UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UM ESTUDO DE CASOS PARA AUXÍLIO À DEFINIÇÃO DE UM MODELO DE PORTABILIDADE NA IMPLEMENTAÇÃO DE ALGORITMOS SIMULADOS EM ROBÔS REAIS

WANDERSON GOMES DE SOUZA

JOÃO PESSOA – PB

Fevereiro – 2014

WANDERSON GOMES DE SOUZA

UM ESTUDO DE CASOS PARA AUXÍLIO À DEFINIÇÃO DE UM MODELO DE PORTABILIDADE NA IMPLEMENTAÇÃO DE ALGORITMOS SIMULADOS EM ROBÔS REAIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos para a obtenção do título de Mestre em Informática.

ORIENTADOR: Prof. Dr. Clauirton de Albuquerque Siebra

JOÃO PESSOA – PB
Fevereiro – 2014

S729u Souza, Wanderson Gomes de.

Um estudo de casos para auxílio à definição de um modelo de portabilidade na implementação de algoritmos simulados em robôs reais / Wanderson Gomes de Souza.-- João Pessoa, 2014.

90f. : il.

Orientador: Clauirton de Albuquerque Siebra

Dissertação (Mestrado) - UFPB/CI

1. Informática. 2. Robótica. 3. Robôs autônomos móveis. 4. Algoritmo de anticolisão. 5. Simuladores.

UFPB/BC *CDU: 004(043)*

WANDERSON GOMES DE SOUZA

UM ESTUDO DE CASOS PARA AUXÍLIO À DEFINIÇÃO DE UM MODELO DE PORTABILIDADE NA IMPLEMENTAÇÃO DE ALGORITMOS SIMULADOS EM ROBÔS REAIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos para a obtenção do título de Mestre em Informática.
ORIENTADOR: Prof. Dr. Clauirton de Albuquerque Siebra

Aprovado no dia 03 de fevereiro de 2014.

BANCA EXAMINADORA

_____
Prof. Dr. Clauirton de Albuquerque Siebra
(Orientador)

_____
Profa. Dra. Natasha Correia Queiroz Lino
(Examinadora Interna)

_____
Prof. Dr. Daniel Scherer
(Examinador Externo)

AGRADECIMENTOS

À Deus, por estar vivo, com saúde, por ter me guiado e dado forças para seguir firme nesta caminhada, superando todas as dificuldades que surgiram no decorrer do Mestrado.

Dedico esse agradecimento especial à pessoa que mais admiro, a minha esposa Mainara, que esteve presente em todos os momentos dessa caminhada, resgatando a minha autoestima, nos períodos mais difíceis e decisivos de minha vida, tornando-se uma grande testemunha das dificuldades superadas.

Aos meus pais, pela educação, paciência, carinho e a base para que meus esforços resultassem em vitórias.

A minha sogra Verônica, por me aconselhar sempre nos momentos certos.

Ao meu professor Dr. Clauirton Siebra, por ter aceitado orientar o meu trabalho, por acreditar em mim e no meu esforço, por ser compreensivo e me ensinado tanto no decorrer de todo o período do Mestrado.

Aos membros da banca de qualificação e defesa: Danielle Rousy, Natasha Lino e Daniel Scherer, por participarem deste momento e contribuírem com enriquecedoras observações.

À Universidade Federal da Paraíba e ao Centro de Informática, pela oportunidade de cursar esta Pós-Graduação.

À grande ajuda dos companheiros que tiveram participações importantes nesse trabalho, como, Gustavo Henrique, Gutierrez, Isaac Lima, Diego Silva, Cecília, Letícia e Erberto.

Aos meus amigos, por estarem sempre presentes nos momentos bons, e também nos de maior dificuldade desde o primeiro dia de aula na Pós.

RESUMO

O desenvolvimento de algoritmos voltados para sistemas robóticos, em ambientes de simulação, auxilia na identificação e antecipação de muitos problemas antes mesmo de serem testados em ambientes reais. Porém, nada garante que um algoritmo avaliado em uma plataforma de simulação funcione quando migrado para robôs reais. A interferência de diversos fatores, como propriedades dos motores e sensores, bem como a falta de uma metodologia necessária para interação simulação/realidade, se caracteriza como um dos principais problemas desta transição, de modo que, existe um grande *gap* conceitual entre o desenvolvimento para ambientes reais e simulados. Este trabalho tem o objetivo de contribuir com a formalização de um modelo conceitual que auxilie no processo de transição, no qual é possível destacar as peculiaridades que carecem de uma maior atenção por parte dos desenvolvedores. Para isso, como estudo de casos, foi desenvolvido um algoritmo de anticolisão, que engloba diversas características de um sistema robótico, sendo realizada sua implementação tanto no simulador *Unity3D*, quanto em robôs reais desenvolvidos na plataforma Arduino. Os resultados foram obtidos e comparados através de uma análise quantitativa de gráficos originados pelo MATLAB. Todas as alterações necessárias foram classificadas e avaliadas, de forma que possamos ter uma ideia do modelo de transição inicial, o qual formalize as principais particularidades do projeto.

Palavras-chaves: Robótica, robôs autônomos móveis, algoritmo de anticolisão, simuladores

ABSTRACT

The development of algorithms oriented to robotic systems, in simulation environments, supports the identification and prediction of many problems before their tests in real environments. However, there is still not a process to ensure that an algorithm evaluated in a simulation platform will similarly work when migrated to real robots. The interference of several factors, such as engine and sensors properties, as well as the lack of a methodology that supports the interaction simulation/reality is considered as one of the main problems in this transition, so that there is a large conceptual development gap between real and simulated environments. This work aims to contribute with the formalization of a conceptual model to assist in the transition process, in which it is possible to highlight the peculiarities that require further attention from developers.. For that end, as a case study, it was developed an anti-collision algorithm, which involves several characteristics of a robotic system. Its implementation was carried out in both Unity3D simulator and real robots developed in the Arduino platform. The results were obtained and compared by a quantitative analysis of graphs using MATLAB. All necessary changes were classified and evaluated, raising up an initial idea of the transition model, which formalizes the main particulars of the project.

Keywords: Robotics, autonomous mobile robots, anti-collision algorithm, simulators

ÍNDICE DE FIGURA

Figura 1 - Etapas do funcionamento do Arduino....16
Figura 2 - Plataforma Arduino....16
Figura 3 - Plataforma *Basic Stamp* 19
Figura 4 - Plataforma *BeagleBoard* 20
Figura 5 - Plataforma Raspberry Pi 21
Figura 6 - Sistema por difusão....23
Figura 7 - Sistema por barreira 23
Figura 8 - Sistema por reflexão 23
Figura 9 - Sensor ultrassônico 24
Figura 10 - Emissão e captação do ultrassom. 24
Figura 11 - Padrões de reflexão....25
Figura 12 - Diagrama de blocos de um circuito de sensor ultrassônico 25
Figura 13 - Leitura de uma única faixa do sensor ultrassônico....26
Figura 14 - Faixa de detecção do sensor ultrassônico....26
Figura 15 - Visão superior do sensor omnidirecional....27
Figura 16 - Interface de desenvolvimento do *Unity3D*....28
Figura 17 - Simulador V-REP 30
Figura 18 - Simulador Microsoft Robotics Developer Studio 31
Figura 19 - Simulador Webots....32
Figura 20 – Sistema de coordenada inercial....33
Figura 21 – Liga padrão (Robôs reais)....36
Figura 22 – Liga simulada (Robôs simulados)....36
Figura 23 - O comportamento de mudança de papel descrito por XABSL 37
Figura 24 - Entradas e saídas do modelo NARMAX 38
Figura 25 - Posição dos oito sensores IR 40
Figura 26 - Esquema de movimentação do robô Khepera 40
Figura 27 - Configuração do *Khepera II* com quatro obstáculos no ambiente de simulação *webots*....41
Figura 28 - A trajetória do robô obtida por GPS, com quatro obstáculos.....41
Figura 29 - Configuração do Khepera II com cinco obstáculos no ambiente de simulação *webots*....41

Figura 30 - A trajetória do robô obtida por GPS, com cinco obstáculos. 41
Figura 31 - Configuração do Khepera II com seis obstáculos no ambiente de simulação *webots*.....42
Figura 32 - A trajetória do robô obtida por GPS, com seis obstáculos. 42
Figura 33 - Integração do MRSim com os simuladores *Player*/Gazebo.....43
Figura 34 - (a) Mostra um robô real *Pioneer* detectando objetos com formato cilíndricos, (b) Um robô virtual realizando leituras com sensor à laser no simulador Gazebo, (c) Mostra o resultado da leitura dos sensores utilizando o simulador *Player*, (d) Mostra o laser de realidade mista visto pelo Gazebo 44
Figura 35 - Chassi do robô 47
Figura 36 - Coordenadas inerciais do robô móvel.....48
Figura 37 – Arquitetura básica 49
Figura 38 - Visão inferior.....50
Figura 39 - Chassi superior 50
Figura 40 - Visão lateral.....50
Figura 41 - Robô real 50
Figura 42 - Esboço do circuito impresso.....52
Figura 43 - Placa de circuito após a impressão 52
Figura 44 - Soldagem dos componentes elétricos 53
Figura 45 - Montagem dos componentes (sensores, atuadores e arduino) 53
Figura 46 - Plataforma Arduino (Modelo Nano).....54
Figura 47 - Plataforma Arduino (Modelo MEGA).....54
Figura 48 - Modelagens feitas no Google *SketchUp* 57
Figura 49 - Modelagem 3D dos robôs utilizando o Google *SketchUp* 57
Figura 50 - Posição dos sensores ultrassônicos em relação aos obstáculos.....59
Figura 51 - Algoritmo de anticolisão em obstáculos com base retangular 63
Figura 52 – Algoritmo de anticolisão em obstáculos com base triangular 63
Figura 53 – Esquema de auxílio para leitura dos gráficos 65
Figura 54 - Medidor de velocidade (RPM) 69

INDICE DE QUADROS

Quadro 1 - Comparativo entre os trabalhos relacionados 45
Quadro 2 - Lista de componentes 51
Quadro 3 - Comparativo entre *BeagleBoard*, Raspberry Pi e Arduino 54
Quadro 4 - Comparativo entre Unity3D e V-REP.....58
Quadro 5 - Diferença de velocidade entre atuadores.....69
Quadro 6 - Número de RPM em diferentes níveis de carga.....72
Quadro 7 – Metodologia aplicada nos experiementos e resultados obtidos 82

INDICE DE GRÁFICOS

Gráfico 1 - Teste com o sensor omnidirecional real em obstáculo retangular 66

Gráfico 2 - Teste com o sensor omnidirecional simulado em obstáculo retangular 67

Gráfico 3 - Teste com sistema de navegação real e simulado em obstáculos triangulares ..67

Gráfico 4 - Teste com sistema de navegação real e simulado em obstáculos triangulares com correção de posicionamento.....68

Gráfico 5 – Teste com atuadores reais 70

Gráfico 6 - Teste com atuadores simulados 71

Gráfico 7 - Velocidade (RPM) x Nível de bateria (%) 72

LISTA DE ACRÔNIMOS E SIGLAS

CAD - *Computer-Aided Drafting*

CARMEN - *Carnegie Mellon Robot Navigation Toolkit*

CCR - *Concurrency and Coordination Runtime*

DSS - *Decentralized Software Services*

GSM – *Global Mobile Communications*

IA – Inteligência Artificial

IAD - Inteligência Artificial Distribuída

IDE - *Integrated Development Environment*

IR - *Infrared*

JIRA - *Japan Industrial Robot Association*

MRDS - *Microsoft Robotics Developer Studio*

NARMAX - *Non-Linear Auto-Regressive Moving Average with Exogeneous*

ODE - *Open Dynamics Engine*

PWM - *Pulse-width modulation*

QTI - *Charge Transfer Infrared*

RAM – Robôs Autônomos Móveis

RM – Robôs Móveis

MRSim – *Mixed Reality Robot Simulation*

RIA - *Robotic Industries Association*

RISC - *Reduced Instruction Set Computing*

ROS - *Robot Operating System*

RPM – Rotação por minuto

RUR - Robôs Universais de *Rossum*

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*

V-REP - *Virtual Robot Experimentation Platform*

XABSL - *Extensible Agent Behavior Specification Language*

XML - *eXtensible Markup Language*

Sumário

1. INTRODUÇÃO 7

1.1 Motivação 8

1.2 Objetivos 10

1.3 Organização do Trabalho 10

2. FUNDAMENTAÇÃO TEÓRICA 12

2.1 Definições 12

2.1.1 Robô autônomo móvel 13

2.2 Microcontroladores 13

2.3 Plataformas 14

2.3.1 Arduino 15

2.3.2 BASIC Stamp 19

2.3.3 BeagleBoard 20

2.3.4 Raspberry Pi 21

2.4 Sensores 22

2.4.1 Sensor Infravermelho 22

2.4.2 Sensor Ultrassônico 24

2.4.3 Sensor Omnidirecional 26

2.5 Simuladores 27

2.5.1 *Unity3D* 27

2.5.2 V-REP 29

2.5.3 CARMEN 31

2.5.4 *Microsoft Robotics Developer Studio* 31

2.5.5 *Webots* 32

2.6 Modelagem cinemática 33

2.7 Resumo 34

3. TRABALHOS RELACIONADOS 35

3.1 Visão Geral 35

3.2 Algoritmo de Evolução 35

3.3 Polinômio NARMAX 37

3.4 Algoritmo de otimização do *Sapo Pulando Embaralhado* 39

3.5 Ambiente de simulação e realidade mista 42
3.6 Comparativo 45
4. APLICAÇÃO DE ALGORITMO ANTICOLISÃO EM AMBIENTE REAL E SIMULADO 47
4.1 Desenvolvimento do Robô real 47
4.1.1 Modelagem cinemática de postura 48
4.1.2 Construção e montagem do robô real 49
4.1.3 Placa de circuito impresso 52
4.1.4 Plataforma Arduino 53
4.1.5 Características do robô 56
4.2 Simulador 56
4.3 Algoritmo 58
4.3.1 Variáveis de estado 59
4.3.2 Algoritmo de anticolisão 61
4.4 Resumo 64
5. EXPERIMENTOS 64
5.1 Teste 1 – Desvio de obstáculos com sensores omnidirecionais 66
5.2 Teste 2 – Desvio de obstáculos com sistema de navegação (bússola) 67
5.3 Teste 3 – Atuadores 69
5.4 Teste 4 – Bateria 71
5.5 Análise dos resultados 73
6. CONSIDERAÇÕES FINAIS 75
7. REFERÊNCIAS BIBLIOGRÁFICAS 77
APÊNDICE 82

1. INTRODUÇÃO

Nas últimas décadas observou-se um crescimento substancial de aplicações que envolvem a robótica, sendo a maioria delas utilizadas para substituir o homem em atividades de alto risco, ambientes inóspitos, trabalhos penosos e insalubres (LEITE, 2011). Estas aplicações visam facilitar e auxiliar a vida humana em diversos aspectos, tais como: busca e salvamento de vítimas, desarmamento de bombas, resgate de vítimas de avalanches, serviços domésticos, missões especiais, vigilância, prevenção de desastres, educação ou mesmo durante atividades do nosso cotidiano, de forma ubíqua ou pervasiva.

A robótica ganhou força na produção industrial, mediante a utilização de braços robóticos que se movem com grande velocidade ao realizar tarefas repetitivas com precisão. No entanto, estes robôs possuem uma grande desvantagem: a falta de mobilidade. Um braço robótico fixo apresenta um intervalo de movimento limitado ao espaço que é aparafusado. Por outro lado, o robô móvel é capaz de locomover-se por todo ambiente físico, com flexibilidade de trabalhar onde quer que seja necessário.

Outra tecnologia que vem sendo bastante empregada está relacionada aos simuladores de robôs. Esses simuladores permitem a construção de ambientes e robôs virtuais, os quais possibilitam experimentações sistemáticas dos algoritmos antes que eles sejam testados em robôs reais. De modo geral, os simuladores oferecem grandes vantagens ao planejar e testar extensivamente estratégias de coordenação, evitando custos com materiais, redução do tempo de trabalho e danos ao robô.

A utilização dos robôs móveis (RM) vem sendo frequentemente explorada e, em alguns casos, é direcionada a um ramo mais amplo e complexo com o uso conjunto de técnicas da Inteligência Artificial. A aplicação desta, em robôs, possibilitou a inserção da capacidade de raciocínio e da tomada de decisões de acordo com as circunstâncias envolvidas e, desde então, esses robôs são tratados como agentes inteligentes.

Ao considerar o robô como substituto do homem, deve-se provê-lo de autonomia para que ele possa trabalhar em conjunto com as demais máquinas. Esta premissa tem sido uma das principais motivações para a pesquisa do robô autônomo móvel.

Podemos definir os robôs autônomos móveis (RAMs) como um artefato de hardware e/ou software capaz de agir conforme um conhecimento pré-adquirido, o qual o torna apto a realizar determinadas tarefas baseada no modelo de mundo e em modelagens de problemas solucionáveis fundamentados na complexidade de tempo polinomial não determinístico.

Para (RUSSEL e NORVIG, 2009), os agentes autônomos são capazes de efetuar tarefas que na maioria das vezes são executadas por pessoas. Esses agentes conseguem perceber seu ambiente e o que acontece em volta por meio de receptores – câmeras, toque, sonar, receptores sonoros – e podem responder a esses estímulos por meio de um conjunto de atuadores – como braços mecânicos, pernas e rodas. Essas percepções podem ser interpretadas de modos divergentes quando se trata de estudos que envolvem plataformas reais e simuladas com robôs autônomos, imersos em ambientes complexos. Essas interações são consideravelmente difíceis de modelar, pois existe um grande número de variáveis independentes que são omitidas. Essas variáveis são particularidades que caracterizam as plataformas de simulação, que ao serem aplicadas em ambientes robotizados podem introduzir diferenças significativas entre o ambiente simulado e real. As diversas opções de terrenos, alta variabilidade em leitura de sensores, taxas de erro, juntamente com cenários operacionais que mudam em tempo real; afetam negativamente a confiabilidade das plataformas de simulação e os algoritmos desenvolvidos em ambientes reais.

Com o objetivo de entender essas diferenças, foi desenvolvido um algoritmo comum capaz de executar instruções em ambientes reais e simulados devido a sua versatilidade e simplicidade. O algoritmo é chamado de anticolisão. Para isso, fez-se necessário entender quais são as peculiaridades que envolvem o processo de validar o código em simuladores e robôs reais. Este entendimento permite uma melhor portabilidade entre os ambientes, sendo de grande auxílio para pesquisadores e desenvolvedores, uma vez que evita a reescrita do código quando atribuída a mesma tarefa para ambas as plataformas.

1.1 Motivação

Os simuladores de robôs são ferramentas importantes para validação de experimentos em diversas áreas de pesquisa em robótica e geralmente são utilizados para preceder o desenvolvimento de robôs reais. Uma de suas principais vantagens é a economia de tempo e esforço na construção de instalações experimentais baseadas no "mundo real". Além disso, as simulações oferecem um método consistente para repetição experimental, eliminando boa parte dos riscos de danos ao robô real. Por outro lado, as experiências com robôs reais ajudam na obtenção de resultados realistas em fases posteriores do desenvolvimento do robô simulado (BALAKIRSKY, CARPIN, *et al.*, 2009); (ZIEMKE, 2003); (SCHLESINGER e PARISI, 2001).

Entretanto, nada garante que o algoritmo validado em uma plataforma de simulação funcione plenamente quando migrado para robôs reais. Isso acontece porque os dados de entrada sofrem influências de fatores externos como: hardware do robô, algoritmo implementado ou ambiente de aplicação. Esse último merece mais atenção, pois o mesmo proporciona a percepção de mundo através de entradas sensorial e cenário operacionais altamente variáveis e não lineares, que mudam de características em tempo real. Entre os principais problemas, envolvendo o modelo de ambiente, estão: variações do tipo de terreno, sensores ruidosos, interferência dos raios solares e erros odométricos. No entanto, o acúmulo de erro pode levar a níveis inaceitáveis de discrepância, prejudicando de forma significativa a performance do sistema de controle e navegação (KYRIACOU, NEHMZOW, *et al.*, 2008).

Outro problema bastante comum é a falta de um procedimento formal baseado na interação simulação-realidade (o que proporcionaria uma metodologia apropriada para controle de robôs autônomos móveis), visto que os programas desenvolvidos utilizam processos empíricos de "tentativa e erro". Esse método está sujeito a muitos erros, acarretando prejuízos materiais (queima de componentes e equipamentos) e de tempo, onde desenvolvedores precisam testar seguidas vezes até encontrar o ponto ideal (KOESTLER e BRAUNL, 2004).

Considerando estes fatores, muitos trabalhos desenvolvidos na IA, em especial focados em robôs autônomos móveis, são voltados para análise de comportamento entre robôs simulados e reais, com o objetivo de fazer com que os algoritmos ou métodos desenvolvidos em simuladores possam ser adaptados e também executados em plataformas reais (BALAKIRSKY, PROCTOR, *et al.*, 2008); (HU e ZEIGLER, 2004).

A motivação do presente trabalho é propor um estudo que analise as diferenças significativas entre os ambientes simulados e realistas. Para isso é utilizado o algoritmo de anticolisão aplicado em um sistema de robô real e outro simulado. Este tipo de algoritmo é utilizado para se obter o domínio dos obstáculos encontrados no ambiente, uma vez que os robôs são colocados em lugares desconhecidos e o algoritmo fornece um conjunto de parâmetros que possibilita localizar obstáculos fixos e desviá-los enquanto percorre o menor trajeto possível. A principal razão para o uso deste tipo de algoritmo, em particular, é o seu uso em quase todos os sistemas de robótica móvel e a facilidade de implementação nas plataformas simuladas/reais.

Apesar da relevância do estudo envolvendo meios simulados e reais, existem poucos registros na literatura que provam a eficiência e a fidelidade da aplicação de algoritmos produzidos e testados nesta perspectiva. A aplicação correta do experimento poderá introduzir uma nova visão para desenvolvedores que, ao projetarem sistemas robóticos utilizando ambientes simulados, possa auxiliar na transferência do código originalmente desenvolvido no simulador para o "mundo real" com pouca ou nenhuma modificação.

1.2 Objetivos

A pesquisa tem como objetivo geral implementar o algoritmo proposto de anticolisão em robôs reais e simulados, de forma que seja possível a análise das principais diferenças entre tais implementações. Deste modo, pretende-se contribuir com a construção do conhecimento para a especificação futura de um modelo de portabilidade que auxilie a transição de algoritmos simulados para robôs reais.

Os objetivos específicos são:

- Desenvolver um protótipo equipado com um sensor omnidirecional, capaz de detectar e desviar obstáculos fixos em ambientes desconhecidos;
- Definir o microcontrolador e o simulador adequado para validação do trabalho;
- Construir o robô físico, juntamente com uma placa de circuito impresso que facilite a inserção e a remoção dos sensores e atuadores, organizar os fios e as conexões do sistema robótico real;
- Implementar o algoritmo de anticolisão e aplica-lo nos robôs simulado e real;
- Testar e validar a pesquisa a partir de experimentos em cenários específicos.

1.3 Organização do Trabalho

O presente trabalho está estruturado em seis capítulos. No primeiro capítulo é apresentada uma introdução geral da pesquisa, mostrando os principais pontos a serem aprofundados no decorrer do trabalho. Em seguida, são abordados os princípios que motivaram a pesquisa e, por fim, os objetivos a serem alcançados.

O segundo capítulo apresenta a fundamentação teórica do trabalho, na qual são mostrados os conceitos necessários para compreensão da pesquisa. Inicialmente é realizada a definição do termo robótica e suas aplicações, seguido de uma especificação sobre os "RAM"

(robôs autônomos móveis). São abordados também conceitos de microcontroladores, as principais plataformas de prototipagem existentes no mercado, sensores e simuladores. E por fim, a modelagem cinemática do robô autônomo móvel desenvolvido.

O terceiro capítulo relata resumidamente alguns dos principais trabalhos da literatura envolvendo algoritmos desenvolvidos para robôs autônomos móveis, incluindo pesquisas focadas no comportamento dos robôs em ambientes simulados e reais.

O quarto capítulo enfatiza com detalhes o trabalho proposto. Inicialmente é explicitado o processo de desenvolvimento do robô e da placa de circuito impresso. Em seguida, é justificada a escolha da plataforma Arduino e do simulador de robôs *Unity3D*, bem como o processo de desenvolvimento e implementação do algoritmo de anticolisão.

Para validação do experimento, no quinto capítulo, são apresentados quatro testes elaborados para a aplicação do algoritmo de anticolisão em ambientes reais e simulados, e como o robô interage com base nas variações propostas na pesquisa.

O sexto capítulo trata das considerações finais da proposta apresentada. Nele é feito um breve resumo do trabalho, das motivações e resultados obtidos. Fala-se também de algumas das contribuições do projeto e dos trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

No presente capítulo são mostrados os conceitos necessários para compreensão do trabalho, partindo dos conceitos gerais de robótica e suas aplicações, até um maior detalhamento sobre os robôs autônomos móveis. Por fim, são abordados conceitos de microcontroladores, as principais plataformas de prototipagem existentes no mercado, sensores, módulos, simuladores e modelagem cinemática.

2.1 Definições

A origem da palavra robô, ou "robota" em Tcheco, significa trabalho árduo ou escravo. Essa expressão foi usada pela primeira vez em uma peça de ficção chamada *R.U.R.*, (*Robôs Universais de Rossum*), do autor Karel Capek, de 1921. A peça conta a história de um cientista que desenvolve robôs para ajudar pessoas a executarem tarefas simples e repetitivas. No entanto, uma vez que esses dispositivos robóticos são usados para lutar em guerras, eles se voltam contra seus criadores e conquistam o mundo.

Na atualidade, os robôs possuem outros significados e utilidades. Portanto, para que se possa entender sobre a Robótica, se faz necessário compreender o conceito de robô e sua função.

Segundo (MARTINS, 2007) “o robô é um dispositivo automático adaptável a um meio complexo, substituindo ou prolongando uma ou várias funções do homem e capaz de agir sobre seu meio”. De acordo com a definição oficializada da Associação das Indústrias de Robótica (RIA) o robô é “um manipulador (re)programável, multifuncional projetado para mover materiais, peças, instrumentos ou outros dispositivos especiais através de vários movimentos programados para realizar uma variedade de tarefas”.

Já na definição da Associação de Robótica Industrial do Japão (JIRA), o robô é definido como um sistema mecânico que possui movimentos flexíveis análogos aos movimentos orgânicos, e combina esses movimentos com funções inteligentes e ações semelhantes as do humano (SCHLUSSEL, 1985). Neste contexto, função inteligente significa: decisão, reconhecimento, adaptação ou aprendizagem.

Logo, podemos entender o significado da Robótica como uma ciência dos sistemas que interagem com o mundo real ou simulado, com pouca ou mesmo nenhuma intervenção humana.

2.1.1 Robô autônomo móvel

Os robôs autônomos móveis (RAM) são caracterizados pela sua capacidade de locomoção e de operação de modo semi ou totalmente autônomo. Os níveis de autonomia são alcançados na medida em que os robôs passem a considerar os aspectos de maior importância, como: capacidade de percepção (leitura do ambiente em sua volta), capacidade de agir (relacionado ao deslocamento e ações produzidas por atuadores e motores) e a presença da inteligência (aptidão de tomar decisões e lidar com situações por mais complexas que sejam).

De acordo com (SILVA, 2008), os robôs móveis atuais possuem sensores de infravermelho, sonar, acelerômetro, temperatura, toque, umidade, entre outros. A utilização desses sensores permite o robô interagir com o ambiente, ajudando na percepção e auxiliando na elaboração do modelo de onde esse robô está inserido. Dessa forma, o valor de entrada dos sensores, provenientes do meio, facilita na tomada de decisões e na realização de tarefas.

Jacobo (JÁCOBO, 2001) cita três características que elucidam a estratégia de decisão e percepção a serem tomadas pelos robôs.

- **Estratégia reativa**: o comportamento do robô depende dos estímulos gerados pela reação provenientes da leitura dos sensores no ambiente;
- **Estratégia deliberada**: as informações do ambiente são previamente processadas e a tarefa e o comportamento do robô são previamente determinados;
- **Estratégia híbrida**: junção das abordagens anteriores.

Os dispositivos robóticos propostos neste projeto se encaixam em uma estratégia de decisão deliberada e sua categoria pertence à Robótica Autônoma Móvel, de acordo com as características de montagem de um protótipo com duas rodas motrizes e duas rodas com esferas do tipo castor, que permitem sua locomoção de forma autônoma. A captação de informações do ambiente é feita com auxilio de sensores posicionados em volta do veículo.

2.2 Microcontroladores

Atualmente vivemos cercados de dispositivos que utilizam algum tipo de circuito integrado. Porém, nem sempre percebemos em quais aparelhos esses microcontroladores são encontrados. Esses pequenos computadores, embutido em um pequeno chip, possuem um grande poder de controle e geralmente são encontrados em dispositivos como TVs, *smartphones*, máquina de lavar roupas, automóveis, câmeras digitais, micro-ondas, brinquedos e especialmente utilizados em dispositivos robóticos.

O trabalho de Souza (SOUZA, 2008) define o microcontrolador ou controlador embutido como um "pequeno" componente eletrônico, dotado de uma "inteligência"

programável, utilizado no controle de processos lógicos. No entanto, (DENARDIN, 2008) conceitua o microcontrolador como um sistema computacional completo, no qual inclui uma CPU (*Central Processor Unit*), memória de dados e programa, um sistema de *clock*, portas de Entrada/Saída (*Input/Output*), além de outros possíveis periféricos tais como, módulos de temporização e conversores Analógico/Digital, integrados em um mesmo componente. Diante dessas definições, podemos entender também que os microcontroladores são circuitos integrados programáveis que possuem arquitetura de um microcomputador, sendo que sua estrutura baseia-se em uma arquitetura RISC (*Reduced Instruction Set Computing*).

As máquinas baseadas em RISC possuem poucas instruções e, em decorrência disso, sua unidade de controle é simples permitindo que alcance uma melhor performance. Algumas vantagens são perceptíveis como: menor consumo de energia, circuitos menores e melhor desempenho ((RICARDO e ALEXANDRE, 1999) apud (FORMIGA, 2005)).

Os microcontroladores estão disponíveis com 8, 16 ou 32 bits, embora os PCs incorporaram, há bastante tempo, arquiteturas acima de 32 bits, com o modo de proteção e memória virtual de sistemas operacionais. A maioria das aplicações para microcontroladores não necessita mais de que oito bits, afirma McComb e Predko (MCCOMB e PREDKO, 2006). Os controladores embutidos possuem basicamente algumas características que levam o desenvolvedor (usuário) a optar por sua finalidade. Sejam essas características, o baixo custo, tamanho reduzido, praticidade, facilidade de desenvolvimento, manutenção e modificação.

Ainda de acordo com (MCCOMB e PREDKO, 2006), cada microcontrolador disponível no mercado segue sua própria convenção, seja ela fabricada pela Microchip, Atmel, Intel, NEC, Texas Instruments, Motorola, Hitachi, entre outros. Apesar da semelhança de funcionalidade básica entre chips de diferentes empresas, aprender a usar cada microcontrolador envolve uma elevada curva de aprendizado. Em consequência disso, os desenvolvedores de robôs tendem a dedicar-se apenas a uma marca ou até mesmo um modelo, uma vez que, aprender uma nova linguagem ou arquitetura de microcontrolador pode exigir muito trabalho, tempo e adaptação. Neste trabalho, em particular, é empregada a plataforma Arduino, a qual é detalhada a seguir.

2.3 Plataformas

As plataformas de robótica são consideradas um conjunto de componentes eletrônicos que integram um circuito. Estes são: (micro) processador/controlador, Portas de entrada e saída, memórias (ROM, RAM e EEPROM), regulador de tensão, portas USB/RS-232, entre outros. A seguir, definimos as principais plataformas destinadas à construção de projetos autônomos e/ou robóticos.

2.3.1 Arduino

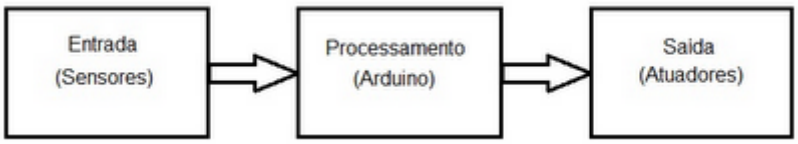
O Arduino é uma plataforma *opensource* de computação física baseada em uma simples placa com entradas/saídas microcontroladas e um ambiente de desenvolvimento que permite a implementação da linguagem Processing/Wiring (ARDUINO, 2005). A plataforma pode ser utilizada para desenvolver projetos autônomos interativos ou ser conectada a um software de computador.

As primeiras ideias do Arduino surgiram na Itália em 2005, através do professor Massimo Banzi e com ajuda do docente David Cuartielles, ambos visaram uma proposta inovadora capaz de interagir com projetos escolares, proporcionando um orçamento menos dispendioso que outras plataformas disponíveis para a época. Além do baixo custo orçamental, o Arduino consegue aproximar usuários que possuem pouco ou nenhum conhecimento em eletrônica e programação. Segundo (WERNECK, 2009) existem outras plataformas de microcontroladores criadas com o mesmo objetivo, mas não mantendo o foco em características como facilidade de uso e preço acessível como o Arduino.

Devido ao surgimento de versões mais robustas, incluindo características como facilidade de manuseio, acessibilidade e flexibilidade, o Arduino tem se tornado popular em meios acadêmicos, escolas, cursos técnicos, por hobistas e amadores, sendo utilizado, em projetos bem elaborados.

O processo de funcionamento do Arduino é simples. O dispositivo interpreta as variáveis de ambiente para transformá-las em sinais elétricos, mediante sensores ligados aos terminais de entrada, controlando e acionando outro componente conectado ao terminal de saída. Através do diagrama de bloco mostrado na Figura 2, é possível identificar os elementos principais do circuito.

Figura 1 - Etapas do funcionamento do Arduino



Fonte: (ARDUINO, 2005)

A arquitetura do Arduino é constituída por um microcontrolador Atmel AVR de 8 bits, fabricada pela Atmel Corporation. As plataformas originais utilizam a série de chips megaAVR, especialmente os ATmega8, ATmega168, ATmega328, ATmega1280 e ATmega2560; porém muitos outros processadores são utilizados por clones.

A maioria das versões possuem um cristal oscilador de 16MHz, 6 portas analógicas e 14 portas digitais de entrada e saída, além de interface serial via conexão USB para comunicação com o computador. Os pinos digitais podem ser configurados através do código fonte para enviar e receber pulsos elétricos de 5v e 3.3v, com corrente máxima de 40 mA e 50 mA, respectivamente. Os pinos 3, 5, 6, 9, 10, 11, possuem saídas PWM – *Pulse Width Modular* (sistema modular de controle de pulsos elétricos), conforme mostra a Figura 1.

Figura 2 - Plataforma Arduino

Fonte: (ARDUINO, 2005)

Na outra extremidade da placa encontram-se seis pinos de entrada analógica, etiquetadas de A0 a A5, usadas para conexão de sensores e componentes eletrônicos diversos, como potenciômetros, botões de pressão, termômetros, entre outros. As entradas analógicas trabalham com 10 bits, que abrange valores entre 0 e 1024, com tensão máxima de 5v. Caso um sensor ou qualquer componente analógico forneça uma tensão de 5v para as portas

analógicas, a placa deverá converter para o valor máximo em bits, que corresponde a 1024. Por outro lado, uma tensão de 0V equivale a 0 bits.

Os projetos e esquemas de hardwares do Arduino são distribuídos sob a licença *Creative Commons*, que permite padronizar o licenciamento e distribuição de conteúdo em geral, de modo a facilitar seu compartilhamento e recombinação, sob a defesa de uma filosofia *copyleft*[1]. Desde 2005, foram disponibilizadas para vendas 11 versões. As principais delas são:

- **Arduino Uno:** é a mais recente versão do Arduino. O "UNO" significa "um" em italiano e representa o lançamento do Arduino 1.0 como base para futuras versões. O Arduino Uno é composto por um microcontrolador ATmega328, com 32kb de Memória Flash, dos quais são utilizados 512 Bytes pelo *bootloader*, 2KB de SRAM e 1KB de EEPROM;
- **Arduino Mega:** é uma placa de microcontrolador baseado no ATmega2560. Ele possui 54 pinos de entradas/saídas digitais, 16 entradas analógicas, 4 UARTs (portas seriais de hardware), um oscilador de cristal de 16 MHz, uma conexão USB, uma entrada de alimentação, uma conexão ICSP e um botão de reset;
- **Arduino Nano**: é uma placa pequena e completa, baseada originalmente no ATmega168, possuindo semelhanças ao Arduino Duemilanove;
- **Arduino Mini**: placa Arduino baseada originalmente nas especificações do microcontrolador ATmega168, porém suporta somente tensão de 9v e 40mA de entrada;
- **Arduino BT (Bluetooth)**: baseia-se nas especificações do microcontrolador ATmega328 com a diferença de acoplar, em seu circuito, um módulo *bluetooth* integrado;
- **LilyPad Arduino**: diferente das outras placas convencionais, a LilyPad tem formato circular, sendo projetada para ser costurada em roupas e outras aplicações que envolvem tecidos. A linha de costura é o próprio condutor elétrico, entre as entradas/saídas das placas e os sensores. Sua arquitetura é composta por um microcontrolador ATmega168, um oscilador de cristal de 8Mhz, 16Kb de Memória Flash com 2Kb de *bootloader*, 1 Kb de SRAM, 512bytes de EEPROM, operando com tensão entre 2.7v e 5.5v e dispensa o regulador de tensão embutido;

[1] O *copyleft* denomina genericamente uma ampla variedade de licenças que permitem, de diferentes modos, liberdades em relação a uma obra intelectual. Seu nome origina-se do trocadilho com o termo "*copyright*"; literalmente, *copyleft* pode ser traduzido como "cópia permitida".

- **Arduino Diecimila**: baseia-se no microcontrolador ATmega168, possuindo metade dos recursos de memória flash, SRAM e EEPROM, comparado ao Arduino Uno;
- **Arduino Duemilanove**: foi lançada em 2009, com a versão ATmega168. As versões seguintes do *Duemilanove* trouxeram como inovação a introdução do microcontrolador Atmel328 e a alternância automática entre a alimentação USB e DC, pois na versão anterior chamada *Diecimila* a alimentação era selecionada manualmente através de *jumpers*.

A plataforma Arduino também disponibiliza um ambiente de desenvolvimento integrado ao hardware (IDE – *Integrated Developement Environment*) para geração dos programas (*sketches*) que serão enviados para a plataforma. Esses *sketches* consistem basicamente de duas funções. A função "*void setup*()" configura os parâmetros iniciais do programa. A função "*void loop*" mantem uma rotina de *loop* que executa repetidamente o código até que ocorra uma interrupção interna ou através de alguma ação externa motivada por um sensor. A linguagem de programação interpretada pelo Arduino é o *Processing*, muito semelhante à linguagem C/C++ e muito utilizada em sistemas embarcados. O ambiente de desenvolvimento é multiplataforma e são compatíveis com os sistemas operacionais Windows, Linux e Mac OS X.

Com o objetivo de aumentar as funcionalidades da plataforma Arduino, várias empresas de hardware desenvolveram placas eletrônicas adicionais para conexão nos terminais do Arduino. Estas placas adicionais são denominadas "*Shields*" e acrescentam várias funções específicas à plataforma, desde controle sobre motores, sensores até sistemas de rede sem fio.

Entre as expansões mais utilizadas, podemos citar:

- **Motor *Shield***: é utilizado para controlar motores de até 18v. Este *shield* inclui uma superfície de montagem em ponte-H, o qual permite uma maior tensão do motor a ser utilizado;
- **Ethernet**: possibilita a conexão com a Internet, utilizando uma biblioteca de rede que suporta os protocolos TCP (*Transmission Control Protocol*) e UDP (*User Datagram Protocol*);
- **MicroSD**: permite ampliar a capacidade de memória usada pelo projeto ou pode funcionar como registrador de *logs* onde se queira gravar determinas informações, como, por exemplo, a leitura de um sensor;

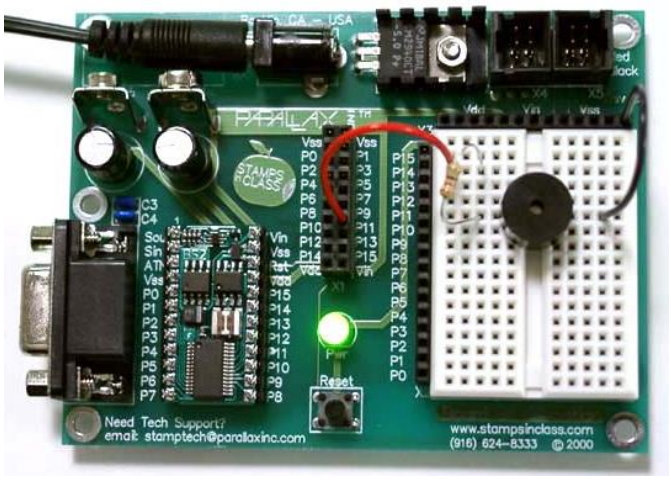
- **GPS *Shield***: este *shield* possibilita obter informações de posicionamento de satélites GPS. Utiliza o padrão *National Marine Electronics Association* (NMEA) que fornece parâmetros tais como longitude e latitude.
- **Kit Joystick**: é um *joystick* que se encaixa sobre o Arduino, transformando-o em um controle simples com 4 botões e 1 manche;
- **GSM *Shield***: utiliza o protocolo (GSM) para enviar mensagens de texto a grandes distâncias;
- **LCD *Shield***: mostra informações do processamento na tela LCD acoplada ao Arduino.

Podemos considerar que o Arduino se destaca em relação a outras plataformas devido a uma grande variedade de componentes, sensores, módulos e atuadores. Possui baixo consumo de energia e uma grande quantidade de expansões que aumentam a capacidade de armazenagem, processamento e adaptabilidade.

2.3.2 BASIC Stamp

Os módulos *BASIC Stamp* são microcontroladores projetados para uso de diversas aplicações embarcadas. Cada módulo acompanha um *chip* interpretador BASIC, memória interna (RAM e EEPROM), regulador de tensão de 5 volts, pinos de entrada e saída (Nível TTL, 0-5 volts), seguido de um conjuntos de comandos matemáticos internos para operações de entrada e saída (Ver Figura 3). Os módulos *BASIC Stamp* são programados de acordo com uma forma simplificada e personalizada da linguagem BASIC, chamada de PBASIC (BASIC STAMP, 2000).

Figura 3 - Plataforma *Basic Stamp*



Fonte: (BASIC STAMP, 2000)

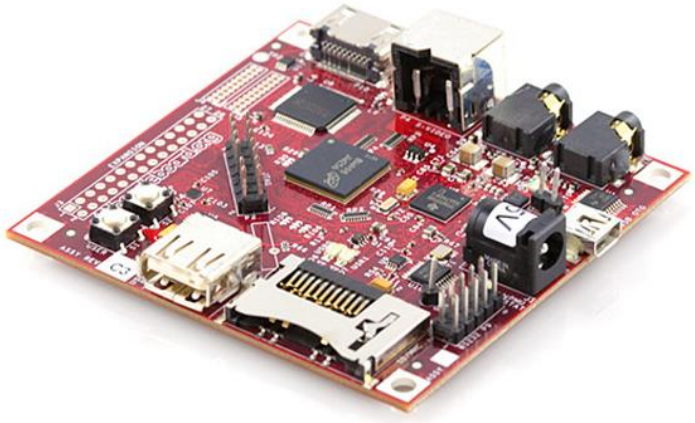
A linguagem PBASIC foi desenvolvida especificamente para controlar o dispositivo BASIC Stamp, com uma proposta de facilitar a aprendizagem e manipulação por parte do usuário. A sua estrutura inclui muitas das principais instruções de outras formas de BASIC (GOTO, FOR ... NEXT, IF ... THEN ... ELSE), bem como algumas instruções especializadas (SERIN, PWM, BUTTON, COUNT e DTMFOUT).

Os microcontroladores BASIC Stamp têm sido utilizados por engenheiros e hobistas desde os primeiros modelos lançados em 1992. Em novembro de 2004, a Parallax chegou à marca de três milhões de módulos BASIC Stamp em uso (PARALLAX, 2004). Até o presente momento, esta tecnologia possui cinco modelos disponíveis: o BASIC Stamp 1, BASIC Stamp 2, 2e BASIC Stamp, 2SX BASIC Stamp e 2p BASIC Stamp.

2.3.3 BeagleBoard

O *BeagleBoard* é um computador em uma placa única, desenvolvido pela *Texas Instruments*, que utiliza um processador de vídeo/imagem OMAP3530 (*Open Multimedia Application Platform*) embutido em microprocessador ARM Cortex A8. Esta placa tem o suporte de uma ampla comunidade ativa, sendo desenvolvida através de uma política baseada em uma iniciativa *opensource* (Figura 4).

Figura 4 - Plataforma *BeagleBoard*



Fonte: (BEAGLEBOARD, 2010)

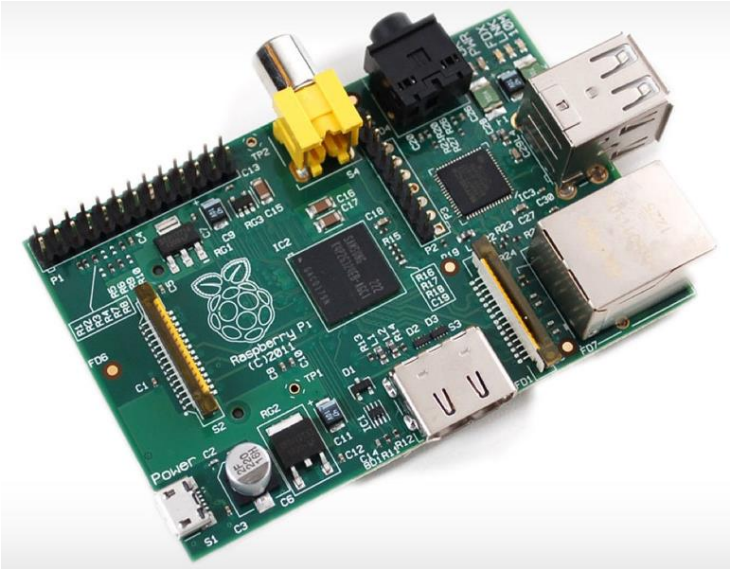
A plataforma mede aproximadamente 7,6 x 7,6cm e possui todas as características e funcionalidades de um computador básico. Com numerosas opções de expansão, pode ser aplicado em uma grande variedade de projetos. O processador de vídeo/imagem OMAP3530 inclui um *encoder* TM320C64x que suporta vídeo de alta definição com processamento digital de sinal para decodificação de áudio e vídeo, e uma unidade de processamento gráfico 2D e 3D com suporte ao OpenGL ES 2.0, incluindo saídas de vídeo que podem ser conectadas através das interfaces S-video ou DVI-D (HDMI) (BEAGLEBOARD, 2010).

O *BeagleBoard* inclui uma interface MMC+/SD/SDIO, USB 2.0, conectores de áudio de 3,5mm de entrada/saída e conectores RS-232 e JTAG. O fornecimento de energia é provido pela USB ou por uma conexão externa de 5 volts.

2.3.4 Raspberry Pi

O Raspberry Pi é um computador com dimensões reduzidas, basicamente do tamanho de um cartão de crédito (Ver Figura 5). A placa foi desenvolvida em 2009 no Reino Unido pela Raspberry Pi *Foundation*, com intuito de fornecer uma plataforma de baixo custo a ser utilizada para fins educacionais. Seu sistema é integrado em um chip gráfico da *Broadcom* BCM2835 que inclui um processador ARM de 700MHz, Memória RAM de 256 Mb e GPU VideoCore IV. A placa não acompanha memória não-volátil. Portanto, é necessário adquirir um cartão de memória SD para armazenamento de dados (RASPBERRY PI, QUICK START GUIDE, 2012).

Figura 5 - Plataforma Raspberry Pi



Fonte: (RASPBERRY PI, QUICK START GUIDE, 2012)

Atualmente existem dois modelos (A e B) disponíveis no mercado. Ambos possuem interfaces HDMI e RCA vídeo, USB, áudio e interface para cartão SD. Porém, o Modelo A possui apenas uma porta USB, enquanto o Modelo B possui duas portas USB com controlador de *Ethernet*. Apesar de não conter a porta *Ethernet*, o Modelo A pode ser conectado a internet através da *Wi-Fi* ou de um adaptador USB de *Ethernet*.

2.4 Sensores

São dispositivos responsáveis por captar informações provenientes do ambiente, tais como informações luminosas, cinéticas, térmicas, etc, relacionando tais informações com uma grandeza física que precisa ser mensurada (*e.g.*, velocidade, corrente, temperatura, posição, aceleração, entre outras). O uso de sensores externos é um dos principais fatores responsáveis por tornar os robôs móveis agentes inteligentes.

Segundo (SUÁREZ, 2000), os sensores possibilitam a entrada de informações em um sistema inteligente. Os dados providos pelos sensores são direcionados ao módulo de processamento sensorial do sistema inteligente, que por sua vez, converterá os sinais vindos dos sensores em informações úteis ao sistema. A seguir é detalhado os principais sensores e módulos utilizados no projeto prático.

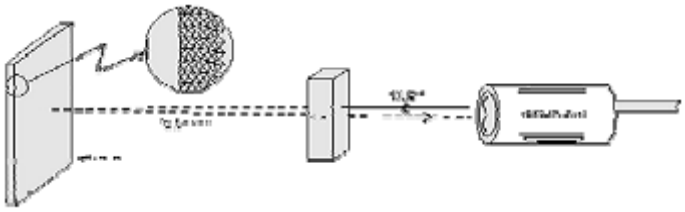
2.4.1 Sensor Infravermelho

São dispositivos sensoriais que detectam a distância de um objeto através da variação da tensão e corrente elétrica. Esses componentes emitem raios infravermelhos invisíveis ao olho humano.

Os sensores IRs, como também podem ser chamados, são divididos em dois grupos principais: os ativos e os passivos. Os ativos possuem um emissor que envia ondas infravermelhas, as quais são captadas pelo dispositivo fotodiodo (passivo). Por sua vez, os passivos recebem apenas o sinal enviado pelo emissor. Dessa forma, o sensor infravermelho subdivide-se em três grupos: sistema por difusão, por barreira e reflexão (BRESSANI, PERINI, *et al.*, 2006).

- **Sistemas por difusão**: os raios infravermelhos são emitidos diretamente sobre o objeto, retornando um feixe de luz ao receptor (Figura 6). Dessa forma, é possível calcular a distância de acordo com o tempo de retorno entre a emissão e recepção;

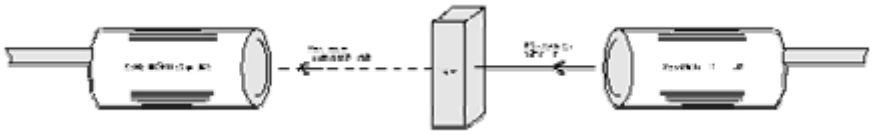
Figura 6 - Sistema por difusão



Fonte: (TOMBA, 2008)

- **Sistemas por barreira**: o emissor é posto diante do receptor infravermelho a uma distância pré-estabelecida de acordo com as características de cada sensor (Figura 7). A interrupção do sinal ocorre quando há um bloqueio entre o emissor e o receptor;

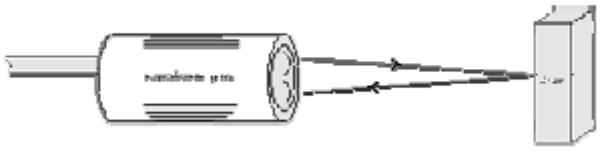
Figura 7 - Sistema por barreira



Fonte: (TOMBA, 2008)

- **Sistemas por reflexão**: os elementos de emissão e recepção estão sobrepostos no mesmo conjunto óptico (Figura 8). Os raios emitidos pelo transmissor refletem em um espelho prismático posto a sua frente e retornam ao elemento receptor.

Figura 8 - Sistema por reflexão



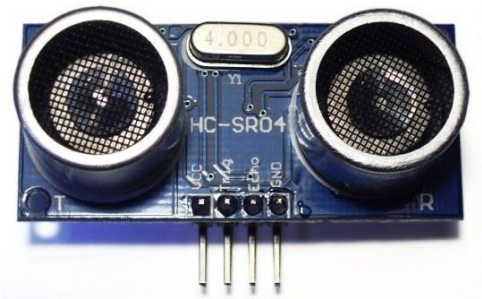
Fonte: (TOMBA, 2008)

Estes sensores alimentarão o sistema de controle robótico, permitindo que ele detecte obstáculos em sua volta. Mais detalhes do sensor omnidirecional, será explicado na seção 2.4.3.

2.4.2 Sensor Ultrassônico

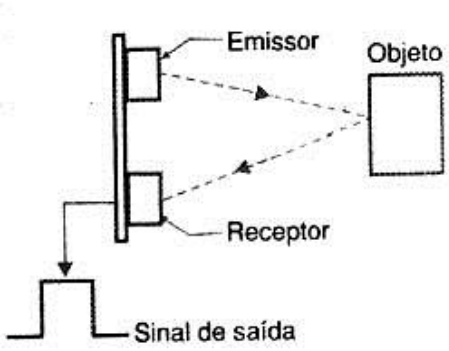
Os sensores ultrassônicos medem a distância entre um ponto de referência e os objetos no campo de atuação do mesmo. Esses dispositivos são baseados no método de pulso-eco, que baseia-se no tempo de trânsito que uma onda ultrassônica gasta para percorrer um trajeto de ida e volta. Trata-se então de medir o intervalo percorrido entre o momento da emissão da onda ultrassônica e o instante do retorno do eco refletido (Figura 9). Esse método foi desenvolvido através de observação dos sons emitidos por morcegos para calcular a distância de um objeto a frente.

Figura 9 - Sensor ultrassônico



Fonte: omecatronico.com.br

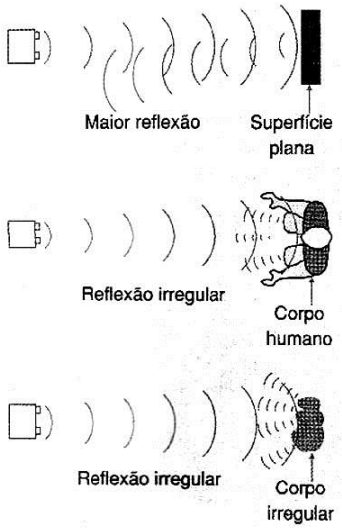
Figura 10 - Emissão e captação do ultrassom.



Fonte: (BRAGA, 2013)

O emissor é formado por um cristal piezoelétrico, no qual emite pulsos de ultrassons de curta duração. Esses pulsos se propagam e refletem no anteparo a frente, que ao serem detectados, retorna um ou mais pulsos de reflexão que serão captados por um microfone, conforme mostra a Figura 10.

Figura 11 - Padrões de reflexão

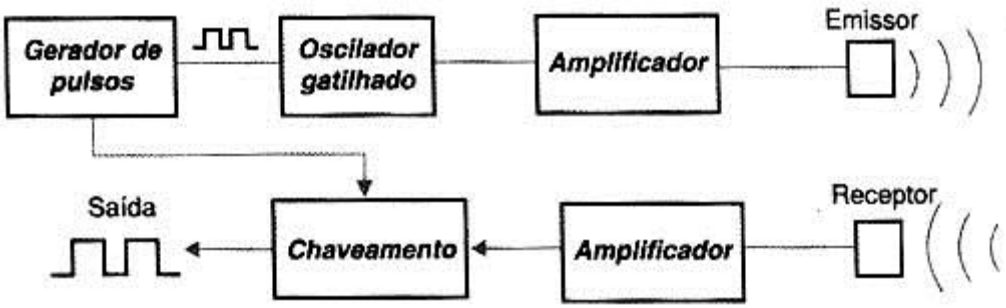


Fonte: (BRAGA, 2013)

O formato do anteparo influencia no padrão de reflexão da onda ultrassônica. Assim, uma superfície plana reflete o som praticamente na mesma direção de onde ele provém, enquanto em outros formatos, podem provocar reflexões em diversos padrões, como mostra a Figura 11.

A Figura 12 mostra um diagrama de blocos que representa um circuito típico de um sensor ultrassônico.

Figura 12 - Diagrama de blocos de um circuito de sensor ultrassônico



Fonte: (BRAGA, 2013)

O oscilador produz ondas ultrassônicas em intervalos regulares, as quais são emitidas pelo gerador de pulsos (transdutor). O sinal monoestável[2] é disparado pelo oscilador que

[2] São sinais enviados por um circuito que tem dois estados, em que, somente um deles é estável.

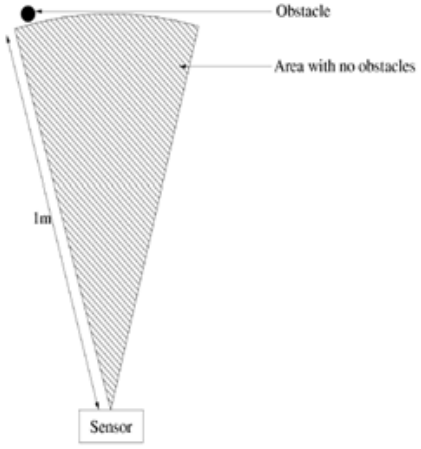
determina o tempo previsto para o retorno do eco. Se o eco voltar em um intervalo maior que o previsto, significa que o objeto está longe o suficiente para não ser detectado.

O sinal monoestável serve para abrir o sensor de retorno ou microfone ultrassônico e também serve para cronometrar o tempo de resposta. Quando o eco é captado dentro do intervalo previsto, o cronômetro é paralisado, obtendo-se uma indicação da distância em que o objeto detectado se encontra.

2.4.3 Sensor Omnidirecional

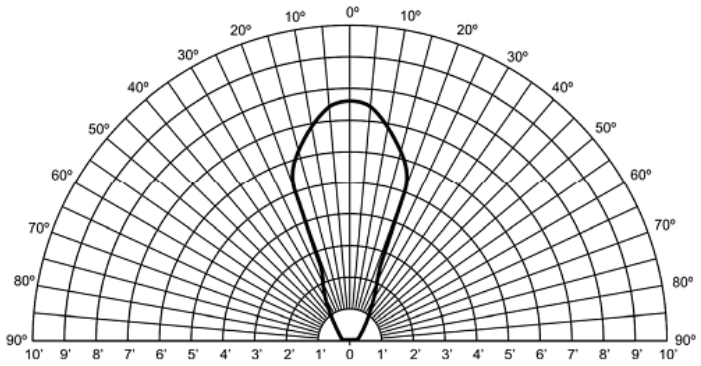
A palavra **omni**, derivada do grego *omne*, significa todo ou inteiro e **direcional** está relacionado com direção ou sentido. Em outras palavras, um dispositivo omnidirecional é um sensor, ou um conjunto de sensores que pode emitir ou receber/captar algum sinal em todas as direções.

Figura 13 - Leitura de uma única faixa do sensor ultrassônico



Fonte: (BRAGA, 2013)

Figura 14 - Faixa de detecção do sensor ultrassônico.



Fonte: (KALMEGH, SAMRA e RASEGAONKAR, 2010)

O sensor omnidirecional é constituído por um conjunto de sensores ultrassônicos ou infravermelhos. Cada unidade de sensor possui uma faixa de detecção que gira em torno de 30º a 45°. Esse valor varia de acordo com a distância do obstáculo. Quanto mais distante o cone alarga, enquanto que quanto mais próximo o cone estreita (Figura 13). Isso aumenta a incerteza nas dimensões espaciais entre o robô e o anteparo. A maioria dos robôs móveis, que usam sensores ultrassônicos para desviar obstáculos, utilizam entre 8 e 12 sensores em um anel em torno do robô, com a finalidade de cobrir todas as direções.

Figura 15 - Visão superior do sensor omnidirecional

Fonte: Autor

No trabalho proposto, foram utilizados oito sensores ultrassônicos espalhados nas margens de uma circunferência de 20 cm de diâmetro, com a finalidade de obtermos uma cobertura completa de 360º. Cada sensor ultrassônico apresenta uma faixa de detecção em torno de -22,5° a 22,5° ou simplesmente 45°, observe as Figuras 14 e 15.

2.5 Simuladores

Simuladores de robôs são ferramentas próprias para pesquisas em robótica as quais auxiliam no desenvolvimento e testes de controle de novos robôs. Esses simuladores proporcionam algumas vantagens, como a economia de tempo e esforço braçal que possibilita trabalhar com instalações experimentais do "mundo real". Em outras palavras, o tempo de simulação geralmente é menor que o desenvolvimento em hardware e, portanto, os resultados dos experimentos surgem com mais rapidez e demanda menos tempo e despesas para construção de protótipos em relação aos robôs reais.

Entre as diversas opções de simuladores disponíveis, podemos destacar algumas ferramentas, discutidas a seguir.

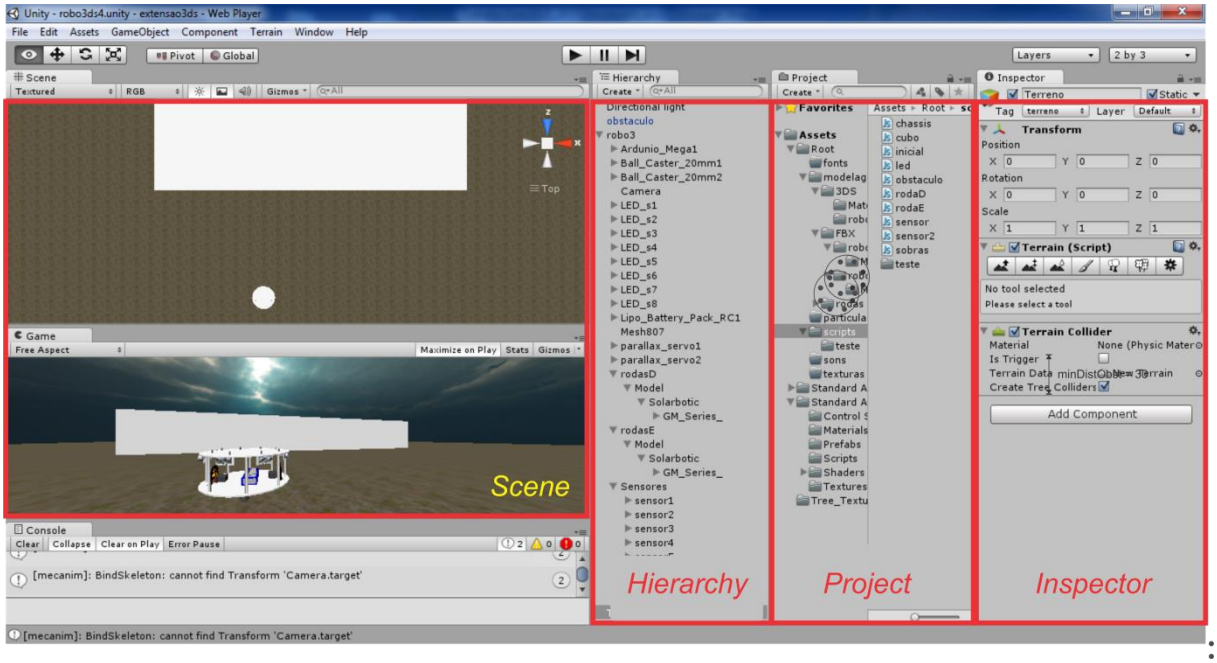
2.5.1 *Unity3D*

O *Unity3D* não é exatamente um simulador, mas pode ser utilizado para tal função. O *Unity3D* é considerado tanto uma IDE quanto um motor físico para criação de jogos. Este simulador foi desenvolvido pela *Unity Technologies* e possui dois tipos de licenças: *UnityPro*, que custa em torno de U$ 1500,00, sendo disponibilizada para testes por um período de 30

dias. Já a versão gratuita é chamada simplesmente de *Unity*, sendo concedida para fins educacionais.

O *Unity3D* possui um ambiente de desenvolvimento integrado por diversas telas e elementos estruturais para criação de jogos/simulações. Os jogos ou simulações são criados através da junção de cenas, que funcionam como etapas do processo de simulação. O fato de separar as cenas em etapas permite que o computador economize processamento, pois somente os elementos ativos na tela serão processados.

Figura 16 - Interface de desenvolvimento do *Unity3D*



Fonte: Autor

A interface de desenvolvimento (IDE) é composta por quatro telas principais que são mostradas quando o sistema é inicializado, sendo que cada uma possui uma função específica (Figura 16). São elas *Scene*, *Hierarchy*, *Project e Inspector*.

- *Scene* – é a principal janela da IDE. Através dessa janela são exibidos todos os elementos da aplicação, como por exemplo, controles que permitem rotacionar, ajustar o posicionamento, aumentar e diminuir o tamanho dos objetos. Na *scene* também é possível navegar por entre os objetos, obtendo a visão em diversos ângulos e distâncias, antes mesmo da aplicação ser executada;

- *Hierarchy* – como o próprio nome indica, o componente *hierarchy* organiza todos os elementos postos em cena de forma hierárquica, exibindo esses os objetos em forma de árvore de visualização;
- *Project* – permite manipular os arquivos que compõe um projeto, os quais podem ser: scripts, modelagens tridimensionais, texturas, arquivos de áudio e um componente utilizado para instanciar vários objetos em uma linguagem de programação orientada a objeto chamada de *prefabs*.
- *Inspector* – exibe, com detalhes, os componentes de um determinado objeto. Cada componente é composto de atributos que são valores manipuláveis ou editáveis que influenciam no comportamento e características de um objeto.

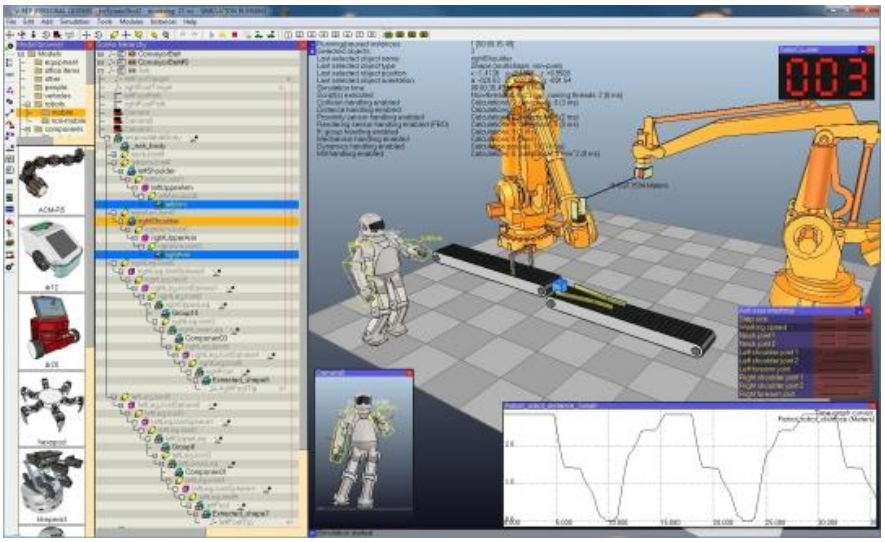
Outro fator importante é a possibilidade do usuário manipular os objetos de forma a atribuir regras ou ações específicas para o seu funcionamento. Para isso são utilizados os *scripts* que são linhas de códigos escritas de forma lógica e salvas em arquivos com extensão referente a uma determinada linguagem de programação. O *Unity3D* possibilita trabalhar com três tipos de linguagens de programação: *Javascript*, *C++* ou *Boo*.

2.5.2 V-REP

O *software Virtual Robot Experimentation Platform* (V-REP) foi desenvolvido pelo Dr. Marc Freese, sendo utilizado na maioria dos casos, como ferramenta de simulação de robôs (Ver Figura 17). A grande vantagem da utilização do V-REP, assim como o *Unity 3D*, é a possibilidade de importar modelos 3D construídos através do *Google SketchUp*, podendo criar simulações 3D rapidamente, ao contrário de outros *softwares* de simulação, que exigem modelagens tridimensionais feitas apenas por ferramentas CAD, tornando o processo de construção lento e complexo.

O simulador V-REP, possui um ambiente de desenvolvimento integrado, baseado em uma arquitetura de controle distribuído: cada objeto/modelo pode ser controlado individualmente através de *scripts, plug-ins,* nós ROS e/ou cliente API remoto. Isso faz do V-REP um simulador versátil e indicado, principalmente, para aplicações envolvendo múltiplos robôs. As linguagens de programação aceitas são C/C++, Python, Java, Lua, Matlab e Urbi.

Figura 17 - Simulador V-REP



Fonte: Autor

O V-REP utiliza dois diferentes motores físicos, sendo eles: *Open Motor Dynamics* (ODE) e o *Bullet* (FREESE, 2011). O usuário V-REP pode optar por qual motor que será utilizado em uma simulação específica. Esses motores físicos possuem atributos que diferem entre sí, que depende da finalidade de cada simulação.

O motor ODE foi criado por Russell Smith, é *opensource* e possui uma biblioteca de alto desempenho para simulação de dinâmica de corpos rígidos (SMITH, 2012). O ODE dispõe de um conjunto avançado de técnicas para detecção de colisão que inclui grandezas físicas, como: força de atrito, gravitacional, aceleração, entre outros. Normalmente, o motor é usado em jogos de computador, ferramentas de criação 3D e de simulação. Existem também aplicações em simulações de veículos, indivíduos virtuais e objetos em ambientes de realidade virtual.

Já o *Bullet* é uma biblioteca de detecção de colisão e dinâmicas de corpos rígidos. Assim como ODE, possui código fonte aberto e livre para uso comercial. É usado principalmente em simulações físicas de jogos (FREESE, 2011).

O V-REP é utilizado para o desenvolvimento de algoritmos rápidos, simulações de automação de fábrica, prototipagem rápida e verificação, robótica educacional, monitoramento remoto, segurança de controle duplo, entre outros (SMITH, 2012).

2.5.3 CARMEN

O *Carnegie Mellon Robot Navigation Toolkit* (CARMEN) é uma coleção de software de código fonte aberto com aplicação no controle de robôs móveis. É um software modular projetado para fornecer primitivas básicas de navegação, incluindo: base e controle de sensor, localização, planejamento de percurso, desvio de obstáculos e mapeamento de ambiente (CARMEN, 2006). As simulações são visualizadas em um ambiente 2D, implementada na linguagem C e Java no sistema operacional Linux. Oferece suporte a atuadores e sensores mais comuns como os de proximidade, GPS e colisão.

2.5.4 *Microsoft Robotics Developer Studio*

É um ambiente baseado no sistema operacional *Windows* para controle e simulação de robôs. O seu uso é destinado a desenvolvedores acadêmicos, hobistas, fins comerciais e lida com diversos dispositivos robóticos (MICROSOFT, 2012).

O *Microsoft Robotics Developer Studio* é baseado em CCR (*Concurrency and Coordination Runtime*, ou seja, Concorrência e Coordenação em Tempo de execução), uma biblioteca cuja programação é realizada no *Framework* .NET da *Microsoft* que visa proporcionar o gerenciamento assíncrono de tarefas paralelas. Esta técnica envolve a utilização de transmissão de mensagens em tempo de execução orientada a serviços leves, DSS (Serviços de Software Descentralizados), que permite o gerenciamento de múltiplos serviços para gerar comportamentos complexos (Figura 18).

Figura 18 - Simulador Microsoft Robotics Developer Studio

Fonte: (MICROSOFT, 2012)

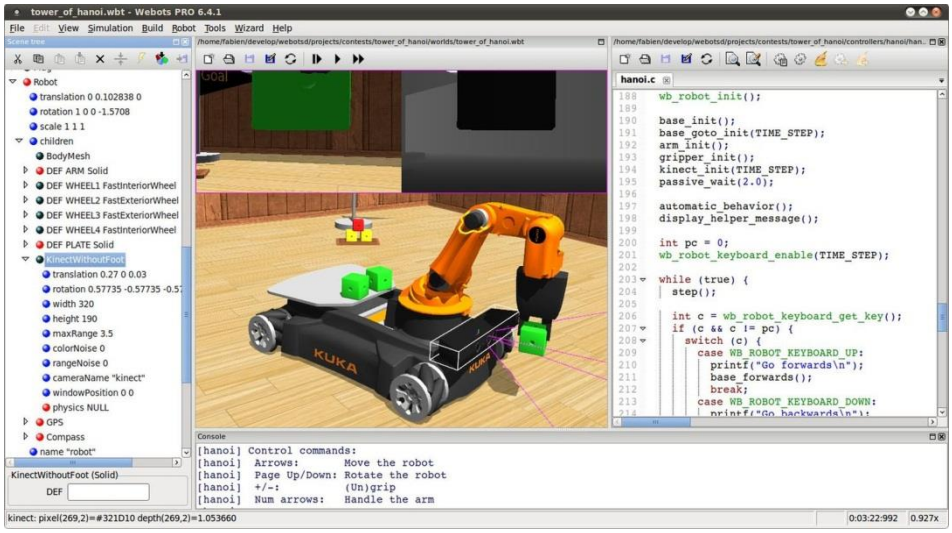
Entre as principais características podemos observar: uma ferramenta de programação visual que permite criar e depurar aplicações para robôs, interface voltadas para web e para plataforma *Windows*, simulação em 3D (incluindo aceleração de hardware), facilidade de manipulação de sensores e atuadores de um robô. A principal linguagem de programação é C#.

O *Microsoft Robotics Developer Studio*, ou simplesmente MRDS, possui suporte para adicionar outros serviços à sua suíte de aplicativos. Por padrão estão disponíveis: Simulador de futebol e competição de Sumô, simulador de labirinto e programa para criar "mundos" com paredes que podem ser explorados por um robô virtual.

2.5.5 *Webots*

É um ambiente de desenvolvimento usado para controlar, programar e simular robôs móveis (WEBOTS, 2012). Foi produzido pelo Instituto Federal Suíço de Tecnologia em Lausanne e atualmente é utilizado por mais de 750 universidades e centros de pesquisa em todo mundo. Através do *Webots* é possível desenvolver projetos robóticos complexos, com um ou mais robôs, semelhantes ou diferentes, compartilhando o mesmo ambiente (Figura 19).

Figura 19 - Simulador Webots



Fonte: (WEBOTS, 2012)

A personalização de cada objeto, tanto quanto a sua forma, textura, cor, atrito e massa são definidas pelo usuário. Uma grande variedade de sensores e atuadores estão disponíveis para equipar cada robô.

O sistema disponibiliza diversas plataformas pré-configuradas como o *Pionner*(2012), Lego NXT e vários módulos com sensores e atuadores mais utilizados. É um software proprietário e sua licença comercial pode chegar a custar U$ 2.300,00.

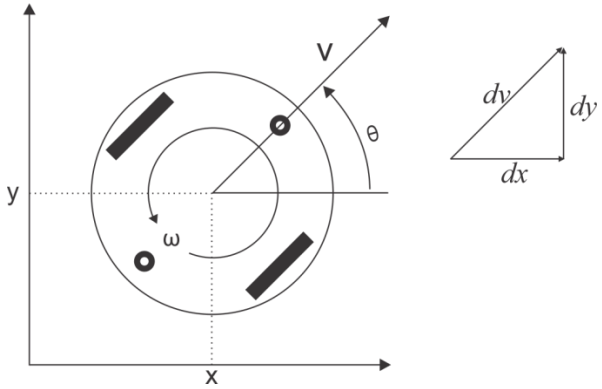
2.6 Modelagem cinemática

O modelo cinemático leva em consideração a geometria do robô e as suas restrições de movimento. As propriedades físicas como a massa e inércia das rodas são desconsideradas. O centro de massa do robô está localizado no meio do eixo que liga as rodas de apoio traseiras e dianteiras.

De acordo com a Figura 20, o eixo (*x*, *y*) corresponde às coordenadas do centro de massa, o ângulo θ é medido a partir da posição do eixo horizontal, *v* é classificado como a magnitude da velocidade de translação do centro de massa, e ω é a velocidade angular do robô. Pressupondo que as rodas não deslizem, o vector *v* é ortogonal em relação ao eixo que liga as rodas traseiras e dianteiras. Considerando o robô com acionamento diferencial, o mesmo possui apenas uma restrição cinemática não-holonômica a qual é demonstrada pela seguinte equação (JIANG, YUICHI e XINGQUAN, 2005):

$$-\dot{x}sen\theta + \dot{y}cos\theta = 0 \tag{2.1}$$

Figura 20 – Sistema de coordenada inercial



Fonte: Autor

Baseando-se na Figura 20, podemos definir o modelo cinemático para deslocamentos incrementais como:

$$\dot{x} = v(t)cos\theta(t) \quad (2.2)$$

$$\dot{y} = v(t)sen\theta(t) \quad (2.3)$$

$$\dot{\theta} = \omega(t) \quad (2.4)$$

As equações cinemáticas do robô são representadas por:

$$| v(t) | < v_{max}, | \dot{v}(t) | < \dot{v}_{max} \quad (2.5)$$

$$| \omega(t)| < \omega_{max}, | \dot{\omega}(t)| < \dot{\omega}_{max} \quad (2.6)$$

De acordo com o princípio do movimento cinemático de um corpo rígido, o movimento de um robô móvel pode ser descrito pela velocidade angular da roda esquerda (φ_L) e a roda direita (φ_R). Os diversos componentes de velocidade estabelecidos na coordenada podem ser expressos nas velocidades das rodas, como a seguir:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{rcos\theta}{2} & \frac{rcos\theta}{2} \\ \frac{rsen\theta}{2} & \frac{rsen\theta}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix} \begin{bmatrix} \varphi_R \\ \varphi_L \end{bmatrix} \quad (2.7)$$

Denota-se **r** como o raio das rodas e **d** o comprimento do eixo entre as duas rodas. No nosso robô estes parâmetros correspondem a 0.008m e 0.054m, respectivamente.

2.7 Resumo

Nesse capítulo foi visto os principais conceitos da robótica, que são pertinentes a este trabalho, e a ramificação dos robôs autônomos móveis como artefato que possui a capacidade de locomoção e de operação de modo parcial ou totalmente autônomo. Neste contexto, foram apresentados a definição de microcontroladores e as plataformas existentes no mercado, com enfoque ao Arduino, *Basic Stamp*, *Beagleboard* e *Raspberry Pi*. Em seguida, foram feitas considerações sobre os sensores infravermelhos, ultrassônicos e omnidirecionais. Definimos ainda os simuladores de robôs e suas vantagens ao permitir trabalhar com instalações experimentais do "mundo real", bem como as suas principais plataformas de desenvolvimento. Por fim, foram abordados os conceitos gerais da modelagem cinemática, os quais serão utilizadas no decorrer do trabalho.

3. TRABALHOS RELACIONADOS

Apresentamos nesse capítulo uma revisão geral acerca dos principais trabalhos da literatura envolvendo a utilização de algoritmos em ambientes de simulação e realidade.

3.1 Visão Geral

Nos últimos anos, diversos simuladores foram criados para auxiliar no desenvolvimento, avaliação e testes na área da robótica. No entanto, diferenças entre simulação e realidade ainda é um grande percalço a ser resolvido. Isso faz com que seja bastante complexo desenvolver métodos e código e transferi-los de uma plataforma simulada para real.

Muitos trabalhos tem se desenvolvido na tentativa de reduzir as diferenças entre ambientes reais e simulados (HU e ZEIGLER, 2005); (HU e ZEIGLER, 2004); (DIXON, DOLAN, *et al.*, 1999); (BALAGUER e CARPIN, 2008); (MEEDEN, 1998); (BALAKIRSKY, PROCTOR, *et al.*, 2008). A maioria desses estudos abordam aspectos comportamentais entre sistemas simulados e reais. Isto é, são aplicados a métodos que visam aprimorar novos movimentos e tomadas de decisões baseadas no processo de aprendizagem de máquina (DAWSON, WELLMAN e ANDERSON, 2010).

Outros autores (CHEN e BILLINGS, 1989) afirmam que muitos problemas estão relacionados a fatores externos. De forma geral, o comportamento do robô é afetado pelo hardware do robô, o algoritmo implementado e o ambiente de operação. A combinação destes três elementos resulta na execução de um cenário de um sistema complexo e altamente variável e predominantemente não linear.

Tendo em vista estes fatores, foram selecionados os principais trabalhos relacionados na área para compor a pesquisa.

3.2 Algoritmo de Evolução

Yuan Xu *et al.* (XU, MELLMANN e BURKHARD, 2010) propuseram em 2010, um estudo a fim de aproximar elementos que caracterizam a simulação dos ambientes reais, fazendo com que as plataformas reais e simuladas pudessem funcionar implementando o mesmo código. Para isso, os autores citam as diferenças/similaridades e o comportamento entre as plataformas como um dos principais pontos a serem analisados, para que em seguida

possam ser utilizados métodos de aprendizagem de máquina vindos a aprimorar o desempenho dos robôs.

Uma equipe de competidores e pesquisadores conhecida como *Humboldt*, a qual participa do evento anual *RoboCup*[3] desde 2008 na modalidade futebol de robôs, fez parte deste estudo. Inicialmente, a equipe competia na liga padrão (robôs reais) e recentemente agregaram a modalidade liga simulada aos seus experimentos (Ver Figura 21 e 22). No mesmo evento, perceberam que as provas realizadas continham os mesmos desafios, tanto na liga simulada quanto na liga padrão. Logo, chegaram à conclusão que desenvolver algoritmos dedicados para problemas semelhantes tornaria o trabalho repetitivo e exaustivo.

A princípio, as diferenças e semelhanças entre a realidade e simulação devem ser analisadas, pois as duas plataformas possuem características divergentes, como por exemplo, os robôs reais que precisam processar imagem por via de câmeras, enquanto a plataforma simulada dispensa o seu uso, visto que, a percepção do simulador é abstrata. Por outro lado, o controlador simulado pode desativar o módulo de processamento de imagem, utilizando o sensor de visão do simulador *SimSpark* para prover percepções.

Figura 21 – Liga padrão (Robôs reais)

Fonte: (XU, MELLMANN e BURKHARD, 2010)

Figura 22 – Liga simulada (Robôs simulados)

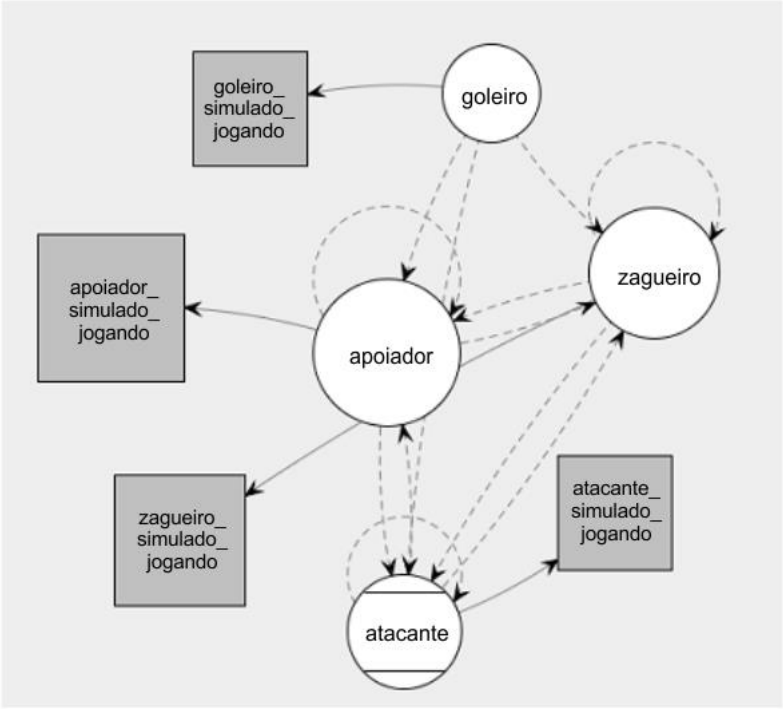
Fonte: (XU, MELLMANN e BURKHARD, 2010)

Outro ponto a ser analisado são as características entre as ligas. O tamanho e a largura do campo da liga de simulação 3D é relativamente maior que a liga padrão. Cada equipe possuem seis robôs na liga de simulação, enquanto apenas três robôs são usados em liga da plataforma padrão. A duração de uma partida da liga da plataforma padrão é duas vezes maior que o jogo de simulação.

[3] O *Robocup* é uma competição a nível mundial que é realizada todos os anos. Visa o estudo e desenvolvimento da Inteligência Artificial (IA) e da Robótica.

A análise do comportamento é realizada através de uma máquina de estado hierárquico, conhecida como XABSL (LOETZSCH, RISLER e JUNGEL, 2006) (*Extensible Agent Behavior Specification Language*). O modelo é utilizado para descrever o comportamento a partir da teoria do jogo de futebol. No projeto existem quatro estados básicos, sendo eles: goleiro, zagueiro, apoiador e atacante (Ver Figura 23). A decisão de mudança entre os estados dependerá das características do robô e do ambiente, ou seja, as condições para decisões são diferentes em cada liga.

Figura 23 - O comportamento de mudança de papel descrito por XABSL



Fonte: Autor adaptado do artigo (XU, MELLMANN e BURKHARD, 2010)

Algumas habilidades foram tratadas como processo comportamental do robô, que se utilizou da aprendizagem de máquina, em especial o algoritmo de evolução, para aprimorar as capacidades de andar, chutar e driblar. E por fim, foi possível observar que plataformas realizaram as mesmas tarefas. Foi a primeira vez, na competição, que o mesmo código fonte foi utilizado em duas ligas: simulação e robôs reais.

3.3 Polinômio NARMAX

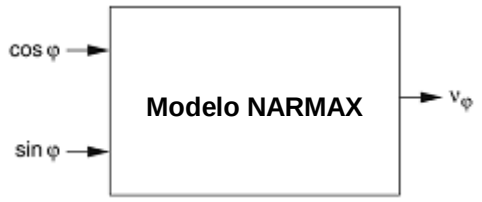
O comportamento de um robô é influenciado por três fatores: o hardware do robô, o algoritmo implementado e o meio em que o robô está inserido. Esses fatores geram um grande e complexo sistema não-linear que, mesmo sendo projetado através de simuladores, não

garante que o resultado saia de acordo com o esperado quando transferido para o ambiente real ou físico, pois ocorrem variações constantes no ambiente de aplicação.

Pensando nisso, (KYRIACOU, NEHMZOW, *et al.*, 2008) desenvolveram uma nova metodologia para criação de simuladores de robôs, que permite prever com exatidão o comportamento e investigar metodicamente como o desempenho dos robôs são influenciados pelo meio, implementando uma base teórica entre a interação robô-ambiente.

Nessa abordagem, foi desenvolvido um modelo matemático transparente, chamado de polinômio de NARMAX. A abordagem de polinômio NARMAX é uma metodologia que prevê variáveis para a identificação de parâmetros e modelos importantes associados a sistemas dinâmicos não-lineares desconhecidos. Através de múltiplas entradas, o polinômio gera uma única saída sem ruído.

Figura 24 - Entradas e saídas do modelo NARMAX



Fonte: Autor adaptado do artigo (KYRIACOU, NEHMZOW, *et al.*, 2008)

Inicialmente, um robô real é utilizado para capturar amostras do ambiente, de modo a colher informações de localização e de valores correspondentes aos sensores utilizados para prever este modelo. Esses dados coletados servem para aproximar a percepção do sensor do robô, em cada nó da grade de localização, que cobre todas as imediações do ambiente. Dois métodos são apresentados para modelar as assinaturas[4] dos sensores de aproximação e de percepção do local.

- Utilizando um polinómio NARMAX único para modelar a assinatura como um todo;
- Usando um conjunto de polinómios NARMAX para modelar a assinatura por partes.

Com o propósito de avaliar a precisão de cada um dos modelos apresentados, foi analisado o comportamento do robô real durante a execução de uma tarefa. O mesmo procedimento foi utilizado com os robôs simulados ao executar tarefas semelhantes. Após a

[4] A assinatura corresponde ao valor capturado por cada sensor em relação ao ambiente.

aplicação do modelo NARMAX, nenhuma diferença significativa de comportamento foi encontrada entre as plataformas reais e simuladas.

3.4 Algoritmo de otimização do *Sapo Pulando Embaralhado*

O planejamento de trajetória de robôs móveis é um problema de tempo polinomial não determinístico, visto que, os métodos tradicionais não são eficazes o suficiente para determinar o menor trajeto possível. Pensando nisso, (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010) desenvolveram um algoritmo evolutivo capaz de resolver o problema de trajetória de robôs, que, por sua vez, atua em ambientes parcialmente desconhecidos com base no algoritmo de otimização *Shuffled Frog Leaping* (SFL), traduzido como, Sapo Pulando Embaralhado. Para validação do problema foi utilizado nos experimentos um robô real Khepera e o simulador de robôs *Webots*.

A relação existente entre o algoritmo do SFL e o problema proposto nesse estudo remete a ideia de que os sapos precisam encontrar o máximo de alimentos o mais rápido possível ao pular sobre as pedras. Os sapos necessitam se comunicar uns com os outros e podem melhorar as suas memes[5] ao compartilhar informações com outros sapos. A estratégia de embaralhamento permite a troca de informações entre buscas de dados locais para mover-se em direção a um melhoramento de forma global.

O algoritmo de otimização SFL permite que o robô móvel navegue através de obstáculos estáticos e encontre o caminho livre até chegar à posição final, sem colidir com objetos em sua volta. O processo de otimização da trajetória do robô é avaliado com base na distância entre os alvos e obstáculos. Assim, ele atualiza constantemente os dados detectado por meio de sensores. Quanto mais próximo à posição do robô estiver para o alvo, melhor será a eficácia do algoritmo. Por outro lado, quanto mais próximo à posição do robô estiver dos obstáculos, menor será a aptidão do algoritmo. Com base nessa afirmação, denota-se $\boldsymbol{T}$ como alvo, cuja coordenada é $\boldsymbol{(X_T, Y_T)}$. Considerando que existem $\boldsymbol{N}$ obstáculos no ambiente, denota-se $\boldsymbol{O = \{O_1, O_2, ..., O_N\}}$ como obstáculos, supõe-se que um ponto e sua coordenadas centrais são $\boldsymbol{(Xo_1, Yo_1), (Xo_2, Yo_2), ..., (Xo_N, Yo_N)}$.

[5] O termo **memes** é derivado do latim, que significa 'lembra-te'. O radial mem- significa pensar. Quando incluído em um contexto coloquial, o termo meme pode significar apenas a transmissão de informação de uma mente para outra. Os memes também podem ser classificados como ideias ou fragmentos de ideias, línguas, sons, desenhos, capacidades, valores estéticos e morais, ou qualquer outra coisa que possa ser aprendida facilmente e transmitida enquanto unidade autônoma. O estudo dos modelos evolutivos da transferência de informação é conhecido como memética.

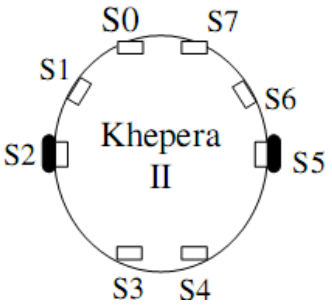
Devido à limitação do intervalo de detecção dos sensores (entre 5 a 10cm), primeiramente é definido às coordenadas do robô. Porém, o tamanho e o formato dos obstáculos no ambiente são totalmente desconhecidos. A cada passo, as informações são atualizadas por sensores e o algoritmo delimita a área em que o robô não deve trafegar, sendo esta área correspondente à região do obstáculo. Caso ocorra à possibilidade de colidir, a direção do movimento é alterada ajustando a velocidade das rodas.

A função P_i, cujas coordenadas são (X_i, Y_i) pode ser expressa da seguinte forma:

$$f(P_i) = w_1 . \frac{1}{\min\limits_{O_j \in O} \left|\left|P_i - O_j\right|\right|} + w_2 . ||P_i - T||$$

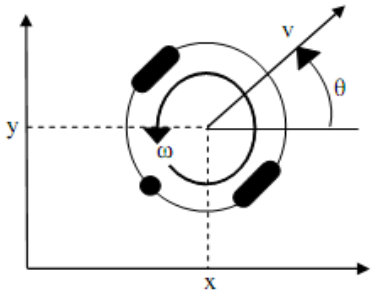
O modelo cinemático do hardware utilizado no estudo é formado por um robô móvel não-holonômico, chamado de Khepera II, com oito sensores acoplados, duas rodas controladas por motores independentes e uma roda castor traseira que impede que o robô incline para trás (Figura 25).

Figura 25 - Posição dos oito sensores IR



Fonte: (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010)

Figura 26 - Esquema de movimentação do robô Khepera



Fonte: (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010)

Considera-se que a massa e a inércia do robô são desconsideradas e o centro de massa está localizado no meio do eixo que liga as duas rodas. O eixo *x* e *y* denotam as coordenadas do centro de massa, θ corresponde ao ângulo medido a partir da posição do eixo horizontal, o

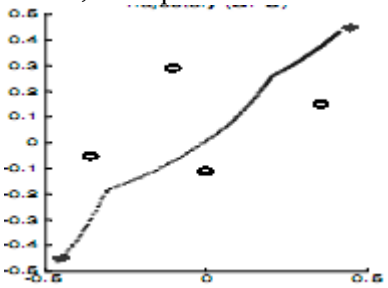
v é classificado como a magnitude da velocidade de translação do centro de massa e o ω é a velocidade angular do robô (Figura 26).

O algoritmo de otimização SFL foi implementado no Khepera II e no ambiente de simulação Webots. O cenário de testes foi montado em um plano com dimensões de 100cm x 100cm, cujo ponto de partida corresponde a S(-49, -49) e a posição do alvo é T(49, 49). Para a simulação, foram utilizados três ambientes diferentes, com 4, 5 e 6 obstáculos, veja as Figuras de 27 a 32.

Figura 27 - Configuração do *Khepera II* com quatro obstáculos no ambiente de simulação *webots*

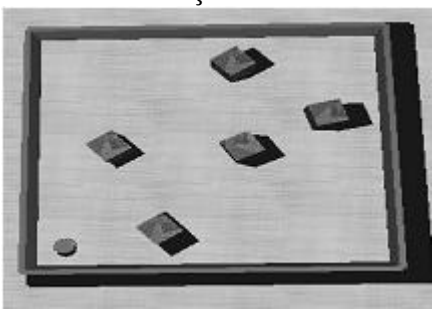
Fonte: (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010)

Figura 28 - A trajetória do robô obtida por GPS, com quatro obstáculos.



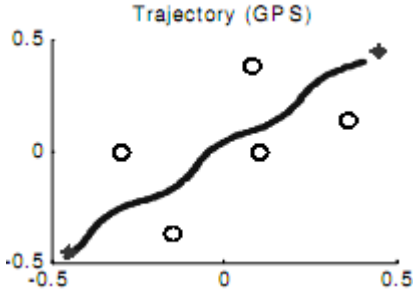
Fonte: (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010)

Figura 29 - Configuração do Khepera II com cinco obstáculos no ambiente de simulação *webots*



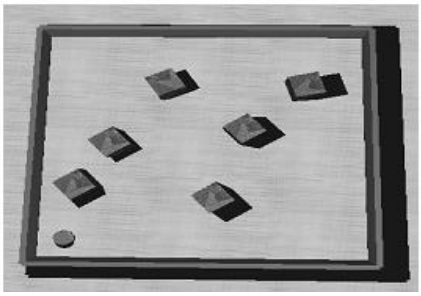
Fonte: (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010)

Figura 30 - A trajetória do robô obtida por GPS, com cinco obstáculos.



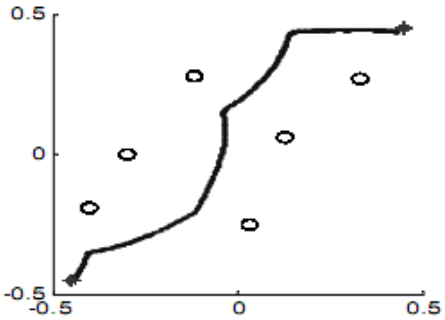
Fonte: (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010)

Figura 31 - Configuração do Khepera II com seis obstáculos no ambiente de simulação *webots*



Fonte: (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010)

Figura 32 - A trajetória do robô obtida por GPS, com seis obstáculos.



Fonte: (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010)

Pode-se observar, nos três casos, que a implementação do algoritmo de otimização SFL no planejamento de trajetória do robô e no simulador é viável. Ou seja, o robô pode atingir o objetivo na condição de não colidir com os obstáculos. Embora o caminho ideal tenha efeito de otimização, não se pode garantir que o trajeto percorrido seja o mais adequado ao utilizar apenas às informações do alvo e dos obstáculos ao mesmo tempo. Logo, os resultados das simulações mostram de forma clara a eficácia e a robustez do algoritmo.

3.5 Ambiente de simulação e realidade mista

Os autores do estudo (CHEN, MACDONALD e WUNSCHE, 2009) propuseram o conceito de Realidade Mista (RM) onde os elementos reais e virtuais se misturam, possibilitando a construção de cenários de testes mais baratos e seguros. Dessa forma, torna-se possível introduzir objetos virtuais em ambiente real através do *framework* de simulação de Realidade Mista. Essa ferramenta inclui um robô real em um processo de simulação utilizando o *software Player* integrado ao Simulador 3D Gazebo.

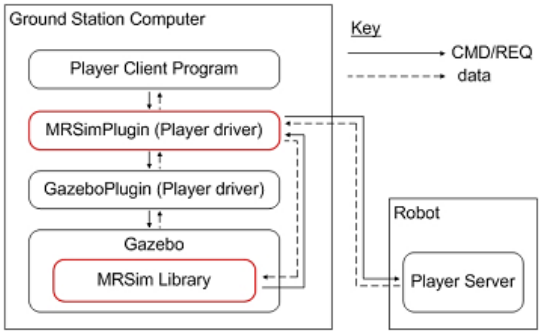
Em suma, o *framework* de simulação de Realidade Mista permite a integração de recursos virtuais para construção de um ambiente de simulação seguro, fornecendo um feedback visual em tempo real e facilitando a integração entre robôs e objetos reais durante a simulação.

O *framework* de simulação RM é composto por: cliente, servidor de simulação de RM, mundo virtual e o real. O cliente é o aplicativo que está sendo executado. O servidor de simulação RM lida com as solicitações e os comandos do cliente, que permite gerenciar o controle de dados produzidos pelos dois ambientes. O mundo real é essencialmente o

ambiente físico no qual tem lugar à experimentação. O mundo virtual é uma réplica do ambiente real, no qual os usuários podem introduzir objetos virtuais para incrementar e recriar diferentes cenários de testes para o cliente.

A ferramenta MRSim (*Mixed Reality Robot Simulation Toolkit*) foi desenvolvida e incorporada ao ambiente de simulação *Player*/Gazebo. O seu funcionamento independe do simulador e utiliza o seu próprio arquivo XML para configurar as propriedades de robôs e seus dispositivos. O MRSim desempenha o papel do servidor de simulação MR responsável por rastrear os estados dos dois mundos. O robô *Pionnier 2* executa as tarefas no "mundo real" e o ambiente virtual é criado e manipulado pelo simulador *Player*/Gazebo. O fluxo de dados no processo de simulação *Player/*Gazebo é mostrado na Figura. 33. O MRSim é formado por dois componentes: MRSimPlugin e a biblioteca MRSim.

Figura 33 - Integração do MRSim com os simuladores *Player/*Gazebo

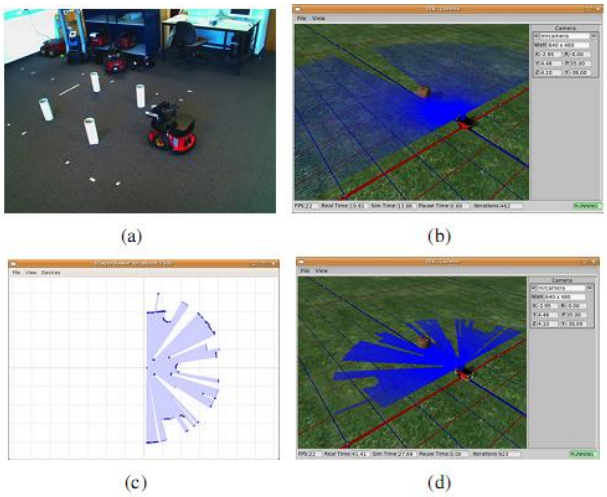


Fonte: (CHEN, MACDONALD e WUNSCHE, 2009)

O MRSimPlugin é responsável por combinar mundo real e dados de simulação para obter uma interação MR. A sua composição é formada por três etapas: Interceptar, Misturar e Publicar. A leitura real dos sensores a *laser* é aumentada para refletir os objetos virtuais agregados. A principio, o MRSimPlugin intercepta mensagens enviadas pelo programa cliente e os envia para Gazebo e o robô real. As leituras retornadas são misturadas, tendo os valores mínimos do intervalo real e virtual direcionados a cada ponto da varredura do laser. Os dados resultantes são então publicados. A Figura 34 de (a) a (d) mostra um exemplo com leituras dos sensores a laser de MR. Os dados do laser MR são exibidos usando o utilitário *Player built-in* e *Player Viewer*. O *player* cliente se conecta ao MRSimPlugin e requisita leituras do sensor. Os dados do sensor a laser aumentados são visualizados no Gazebo através da biblioteca MRSim, que por sua vez, solicita dados de MR do MRSimPlugin.

A biblioteca MRSim constrói o ambiente MR e controla as visualizações MR. Ele monitora a renderização da simulação, subsistemas físicos e faz com que mude diretamente para o mundo virtual criado pelo Gazebo. A interface da Realidade Aumentada (AR) e interface de Virtualidade Aumentada (AV) utilizam usando a biblioteca MRSim.

Figura 34 - (a) Mostra um robô real *Pioneer* detectando objetos com formato cilíndricos, (b) Um robô virtual realizando leituras com sensor à laser no simulador Gazebo, (c) Mostra o resultado da leitura dos sensores utilizando o simulador *Player*, (d) Mostra o laser de realidade mista visto pelo Gazebo



Fonte: (CHEN, MACDONALD e WUNSCHE, 2009)

De acordo com os autores, a interação entre o robô real e os objetos virtuais foi bem sucedida. O robô navegou no ambiente evitando obstáculos reais e virtuais detectados pelo sensor de *laser*, salvo por colisões com pequenos objetos virtuais que não puderam ser detectados pelo sensor.

A introdução de objetos virtuais em ambientes físicos permitiu não só a redução de custos com equipamentos caros, mas também forneceu uma rica disponibilidade de recursos. Isso é importante em ambientes onde alguns objetos são difíceis de serem recriados em experiências reais, como é o caso de ambientes com radiação ou a própria fumaça produzida por incêndios.

3.6 Comparativo

Este capítulo abordou alguns dos principais trabalhos relacionados envolvendo conhecimento teórico-prático sobre sistemas robóticos simulados e reais. O Quadro 1 apresenta um comparativo resumido entre os trabalhos relacionados.

Quadro 1 - Comparativo entre os trabalhos relacionados

Trabalho	Inclui aprendizagem	Ambiente de simulação	Hardware	Algoritmo / Técnica	Características
Xu, Mellmann e Burkhard	Sim, Aprendizagem de máquina	*SimSpark*	Nao (*Aldebaran Robotics)*	Algoritmo de Evolução	Usa o mesmo algoritmo no ambiente simulado e real
Kyriacou, Nehmzow, Iglesias e Billings	Não	*Player/Stage* e Webots	Magellan Pro	Polinômio de NARMAX	Reduz os erros captados pelos sensores e gera uma saída sem ruídos
Hassanzadeh, Madani e Badamchizadeh	Sim, Aprendizagem de máquina	Webots	Khepera II	Algoritmo de Otimização do Sapo pulando embaralhadamente	Permite que o robô móvel navegue através de obstáculos estáticos sem colidir com objetos em sua volta.
Chen, MacDonald e Wunsche	Não	*Player* / Gazebo	*Pioneer* 2	Framework de Realidade Mista	Interage com objetos reais e virtuais

Fonte: Autor

No primeiro trabalho os autores discutiram a respeito das principais diferenças/similaridades e o comportamento entre as plataformas reais e simuladas. Para experiência, foram usados o Robô Nao da *Aldebaran Robotics* e o simulador *SimSparks*. A utilização da aprendizagem de máquina agregada à implementação do algoritmo de evolução foi essencial para aprimorar as habilidades de andar, chutar e driblar. Por fim, foi possível executar um único código tanto na plataforma real quanto na simulada.

No segundo trabalho, foi apresentado um "modelo matemático transparente", chamado de polinômio de NARMAX. Segundo (KYRIACOU, NEHMZOW, *et al.*, 2008), com esse método tornou-se possível inserir múltiplas entradas, gerando uma única saída sem ruído. O hardware utilizado na experiência foi o Magellan Pro e o simulador *Webots* em conjuntos com os simuladores *Player/Stage*.

De acordo com (HASSANZADEH, MADANI e BADAMCHIZADEH, 2010), nem sempre é possível planejar a melhor trajetória utilizando métodos tradicionais. Dessa forma,

foi desenvolvido o algoritmo de otimização do sapo pulando embaralhado (SFL). Esse algoritmo permite que o robô móvel navegue através de obstáculos estáticos e encontre o caminho livre até chegar à posição alvo, sem colidir com objetos em sua volta. Para experiência, foi utilizado o robô Khepera II e o ambiente de simulação Webots. É possível perceber a presença da aprendizagem de máquina, quando os robôs detectam novos obstáculos, delimitam a área e comunicam a outros robôs o caminho que deve ser evitado.

O quarto trabalho apresenta um ambiente capaz de introduzir objetos virtuais como parte de um ambiente físico real através de um *framework* de simulação de realidade mista. A ideia visa integrar técnicas de realidade e virtualidade aumentada, preservando vantagens de ambos os lados. A introdução de objetos virtuais em ambientes físicos permite não só economizar custos com cenários de testes ou com equipamentos caros, mas também oferecer uma rica simulação de recursos com segurança. Foram utilizados nos experimentos o robô real *Pioneer 2* e o *software Player* integrado ao Simulador 3D Gazebo.

Apesar da utilização de técnicas, robôs reais e ambientes de simulação distintos, nenhum destes trabalhos demonstrou com clareza as particularidades que diferem as plataformas reais e simuladas. Assim, essa dissertação teve como objetivo geral, contribuir para a formalização de um modelo conceitual que evidencie o processo de transição, destacando as peculiaridades que carecem de uma maior atenção por parte dos desenvolvedores e projetistas.

Com base nesses fatores foi possível desenvolver um estudo baseando-se nas características de cada componente, dentre eles: sensores ultrassônicos (alcance, semiângulo de abertura, erro de reflexão), módulo de navegação, cinemática do robô (motores, rodas, deslocamento e velocidade de rotação) e por fim, a influência da bateria do comportamento dos robôs reais e simulados.

4. APLICAÇÃO DE ALGORITMO ANTICOLISÃO EM AMBIENTE REAL E SIMULADO

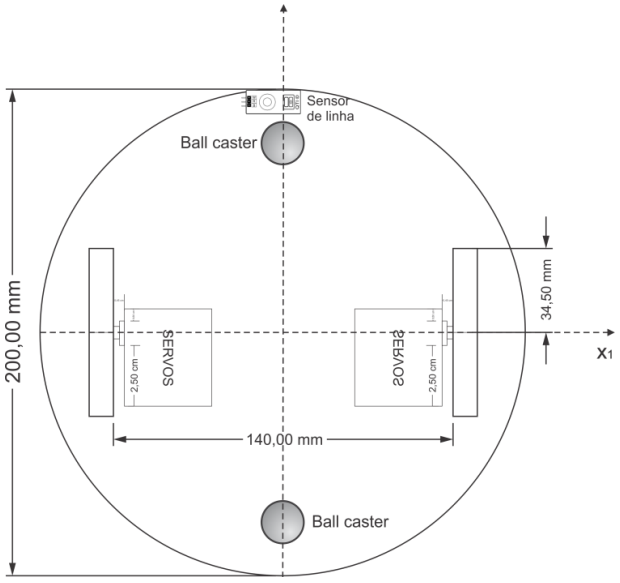
A proposta desse trabalho é implementar o algoritmo de anticolisão em robôs reais e simulados, de forma que possamos destacar as diferenças entre a sua implementação em ambiente simulado e real devido a sua versatilidade e simplicidade. Com essa proposta, espera-se que o trabalho possa contribuir com a construção do conhecimento para obtermos uma especificação inicial de um modelo de portabilidade que auxilie a transição de algoritmos em ambientes simulados para robôs reais.

Os principais aspectos que norteiam a pesquisa são: o processo de desenvolvimento do robô real, enfatizando a modelagem cinemática de postura, a construção e montagem do protótipo utilizando à ferramenta Arduino, a escolha do simulador para os experimentos e o desenvolvimento do algoritmo, incluindo as variáveis de estado e o próprio algoritmo de anticolisão.

4.1 Desenvolvimento do Robô real

O protótipo desenvolvido para essa experiência é um robô móvel de base circular de 200mm de diâmetro. Possui duas rodas convencionais fixas no chassi, centralizado, com eixos colineares, paralelos, com distância de 140mm uma da outra. Cada roda fixa mede 34,50 mm de raio. Além disso, o robô possui duas rodas do tipo *ballcaster* orientada fora do centro e servindo de apoio, sem nenhum controle, a qual não foi considerada para o seu modelo cinemático. O controle de direção é realizado através da variação da velocidade de rotação das rodas fixas (motrizes). A Figura 35 esquematiza o chassi do robô.

Figura 35 - Chassi do robô



Fonte: Autor

4.1.1 Modelagem cinemática de postura

Podemos considerar o protótipo como um robô autônomo móvel que se desloca de um ponto a outro sob uma superfície plana, sendo referenciado por um ponto fixo e centralizado, chamado de *P*. A modelagem do robô é feita a partir de um objeto circular locomovendo-se no espaço, como mostra a Figura 36. O espaço de configuração possuem três dimensões, uma de rotação e duas de translação. Podemos entender as coordenadas de postura como $\xi(x,y,\theta)$, onde x e y são coordenadas do ponto médio *P*, na base inercial, localizada entre as duas rodas motrizes e $\theta \in [0,2\pi]$ é o ângulo entre o eixo x da base inercial e o eixo principal do robô, que varia durante a locomoção. Conforme vimos na seção 2.6, o movimento é limitado pela equação (LATOMBE, 1996)

$$-\dot{x}sen\theta + \dot{y}cos\theta = 0$$

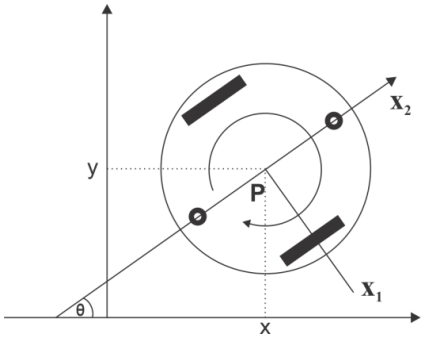
Observa-se que o robô possui uma restrição não-holonômica, isto é, move-se somente na direção normal ao eixo das rodas motrizes, atendendo as condições de rolamento puro e sem deslizamento.

O modelo cinemático de postura genérico para robôs móveis com duas rodas motrizes não-holonômicas é definida como:

$$\begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{Bmatrix} = \begin{bmatrix} -sen\theta & 0 \\ cos\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} \eta_1 \\ \eta_2 \end{Bmatrix}$$

De acordo o princípio do movimento cinemático de postura dos robôs móveis, o movimento pode ser descrito pela velocidade angular e linear da roda direita e esquerda, onde $|\eta_1| \leq V_{max}$ e $|\eta_2| \leq W_{max}$ são velocidades máximas angular e linear no ponto *P*.

Figura 36 - Coordenadas inerciais do robô móvel

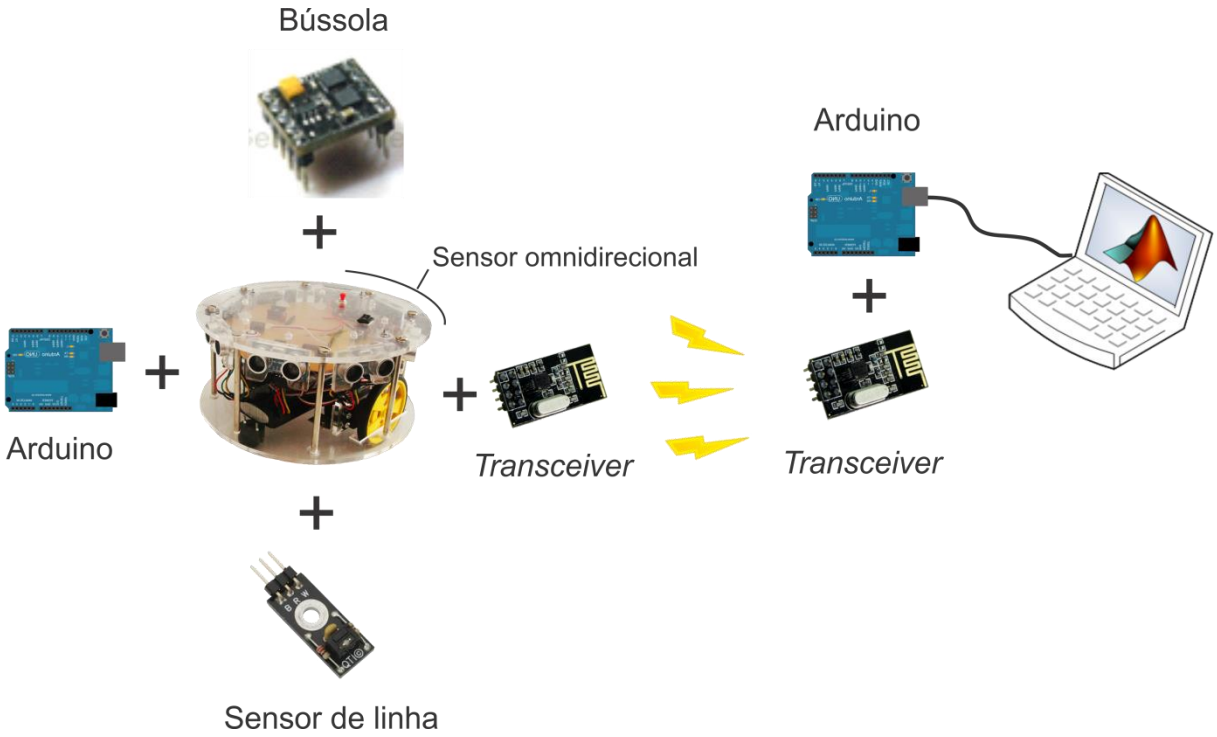


Fonte: Autor

4.1.2 Construção e montagem do robô real

Para construção do robô móvel foi utilizado duas placas Arduino, uma delas foi utilizada no robô móvel, juntamente com sensores, atuadores e módulos. A outra placa Arduino foi conectada a um computador e integrada ao MATLAB, que ficará responsável pela coleta dos dados e geração os gráficos. A Figura 37 mostra a arquitetura básica e a organização dos componentes.

Figura 37 – Arquitetura básica



Fonte: Autor

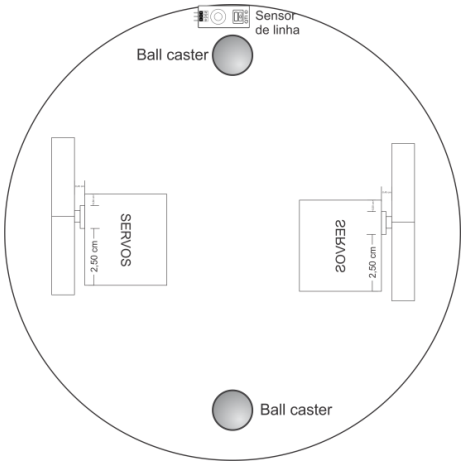
O ponto chave para construção do hardware foi o desenvolvimento do sensor de omnidirecional. Conforme foi mostrado na seção 2.4.3, o mesmo é composto por oito sensores de distância ultrassônicos justapostos igualmente às margens do chassi superior em uma circunferência de 200mm de diâmetro. Cada unidade possui uma faixa de detecção em torno de 45°, com a finalidade de obter uma cobertura completa de 360° (Figura 39).

O módulo de bússola ou HMC5883L apresenta um sistema de navegação bastante utilizado para alinhar o robô móvel em torno do seu eixo. Esse chip contém uma superfície com baixo campo magnético e uma interface digital para aplicações de baixo custo. O modelo atual oferece vantagens em relação a outras tecnologias de sensor magnético, pois foi projetado para medir a direção e a magnitude dos campos magnéticos da Terra a partir de miligraus a 8 graus, garantindo alta precisão e linearidade nos eixos.

O sensor de linha ou QTI utiliza um LED (*light emitting diode*) e um fototransistor infravermelho capaz de detectar a luz refletida pelo diodo, sem contato com os objetos. Esse dispositivo foi usado para auxiliar na percepção do robô autônomo móvel ao detectar as variações de tonalidade do solo, uma vez que a superfície foi previamente demarcada com linhas delimitando o tamanho e as dimensões dos obstáculos.

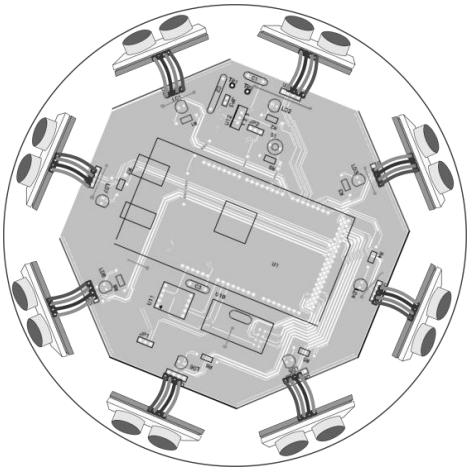
Para estabelecer a comunicação entre o robô e a unidade de coleta de dados foram utilizados "transceivers", traduzindo para o português podemos chamar de transceptor, e o seu significado é o resultado da junção das palavras **trans**missor e re**ceptor**, cuja função é combinar um transmissor e um receptor utilizando componentes de circuito comuns para ambos em um só dispositivo. Cada robô utiliza apenas um *transceiver* para emitir e receber dados. O uso da biblioteca "Mirf" foi imprescindível para estabelecer a comunicação entre os módulos. Através dessa biblioteca é possível configurar a "pinagem" de entrada/saída dos *transceivers* e definir quais os módulos que enviam e recebem os dados, levando em conta a simplicidade em relação às outras bibliotecas disponíveis.

Figura 38 - Visão inferior



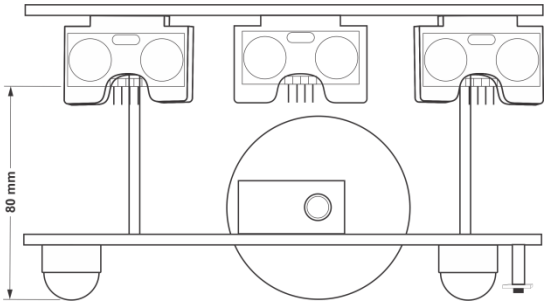
Fonte: Autor

Figura 39 - Chassi superior



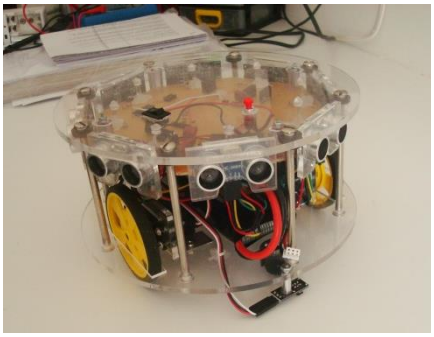
Fonte: Autor

Figura 40 - Visão lateral



Fonte: Autor

Figura 41 - Robô real



Fonte: Autor

A estrutura básica do robô é subdividida em dois níveis. Na parte inferior estão localizados os servos motores, bateria e as quatro rodas, sendo duas delas motrizes e duas de apoio. Na parte superior estão localizados o microcontrolador, o circuito impresso, os sensores e os módulos. A distância dos sensores ultrassônicos até superfície (chão) corresponde a uma altura de 80 mm, (i.e. o sensor omnidirecional consegue identificar um objeto com a altura mínima de 35 mm (BRAGA, 2013)).

A base do robô foi construída com acrílico transparente liso de 4mm de espessura. Para o corte foi utilizado uma mini retífica dremel com discos de corte de 15/16". O modelo de chassis circular foi desenhado propositalmente para que as extremidades do chassi não fiquem presas em caso de contato com o obstáculo. O software vetorial utilizado na criação dos modelos foi Corel Draw X5, (Figuras 38 a 40). A Figura 41 mostra o robô após a conclusão da montagem.

O Quadro 2 mostra a lista dos principais componentes utilizados na montagem do robô móvel.

Quadro 2 - Lista de componentes

Qtde	Descrição		Qtde	Descrição	
2		**Arduino Mega 2560**- É um microcontrolador baseado no ATmega1280. Possui 54 pinos de entradas/saídas digitais, 16 entradas analógicas, um oscilador de cristal de 16 MHz, uma conexão USB, uma entrada de alimentação e um botão de reset.	**2**		**Rodas de 8" para servo** - Projetada para encaixe nos servomotores.
8		**Sensores ultrassônicos** - são sensores de proximidade que trabalham livres de contato físico e detectando objetos à distâncias de até 8m.	**2**		***Ball caster*** – É um tipo de roda não motriz utilizada para dar sustentação ao robô. É formado por uma esfera e permite girar para todas as direções.
1		**Módulo bússola** – Detecta a mudança de orientação em torno do seu eixo.	**1**		Bateria de polímero de lítio ionizado com tensão de 7,4v, capacidade de 5000mAh.
2		**Transceiver** - É um dispositivo que combina um transmissor e um receptor utilizando componentes de circuito comuns para ambas as funções em um só aparelho.	**2**		O **iMAX B6-AC** é um carregador e balanceador de carga para baterias LiPo. / Li-ion / LiFe / NiMh / NiCad / Pb / Lead Acid

1		O **sensor de linha** ou **sensor ótico** permite a detecção de contraste de luminosidade. É composto por um LED emissor de infravermelho e um fototransistor.	**8**		**LEDs** – Os leds são utilizados como respostas dos sensores ultrassônicos, uma vez sem funcionar indica que o sensor não está sendo acionado ou com problema.
2		**Servomotor** – Componentes eletromecânico, diferente dos motores contínuos, recebem um sinal de controle; verificam a posição atual; atuam no sistema indo para a posição desejada.	**1**		**Botão L/D** – Botão responsável pelo acionamento do robô.

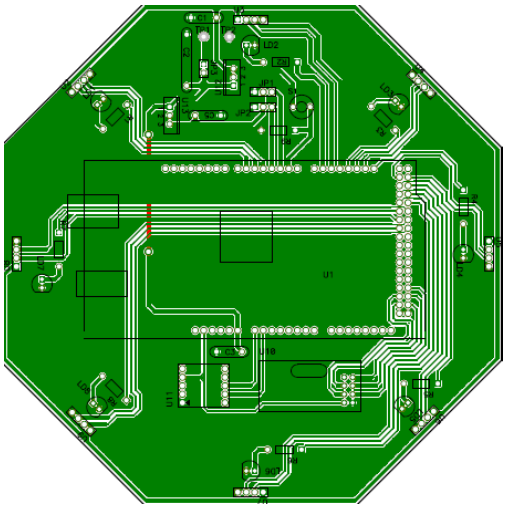
Fonte: Autor

Para atendermos a estas especificações foi necessário escolher as dimensões, motorização, sensores, microcontrolador, entre outros componentes, sempre com o objetivo de deixar uma plataforma flexível o suficiente para que possa ser reaproveitada em trabalhos futuros.

4.1.3 Placa de circuito impresso

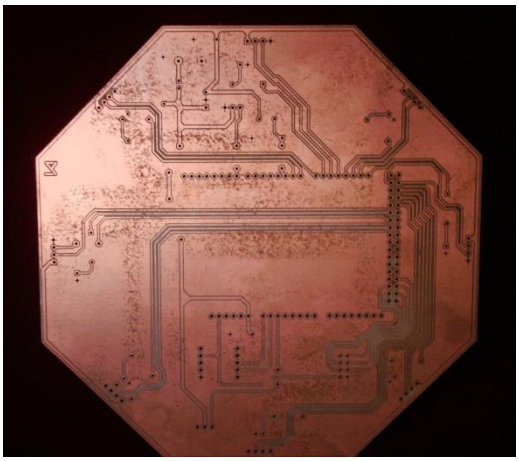
A placa de circuito impresso foi desenvolvida com o objetivo de manter os fios e componentes (sensores, módulos e peças eletrônicas) organizados, de forma a evitar qualquer falha durante o experimento.

Figura 42 - Esboço do circuito impresso



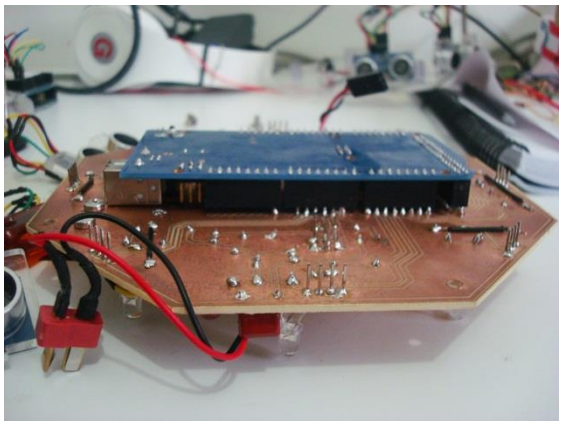
Fonte: Autor

Figura 43 - Placa de circuito após a impressão



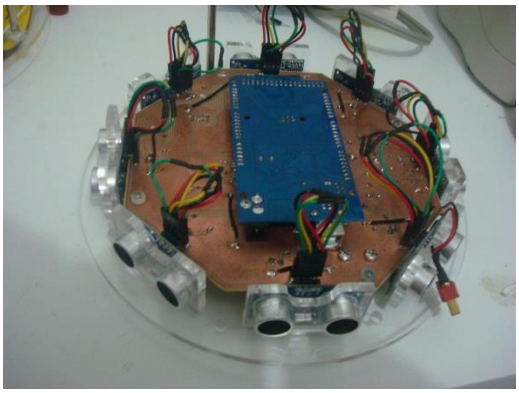
Fonte: Autor

Figura 44 - Soldagem dos componentes elétricos



Fonte: Autor

Figura 45 - Montagem dos componentes (sensores, atuadores e arduino)



Fonte: Autor

Para desenvolver a placa foi utilizado o software P-CAD 2006 na criação do *layout* do circuito, como mostra a Figura 42. Após o desenho, o *layout* foi encaminhado para o programa *boardmaster* que controla a fresadora LPKF ProtoMat S52 integrado ao software *circuitcam* que realiza o mapeamento do circuito. Depois da impressão da placa (Figura 43), iniciou-se o processo de soldagem manual das peças eletrônicas, entre eles: resistores, capacitores, barras de pino e LEDs, conforme ilustra a Figura 44. O último passo foi à montagem dos componentes (sensores, atuadores e arduino), como mostra a Figura 45.

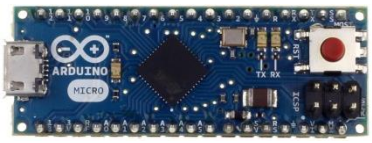
4.1.4 Plataforma Arduino

A plataforma escolhida para desenvolvimento e criação do protótipo foi o Arduino. Como mencionado na seção 2.3.1, o Arduino possui características que se destacam diante de outras plataformas existentes no mercado. Um dos principais fatores que determinaram essa escolha foi o valor acessível da placa e de seus componentes, além da simplicidade e da disponibilidade para compra das plataformas, tanto no Brasil, como no exterior.

Outro fator decisivo para a escolha do Arduino se deu pela quantidade de versões disponíveis no mercado e pelos diversos tamanhos de plataformas que variam entre as dimensões 17,8mm x 43,2 mm no Arduino Nano (Figura 46) e 101,98mm x 53,63mm no

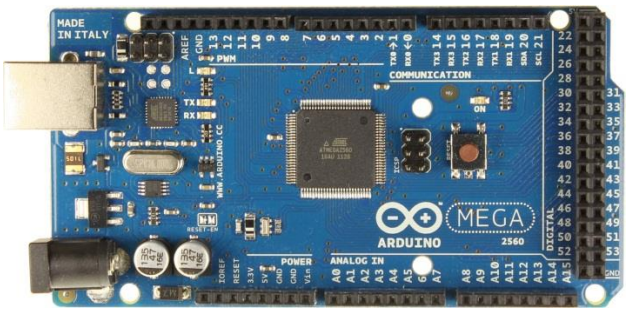
Arduino Mega (Figura 47), oferecendo ao usuário a possibilidade de optar pela versão que mais se adapta às particularidades e necessidades de cada projeto.

Figura 46 - Plataforma Arduino (Modelo Nano)



Fonte: Arduino (2012)

Figura 47 - Plataforma Arduino (Modelo MEGA)



Fonte: Arduino (2012)

Neste trabalho será utilizado o modelo MEGA que embarca um microcontrolador ATMega2560. Possui basicamente uma CPU (Unidade Central de Processamento), memórias SRAM de 8Kb, EEPROM de 4Kb e Flash de 256Kb, um oscilador de cristal de 16 MHz, 54 pinos digitais de entrada e saída, 16 pinos analógicos, 14 pinos PWM (pulse width module), 4 UARTs (portas seriais de hardware), conversor de sinal analógico/digital e entre outros recursos.

O Quadro 3 mostra um comparativo resumido das principais plataformas de robótica disponíveis no mercado.

Quadro 3 - Comparativo entre *BeagleBoard*, Raspberry Pi e Arduino

	Dispositivos		
Plataformas	**Beagleboard**	**Raspberry Pi**	**Arduino**
Modelo	Rev. C4	Model-B	Mega
Software			
Sistema Operacional	Android, Linux, Windows CE, RISC OS	Linux, RISC OS	-
Ambientes de Desenvolvimento	Eclipse, Android ADK, Screktch	OpenEmbedded, QEMU, Stratchbox, Eclipse	Arduino IDE, Eclipse
Linguagem de Programação	Python, C	Python, C, BASIC	Wiring (C/C++)
Arquitetura	32 bits	32 bits	8 bits

Hardware			
(Micro)Processador / Controlador	TI DM3730 (ARM)	BCM 2835 (ARM)	ATMega2560
Clock	720 Mhz	700 Mhz	16 Mhz
RAM	256 MB	256 MB	8 KB
ROM	256MB Flash	SD	256 KB
Portas E/S	22	8	54 Entrada/Saída
USB	USB 2.0	2 portas USB 2.0	USB 2.0
Áudio	Stereo In/Out	Stereo Out, In com microfone	-
Vídeo	DVI-D, S-Vídeo	HDMI / NTSC ou PAL	-
Itens diversos	SD/MMC, RS-232, ITAG, USB, OTG, LCD	SD / Ethernet 10/100 e JTAG	Diversos *shields* disponíveis para aumentar a capacidade
Dimensões	76.2 mm × 76.2 mm	85.60mm × 53.98mm	101.98mm x 53.63mm
Peso	37g	45g	40g
Média de custo			
	$89,00	$ 35,00	$ 29,00

Fonte: Autor

De acordo com o Quadro 3, podemos notar a similaridade entre as plataformas *Beagleboard* e RaspberryPi. Portanto, iremos focar na análise entre o Arduino e o Raspeberry Pi por representar diferenças acentuadas.

O Raspberry Pi é um microcomputador do tipo “Single-board computer” baseado no microprocessador ARM. A plataforma possui dimensões de um cartão de crédito que pode ser conectado em TV/Monitor ou teclado. O Raspberry Pi é um computador pequeno que pode ser utilizado para realizar tarefas como processadores de texto, vídeo, áudio e jogos. O principal objetivo da plataforma é alcançar interessados em aprender programação, bem como conceitos básicos de ciência da computação.

Já o Arduino é uma plataforma que já está há muito tempo consolidada no mercado, possui um microcontrolador que consome pouca energia e proporciona ao usuário o controle completo do hardware. Possui IDE própria que possibilita escrever pequenos programas que fazem interface com vários sensores, atuadores, módulos e outros microcontroladores. Por outro lado, o Raspberry foi projetado para atuar em alto nível computacional. O seu hardware integra funções (de internet, vídeo e áudio), possui quantidade elevada de memória RAM e

espaço de armazenamento através de cartões SD. Resumidamente, o Raspberry Pi contém maior capacidade computacional, enquanto o Arduino proporciona um fácil aprendizado de eletrônica e computação física.

Após analisarmos as duas principais plataformas (Raspberry Pi e Arduino), podemos considerar que esses dois dispositivos não competem entre si, pois os mesmos possuem finalidades e propósitos diferentes. Logo, com base no trabalho proposto, o Arduino Mega se destaca pelo grande número de portas de entrada e saída, baixo consumo de energia e uma grande variedade de expansões que aumentam a sua capacidade, aproveitando, principalmente, de boa parte dos recursos que a tecnologia oferece.

4.1.5 Características do robô

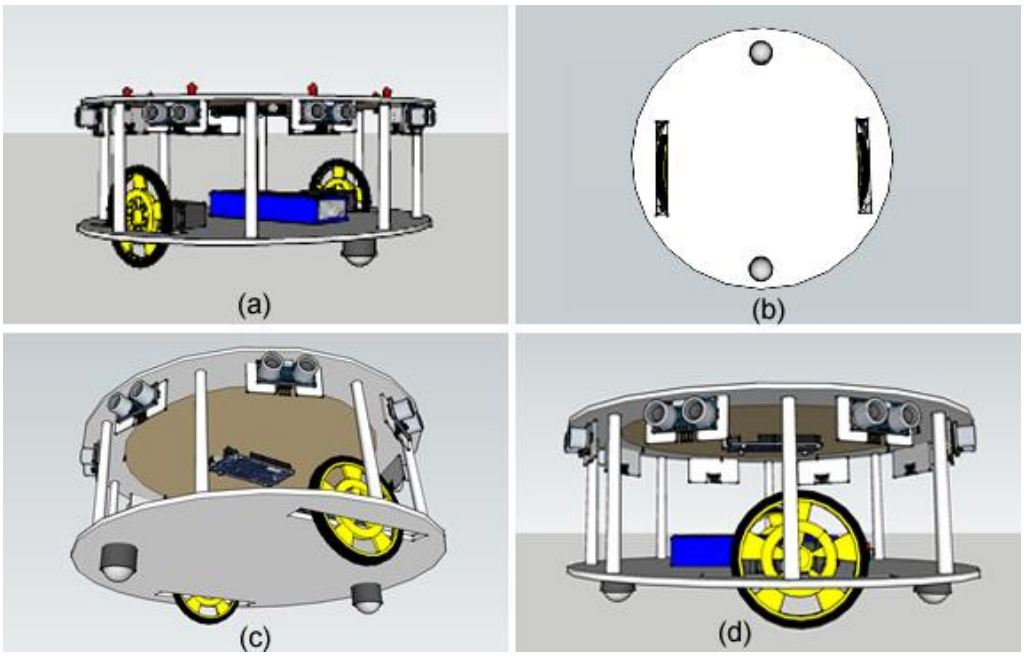
O formato circular do chassi e a presença de oito sensores ultrassônicos distribuídos em torno de uma circunferência de 20 cm diâmetro permitiu uma cobertura completa dos objetos e obstáculos em sua volta. O robô possui também duas rodas de apoio omnidirecionais que facilitam a locomoção em torno dos obstáculos. A placa de circuito impresso foi confeccionada com a finalidade de organizar os fios e evitar falhas durante os testes. E por fim, a escolha da plataforma Arduino como alternativa de baixo consumo de energia, possibilitando um aumento da capacidade devido a uma ampla variedade de expansões disponíveis. Por tanto, podemos concluir que as características básicas do robô autônomo móvel desenvolvido se encaixam com o padrão estabelecido para o experimento.

4.2 Simulador

A plataforma de simulação escolhida para ser utilizada no trabalho foi o *Unity3D*. Como citado na seção 2.5.1, apesar do ambiente ser voltado para criação de jogos, o mesmo oferece recursos e funcionalidades suficientes para simulação de robôs. Uma das principais vantagens do motor de jogo *Unity3D* é a possibilidade de importar modelagens criadas pelo software Google *SketchUp*[6], o que torna o processo de criação e modelagem 3D rápido e simples, ao contrário de outros softwares de simulação, que exigem modelagens tridimensionais feitas apenas por ferramentas CAD, que apresentam-se de forma complexa e lenta.

[6] *SketchUp* é um software proprietário, pertencente ao grupo Google, utilizado para a criação de modelos em 3D no computador. A sua principal vantagem é a facilidade de modelagem de estudos de formas e volumes tridimensionais.

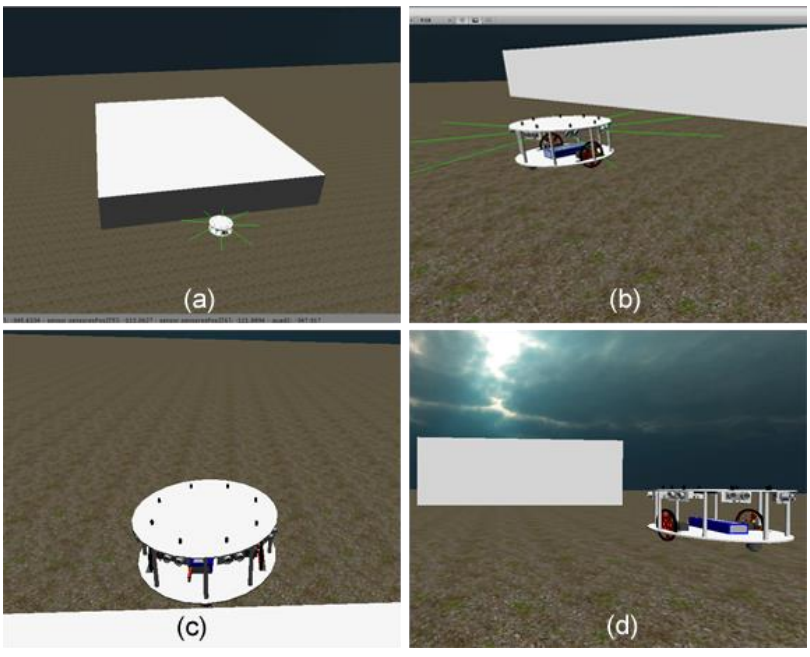
Figura 48 - Modelagens feitas no Google *SketchUp*



Fonte: Autor

A Figura 48 de (a) a (d), ilustra as diversas faces do modelo de robôs desenvolvido através do software Google *SketchUp*, e em seguida, importado para o simulador *Unity3D*. A partir da familiarização com o ambiente de desenvolvimento o passo seguinte foi identificar os componentes que determinam o desenvolvimento da simulação. Posteriormente foram implementadas as primeiras versões do algoritmo de anticolisão.

Figura 49 - Modelagem 3D dos robôs utilizando o Google *SketchUp*



Fonte: Autor

Apesar do *Unity3D* ser um motor de jogo versátil, o mesmo foi desenvolvido para atuar em ocasiões que envolvem múltiplos objetos em cena. Sua estrutura permite executar unidades gráficas de processamento apresentando em um melhor desempenho, além de abstrair os cálculos complexos do desenvolvedor aumentando sua produtividade. O *Unity3D* possui internamente um sistema de colisão que envolve formas básicas como cubos, cápsulas e esferas, sendo fundamentais para composição do cenário. A Figura 49 de (a) a (d), mostra a formação de obstáculos através de estruturas básicas como o cubo.

Quadro 4 - Comparativo entre Unity3D e V-REP

	Unity 3D	V-REP
Modelagem	3D	3D
Liguagem de programação	C++, *Javascript* e *Boo*	C/C++, *Python* e LUA
Importar modelos do Google SketchUp	Sim	Sim
Licença	Versão gratuita e proprietária (U$1500,00)	Versão gratuita (educacional) e proprietária (comercial)
Nível de aprendizagem	Fácil	Moderado

Fonte: Autor

O Quadro 4 mostra o comparativo entre os simuladores *Unity3D* e V-REP, bem como características e funcionalidades semelhantes. Porém, o fator preponderante para a escolha da o *Unity3D* deve-se aos seguintes elementos: facilidade de uso, importar modelos do Google *SketchUp* e a possibilidade de programar nas linguagens C++, *Javascript* e *Boo*. Desta forma, acredita-se que a IDE e motor de jogo *Unity3D* ofereça as condições mais adequadas para o desenvolvimento do trabalho proposto.

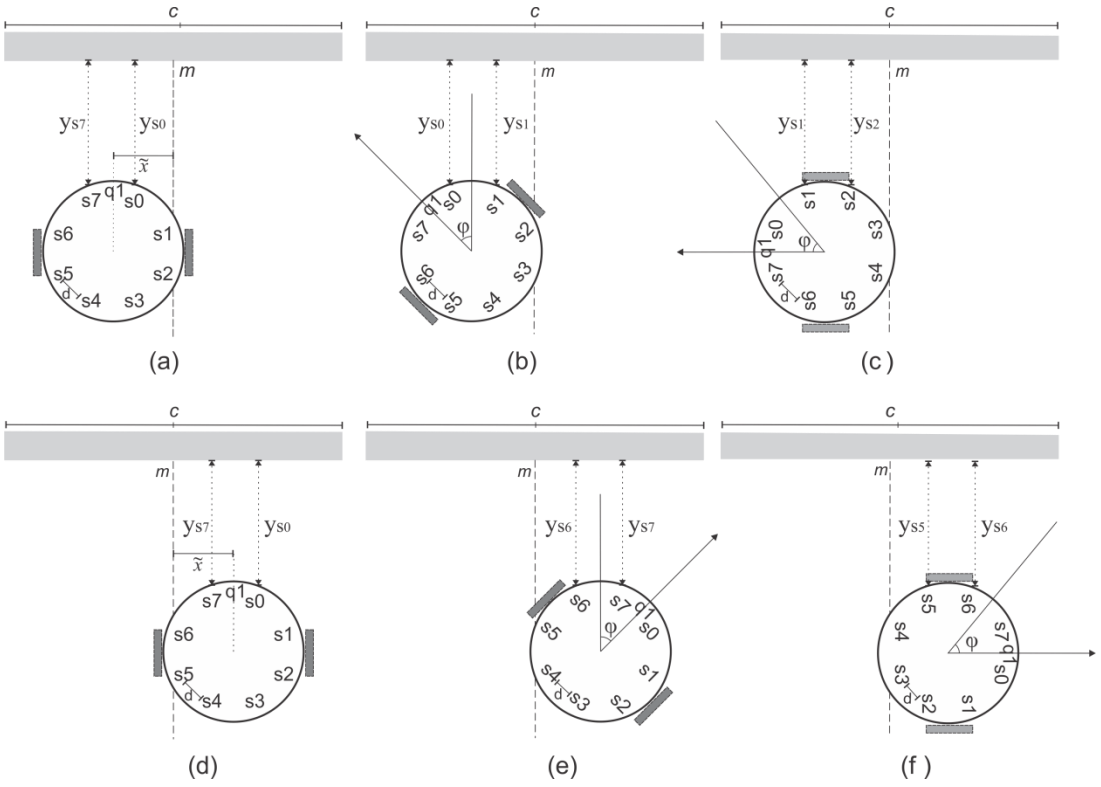
4.3 Algoritmo

Para que seja possível desenvolver qualquer algoritmo para sistemas robóticos, faz-se necessário o entendimento de alguns conceitos como: tipo de sensores utilizados, a posição que o sensor está disposto no robô e as variáveis de estado. A seguir é visto o método obtido para leitura dos sensores através das variáveis de estado. Em seguida, será mostrado o algoritmo de anticolisão desenvolvido para o trabalho.

4.3.1 Variáveis de estado

Como foi visto anteriormente, a velocidade linear do robô é declarada como constante e a velocidade angular é considerada variável de controle. São variáveis de estado do sistema a função do erro de posição do robô ($\tilde{x}$) e o erro de orientação em relação ao obstáculo (φ). Para que o robô e o conjunto de sensores funcionem de maneira coerente, é necessário que os sensores recebam e tratem os valores de estado a cada ciclo de controle. Neste trabalho foi utilizada uma forma de calcular estas variáveis de estado através da junção dos sensores ultrassônicos (FREIRE, 2002) e sensor de linha.

Figura 50 - Posição dos sensores ultrassônicos em relação aos obstáculos



Fonte: Autor

A Figura 50 mostra um robô móvel à frente de um obstáculo retangular, bem como a posição dos sensores ultrassônicos distribuídos em volta de sua circunferência. A princípio o robô decidirá para qual lado do obstáculo deverá desviar. Para isso, o robô utiliza o sensor de luz (*q1*) para detectar a sua posição em relação à linha média (*m*) do obstáculo,

$$m = \frac{c}{2}$$

onde *m* é a mediana e *c* corresponde ao comprimento da maior aresta do retângulo.

Os sensores ultrassônicos foram incluídos no cálculo das variáveis de estado. Estes sonares são responsáveis por obter as medidas entre o robô e o obstáculo. A distância entre o robô e a parede é calculada como:

$$\begin{cases} d_{esq} = \frac{y_{s7} + y_{s0}}{2} + \frac{y_{s0} + y_{s1}}{2} + \frac{y_{s1} + y_{s2}}{2} \\ d_{dir} = \frac{y_{s7} + y_{s0}}{2} + \frac{y_{s7} + y_{s6}}{2} + \frac{y_{s6} + y_{s5}}{2} \end{cases}$$

onde *ys0, ys1, ys2, ys5, ys6* e *ys7* são distâncias correspondentes a cada sensor e o obstáculo.

A distância do centro robô e a linha média que separa a metade do obstáculo ($\tilde{x}$) define para qual lado do obstáculo o robô deve percorrer. Seu valor pode ser obtido através da equação.

$$\tilde{x} = \frac{d_{dir} - d_{esq}}{2}$$

Para determinar a outra variável de estado, o erro de orientação em relação ao obstáculo, é preciso conhecer a diferença entre as medidas dos sensores frontais,

$$dif = (y_{si} - y_{sj})$$

Assim, o ângulo φ mostrado na figura é dado por,

$$\varphi = sen^{-1}\left(\frac{dif}{d}\right)$$

onde *d* é a distância entre os sensores ultrassônicos.

As informações providas pelos sensores podem ocasionar medidas com erros, dependendo de circunstâncias com as quais estão envolvidas, como: incidência de luz solar nos sensores de luz ou nos sensores ultrassônicos quando o robô possui uma inclinação grande em relação ao obstáculo. No caso dos sonares, se o plano estiver em um posicionamento maior que o semiângulo do módulo emissor, o sinal será refletido para longe do sonar e o objeto não será detectado pelo transdutor (Ver Figura 10). No robô proposto neste trabalho, são utilizados sensores ultrassônicos HC-SR04. Para este sensor, o ângulo de inclinação máxima para que as medidas fornecidas por ele sejam confiáveis é de aproximadamente 15°, o qual é chamado de semiângulo de abertura.

4.3.2 Algoritmo de anticolisão

Com base na análise das variáveis de estado, foi possível desenvolver o algoritmo de anticolisão baseando-se nas características de cada sensor, como: alcance, semiângulo de abertura e mediante o comportamento do conjunto de sensores ao se depararem com os obstáculos. O modelo do algoritmo de anticolisão possibilita uma interação coerente do robô e o os obstáculos em sua volta, uma vez que os RAMs possam tomar decisões, (i.e., os robôs calculam a menor distância entre a metade do obstáculo até a sua aresta, de forma que possam navegar entre obstáculos pelo menor trajeto possível). No ambiente real os sensores de linha são responsáveis por prover informações para tomada de decisão e discernir o menor lado do obstáculo a ser contornado.

Na simulação esse procedimento não é necessário, pois o próprio sistema simulado calcula internamente a posição do obstáculo em relação ao robô, omitindo o uso de sensores que detectam a divisão mediana de cada obstáculo. O sistema de navegação (bússola) foi instalado para amenizar a fuga de rota. Foi utilizado para implementação do algoritmo de anticolisão a linguagem C++, tanto no simulador quanto na IDE do Arduino.

Segue abaixo o pseudo-código do algoritmo de anticolisão:

Algoritmo 1 - Estado: Seguir em frente

Avança ao longo do caminho à frente, a menos que:
Se o obstáculo bloqueia o caminho adiante, então
    Execute o estado CONTORNAR OBSTÁTULO
Senão
    Seguir em frente
Fim

Algoritmo 2 – Estado: Contornar obstáculo

Move-se em torno do obstáculo no sentido horário, quando:
Se (S7 < DistMin ∧ S0 < DistMin) ∨ (S0 < DistMin ∧ S1 < DistMin), então
    Move-se para esquerda
Então se S1 > DistMin ∧ S2 < DistMin, então
    Move-se para direita
Fim Se

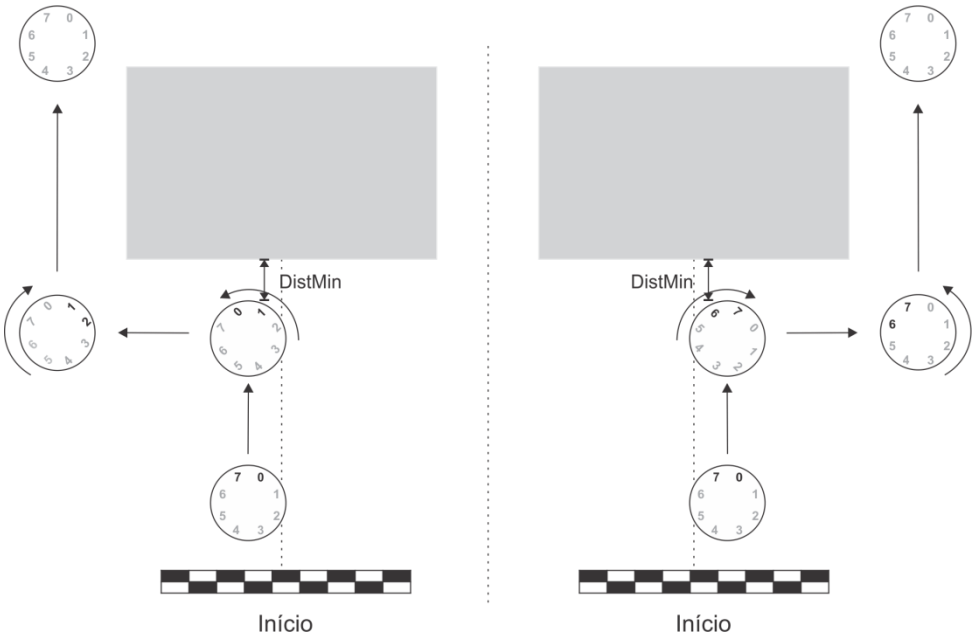
Move-se em torno do obstáculo no sentido anti-horário, quando:
decisão = 1 - decisão
Se (S7 < DistMin ∧ S0 < DistMin) ∨ (S7 < DistMin ∧ S6 < DistMin) ∧ decisão == 1, então
    Move-se para direita
Então se (S6 > DistMin ∧ S5 < DistMin) ∧ decisão == 1, então
    Move-se para esquerda
Fim Se

Segue abaixo o significado das siglas utilizadas no algoritmo de anticolisão:

- **DistMin** = Menor distância entre o robô e o obstáculo
- **decisão** = Decide qual o menor lado do obstáculo que o robô deverá desviar
- **S0 =** Sensor 1
- **S1 =** Sensor 2
- **S2** = Sensor 3
- **S5 =** Sensor 6
- **S6 =** Sensor 7
- **S7** = Sensor 8

A Figura 51 mostra a sequência de passos que o robô levou para percorrer todo o trajeto em um obstáculo de base retangular.

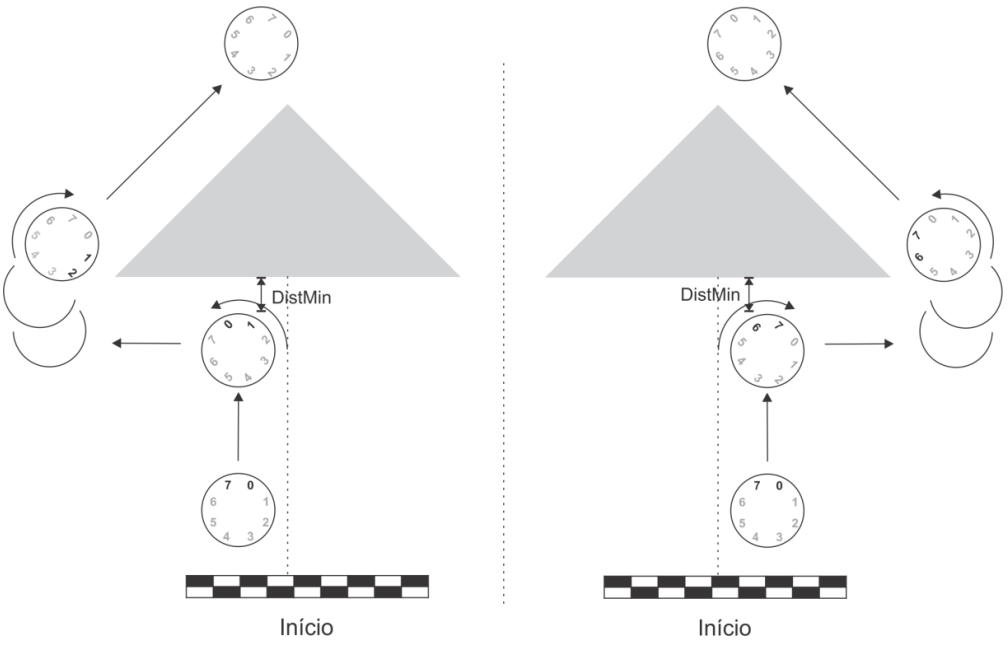
Figura 51 - Algoritmo de anticolisão em obstáculos com base retangular



Fonte: Autor

A Figura 52 ilustra um obstáculo com base triangular. Observe que os ângulos internos dos vértices correspondem a 60°, que representa o limite para que o robô não perca contato com a parede do obstáculo.

Figura 52 – Algoritmo de anticolisão em obstáculos com base triangular



Fonte: Autor

O algoritmo de anticolisão mostrou-se eficiente ao ser utilizado em obstáculos com formato de prisma retangular e triangular, pois o semiângulo de abertura dos sensores ultrassônicos abrange o intervalo angular delimitado em cada aresta do obstáculo, visto que pelo menos um dos sensores deve estar em contato com a parede. O mesmo não pode ser dito em formatos cilíndricos, pois dependendo do tamanho da base os sensores frontais não conseguem perceber simultaneamente a barreira à frente. Para correção desse problema, deve-se reposicionar os sensores frontais ou inserir mais sensores em sua volta.

4.4 Resumo

Nesse capítulo foi apresentado o processo de construção do robô real através da modelagem cinemática de postura de robôs não-holonômicos e dos componentes sensoriais e módulos de comunicação utilizados para compor o protótipo. Foi visto também o procedimento de confecção da placa de circuito impresso e a escolha da plataforma Arduino como uma alternativa de baixo custo e que oferece a possibilidade de expansão das capacidades do robô com outros recursos que a tecnologia oferece. Foram discutidos os motivos pela preferência do simulador e motor de jogo *Unity3D* enaltecendo vantagens como facilidade de uso, possibilidade de programar em linguagem C++ e importação de modelos feitos no Google *SketchUp*. E por fim, foram abordados conceitos gerais das variáveis de estado dos elementos sensoriais e a descrição do algoritmo de anticolisão aplicado em ambiente real e simulado.

5. EXPERIMENTOS

Os experimentos para validação desta pesquisa foram realizados através de testes utilizando um robô simulado na plataforma *Unity3D* e um robô real equipado com duas placas Arduino, sensores, módulos, atuadores e componentes eletrônicos. Os testes com robô real foram feitos no auditório da UFPB Virtual. O objetivo foi colher os dados e compará-los com os experimentos realizados no simulador *Unity3D*.

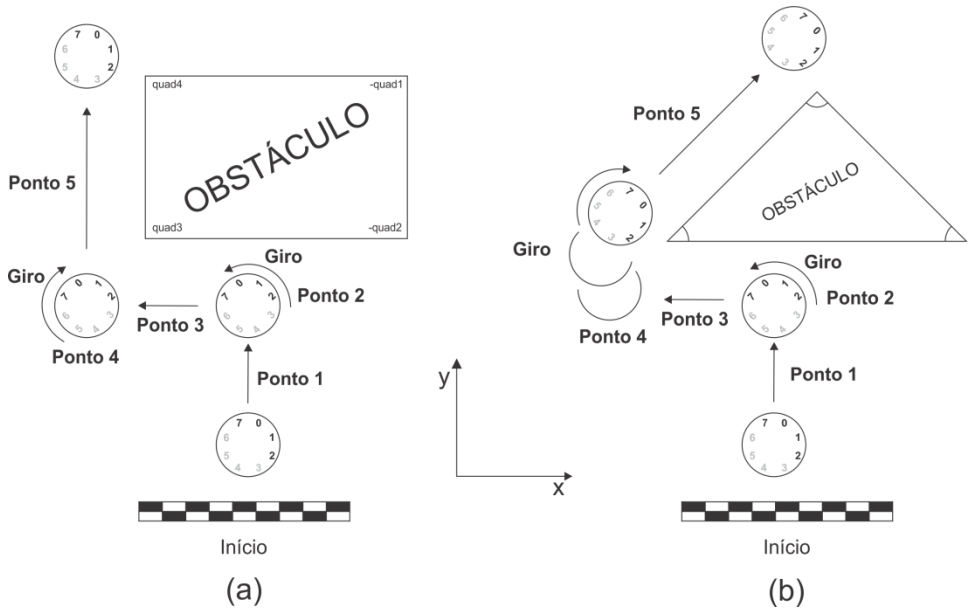
O objetivo da validação é mostrar as principais características responsáveis pela diferenciação dos resultados nas plataformas reais e simuladas. Com isso, pretende-se contribuir para a especificação futura de um modelo de portabilidade que auxilie na transição de algoritmos simulados para robôs reais. De fato, os simuladores possuem características particulares que, ao se transferir o código fonte para o ambiente real, este código torna-se suscetível à interferência de fatores externos. Por exemplo, as entradas sensoriais em robôs

reais são bastante ruidosas e na maioria das vezes imprecisas, causando perda e distorção de informações.

Durante os experimentos foram realizados quatro testes implementando o algoritmo de anticolisão como estudo de casos. O primeiro e segundo testes consistiram em analisar os sensores omnidirecionais e o sistema de navegação em ambientes reais e simulados diante de obstáculos com formato prisma retangular e triangular respectivamente. No terceiro teste foi realizado um experimento simples para obtenção das diferenças de velocidade entre os atuadores (motores). O último teste avaliou o desempenho dos robôs reais mediante as intervenções de fatores externos. Para isso, foram avaliadas as diferenças de velocidades entre os motores mediante a modificação da bateria em diversos níveis de carga. Dessa forma, espera-se que seja possível obter resultados adequados em relação a uma futura transição de algoritmos, diminuindo a prevalência de erros na plataforma real.

Para facilitar o entendimento, os dois primeiros testes foram divididos em cinco pontos cada. A Figura 53 mostra o esquema que servirá de auxílio para leitura dos gráficos.

Figura 53 – Esquema de auxílio para leitura dos gráficos



Fonte: Autor

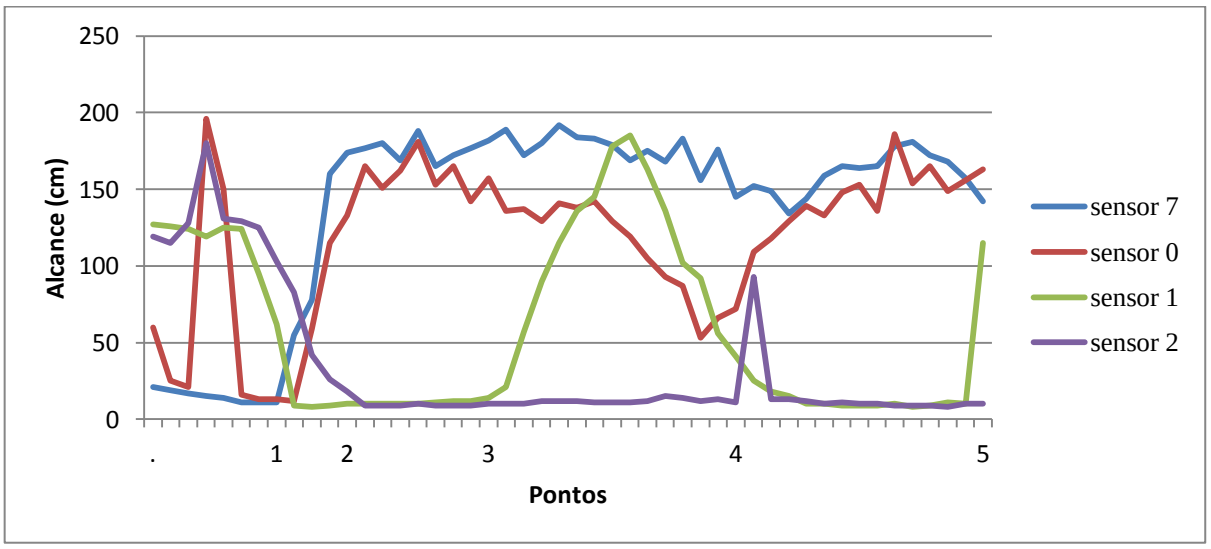
Para os testes 1 e 2 foram adotados um modelo de teste simplificado onde o robô percorrerá apenas no sentido horário. Além disso, os registros foram reduzidos para duas amostras por segundo, (i.e. as amostras foram registradas a cada 500ms) devido à grande quantidade de registros gerado em *logs*. O tempo percorrido entre o início e o fim do teste, em

cada etapa, durou aproximadamente 27 segundos para os robôs reais e 25 segundos para os robôs virtuais.

5.1 Teste 1 – Desvio de obstáculos com sensores omnidirecionais

O objetivo desse teste é analisar o comportamento dos sensores omnidirecionais diante de obstáculos com base retangular. O algoritmo de anticolisão foi implementado nos robôs reais e simulados com a finalidade de propor que os robôs realizem a mesma tarefa (i.e. detectar obstáculos e desviá-lo). Nesse teste foram avaliadas as principais diferenças que envolvem os meios, levando em consideração a alta taxa de erro produzido pelos sensores omnidirecionais (ultrassônicos).

Gráfico 1 - Teste com o sensor omnidirecional real em obstáculo retangular

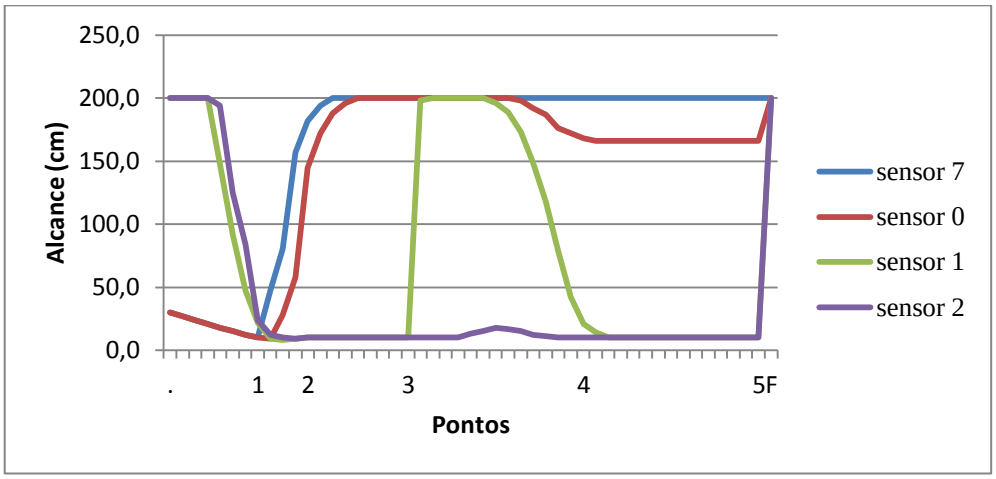


Fonte: Autor

O Gráfico 1 mostra o alcance de cada sensor real utilizado no experimento, visto que o alcance máximo que um sensor ultrassônico pode chegar é de 200cm. Nesse mesmo gráfico é possível perceber que diversos pontos sofreram interferências provenientes de ações externas.

Podemos observar que o estado mais agravante está entre os pontos 0 e 1 com o sensor0. O sonar emitiu um pulso de onda ultrassônica e não obteve resposta em tempo hábil, com isso o sensor atribuiu o valor de alcance com amplitude máxima. A situação semelhante ocorreu no sensor2 entre os pontos 4 e 5, pois houve atraso no recebimento do sinal e o valor foi registrado fora dos padrões.

Gráfico 2 - Teste com o sensor omnidirecional simulado em obstáculo retangular



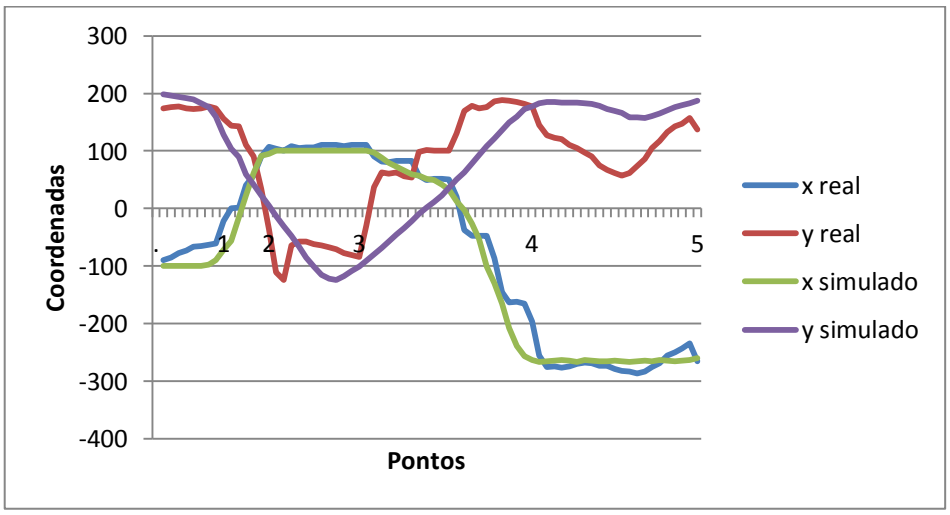
Fonte: Autor

Diferentemente, os sensores omnidirecionais simulados apresentam linhas suavizadas. De acordo com o Gráfico 2 percebe-se que não houve interferências de fatores externos. A ausência de superfície para rebater o sinal emitido fez com que o simulador registrasse a amplitude máxima que é 200cm.

5.2 Teste 2 – Desvio de obstáculos com sistema de navegação (bússola)

Esse teste é similar ao anterior, porém foi avaliado o comportamento do sistema de navegação diante de obstáculos com base triangular. A princípio foi gerado um *log* com os registros obtidos sem utilizar o sistema de navegação e em seguida, o sistema de navegação foi ativado e as diferenças de coordenadas foram confrontadas com os gráficos obtidos pelo simulador.

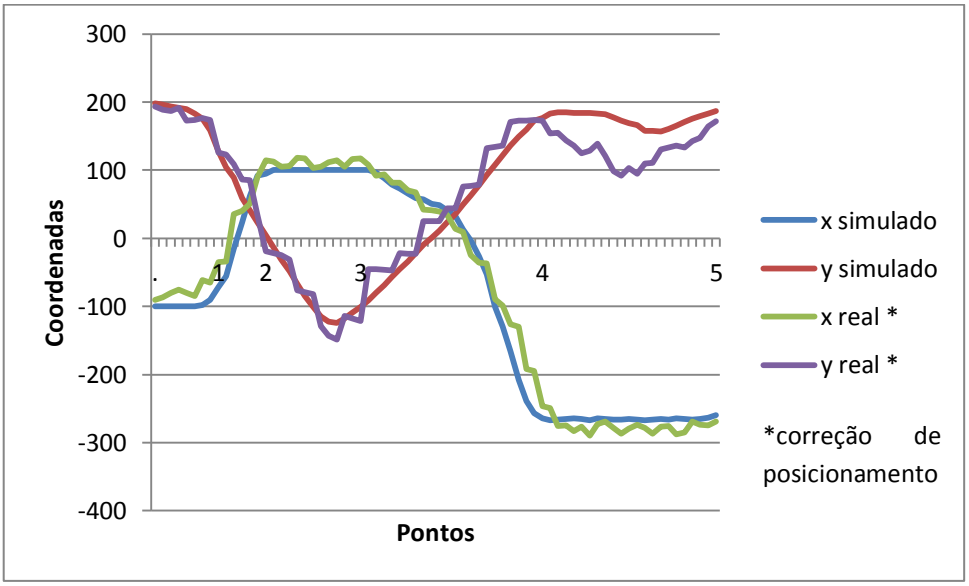
Gráfico 3 - Teste com sistema de navegação real e simulado em obstáculos triangulares



Fonte: Autor

O Gráfico 3 mostra as coordenadas de posicionamento em função dos pontos chaves para desvio de obstáculos (Ver Figura 53b). O eixo x e y real representa a posição do robô real no espaço físico, enquanto os eixos x e y simulado mostram as coordenadas de posicionamento do robô simulado. Nesse mesmo gráfico é possível observar os trajetos realizados pelos robôs reais e simulados sem a presença do sistema de navegação (i.e. a bússola serviu apenas para fornecer parâmetros para comparação). Dessa forma, apenas os sensores omnidirecionais ficaram responsáveis por guiar o robô em torno dos obstáculos. No entanto, percebe-se que existe uma grande disparidade entre o trajeto dos robôs reais e simulados, isso porque os erros provenientes dos sensores omnidirecionais ainda persistem, juntamente com a imprecisão dos servos. Por esse motivo, surge a necessidade de um sistema de correção de posicionamento.

Gráfico 4 - Teste com sistema de navegação real e simulado em obstáculos triangulares com correção de posicionamento



Fonte: Autor

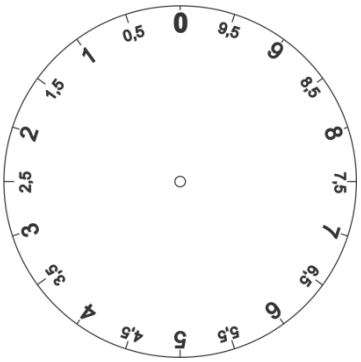
O Gráfico 4 apresenta as coordenadas do robô com correção de posicionamento em função dos pontos chaves para desvio de obstáculos (Ver Figura 53b). Para esse teste o sistema de navegação foi utilizado com a finalidade de amenizar a fuga de rota e por consequência disso, resultou no trajeto com aparência ondulada, pois a cada leitura da bússola o robô tende a se manter sempre em linha reta, provocando o efeito “zig-zag”. Vale a pena ressaltar que, assim como os sensores omnidirecionais, a própria bússola contém erros de

coordenada. Porém, esses erros ocorrem em uma amplitude menor que os sonares, fazendo com que o sistema de navegação o sobreponha.

5.3 Teste 3 – Atuadores

Este teste baseia-se na proposta sugerida pela empresa (PARALLAX, 2012) para obter a velocidade de rotação por minuto (RPM) e a direção dos servos motores para os valores de pulso entre 1300ms e 1700ms com intervalos de 50ms. Estas medições de velocidade ajudaram a obter um modelo prático que define as diferenças de rotação entre os motores.

Figura 54 - Medidor de velocidade (RPM)



Fonte: Autor

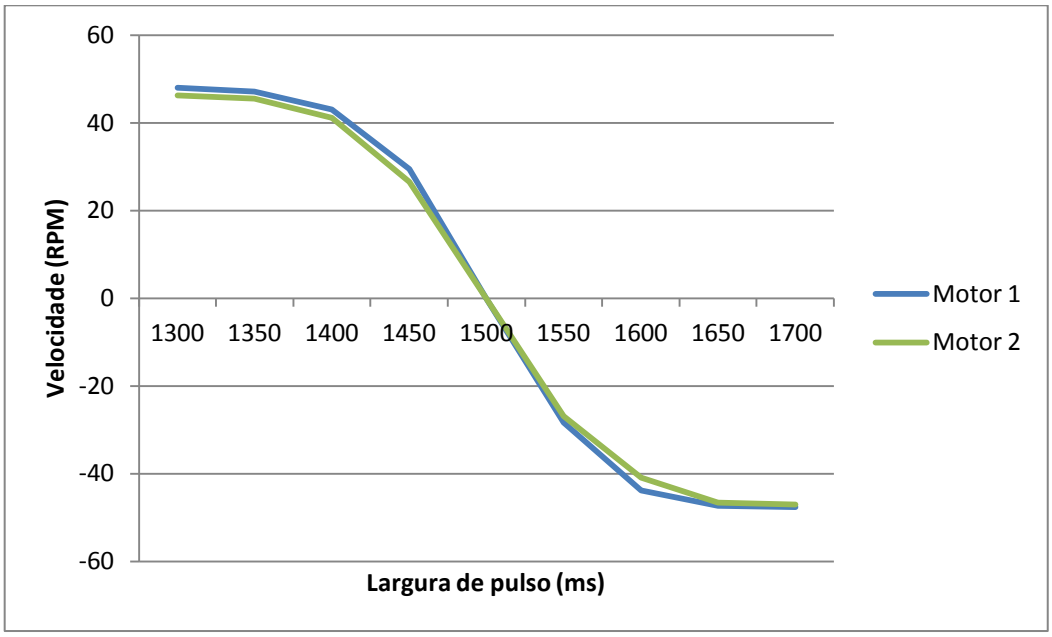
Inicialmente foi desenvolvido um medidor de RPM (Ver Figura 54) e fixado às rodas do robô. Em seguida, o robô foi posto em linha reta e deslocado de um ponto a outro no intervalo de 6 segundos. Durante esse tempo contabilizou-se o número de voltas completas em cada uma das rodas. Os valores obtidos foram multiplicados por 10 para obter as rotações de cada motor no intervalo de 1 minuto. Posteriormente, o ciclo de teste foi repetido a cada 50ms, dentro do limite entre 1300ms e 1700ms. Após todos os testes concluídos foi possível encontrar a diferença de rotação de cada motor em relação à largura de pulso.

Quadro 5 - Diferença de velocidade entre atuadores

Motor 1		**Motor 2**	
CPW	**RPM**	**CPW**	**RPM**
1300	48	**1700**	46,2
1350	47,1	**1650**	45,5
1400	43,1	**1600**	41,2
1450	29,5	**1550**	26,5
1500	0	**1500**	0
1550	-28,3	**1450**	-26,9
1600	-43,8	**1400**	-40,9
1650	-47,3	**1350**	-46,6
1700	-47,6	**1300**	-47

Fonte: Autor

Gráfico 5 – Teste com atuadores reais



Fonte: Autor

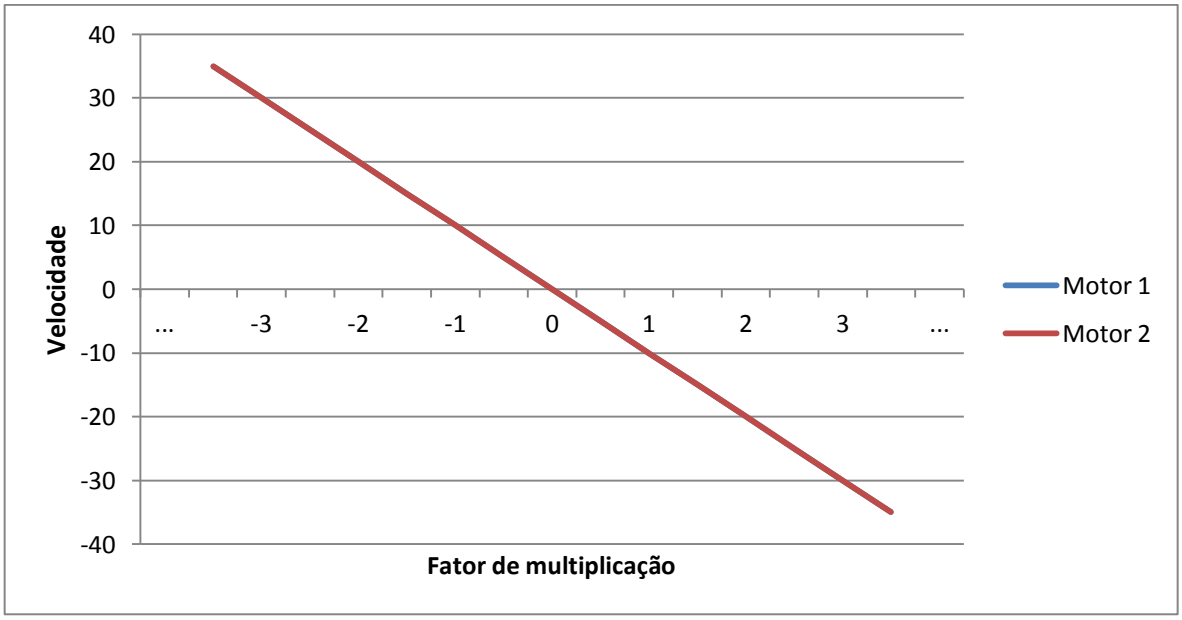
O Gráfico 5 mostra a relação entre a velocidade em rotações por minuto e o controle de largura de pulso em milissegundos, considerando os dois servos instalados em um chassis de robô. Nota-se que curvatura de velocidade de rotação são divergentes e o robô tende a desviar ligeiramente para um dos lados. Por outro lado, o mesmo gráfico mostra que a velocidade tende a se aproximar em 1500ms, pois nesse ponto a velocidade angular é a mesma para ambos os motores. O sentido da rotação é expresso pelo sinal de positivo (horário) ou negativo (anti-horário), que também pode ser chamado de curvatura de transferência, que varia de -47,6 a 48 RPM para o motor 1 e -47 a 46,2 para o motor 2, conforme é mostrado no Quadro 5. Para aplicações que envolvem navegação robótica e não possui ajuste de rota com auxílio de sistemas de navegação, essas correções podem ser compensadas por meio de softwares ou com o uso de *encoders*[7].

É importante salientar que os simuladores trabalham de forma diferente dos robôs reais, começando pela medição da velocidade. Nos robôs reais costuma-se definir as unidades de deslocamento em função do tempo em m/s, km/h ou até em rotações por minuto (em caso de velocidades constantes). No ambiente simulado, a menor unidade é chamada de *Frame*. Dessa forma, para se calcular a velocidade basta utilizar a linha de comando "Time.deltaTime", que representa o tempo decorrido em segundos desde que o último *Frame*

[7] é um dispositivo eletromecânico que conta ou reproduz pulsos elétricos a partir do movimento rotacional de seu eixo. Fornecem medidas e controles precisos em velocidades de rotação, velocidades lineares, posicionamentos angulares, robótica e outras aplicações em processos diversos.

foi computado. Com isso, é possível definir a velocidade empregada na simulação utilizando o fator de multiplicação antes do comando “Time.deltaTime”. Por exemplo, “velocFrente = 20*Time.deltaTime”. Se o valor de transição de um *Frame* para outro levar 1 segundo e o fator de multiplicação for 20, então a velocidade será 20m/s.

Gráfico 6 - Teste com atuadores simulados



Fonte: Autor

O Gráfico 6 apresenta o teste semelhante do que foi feito com robôs reais. A velocidade negativa representa o movimento do motor no sentido anti-horário e o positivo no sentido horário. Nota-se que não há diferenças de sincronismo entre os motores simulados 1 e 2. Também é possível notar que não há limite de velocidade, diferentemente do que acontece com os motores reais.

5.4 Teste 4 – Bateria

O principal objetivo desse teste é provar que o desempenho dos sistemas robóticos sofre influências de fatores externos, e essas características são omitidas ao serem transferidas para as plataformas simuladas. Nesse caso a bateria foi testada e modificada em diferentes níveis de carga. Para o experimento foi utilizado uma bateria de polímero de lítio ionizado com duas células e tensão máxima de 7,4v.

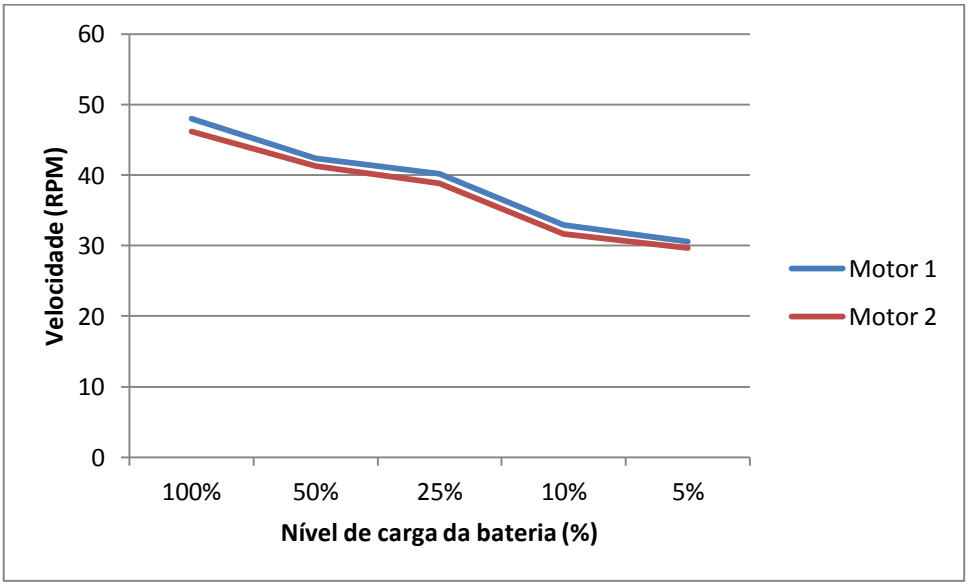
Quadro 6 - Número de RPM em diferentes níveis de carga

	RPM em nível de bateria 100%	RPM em nível de bateria 50%	RPM em nível de bateria 25%	RPM em nível de bateria 10%	RPM em nível de bateria 5%
Motor 1	48	42,4	40,2	32,9	30,6
Motor 2	46,2	41,3	38,8	31,7	29,7

Fonte: Autor

A princípio o experimento foi realizado com a capacidade máxima de carga (100%) e aplicado o mesmo teste realizado no tópico 5.3. Em seguida, com auxílio do balanceador de carga chamado de IMAX B6AC, foi possível reduzir a carga gradativamente com os seguintes valores (50%, 25%, 10% e 5%), conforme mostra o Quadro 6. Para simplificar o teste, os motores foram configurados em sua velocidade plena, que corresponde a 1300ms para o motor 1 e 1700ms para o motor 2.

Gráfico 7 - Velocidade (RPM) x Nível de bateria (%)



Fonte: Autor

O Gráfico 7 mostra o nível de carga da bateria em função da velocidade dos motores em RPM. É possível observar o declínio acentuado de desempenho entre níveis de carga 25% a 10%. Isso porque o torque fornecido pelos motores está ligado à quantidade de carga provida pela bateria. Quanto maior a carga maior será o torque aplicado. Abaixo do nível de 5%, os motores não geraram torque suficiente para movimentar o robô.

No ambiente simulado não foi possível à reprodução de gráficos, pois até o momento, nenhum dos simuladores analisados nessa pesquisa disponibilizava um componente para simulação de carga.

5.5 Análise dos resultados

De modo geral, o principal objetivo da validação é mostrar as principais peculiaridades que diferenciam as plataformas reais das simuladas. Com base nos resultados obtidos, pretende-se colaborar para definição futura de um modelo conceitual que auxilie na transição de algoritmos simulados para robôs reais. Com isso, foi realizado quatro testes implementando o algoritmo de anticolisão como estudo de casos.

O primeiro teste objetivou a análise do comportamento dos sensores omnidirecioinais diante de obstáculos com base retangular, e com os resultados foi possível perceber que diversos pontos sofreram interferências de ações externas (i.e. os sensores ultrassônicos sofreram interferência no padrão de reflexão de onda ultrassônica). Ainda no primeiro teste, não foi identificado qualquer sinal de interferência no ambiente simulado, portanto, o Gráfico 2 serviu de referência para os sensores reais.

O segundo experimento seguiu o mesmo modelo do teste anterior. Porém foi analisado o sistema de navegação em obstáculos com base triangular. Na primeira parte do teste foi utilizado bússolas apenas para acompanhar o movimento do robô. Os resultados apontaram para uma ampla variação entre o mesmo trajeto realizado pelos robôs reais e simulado. A segunda parte do teste envolveu o sistema de navegação com bússolas e resultou em um gráfico com menor interferência (variações). Com esse teste constata-se que o sistema de navegação é imprescindível para validação de um modelo inicial de portabilidade.

O terceiro teste analisou as diferenças de velocidades dos atuadores em rotações por minuto em função da largura de pulso. No primeiro momento constatou-se que os motores dos robôs tendem a desviar ligeiramente para um dos lados, devido à inconsistência dos atuadores e do solo. Para corrigir esse problema, deve ser utilizado um sistema de navegação ou *encoders*. Ainda no terceiro teste, observa-se que os simuladores não possuem limites de velocidade e o gráfico gerado resulta em função polinomial de primeiro grau.

No quarto e último experimento foi realizado testes com bateria em diferentes níveis de carga. Os resultados apontaram para uma perda significativa de performance entre 25% e 10% de carga, que corresponde a um comprometimento de aproximadamente 20% do desempenho. Isso quer dizer que os motores terão 20% a menos de torque para deslocar o robô. Por outro lado, não foi possível reproduzir um teste com bateria em ambientes simulados, pois os simuladores atuais não fornecem esse tipo de suporte.

O objetivo geral do experimento foi alcançado com êxito. Após a aplicação dos testes foi possível observar que tais implementações contribuíram para construção do conhecimento direcionado a uma especificação futura de um modelo de portabilidade que auxilie a transição de algoritmos simulados para reais.

6. CONSIDERAÇÕES FINAIS

Essa pesquisa apresentou uma proposta que visa fornecer informações (conforme mostrado no apêndice) para a formalização de um modelo conceitual que possa indicar os principais problemas existentes na transição de algoritmos validados em ambientes virtuais para robôs reais. Como estudo de caso, foi desenvolvido um algoritmo chamado de anticolisão e executado nos dois ambientes, de forma que os robôs reais e os simuladores pudessem realizar a mesma tarefa utilizando o mesmo algoritmo. O trabalho foi desenvolvido utilizando a plataforma Arduino integrado a um conjunto de sensores, módulos e atuadores entre eles: sensor omnidirecional, sensor de linha, bússola, transceiver e servomotores. O ambiente escolhido para simulação foi o motor de jogo *Unity3D*.

O trabalho abordou, além da identificação de pontos que devem ser observados em um modelo de portabilidade, outros fatores fundamentais para o sucesso da proposta, como a escolha dos ambientes de simulação (*Unity3D*) e a plataforma de robótica (Arduino). Para a realização do trabalho foi necessário o desenvolvimento completo de um robô autônomo móvel equipado com sensor omnidirecional, a implementação do algoritmo de anticolisão e construção da placa de circuito impresso. É relevante destacar que uma importante contribuição dessa dissertação foi à preparação de todo o ambiente de experimentação, tanto real como simulado, para futuras pesquisas na área.

O arcabouço teórico da pesquisa foi fundamentado na definição da robótica aplicada a diversos segmentos, conceitos de robôs autônomos móveis e as principais plataformas de robóticas existentes. Foi mostrada também uma visão geral dos simuladores, sensores e o modelo cinemático de postura. Com isto, realizou-se uma pesquisa bibliográfica e foram selecionados alguns trabalhos que nortearam o conhecimento em relação ao estado da arte envolvendo os aspectos comportamentais entre ambientes reais e simulados.

Nos experimentos foram realizados quatro testes envolvendo cenários diversos, elementos sensoriais diferentes, atuadores e bateria. Em todos os casos verificou-se aspectos que evidenciam a presença das variáveis independentes que são omitidas na plataforma simulada e estão presentes no ambiente real. Com base nisso, mostrou-se que é possível construir um modelo de transição inicial que formalize as principais particularidades envolvendo simuladores e robôs reais.

As variações do ambiente real são de fato um dos principais problemas a ser enfrentado, pois devido à complexidade do ambiente operacional, algumas variáveis tais como, características do terreno ou sensibilidade dos sensores podem ser ignoradas ou omitidas nos simuladores. Sem a presença de um modelo conceitual que viabilize o elo entre os meios, torna-se incerto garantir a fidelidade entre o simulado e o "mundo real", visto que essas premissas podem reduzir o realismo da simulação o bastante para tornar o algoritmo desenvolvido pouco eficiente para implantação em um ambiente real. É relevante enfatizar a importância do ambiente de experimentação proposto pelos simuladores em termos de redução de custos e riscos por meios de testes em robôs reais, além de possibilitar a capacidade de testar repetidamente e coletar dados mensuráveis e quantificáveis. Baseando-se nesses fatos, acredita-se que um modelo de portabilidade a ser proposto possa contribuir na melhoria da qualidade e eficiência dos algoritmos desenvolvidos em simuladores.

Algumas dificuldades foram enfrentadas no decorrer deste trabalho. Por tratar-se de uma pesquisa pioneira dentro da própria instituição, poucos materiais de apoio encontraram-se disponíveis para pesquisa. Outros problemas como a falta de peças, atrasos por parte do fornecedor e a falta de manual técnico para interligação de sensores e módulos contribuíram acentuadamente para complexidade do projeto.

Alguns trabalhos futuros devem ser desenvolvidos para que o estudo atual possa ser utilizado na criação de um modelo de portabilidade, sendo eles: (i) Expandir o campo de análise para outros tipos de variáveis de ambiente. Nesse caso, o maior desafio está em torno da quantidade elevada de variáveis presentes na natureza, (ii) utilizar sistemas multirobóticos, outros simuladores e plataformas robóticas distintas, incluindo um modelo de mobilidade holonômico, (iii) melhorar o algoritmo de anticolisão para que o mesmo possa abranger uma diversidade maior de cenários e sensores, e (iv) avaliar o simulador de robôs como uma ferramenta de potencialidade educacional.

7. REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO. Página do Arduino, 2005. Disponivel em: <http://arduino.cc/en/>. Acesso em: 29 Setembro 2012.

BALAGUER, B.; CARPIN, S. **Where Am I? A Simulated GPS Sensor for Outdoor Robotic Applications**. Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR '08). Heidelberg: [s.n.]. 2008. p. 222 - 233.

BALAKIRSKY, S. B. et al. **An Integrated Control and Simulation Environment for Mobile Robot Software Development**. Proceedings of IDETC/CIE 2008 ASME 2008 International Design Engineering Technical Conference & Computers and Information in Engineering Conference. New York, NY: [s.n.]. 2008.

BALAKIRSKY, S. et al. **From Simulation to Real Robots with Predictable Results:** Methods and Examples. Performance Evaluation and Benchmarking of Intelligent Systems, R. Madhavan, E. Tunstel, and E. Messina. Gaithersburg, MD: [s.n.]. 2009. p. 113-137.

BASIC STAMP. Basic Stamp - Programming Manual, 2000. Disponivel em: <http://www.sfu.ca/phys/430/basicstampman.pdf>. Acesso em: 09 Outrubro 2012.

BEAGLEBOARD. BeagleBoard-xM Rev C - System Reference Manual, 04 Abril 2010. Disponivel em: <http://beagleboard.org/static/BBxMSRM_latest.pdf>. Acesso em: 15 Outubro 2012.

BRAGA, N. C. Instituto Newton C. Braga, 2013. Disponivel em: <http://www.newtoncbraga.com.br/>. Acesso em: 17 Setembro 2013.

BRESSANI, D. et al. **Sensor de Medição de Distância**. Dissertação de Mestrado. [S.l.]: [s.n.]. 2006.

CARMEN. Carmen: Robot Navigation Toolkit, 2006. Disponivel em: <http://carmen.sourceforge.net/doc/>. Acesso em: 11 Novembro 2012.

CHEN, I. Y.-H.; MACDONALD, B.; WUNSCHE, B. **Mixed Reality Simulation for Mobile Robots**. IEEE International Conference on Robotics and Automation - Kobe International Conference Center. Kobe, Japan: [s.n.]. 2009.

CHEN, S.; BILLINGS, S. **Representations of non-linear systems:** The narmax model. International Journal of Control 49. [S.l.]: [s.n.]. 1989. p. 1013-1032.

DAWSON, S.; WELLMAN, B. L.; ANDERSON, M. **Using Simulation to Predict Multi-robot Performance on Coverage Tasks**. International Conference on Intelligent Robots and Systems. Taipei, Taiwan: [s.n.]. 2010. p. 18-22.

DENARDIN, G. W. Microcontroladores. **Universidade Tecnológica Federal do Paraná**, 26 Março 2008. Disponivel em: <http://pessoal.utfpr.edu.br/gustavo/apostila_micro.pdf>. Acesso em: 06 Fevereiro 2013.

DIXON, K. et al. **RAVE:** A Real and Virtual Environment for Multiple Mobile Robot Systems. Proceedings of the 1999 IEEE/RJS International Conference on Robotics and Systems (IROS '99). [S.l.]: [s.n.]. 1999.

FORMIGA, M. M. **Comunicação de dados para um sistema de telemetria de baixo custo**. Dissertação Mestrado do Curso de Pós-graduação em Informática da Universidade Federal de Campina Grande. Campina Grande: [s.n.]. 2005.

FREESE, M. Virtual Robot Experimentation Platform., 2011. Disponivel em: <http://v-rep.eu>. Acesso em: 15 Dezembro 2012.

FREIRE, E. O. **Controle de Robôs Móveis por Fusão de Sinais de Controle Usando**. UFES. Espirito Santo. 2002.

HASSANZADEH, I.; MADANI, K.; BADAMCHIZADEH, M. A. **Mobile Robot Path Planning Based on Shuffled Frog Leaping Optimization Algorithm**. 6th annual IEEE Conference on Automation Science and Engineering. Toronto, CA: [s.n.]. 2010.

HU, X.; ZEIGLER, B. P. **Measuring Cooperative Robotic Systems Using Simulation-Based Virtual Environment**. Performance Metrics for Intelligent Systems Workshop. Tucson AZ, USA: [s.n.]. 2004.

HU, X.; ZEIGLER, B. P. **A simulation-based virtual environment to study cooperative robotic systems**. Integrated Computer-Aided Engineering. Amsterdam: [s.n.]. 2005. p. 353-367.

JÁCOBO, J. E. A. **Desenvolvimento de um Robô Autônomo Móvel Versátil utilizando Arquitetura Subsumption**. Dissertação de mestrado do curso de engenharia elétrica da Universidade Estadual de Campinas. Campinas: [s.n.]. 2001.

JIANG, X.; YUICHI, M.; XINGQUAN, Z. **Predictive Fuzzy Control for a Mobile Robot with Nonholonomic Constraints**. 12th International Conference on Proceedings.Advanced Robotics, 2005. ICAR '05. Seattle, WA: [s.n.]. 2005.

KALMEGH, S. K.; SAMRA, D. H.; RASEGAONKAR, M. N. **Obstacle avoidance for a mobile exploration robot using a single ultrasonic range sensor**. International Conference on Emerging Trends in Robotics and Communication Technologies (INTERACT). Chennai, Índia: [s.n.]. 2010.

KOESTLER, A.; BRAUNL, T. **Mobile robot simulation with realistic error models**. ICARA 2004. Palmerston North, New Zealand: [s.n.]. 2004. p. 46–51.

KYRIACOU, T. et al. **Accurate robot simulation through system identification**. 10th British Conference on Mobile Robotics - Towards Autonomous Robotic Systems. United Kingdom: [s.n.]. 2008. p. 1082-1093.

LATOMBE, J.-C. **Robot Motion Planning**. Massachusetts: Kluwer Academic Publishers, 1996. 651 p.

LEITE, A. C. **Servovisão adaptativa e controle de força para robôs manipuladores com cinemática e dinâmica incertas interagindo com ambientes não-estruturados**. Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa em Engenharia. Rio de Janeiro, p. 182. 2011.

LOETZSCH, M.; RISLER, M.; JUNGEL, M. **XABSL - A Pragmatic Approach to Behavior Engineering**. Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on. Paris: [s.n.]. 2006. p. 5124 - 5129.

MARTINS, A. **O que é robótica**. São Paulo: Ed. Brasiliense, 2007. 16 p.

MCCOMB, G.; PREDKO, M. **The Robot Builder's Bonanza**. New York, USA: McGraw-Hill, 2006.

MEEDEN, L. **Bridging the Gap Between Robot Simulations and Reality With Improved Models of Sensor Noise**. Proceedings of the Third Annual Genetic Programming Conference. San Francisco, CA: [s.n.]. 1998.

MICROSOFT. Microsoft Robotics Developer Studio, 2012. Disponivel em: <http://www.microsoft.com/robotics/>. Acesso em: 16 Novembro 2012.

PARALLAX. BASIC Stamp Syntax and Reference Manual, Novembro 2004. Disponivel em: <http://www.parallax.com/dl/docs/prod/stamps/basicstampman.pdf>. Acesso em: 18 Dezembro 2012.

PARALLAX. Controlling Servo Speed and Direction. **Learn Parallax.com**, 2012. Disponivel em: <http://learn.parallax.com/node/187>. Acesso em: 17 Dezembro 2013.

RASPBERRY PI, QUICK START GUIDE. The Raspberry Pi – Single Board Computer, 2012. Disponivel em: <http://www.farnell.com/datasheets/1524403.pdf>. Acesso em: 15 Outubro 2012.

RICARDO, Z.; ALEXANDRE, M. **Introdução aos sistemas embutidos**. [S.l.]: [s.n.]. 1999.

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**. 3ª. ed. [S.l.]: Campus, 2009.

SCHLESINGER, M.; PARISI, D. The Agent-Based Approach: A New Direction for Computational Models of Development. **Developmental Review**, v. 21, p. 121-146, 2001.

SCHLUSSEL, K. **Robotics and Artificial Intelligence Across the Atlantic and Pacific**. IEEE Transactions on Industrial Electronics. [S.l.]: [s.n.]. 1985. p. 244-251.

SILVA, F. A. O. **Desenvolvimento de um robô autônomo que desvia de obstáculos - RADO**. São Paulo: [s.n.]. 2008.

SMITH, R. Open Dynamics Engine, 2012. Disponivel em: <http://www.ode.org>. Acesso em: 02 Dezembro 2012.

SOUZA, D. **Desbravando o PIC – ampliado e atualizado para PIC 16F628A**. São Paulo: Editora Érica Ltda, 2008.

SUÁREZ, L. L. **Conhecimento Sensorial - Uma Análise Segundo a Perspectiva da Semiótica**. Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e Computação. Campinas - SP: [s.n.]. 2000. p. 144.

TOMBA, C. H. S. **Sensor infraverlho de olhos para ceratômetro em lâmpada de fenda com transmissão via bluetooth**. USP. São Carlos - SP. 2008.

WEBOTS. Cyberbotics: Professional mobile robot simulation, 2012. Disponivel em: <http://www.cyberbotics.com/overview>. Acesso em: 14 Novembro 2012.

WERNECK, P. Introdução ao Arduino. **Eletrônica Online:** O seu portal para o universo da eletrônica, 22 Junho 2009. Disponivel em: <http://www.sabereletronica.com.br/secoes/leitura/1307>. Acesso em: 2012 Outubro 15.

XU, Y.; MELLMANN, H.; BURKHARD, H.-D. **An Approach to Close the Gap between Simulation and Real Robots**. Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots (SIMPAR'10). Heidelberg: Springer-Verlag. 2010. p. 533-544.

ZIEMKE, T. On the Role of Robot Simulations in Embodied Cognitive Science. **AISB Journal**, United Kingdon, p. 389-399, 2003.

APÊNDICE

Quadro 7 – Metodologia aplicada nos experiementos e resultados obtidos

	Teste 1	**Teste 2**	**Teste 3**	**Teste 4**
Metodologia aplicada	• O primeiro teste objetivou a análise do comportamento dos sensores omnidirecioinais diante de obstáculos com base retangular em plataformas reais e simuladas.	• **Sem sistema de navegação**: Foi utilizada bússola apenas para acompanhar o movimento do robô. • **Com sistema de navegação**: Foi utilizado o sistema de navegação para evitar a fuga de rota.	• Analisou as diferenças de velocidades dos atuadores em rotações por minuto em função da largura de pulso em plataformas reais e simuladas.	• Foi realizado testes com bateria com os seguintes níveis de carga: 100%, 50%, 25%, 10% e 5%.
Resultado obtido	• Nos robôs reais os sensores ultrassônicos sofreram interferência no padrão de reflexão de onda ultrassônica. • Nos robôs simulados não foi detectado qualquer indício de interferência.	• **Sem sistema de navegação**: Os resultados apontaram para uma ampla variação entre o mesmo trajeto realizado pelos robôs reais e simulado. • **Com sistema de navegação**: Resultou em um gráfico com menor interferência (variações).	• Constatou-se que os motores dos robôs reais tendem a desviar ligeiramente para um dos lados, devido à inconsistência dos atuadores e do solo. O sistema de navegação foi utilizado para correção do problema. • No robô simulado não há diferença de rotação entre os motores.	• Houve uma perda significativa de performance entre 25% e 10% de carga, que corresponde a um comprometimento de aproximadamente 20% do desempenho (i.e. o motor terá 20% a menos de torque para movimentar o robô). • Não foi possível realizar teste com nível de carga em simuladores.

Fonte: Autor