erro Ymm
1	37.55	119.23
2	51.14	122.01
3	52.53	114.9
4	38.84	121.76
5	39.01	124.27

Tabela 6.7: Erros nas Trajetórias de Sentido Anti-horário.

Trajetória	erro $X mm$	erro $Y mm$
1	-75.52	101.46
2	-71.86	76.02
3	-78.84	97.97
4	-75.34	91.29
5	-74.23	106.54

Com os erros das Tabelas 6.6 e 6.7 calcularam-se os centros de gravidade (média dos erros) usando as Equações 4.28 e 4.29:

Centro de gravidade de *sentido horário*:

$$\begin{aligned} X_{cgh} &= 43.814mm \\ Y_{cgh} &= 120.434mm \end{aligned}$$

Centro de gravidade de *sentido anti-horário*:

$$\begin{aligned} X_{cgah} &= -75.158mm \\ Y_{cgah} &= 94.656mm \end{aligned}$$

Calcularam-se o α, o β e o R, através das Equações 4.31, 4.32 e 4.33:

$$\begin{aligned} \alpha &= 0.7482829 \\ \beta &= -2.8402473 \\ R &= -6054.323 \end{aligned}$$

Finalmente, utilizando-se as Equações 4.25 e 4.24, foram calculados E_d e E_b:

$$\begin{aligned} E_d &= 0.9779477 \\ E_b &= 1.008384 \end{aligned}$$

Com E_d e E_b modificaram-se os valores das variáveis cinemáticas do protótipo (descritas na Seção 4.3) para:

$$\begin{aligned} b &= 136.13mm \\ r_e &= 41.72mm \\ r_d &= 40.8mm \end{aligned}$$

Após as correções acima, realizaram-se novas trajetórias. A Figura 6.9 mostra as trajetórias realizadas, com a correção (`'Odometro-UMBmark'`), sem a correção (`'Odometro'`) e a "real" (`'Camera'`).

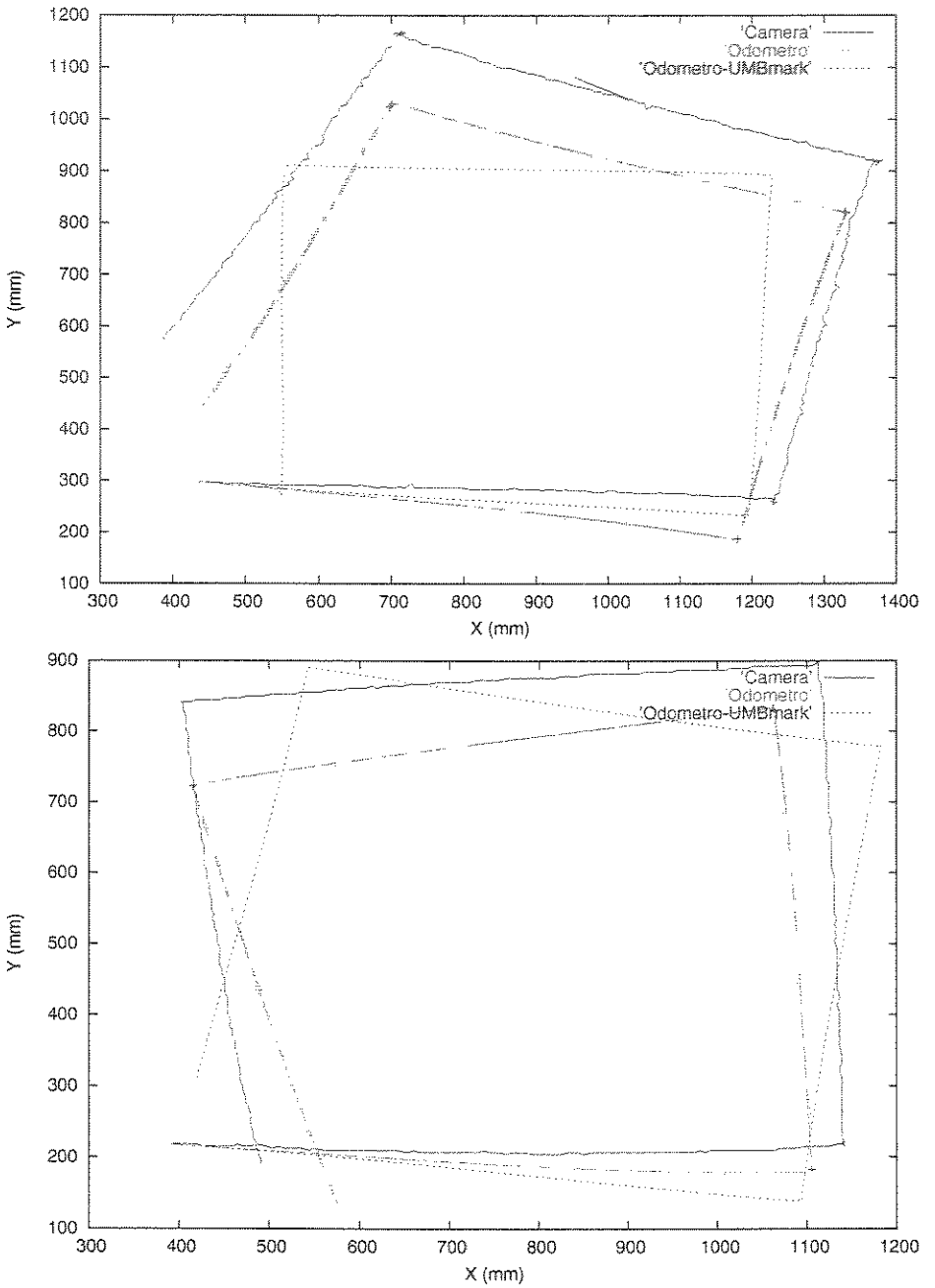


Figura 6.9: Trajetórias Após Correção UMBmark.

6.3.2 Correções Empíricas no Sistema de Localização

Analisando as trajetórias da Figura 6.9, verifica-se que a técnica de correção UBMmark não produziu resultados satisfatórios, isto é, o robô, utilizando as variáveis cinemáticas após a correção UBMmark, teria em mente uma trajetória (`'Odometro-UMBmark'`) mais longe da "real" (`'Camera'`) do que se não a estivesse utilizando (`'Odometro'`). Pode se atribuir esse fato, provavelmente, a escorregamento das rodas acima do que seria aceitável.

Uma forma empírica para minimização de tais erros foi determinada. Analisando as trajetórias mostradas nas Figuras 6.7 e 6.8, observa-se que as trajetórias dadas pelo sistema de odometria (`'Odometro'`) são "aproximadamente" uma redução (mesma forma, porém, em escalas menores) das trajetórias do sistema de localização visual (`'Camera'`). Isso indica que os valores das variáveis cinemáticas r_e e r_d (usados pelo sistema de odometria) podem ser aumentados, porém, isso acarretaria mudança na variação angular, logo, precisa-se de uma relação que aumente o valor destas variáveis, mas não altere o $\Delta\theta$. Analisando as equações do modelo cinemático pode-se obter tal relação.

Assumindo que os raios são iguais:

$$r = r_e = r_d \tag{6.2}$$

Substituindo a Equação 6.2 nas Equações 4.1 e 4.2:

$$DL_e = 2\pi r \frac{N_e}{N_r} \tag{6.3}$$

$$DL_d = 2\pi r \frac{N_d}{N_r} \tag{6.4}$$

Agora substituindo as Equações 6.3 e 6.4 na Equação 4.20:

$$\Delta\theta = \frac{\left(2\pi r \frac{N_d}{N_r}\right) - \left(2\pi r \frac{N_e}{N_r}\right)}{b} \tag{6.5}$$

$$\Delta\theta = \frac{\frac{2\pi r}{N_r}(N_d - N_e)}{b} \tag{6.6}$$

$$\Delta\theta = \frac{2\pi r}{N_r b}(N_d - N_e) \tag{6.7}$$

$$\Delta\theta = \frac{r}{b}\frac{2\pi}{N_r}(N_d - N_e) \tag{6.8}$$

Como o objetivo é manter $\Delta\theta$ sem alteração e aumentar o r. Analisando a Equação 6.8 observa-se que a relação procurada é:

$$\frac{r_{atual}}{b_{atual}} = \frac{r_{nominal}}{b_{nominal}} \tag{6.9}$$

Assim, foram realizadas novas trajetórias com valores de r aumentados gradativamente em cada uma delas, com seus respectivos valores de b (estabelecido pela Equação 6.9), até a obtenção de uma trajetória 'boa', a qual pode ser vista na Figura 6.10.

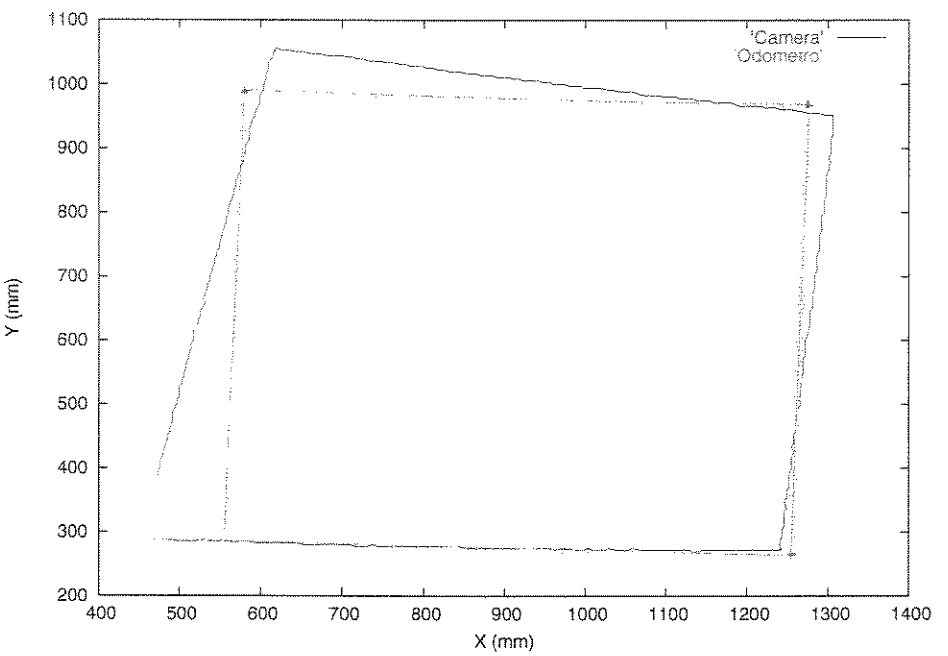


Figura 6.10: Trajetória Após Correção Empírica.

Confrontando a trajetória, cujos erros foram aferidos pelo método Empírico (ver Figura 6.10) com as 10 trajetórias iniciais (ver Figuras 6.7 e 6.8), observam-se expressivas melhoras. Esta solução foi para uma classe específica do problema, ou seja, funcionou bem para um caso específico, o caso deste protótipo. Mas, como houve uma melhora expressiva, os valores das variáveis cinemáticas utilizados serão os aferidos pelo método Empírico, sendo:

$$\begin{aligned} b &= 145.58824mm \\ r_e &= 44.00mm \\ r_d &= 44.00mm \end{aligned}$$

6.3.3 Experimentos de Correção da Odometria

Realizaram-se experimentos para determinação de um intervalo de tempo em função de um erro suportável. Estes experimentos consistem, basicamente, no cálculo do erro acumulado em trajetórias retilíneas em função do tempo. Os resultados desses testes podem ser vistos no gráfico mostrado na Figura 6.11, o qual ilustra o crescimento médio do erro ao longo de cinco trajetórias realizadas pelo robô. A partir desse gráfico, podem-se determinar intervalos de tempo plausíveis para evitar que o robô perca sua referência.

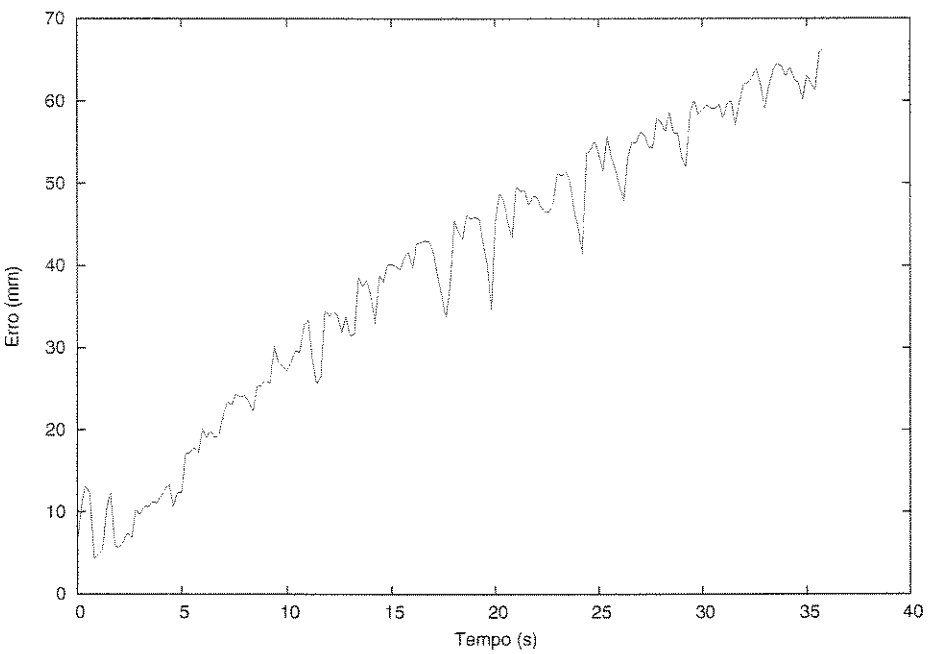


Figura 6.11: Erro Acumulado.

Ressalta-se que o erro médio obtido nesse experimento depende do tipo do piso do ambiente de trabalho do robô. Um teste desse pode ser realizado automaticamente pelo robô, assim, ele poderá adaptar-se a pisos diferentes.

Analisando o gráfico da Figura 6.11, observa-se que para se obter um erro médio máximo menor que $40mm$ (erro considerado plausível para nossa aplicação) é preciso que a localização dada (pelo sistema de localização visual) ocorra em intervalos menores que $15s$. Assim, escolhem-se os intervalos de $1s$, $4s$, $7s$, $10s$ e $13s$, para realização de testes. E para cada intervalo foi realizada uma trajetória. Essas trajetórias são apresentadas nas Figuras 6.12, 6.13, 6.14, 6.15 e 6.16.

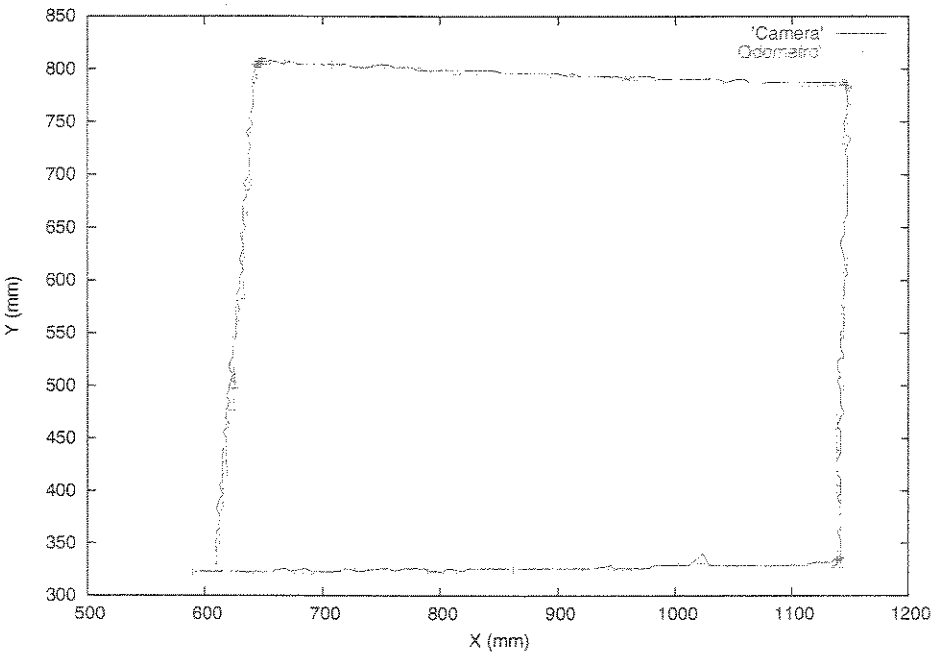


Figura 6.12: Trajetória com Correção (Intervalo 1s).

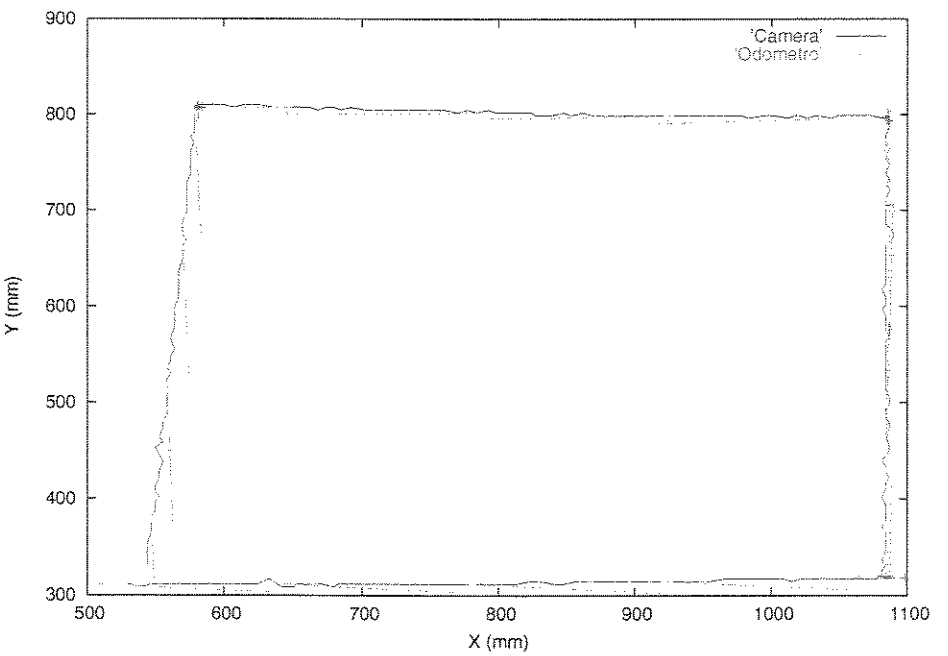


Figura 6.13: Trajetória com Correção (Intervalo 4s).

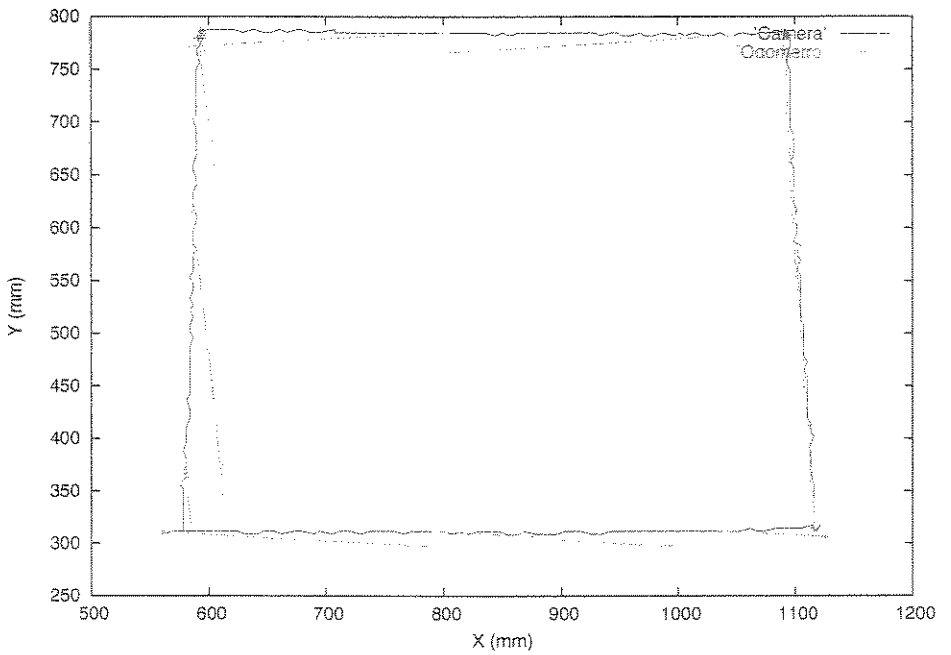


Figura 6.14: Trajetória com Correção (Intervalo 7s).

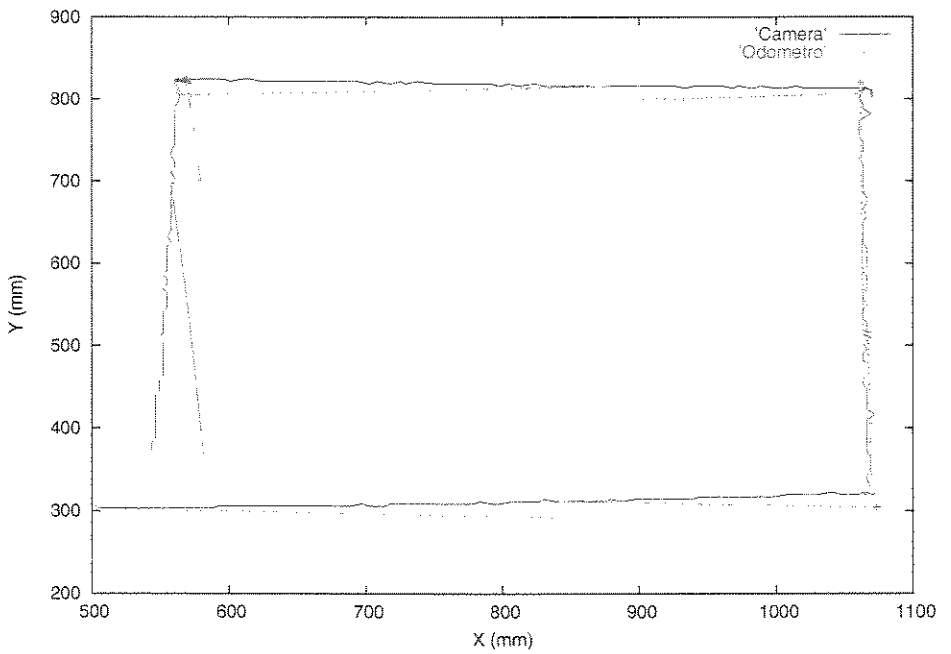


Figura 6.15: Trajetória com Correção (Intervalo 10s).

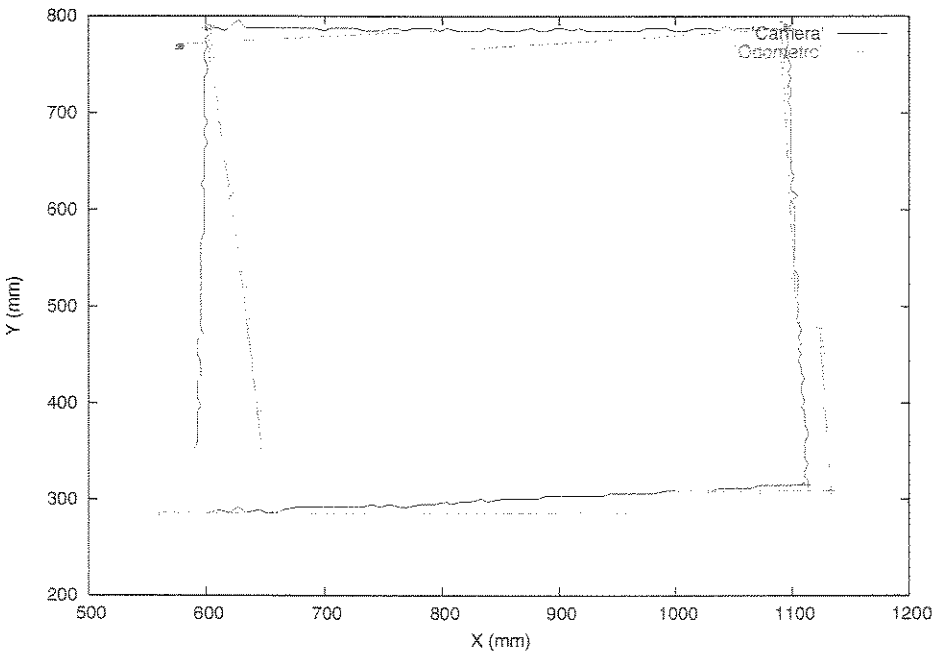


Figura 6.16: Trajetória com Correção (Intervalo 13s).

6.3.4 Considerações sobre o Sistema de Localização

Foi a partir da inclusão da correção de erro, usando o sistema de localização visual, que houve redução significativa do erro de odometria. Com a utilização da técnica de correção intermitente, a câmera foi liberada para execução de outras tarefas, permitindo que o robô possa inclusive sair do campo visual da câmera. Observa-se que em um caso especifico o robô podê ficar intervalos de 13 segundos, sem perda significativa de localização (ver Figura 6.16).

6.4 Testes com o Sistema de Localização Visual (Horizontal)

O Sistema de Visão Horizontal, descrito no Capítulo 5, com a câmera colocada no robô Mestre, foi também testado. Inicialmente, foram conduzidos experimentos, a fim de se obter a transformação entre $(D, X) \rightarrow (z, x)$. A Figura 6.17 mostra algumas imagens adquiridas para determinação da função f.

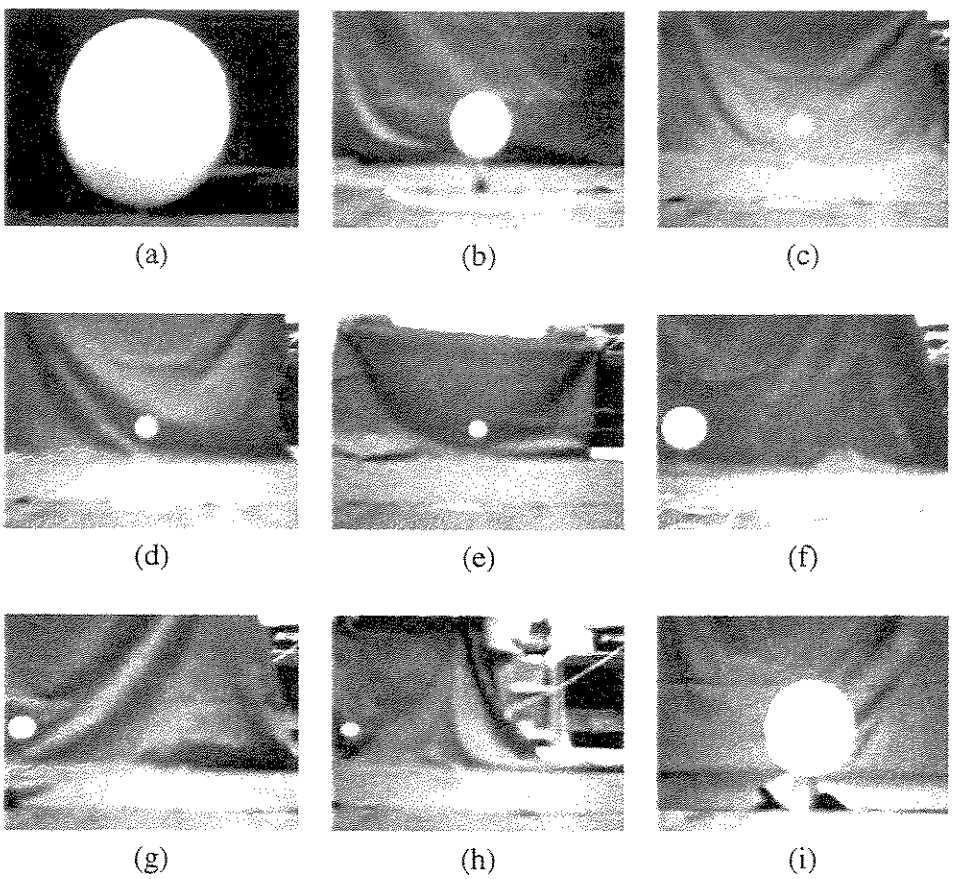


Figura 6.17: Imagens Adquiridas para Determinar uma Aproximação para f.

A função f é responsável pela transformação. Basicamente, têm-se 12 variáveis e cada imagem fornece 2 equações mais uma restrição (uma matriz transformada ortonormal). Então, 4 imagens seriam suficientes para encontrar todos os coeficientes. Como foram tomadas 9 imagens, usou-se a abordagem dos mínimos quadrados, para se determinar a função f. A Tabela 6.8 resume os dados mensurados.

Para validar a função f obtida, foram escolhidos 9 pontos previamente conhecidos e em cada ponto foi colocado e retirado o pedestal com a esfera. Utilizando o Sistema de Visão obteve-se

Tabela 6.8: Dados Mensurados.

Image	D	X	z(cm)
Figura 6.17(a)	210	0	11
Figura 6.17(b)	78	0	33
Figura 6.17(c)	49	0	69
Figura 6.17(d)	30	0	114
Figura 6.17(e)	24	0	147
Figura 6.17(f)	57	160	47
Figura 6.17(g)	33	290	86
Figura 6.17(h)	22	500	143
Figura 6.17(i)	110	0	15

o diâmetro (D) e ponto g (centro da circunferência) de coordenadas X e Y. Com esses valores, a função f retorna a coordenada x da esfera, denominada de x_c e a coordenada z da esfera, denominada de z_c. Como esses pontos eram previamente conhecidos pode-se comparar tais coordenadas, calculadas pela função, com as coordenadas conhecidas dos pontos denominadas de x_r e z_r.

A Figura 6.18 mostra as imagens capturadas e a Tabela 6.9 resume os dados correspondentes, calculados pelo Sistema de Localicação (X, Y, e D), pela função f (x_c e z_c) e os valores reais (x_r e z_r)

Tabela 6.9: Comparando os Dados.

Image	X	Y	D	z_c	x_c	z_r	x_r
Figura 6.18(a)	233	154	28	88	-13	87	-15
Figura 6.18(b)	173	152	28	88	-1	87	0
Figura 6.18(c)	95	152	24	85	14	87	15
Figura 6.18(d)	264	152	36	53	-14	57	-15
Figura 6.18(e)	171	151	38	56	0	57	0
Figura 6.18(f)	63	151	34	59	16	57	15
Figura 6.18(g)	294	151	46	45	-17	42	-15
Figura 6.18(h)	167	149	46	45	1	42	0
Figura 6.18(i)	32	149	42	41	14	42	15

Analisando a Tabela 6.9 observou-se que o erro máximo encontra-se em torno de 10%. Notou-se também, que esse erro aumenta conforme a distância entre o robô Mestre e o robô Escravo.

Num dos testes práticos realizados, o robô Mestre foi colocado em frente a uma esfera posicionada sobre o pedestal. Caso o diâmetro da circunferência seja menor que determinado

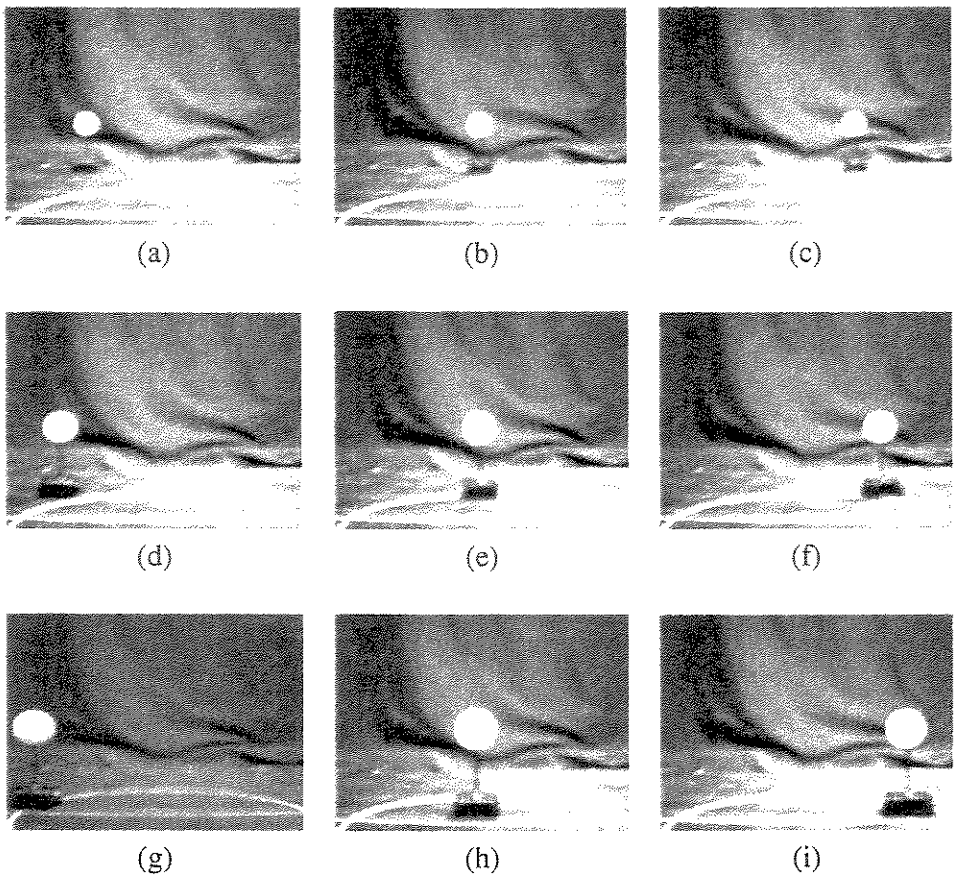


Figura 6.18: Imagens das Esferas Sobre os Pontos Conhecidos.

valor, o robô irá ao encontro da esfera, caso contrário, o robô irá para mais longe da esfera. Convém lembrar que até então, não houve nenhum controle efetivo no robô.

Também foi testada a performance do Sistema de Visão Horizontal, em termos de tempo gasto para todos os cálculos. Para o sistema determinar apenas posição (esfera branca), pode ser executado em tempo real, até 30 quadros por segundo. Essa taxa é bem superior à taxa do RemoteRCX.

Capítulo 7

Conclusões Finais e Perspectivas

Neste trabalho vários experimentos foram realizados, visando determinar os potenciais e limitações dos sistemas operacionais e compiladores disponíveis para o processador que controla o *hardware* da linha LEGO. Este trabalho foi essencialmente importante, uma vez que permitiu avaliar os melhores *softwares* e descobrir o potencial não explorado da tecnologia LEGO, incluindo seus compiladores, sistemas operacionais e do próprio *hardware*. Vários cursos de graduação estão adotando esta tecnologia, variando desde Engenharia Mecânica até Computação. Acredita-se que a utilização do compilador e sistema operacional BRICKOS seja uma tendência futura, na área de pesquisa, uma vez que testes atuais mostraram que eles permitem melhor explorar o *hardware*, com técnicas como: semáforos, multi-tarefas, paginação de memória, dentre outras, que tornam mais simples a implementação de tarefas/objetivos (programação).

Ainda, vários métodos e técnicas foram descritos e propostos neste trabalho, visando prover ferramentas para serem utilizadas em robôs. Os robôs usados podem agir de forma autônoma, executando procedimentos que foram apenas desenvolvidos (programados) em computadores e transferidos para as suas unidades de controle, a fim de sua execução autônoma, ou podem ser controlados a partir de um computador central, que recebe informações de uma câmera instalada sobre um robô Mestre e transmite comandos, usando um protocolo de comandos também proposto e desenvolvido neste trabalho. Esse protocolo de comandos, o RemoteRCX, permite a operação conjunta de vários robôs. Isso pode ser usado, por exemplo, para implementar um time de futebol de robôs.

Foram ainda realizados vários testes como a determinação dos erros de odometria (relativo) os quais, permitiram perceber que um método de correção visual (absoluto) é necessário para ratificar os mesmos.

Como mostrado no Capítulo 6, o método visual de localização proposto no Capítulo 5, mostrou-se eficiente para detecção dos marcos, provendo a posição dos robôs. A taxa de processamento necessária para localizar as esferas está muito além da taxa de operação do *link* de comunicação (protocolo RemoteRCX), permitindo tempo suficiente para que o robô possa

ainda realizar a segunda etapa do processo (raciocínio), caso seja necessário. Esses requisitos são essenciais para aplicações, onde se faz necessário o processamento em tempo real, como ocorre em problemas de navegação de um robô móvel. Os experimentos realizados, visando determinar a precisão da metodologia, dizem que a proposta é exeqüível do ponto de vista de precisão e robustez, isto é, o método permite que o robô consiga operar com uma margem de erro pequena e a busca pelos marcos foi conseguida com sucesso em todos os experimentos realizados.

7.1 Contribuições

Acredita-se que as técnicas desenvolvidas sejam de extrema valia a quem trabalha com a plataforma LEGO. Todas ferramentas ficaram disponibilizadas, sem restrições e serão cadastradas como de *open-source* [37, 45]. Acredita-se que o protocolo de comandos desenvolvido (RemoteRCX), seja uma das principais contribuições deste trabalho. Convém ressaltar que vários trabalhos estão sendo desenvolvidos atualmente (na UFRN e UNICAMP) sobre o RemoteRCX. Num desses trabalhos, um protótipo está sendo usado num museu para guiar pessoas ou seus *avatares* quando estiverem conectadas pela *Internet*.

Uma aplicação interessante, parcialmente realizada, é dotar vários robôs (sem câmeras) com marcas. Nesse caso, as marcas desses robôs sem câmera (robôs Escravos) servem para determinar a localização de um robô dotado de câmera (robô Mestre), desde que as posições de pelo menos dois desses robôs Escravos (visíveis) sejam conhecidas a um dado momento (denominou-se isso de interseção a ré). As mesmas marcas podem servir como identificadores ao robô Mestre, para determinar a localização dos robôs Escravos, desde que a posição e orientação do Mestre seja conhecida (a esta técnica denominou-se de interseção avante). Assim, podem-se alternar entre essas duas situações, para se ter a localização de todo o conjunto num determinado instante, com uma certa precisão. O tipo e magnitude do erro inerente à técnica deverá ser estudado futuramente

As Figuras 7.1 e 7.2 ilustram a aplicação desta técnica, com um robô Mestre, realizando o acompanhamento de dois robôs Escravos dentro de seu campo de visão. Nessa execução ideal, a ser ainda implementada (em trabalhos posteriores), o robô Mestre seria colocado em uma posição arbitrária e os robôs Escravos poderiam se mover somente dentro do cone de visibilidade do robô Mestre. Esses dados seriam fornecidos como retro-alimentação para o procedimento de controle, rodando em um computador *host*. Com isso, seria possível enviar os robôs Escravos a posições restritas ao campo de visão do Mestre, mantendo o acompanhamento da posição deles, executando a 10 quadros por segundo. Os procedimentos e ferramentas desenvolvidos serão de extrema valia na execução deste experimento futuro.

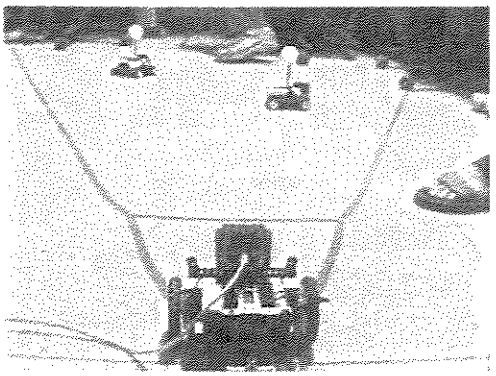


Figura 7.1: Robô Mestre e os Robôs Escravos.

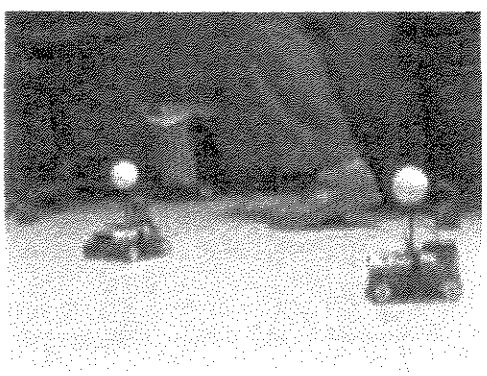


Figura 7.2: Visão do Robô Mestre.

7.2 Limitações

Os *kits* LEGO são extremamente limitados às operações ou aplicações, onde robôs mais robustos são necessários. Porém, na área de pesquisa, eles são ferramentas extremamente úteis, pela fácil prototipagem que permitem. Acredita-se que as ferramentas desenvolvidas e pesquisas condensadas neste contexto poderão ser largamente utilizadas por pesquisadores na área de robótica. Neste sentido, muitos trabalhos atuais são ainda utilizados sobre plataformas de simulação. Acredita-se que com estas ferramentas e alguns *kits*, a simulação pode deixar vez à execução em robôs "pseudo-reais". Ainda, as ferramentas desenvolvidas podem ser adaptadas ligeiramente para serem utilizadas por robôs mais robustos.

7.3 Perspectivas

Para testes futuros, planeja-se o desenvolvimento de um controlador posicional-derivativo. Uma nova versão de *hardware* e *software* para esferas com duas cores está sendo planejada na UFRN. Nesta nova versão, generalizaram-se as suposições anteriores, para encontrarem esferas de duas cores. Um algoritmo similar pode ser executado para encontrarem esferas, que possuam as cores branca e azuis/vermelhas. Nesse caso, podem-se aplicar filtros passa-banda na imagem, realçando os *pixels* vermelhos e/ou azuis e brancos. O *treshold* acima pode ser então aplicado, nesse caso, procurando por *pixels*, em duas imagens binárias que satisfaça um `'ou'` entre as cores, vermelho ou branco e depois azul ou branco. Esse realce, a fim de fornecer orientação, certamente torna o algoritmo um pouco mais lento, mas, ainda, permitindo processamento em tempo real. O próximo passo seria encontrar um ponto interno à projeção da esfera e o algoritmo age então de mesma forma que no método anterior. A seguir, o diâmetro pode ser encontrado de forma similar, provendo os parâmetros necessários para que se tenha posição. Para prover orientação, têm-se, apenas que achar as bordas ou separações entre azul/vermelho e branco. Isto pode ser feito após a determinação do centro e diâmetro da projeção da esfera na imagem, procurando na linha horizontal central da esfera localizada por mudanças abruptas na imagem original.

Serão feitos experimentos em que dois robôs Escravos tenham sua posição determinada a partir do robô Mestre e, então, este se movimente e ratifique sua posição, a partir dos dois robôs Escravos e, assim, sucessivamente. Denominando esta técnica de localização dinâmica, o que exige a implementação de um método robusto para calibração de câmera, que será avaliado, bem como os métodos de interseção para determinar a posição tanto dos Escravos a partir do Mestre quanto do Mestre a partir dos Escravos. Dessa forma, poder-se-a testar a forma dinâmica da proposta, verificando o crescimento do erro de localização.

Várias das ferramentas propostas nesta dissertação poderão ser testadas em um robô que está sendo construído atualmente no Laboratório de Sistemas Inteligentes (LabSis) da UFRN. Pretende-se, ainda, implementar e testar, nesta nova plataforma, procedimentos de baixo nível, como: levantar objetos, seguir linhas e outros mais. Dependendo do problema proposto ao Mestre, cada robô Escravo, com comportamentos especializados, podem ser enviados a uma determinada posição. É iminente definir uma metodologia para calcular e modelar o erro da forma dinâmica do sistema (robôs móveis com marcas).

Referências Bibliográfi cas

[1] Activ - ActivMedia Robotics Corporate, November 2002. Available at `http://www.activrobots.com`.

[2] Pablo Javier Alsina. Disciplina 'Sistemas Robóticos Autônomos', Engenharia de Computação - UFRN, 2002. Disponível em `http://www.dca.ufrn.br/~pablo/sisrob.html`.

[3] Dave Baum. *NQC Programmer's Guide.* Version 2.2 r1.

[4] Dave Baum. *NQC User Manual.* Version 2.3 r1.

[5] Dave Baum. NQC - 'Not Quite C' is a simple language with a 'C'-like syntax that can be used to program Lego's RCX programmable brick (from the Mindstorms set), November 2002. Available at `http://www.baumfamily.org/nqc`.

[6] Reinaldo A. C. Bianchi and Anna Helena Reali-Costa. O Sistema de Visão Computacional do time FUTEPOLI de Futebol de Robôs. In *Congresso Brasileiro de Automática*, 13, pages 2156–2162, Florianópolis, 2000. UFSC / Sociedade Brasileira de Automática. Anais.

[7] Johann Borenstein, Commander H. R. Everett, and Ligiang Feng. *"Where am I?" Sensors and Methods for Mobile Robot Positioning.* The University of Michigan, April 1996. Edited and compiled by J. Borenstein.

[8] BrickOS - An open source embedded operating system and provides a 'C' and 'C++' programming environment for the Lego Mindstorms Robotics Kits, November 2002. Available at `http://brickos.sourceforge.net`.

[9] Mario Fernando Montenegro Campos. Disciplina 'Introdução à Robótica', Departamento de Ciência da Computação - UFRN, 2003. Disponível em `http://www.lrvpa.dcc.ufmg.br/cursos/roboticamovel`.

[10] João Vilhete Viegas D'Abreu. Disciplina 'Introdução à Engenharia de Controle e Automação', Engenharia de Controle e Automação (Mecatrônica) - UNICAMP, 2001.

[11] João Vilhete Viegas D'Abreu. *Integração de Dispositivos Mecatrônicos para Ensino-Aprendizagem de Conceitos na Área de Automação*. PhD thesis, Universidade Estadual de Campinas - UNICAMP, 2002.

[12] João Vilhete Viegas D'Abreu, Luiz Marcos Garcia Gonçalves, Maria F. Garcia, and Luciane T. S. Garcia. Uma Abordagem Prático-Pedagógica para o Ensino de Robótica em Ciência e Engenharia de Computação. In Sérgio C. Pinto, editor, *Proc. of I XIII Simpósio Brasileiro de Informática na Educação*, pages 428–439, São Leopoldo, RS, Brasil, Novembro 2002. Editora UNISINOS.

[13] João Hilário de Ávila Valgas Filho. Uma Metodologia de Correção Dinâmica de Erros de Odometria em Robôs Móveis. Master's thesis, Universidade Federal de Minas Gerais - UFMG, 2002.

[14] Anuj Dev, Bem Krose, and Franciscus Groen. Navigation of a Mobile Robot on the temporal Development of the Optic Flow. In *IEEE/RSJ/GI INT. CONF. On Intelligent Robots and Systems IROS'97*, pages 558–563, 1997.

[15] Anfranserai Morais Dias. Posicionamento de um Manipulador Redundante Utilizando Realimentação Visual. Master's thesis, Universidade Federal do Rio Grande do Norte - UFRN, 2002.

[16] Gedson Faria and Roseli Aparecida Francelin Romero. Explorando o Potencial de Algoritmos de Aprendizado com Reforço em Robôs Móveis. In *IV Congresso Brasileiro de Redes Neurais*, pages 237–242, São José dos Campos - SP, 1999. ITA.

[17] Jean Marie Farines and José E. R. Cury. Disciplina 'Introdução à Engenharia de Controle e Automação - DAS-5411', Departamento de Automação e Sistemas CTC-UFSC, 2003. Disponível em `http://www.lcmi.ufsc.br/das5411`.

[18] FIRA - Federation of International Robot-soccer, December 2003. Available at `http://www.fira.net`.

[19] Luiz Marcos Garcia Gonçalves. Disciplina 'Robótica: Sistemas Sensorial e Motor', Ciência da Computação - UNICAMP, 2001. Disponível em `http://www.ic.unicamp.br/~lmarcos/courses/mo810`.

[20] Luiz Marcos Garcia Gonçalves. Disciplina 'Percepção Robótica', Engenharia de Computação - UFRN, 2002. Disponível em `http://www.dca.ufrn.br/~lmarcos/courses/robotica`.

[21] Luiz Marcos Garcia Gonçalves and Fernando W. Silva. Control Mechanisms and Local Perception to Support Autonomous Behavior in Virtual Animated Agents. *Computer Graphics Journal*, March 2002.

[22] GPL - GNU GENERAL PUBLIC LICENSE, March 2003. Available at `http://www.gnu.org/copyleft/gpl.html`.

[23] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. In *IEEE Transactions on Robotics and Automation*, volume 12, pages 651–670, October 1996.

[24] K-Team S.A., November 2002. Available at `http://www.k-team.com`.

[25] LEGO Mindstorms, November 2002. Available at `http://mindstorms.lego.com`.

[26] LEGO-Lab - University of Aarhus, 2003. Available at `http://www.legolab.daimi.au.dk`.

[27] LegOS - A open source embedded operating system and provides a 'C' and 'C++' programming environment for the Lego Mindstorms Robotics Kits, November 2002. Available at `http://legos.sourceforge.net`.

[28] LejOS - Java for the RCX, November 2002. Available at `http://lejos.sourceforge.net`.

[29] LNP - LegOS Network Protocol, November 2002. Available at `http://legos.sourceforge.net/HOWTO/x405.html`.

[30] Franz Alberto Sandi Lora, Elder M. Hemerly, and Walter Fetter Lages. Estimação em Tempo Real de Posição e Orientação de Robôs Móveis Utilizando Sensores com Diferentes Taxas de Amostragem. In *Anais do 3º Simpósio Brasileiro de Automação Inteligente*, pages 453–458, 1997.

[31] Franz Alberto Sandi Lora, Elder M. Hemerly, and Walter Fetter Lages. Sistema para Navegação e Guiagem de Robôs Móveis Autônomos. In *SBA Controle & Automação*, volume 9, pages 107–118, Set., Out., Nov. e Dez. 1998.

[32] Lynxmotion, Inc, November 2002. Available at `http://www.lynxmotion.com`.

[33] Maja J. Matarić. Reinforcement Learning in the Multi-Robot Domain. *Autonomous Robots*, 4(1):73–83, March 1997.

[34] 6270 Course of Massachusetts Institute of Technology - MIT, 2003. Available at `http://web.mit.edu/6.270`.

[35] MPL - Mozilla and Netscape Public Licenses, March 2003. Available at `http://www.mozilla.org/MPL`.

[36] NEC Corporation, December 2003. Available at `http://www.nec.com`.

[37] Open-Source - Initiative (OSI) is a non-profit corporation dedicated to managing and promoting the Open Source Definition for the good of the community, November 2002. Available at `http://www.opensource.org`.

[38] Mark Overmars. *Programming Lego Robots using NQC*. Department of Computer Science Utrecht University, TB Utrecht the Netherlands, October 1999. Version 3.03.

[39] RcxCC - A front-end for NQC, November 2002. Available at `http://www.cs.uu.nl/people/markov/lego/rcxcc`.

[40] Carlos Henrique Costa Ribeiro, Anna Helena Reali-Costa, and Roseli Aparecida Francelin Romero. Robôs Móveis Inteligentes: Princípios e Técnicas. In *Jornada de Atualização em Inteligencia Artificial*. Congresso da Sociedade Brasileira de Computação, 2001.

[41] RoboCup - Robot World Cup Initiative, December 2003. Available at `http://www.robocup.org`.

[42] ROBOLAB - Software for programming and controlling the RCX is an icon-based, November 2002. Available at `http://www.lego.com/eng/education/mindstorms`.

[43] SIROS - Sistemas Robóticos com SuperLogo, Janeiro 2003. Disponível em `http://www.nied.unicamp.br/~siros`.

[44] AIBO, March 2003. Available at `http://www.us.aibo.com`.

[45] SourceForge - The world's largest Open Source software development web site, providing free hosting to tens of thousands of projects, November 2002. Available at `http://sourceforge.net`.

[46] Douglas Machado Tavares, Aquiles Medeiros Burlamaqui, Viviane André Antunes, George Christian Basilio Thó, Anfranserai Morais Dias, Tatiana Aires Tavares, Guido Lemos Filho, and Luiz Marcos Garcia. A Real Time Platform for Playing with Robots and Avatars in Mixed Reality Spaces. In *IV Workshop de Tempo Real*, Natal-RN, Brasil, March 2003. 21th SBRC2003.

[47] Douglas Machado Tavares, Aquiles Medeiros Burlamaqui, Anfranserai Morais Dias, Meika Iwata Monteiro, Viviane André Antunes, George Christian Basilio Thó, Tatiana Aires Tavares, Carlos Magno de Lima, Luiz Marcos Garcia Gonçalves, Pablo Javier Alsina, Adelardo A. Dantas de Medeiros, and Guido Lemos Filho. Hyperpresence - An Application Environment for Control of Multi-User Agents in Mixed Reality Space. In *Conference held at Orland*, Florida, USA, March 2003. 36th Annual Simulation Symposium.

[48] VERLab - Laboratório de Visão Computacional e Robótica, Agosto 2003. Disponível em `http://www.verlab.dcc.ufmg.br`.