UNIVERSIDADE ESTADUAL PAULISTA
"Júlio de Mesquita Filho"

Pós-Graduação em Ciência da Computação

Diego Renan Bruno

Algoritmo Neurogenético com vistas para o Planejamento de Rotas de Robôs Móveis Autônomos

UNESP

2016

Diego Renan Bruno

Algoritmo Neurogenético com vistas para o Planejamento de Rotas de Robôs Móveis Autônomos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas, Campus de São José do Rio Preto da Universidade Estadual Paulista Júlio de Mesquita Filho, como requisito para a obtenção do título de Mestre em Ciência da Computação.

Linha de Pesquisa: Arquitetura de Computadores e Sistemas Distribuídos

Orientador: Prof. Dr. Norian Marranghello

UNESP

2016

Bruno, Diego Renan.
Algoritmo neurogenético com vistas para o planejamento de rotas de robôs móveis autônomos / Diego Renan Bruno. -- São José do Rio Preto, 2016
183f. : il., tabs.

Orientador: Norian Marranghello
Dissertação (mestrado) – Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Biociências, Letras e Ciências Exatas

1. Computação - Matemática. 2. Redes neurais (Computação) 3. Algoritmos genéticos. 4. Robôs móveis. 5. Processamento de sinais – Técnicas digitais. I. Marranghello, Norian. II. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU – 518.721

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Diego Renan Bruno

Algoritmo Neurogenético com vistas para o Planejamento de Rotas de Robôs Móveis Autônomos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas, Campus de São José do Rio Preto da Universidade Estadual Paulista Júlio de Mesquita Filho, como requisito para a obtenção do título de Mestre em Ciência da Computação.

Linha de Pesquisa: Arquitetura de Computadores e Sistemas Distribuídos

BANCA EXAMINADORA

Prof. Dr. Norian Marranghello
UNESP – São José do Rio Preto
Orientador

Prof. Dr. Aledir Silveira Pereira
UNESP – São José do Rio Preto

Prof. Dr. Henrique Dezani
FATEC –São José do Rio Preto

São José do Rio Preto, 27 de Abril de 2016.

“A ciência é, portanto, uma perversão de si mesma,
a menos que tenha como fim último
melhorar a humanidade.”

-Nikola Tesla

"Faça ou não faça. A tentativa não existe."

-Mestre Yoda *(Star Wars)*

Agradecimentos

Agradeço primeiramente ao Mestre Menino Jesus de Praga por sempre me guiar nos caminhos dos estudos, trazendo-me paz, sabedoria e humildade, pois, sem Ele este trabalho não seria possível.

Agradeço, também, aos meus pais, Pedro e Mariluce, que sempre me apoiaram em fazer dos estudos um potencial diferenciado em minha vida. Não deixo de agradecer também ao meu irmão Danilo, que sempre teve suas curiosidades sobre o meu trabalho e sempre esteve ao meu lado durante os meus estudos. Agradeço, também, às minhas avós Ires e Isabel, que sempre contribuíram para a pessoa que sou hoje.

Gostaria de agradecer o Professor Norian Marranghello, que me orientou desde o meu último ano de graduação na pesquisa para a robótica e sempre mostrou grande dedicação em sua orientação, sempre acreditando em minha capacidade e me incentivando a crescer como um bom pesquisador. Sua orientação contribui de maneira grandiosa na minha formação acadêmica.

Agradeço também aos meus amigos da Unesp e que hoje são meus amigos para a vida toda, Cláudio Fuzita (por ser meu amigo e irmão nesta caminhada), Fernando Persan, Willian Ferreira, Alessandra Braga, Gabriel Covello Furlanetto, Wanderson Rangel, Diego Emygdio, Alciano Oliveira, Eduardo Lima Nascimento, Toni Amorim, Luis Maschi e a todos os outros amigos que fiz nesta universidade, pela amizade, pela contribuição de maneira direta ou indireta ao meu trabalho e pelo apoio e incentivo de sempre.

Agradeço à Unesp, ao Departamento de Ciências da Computação e Estatística e a todos os seus docentes que estiveram neste caminho de estudos, seja como professor de algumas disciplinas ou como banca de avaliação de alguns trabalhos que apresentei durante este tempo que estive como aluno.

Agradeço também aos amigos da Fatec Catanduva, Marco Di Foggi, Denys Riva, Denis Mosconi, Marcio Marques e Michel Mazenini, sempre presentes em minha caminhada e aos que passaram pela minha vida, que me apoiaram e torceram por mim.

Agradeço também aos amigos da USP de São Carlos, Fernando Santos Osório e Rafael Berri, por me mostrarem novos caminhos na robótica móvel.

De uma maneira especial não poderia deixar de agradecer também os meus amigos de trabalho, Maria Teresa Catanho e José Claudinei Cordeiro, amigos que fizeram grande diferença neste caminho de estudos, não só pela amizade, mas também pelo incentivo e companheirismo. Agradeço também à diretora da escola em que trabalho, Dona Eliéte, pelo incentivo aos estudos.

A todos a minha profunda gratidão, e que a força esteja sempre com cada um de vocês, contem comigo sempre que precisarem.

Resumo

Neste trabalho foi desenvolvido um sistema de controle híbrido bioinspirado para o planejamento de rota com vistas para a robótica móvel autônoma, baseado em redes neurais artificiais e algoritmos genéticos. O controlador tem como principal objetivo auxiliar o robô móvel em sua navegação quando aplicado em ambientes dinâmicos. Para este trabalho, o ambiente dinâmico utilizado é um “chão de fábrica” industrial, em que alguns obstáculos não são fixos e permanecem em movimentação constante. O controlador desenvolvido neste trabalho pode ser adaptado facilmente para operar em outros ambientes dinâmicos. Independentemente do ambiente utilizado, o controlador deve ser capaz de traçar uma rota possível entre o ponto inicial e o ponto de objetivo, tendo o potencial de evitar todo tipo de obstáculo que surgir nessa rota, seja um obstáculo estático ou dinâmico. O algoritmo foi implementado na linguagem *C* e simulado no *software* de modelagem e simulação de robôs V-REP (*Virtual Robot Experimentation Platform*). O controlador neurogenético mostrou ser eficiente para auxiliar o robô em sua navegação quando aplicado em ambientes dinâmicos.

Palavras chave: Redes Neurais Artificiais. Algoritmos Genéticos. Robôs Móveis. Planejamento de rotas.

Abstract

In this work, a bioinspired hybrid control system was developed for route planning, aiming autonomous mobile robots based on artificial neural networks and genetic algorithms. The main objective of the controller is to assist the mobile robot in its navigation when applied in dynamic environments. For this work, the dynamic environment chosen was a " factory floor", in which some industrial obstacles are not fixed and remain in constant movements. The controller developed in this work can easily be adapted to operate in other dynamic environments. Regardless the environment chosen in this work, the controller must be able to map out a possible route between the starting point and the goal point with the potential to avoid all types of obstacles that appear along the routes, either a static or a dynamic one. The algorithm was implemented in *C* language and simulated on a robots modeling and simulation *software* called V-REP (Virtual Robot Experimentation Platform). The neurogenetic controller was efficient to assist the mobile robot in its navigation when applied in dynamic environments.

Keywords: Artificial Neural Networks. Genetic Algorithm. Mobile Robots. Route Planning.

Sumário

Capítulo 1

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Capítulo 6

Capítulo 7

Capítulo 8

Capítulo 9

Capítulo 10

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviações

AD	Algoritmo de Distinção
AG	Algoritmo Genético
AGM	Algoritmo Genético Modificado
ADALINE	*Adaptative Linear Neuron ou Adaptative Linear Element*
AMTS	*Automated Material Transport System*
API	*Application Programming Interface*
DARPA	*Defense Advanced Research Projects Agency*
DLA	Distância da Latitude
DLO	Distância da Longitude
GJK	Gilbert-Johnson-Keerth
ENPC	*École Nationale des Ponts et Chaussées*
GM	Grau de Movimentação
GPS	*Global System Position*
HEX	Hexadecimal
IFSP	Instituto Federal de São Paulo
INRIA	*Institut National de Recherche en Informatique et em Automatique*
NI	Nível de Interferência
PDI	Processamento Digital de Imagens
PUC	Pontifícia Universidade Católica
RNA	Rede neural artificial
RGB	*Red, Green and Blue*
TO	Taxa de Ocupação
VC	Visão Computacional
V-REP	*Virtual RobotExperimentation Platform*

Capítulo 1

Introdução

A robótica móvel utiliza técnicas de grande complexidade e de grande interesse científico com vistas para a navegação autônoma. Técnicas de otimização para este problema são desenvolvidas constantemente, buscando soluções com o objetivo de controlar um robô em um ambiente dinâmico, livre de colisões com obstáculos estáticos e dinâmicos (FERREIRA, 2015). O caminho feito pelo robô também deve ser o mínimo possível, assim, reduzindo o tempo e a sua energia.

Muitas pesquisas na área da Ciência da Computação estão ligadas ao desenvolvimento de sistemas inteligentes para a robótica móvel, sendo que neste trabalho, estes sistemas inteligentes estão relacionados ao controlador do robô e sua capacidade de locomoção. Como principal objetivo, a autonomia do sistema robótico, está potencialmente ligada ao desenvolvimento de algoritmos inteligentes. O sistema inteligente de um robô é composto por três elementos básicos, sendo estes: sensoriamento, planejamento e ação (GAT, 1998).

Para um melhor entendimento do problema de rotas de robôs móveis, pode-se citar o evento DARPA (*Defense Advanced Research Projects Agency*) *Grand Challenge* (DARPA, 2015) ; (HATA, 2010). O desafio tem como objetivo uma corrida realizada entre robôs (carros autônomos não tripulados). O evento contou com duas edições, a primeira edição, em 2004, teve como objetivo atravessar o deserto de Mojave, nos Estados Unidos, com 240 km de extensão, no entanto nenhum dos 107 robôs participantes completou a prova (HATA, 2010). Um ano após foi lançado o desafio novamente, tendo um progresso, dos 195 robôs participantes, 5 completaram a prova. No ano de 2007, a complexidade do desafio foi aumentada, sendo utilizado como ambiente dinâmico, o estado da Califórnia, também nos Estados Unidos (HATA, 2010). O desafio foi nomeado como DARPA *Urban Challenge*, teve como maior complicação, as regras de trânsito local, para se locomover com segurança sobre este ambiente dinâmico. O desafio teve 55 robôs inscritos, e apenas três deles cumpriram todas as etapas da prova (HATA, 2010) ; (DARPA, 2015). O desenvolvimento de algoritmos inteligentes para o controle e para o alcance dos objetivos acompanhados de vários desafios por parte dos robôs é uma tarefa de grande complexidade, sendo que a pesquisa científica nessa área dispõe de grande aplicação para resolução de problemas de navegação dos mesmos (HATA, 2010).

1.1. Motivação

A robótica móvel tem uma potencial contribuição para a automação de processos industriais, desenvolvendo funções em que é exigido do robô a sua mobilidade. Na literatura, existe uma grande quantidade de estudos sobre veículos autônomos aplicados na automação de processos industriais e outros serviços (SECCHI, 2008) ; (TUNCER; YILDIRIM, 2012).

Os robôs móveis podem ser aplicados a outros serviços em prol do aumento qualitativo do bem-estar da humanidade, por exemplo: os robôs enfermeiros para o tratamento de enfermos, os robôs autônomos para a transferência de objetos, os veículos não tripulados de exploração em planetas não habitáveis (SECCHI, 2008). Alguns trabalhos relacionados a estes casos foram desenvolvidos (MORAVEC, 1999; SECCHI, 2008; RAJA; PUGAZHENTHI, 2012).

A principal motivação deste trabalho é a contribuição aos robôs manipuladores (estáticos) e aos humanos que atuam nas células robóticas, a fim de não perder tempo na espera de peças e evitar, também, o perigo de um humano interagir com ambientes cheios de riscos para buscar as peças no depósito.

O planejamento de rotas para os robôs móveis e suas técnicas de controle com vistas para a navegação autônoma foram o foco principal de grande parte da pesquisa na década de setenta do século XX (SHILTAGH; JALAL, 2013).

As técnicas de planejamento de rotas de robôs móveis autônomos são classificadas principalmente em duas categorias, sendo estas os métodos clássicos e os métodos inteligentes (QU; XING; ALEXANDER,2013) ; (FERREIRA, 2015). Dentre as várias técnicas inteligentes, as Redes Neurais Artificiais (RNAs) e os Algoritmos Genéticos (AGs) são facilmente encontrados na literatura por apresentarem grande potencial para a resolução destes problemas em robótica móvel (ABINAYA ; KUMAR, 2014) ; (KHELCHANDRA ; HUANG, 2014) ; (SANTHOSH, 2014) ; (ABBASPOUR; ALPOUR, 2015) ; (BESSA ; BARROSO, 2015) ; (FETANAT ; HAGHZAD , 2015) ; (LUO ; YANG, 2015) ; (LYRIO ; SANTOS, 2015) ; (PANDA ; CHOUDHURY, 2015). Neste trabalho, as RNAs são aplicadas no reconhecimento de obstáculos, e os AGs, por seu grande potencial em otimização, são aplicados em novas soluções para obstáculos desconhecidos (SHILTAGH; JALAL, 2013) ; (FERREIRA, 2015).

1.2. Objetivos e expectativas do trabalho

Neste trabalho, uma Rede Neural Artificial (RNA) e um Algoritmo Genético (AG) são usados para auxiliar o robô móvel a se movimentar, detectar e identificar os obstáculos no meio ambiente, aprender o meio ambiente e alcançar o objetivo desejado em um ambiente dinâmico. Este trabalho teve como objetivo o desenvolvimento dos algoritmos para trabalharem de maneira híbrida, modelando, implementando e simulando este controlador em um robô móvel em seu ambiente de trabalho. Utiliza-se, para este fim, o simulador V-REP (*Virtual Robot Experimentation Platform*).

No caso do robô móvel encontrar qualquer obstáculo (reconhecido ou não reconhecido pela RNA), o controlador deve ser capaz de criar uma nova rota auxiliar segura, ou seja, livre de colisão com obstáculos. O algoritmo híbrido e seu banco de conhecimento podem ser modificados para que o robô trabalhe em um ambiente diferente do que foi proposto neste trabalho. A implementação do algoritmo híbrido em um ambiente de simulação tem vistas para uma futura implementação do controlador neurogenético em *hardware*, na plataforma *Kihon* (MATSUMURA, 2014).

1.3. Organização do Texto

Este texto é organizado da seguinte forma:

No capítulo 1, foi feita uma introdução e motivação para este trabalho, bem como os objetivos principais para o desenvolvimento do mesmo.

No capítulo 2, foi apresentada uma revisão bibliográfica sobre conceitos referentes a robôs móveis e seus algoritmos para rota autônoma, apresentando técnicas bioinspiradas para a solução de problemas de navegação do robô.

No capítulo 3, foram apresentadas todas as técnicas utilizadas neste trabalho para a implementação do controlador de movimento do robô móvel autônomo. Também foi apresentada a metodologia para o desenvolvimento de cada um dos algoritmos de controle e as estratégias que cada um utiliza para a solução de um obstáculo, bem como o ambiente de simulação utilizado para robôs, a modelagem e a simulação da RNA *Perceptron* no ambiente de simulação *Scilab.*

No capítulo 4, foi apresentada a Visão Computacional (VC) aplicada à análise e reconhecimento de obstáculos frequentes com vistas para o tratamento de dados para a RNA. Nesse capítulo, o objetivo é apresentar algoritmos que foram utilizados pelo sistema de visão computacional do robô para a extração de características dos obstáculos. Todos os algoritmos foram modelados e simulados no ambiente *Scilab a priori.* O sistema de visão foi efetivamente implementado em linguagem LUA e também foi apresentado nesse capítulo.

No capítulo 5, foram apresentados uma análise e um levantamento de dados em ambientes reais para a base de conhecimento em prol da modelagem e simulação dos ambientes dinâmicos criados no simulador de robôs V-REP. Neste capítulo foi também apresentada uma análise sistemática com vistas para a problemática de cada tipo de obstáculo.

No capítulo 6, foi apresentada a metodologia aplicada à implementação do controlador híbrido e toda a técnica utilizada para a implementação do controlador neurogenético. Também foram apresentados métodos e ferramentas utilizados para a implementação do controlador neurogenético, assim como os testes e resultados obtidos por meio de simulações feitas no ambiente modelado. As simulações aqui apresentadas tiveram como objetivo a validação do ambiente dinâmico modelado.

No capítulo 7, foram apresentados os testes comportamentais e de custo computacional do algoritmo neurogenético. Os testes foram feitos num conjunto de algoritmos que formam o sistema de controle híbrido desenvolvido neste trabalho, intitulado como algoritmo neurogenético. Ainda nessa seção, foram feitas análises de comportamento e custo computacional em cada um dos códigos por meio de simulações no ambiente dinâmico modelado com base em dados coletados nos ambientes reais para a validação do algoritmo.

No capítulo 8, foi apresentada a validação do controlador neurogenético com vistas para as funções propostas ao robô neste trabalho. Neste capítulo também foi apresentado o controlador neurogenético com vistas para auxiliar o robô no desenvolvimento de suas funções propostas neste trabalho. O objetivo neste capítulo foi apresentar diferentes situações para que o controlador pudesse apresentar sua eficiência.

No capítulo 9, foram apresentados os resultados quantitativos do algoritmo neurogenético que foram obtidos com base na análise referente ao custo computacional e de tempo do AG e da RNA de maneira independente. Esta avaliação teve como objetivo analisar o custo de tempo dos algoritmos pertencentes ao controlador neurogenético para gerar uma nova solução de desvio para obstáculos.

No capítulo 10, foram apresentadas as conclusões, comparações entre os resultados, problemas encontrados e sugestões de trabalhos futuros referente ao trabalho desenvolvido nesta dissertação de mestrado.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é apresentada uma revisão bibliográfica sobre conceitos referentes a robôs móveis e seus algoritmos para rota autônoma, contendo técnicas bioinspiradas para a sua solução. Neste capítulo também é apresentado alguns algoritmos bioinspirados híbridos utilizados atualmente para a robótica móvel autônoma.

2.1. Algoritmos evolucionários e inteligência coletiva para rota autônoma de robôs móveis

A utilização da computação e automação principalmente no ambiente industrial no século XX gerou potenciais melhorias no trabalho do ser humano. Estas mudanças contribuíram para a revolução da mão de obra, dando uma nova perspectiva para o homem, que antes desenvolvia trabalhos de risco e, também, dando maior qualidade aos seus produtos desenvolvidos de maneira automática (FERREIRA, 2015). O século XXI chega com grande força na robótica, sendo comum ver sistemas robóticos em indústrias, fazendo atividades antes feitas manualmente, e também em outros setores, como: laboratórios farmacêuticos, em salas de cirurgias ou no cuidado de pacientes debilitados, robôs que procuram por sobreviventes em desastres naturais e acidentes nucleares e até mesmo na exploração extraterrestre (SECCHI, 2008) ; (FERREIRA, 2015).

A palavra robô tem diversas definições na literatura, segundo o dicionário Aurélio (HOLANDA, 2010), robô é um sistema automatizado, na maioria dos casos com aparência física parecida a de um humano, e que realiza trabalhos e movimentos de um trabalhador, o que sabemos que não é aceitável para todos os casos, pois hoje existem robôs de diferentes tipos, e que muitas vezes executam funções nada semelhantes à de um humano. Para exemplo, temos um robô militar, que pode ser observado na figura 2.1 (a), que tem a função de poupar um humano de uma tarefa de alto risco, porém, não desenvolve nem uma função mecânica semelhante ao movimento de um homem. Na figura 2.1 (b) pode ser observado o robô *Curiosity*, utilizado para a investigação sobre o clima, a existência de vida, e a coleta de dados para uma futura missão tripulada para marte.

Isaac Asimov, um escritor e bioquímico americano, nascido na Rússia e autor de obras de ficção científica, definiu três leis básicas para a robótica, sendo elas (ASIMOV, 2014)**:**

- Primeira Lei: um robô não pode ferir um ser humano, ou, por omissão, permitir que um ser humano sofra algum mal.

- Segunda Lei: um robô deve obedecer às ordens que lhe sejam dadas por seres humanos, exceto nos casos em que tais ordens contrariem a primeira lei.

- Terceira Lei: um robô deve proteger sua própria existência enquanto tal proteção não entrar em conflito com a primeira ou segunda lei.

Figura 2.1:(a) Robô militar *Warrior* X700 da iRobot (SECCHI, 2008); (b) Robô *curiosity* (NASA, 2015)

Na figura 2.2(a) pode ser observado dois exemplos de robôs móveis da ficção científica (WARS, 1977), o robô humanoide C-3PO (à esquerda), sendo este um robô móvel com pernas e mecanismos voltados às funções humanas e o robô R2-D2, um robô móvel com rodas. Os dois robôs, mesmo sendo de aparência e características mecânicas distintas, são classificados igualmente como robôs móveis, e por serem independentes de auxílio humano, são autônomos. Na figura 2.2(b) é apresentado o robô B9, humanoide do seriado americano "*Lost in Space*", programado para auxiliar a família Robinson na colonização de alguns planetas no espaço (SPACE, 1965).

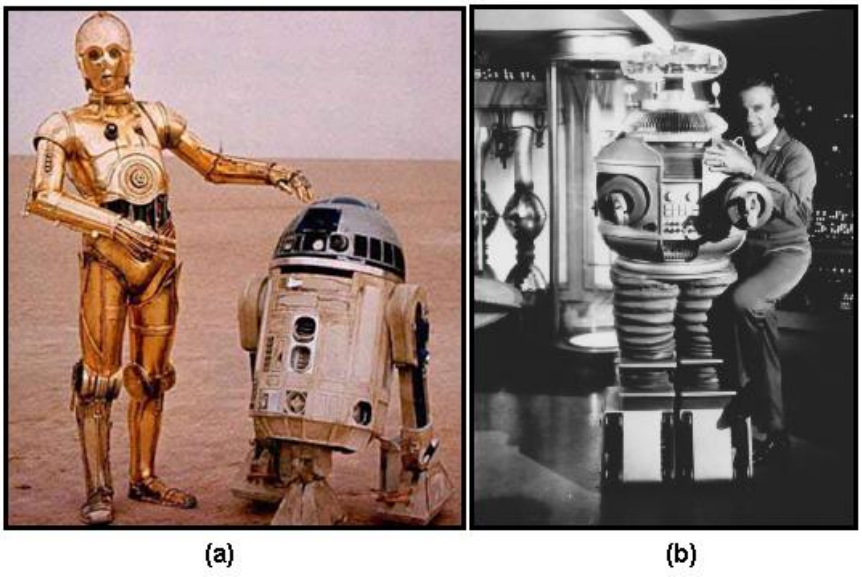


Figura 2.2: (a) Robôs do filme *Star Wars* (WARS, 1977); (b) Robô B9 do seriado "*Lost in Space*" (SPACE, 1965)

Na automação de processos industriais, a robótica é de grande potencial para soluções que se preocupam em criar mecanismos para executar funções humanas, utilizando sensoriamento para captar informações do mundo externo, e atuadores para executar as funções de ação (CRAIG, 2005) ; (FERREIRA, 2015). Na robótica móvel, o robô precisa perceber o seu ambiente e, para isso, utiliza os seus dados sensoriais, o que possibilita sua localização e conhecimento do ambiente, sendo este último aplicado no sistema de ações do robô que necessita de um conhecimento *a priori* do ambiente antes de tomar uma potencial escolha de solução (RAJA; PUGAZHENTHI, 2012) ; (FERREIRA, 2015).

Na automação industrial, além dos robôs móveis, alguns robôs manipuladores estáticos já existiam e ainda são de extrema importância para os processos automáticos, estes, mais conhecidos como “braços” robóticos, são utilizados, por exemplo, para a manipulação de elementos químicos prejudiciais à saúde humana em indústrias farmacêuticas e, solda e pintura em indústrias metalúrgicas (SECCHI, 2008).

Podemos destacar algumas vantagens potências da robótica na indústria: aumento da produtividade, flexibilidade, qualidade e melhoria na segurança (SECCHI, 2008). A desvantagem dos robôs manipuladores industriais, é que estes são fixos, consequentemente não podendo mover-se em alguma situação em que esta função seja exigida (SIEGWART; NOURBAKHSH, 2004) ; (FERREIRA, 2015).

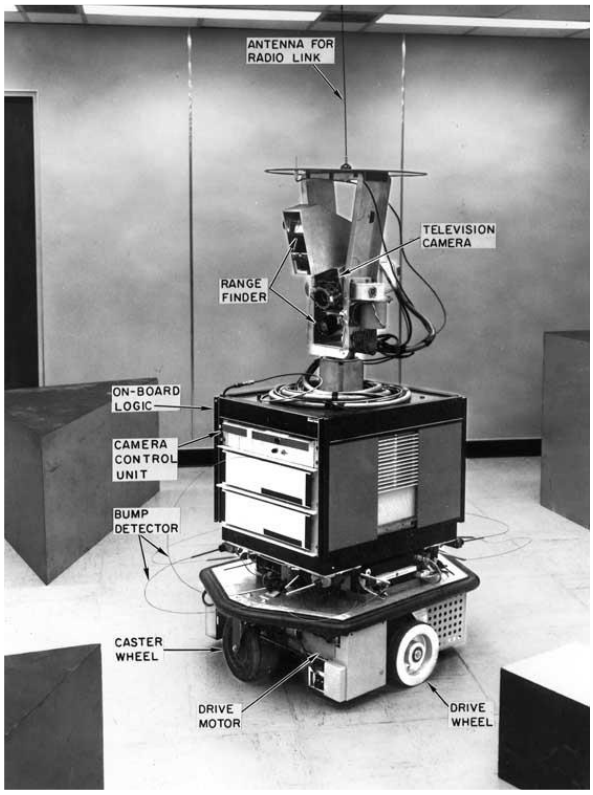


Figura 2.3: *Shakey*, o primeiro robô móvel (MORAVEC, 1999).

Para a solução desta desvantagem temos a robótica móvel, que é capaz de trabalhar com as habilidades de locomoção de um sistema robótico em que é possível o deslocamento no ambiente de interesse (SIEGWART; NOURBAKHSH, 2004) ; (FERREIRA, 2015). Na figura 2.3 pode ser visto o robô *Shakey* do Instituto de pesquisa de Stanford (*Stanford Research Institute*) nos anos 70, o primeiro robô móvel com capacidade de "raciocínio" sobre suas ações, e na figura 2.4 (a) pode ser observado um robô móvel de transporte de material automatizado da *Carnegie Mellon University*, utilizado para alimentar máquinas e robôs com peças em diferentes células robóticas, (b) robô enfermeiro hospitalar, utilizado para dar suporte aos enfermos (SECCHI, 2008).

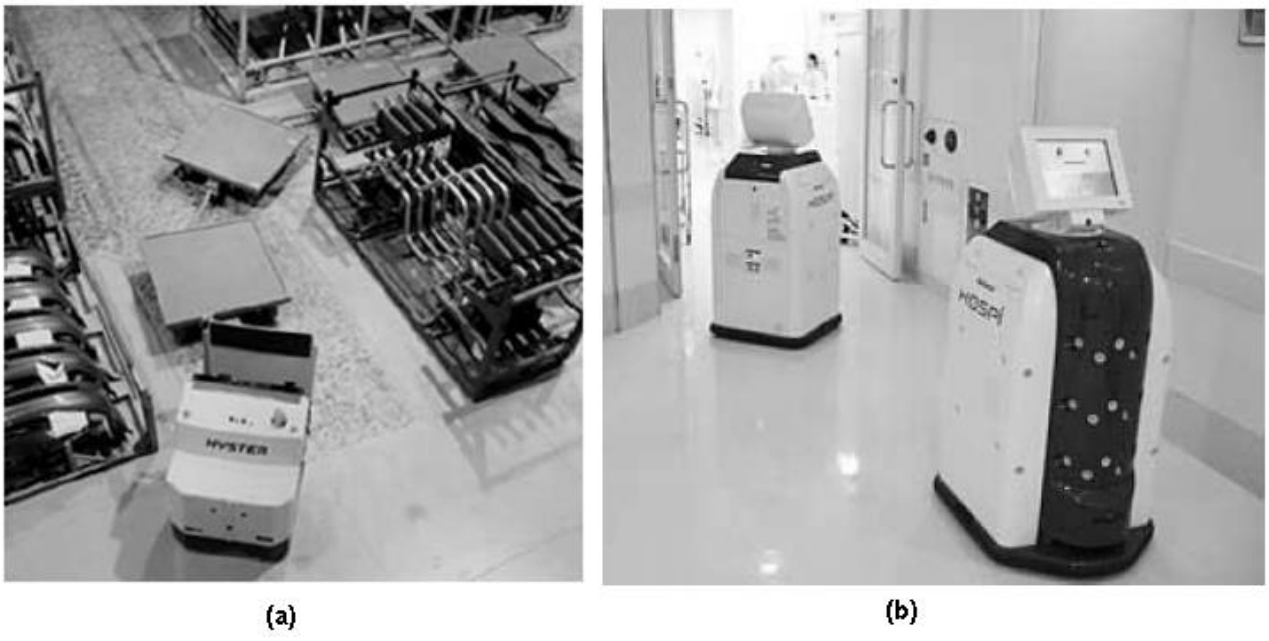


Figura 2.4: Exemplos de robôs móveis: (a) Sistema de Transporte de Material Automatizado (AMTS sigla em inglês) da *Carnegie Mellon University* (b) Robô enfermeiro Hospi desenvolvido pela empresa Matsushita. (SECCHI, 2008).

As técnicas de planejamento de rotas de robôs móveis autônomos são classificadas principalmente em duas categorias, sendo estas: métodos clássicos e métodos inteligentes (QU; XING; ALEXANDER, 2013) ; (FERREIRA, 2015). Para os métodos inteligentes, temos a computação bioinspirada e seus algoritmos baseados em técnicas abstraídas de vários tipos de sistemas naturais biológicos.

Os sistemas de computação bioinspirados, utilizam soluções heurísticas. Os métodos heurísticos são exploratórios, ou seja, não possuem uma especialidade para a resolução de um determinado problema (BUENO, 2009). Sendo assim, os métodos heurísticos fazem uma busca sobre a solução ótima, tendo em alguns casos como ponto de partida uma solução viável (ou uma região de interesse viável), e a partir desta, fazem aproximações para encontrar uma solução ótima para o problema a ser tratado (BUENO, 2009).

Os métodos heurísticos não são de baixo potencial para soluções de navegação robótica pelo fato de não serem determinísticos, e sim, podem ser comparados a inteligência humana, que utiliza o conhecimento *a priori* para tomar decisões potenciais em uma região de interesse (MOHANTY; PARHI, 2013) ; (FERREIRA, 2015). No gráfico da figura 2.5 é apresentado o uso das técnicas clássicas e heurísticas para navegação autônoma de robôs móveis nas últimas décadas. O gráfico deixa claro que os métodos heurísticos (inteligentes) têm sido mais utilizados nos últimos anos.

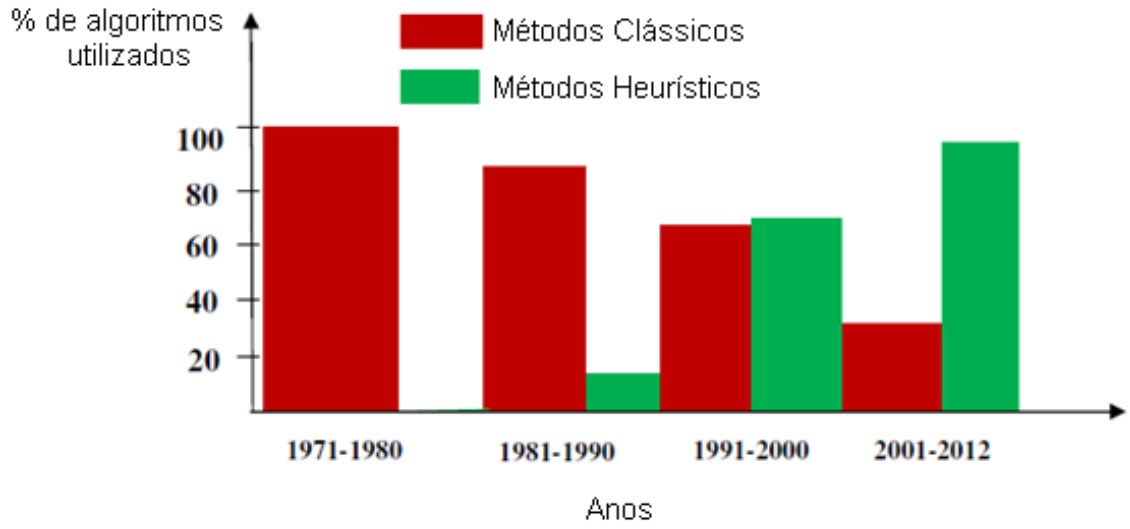


Figura 2.5: Aplicação de algoritmos clássicos e heurísticos com vistas para navegação autônoma de robôs móveis, adaptado de (MOHANTY; PARHI, 2013)

A computação bioinspirada tem por objetivo construir sistemas computacionais semelhantes aos seres vivos com vistas para a criação de algoritmos inspirados nesta natureza, resolvendo problemas dos mais diversos existentes na computação, sendo este um dos seus maiores potenciais (SIERAKOWSKI, 2006). Com esta lógica é possível desenvolver algoritmos inspirados em comportamento de grupos de seres vivos, comportamento de um sistema nervoso, comportamento de regras nebulosas desenvolvidas pela experiência de vida dos seres humanos e também por meio do comportamento da evolução da natureza (SIERAKOWSKI, 2006) ; (MOHANTY; PARHI, 2013).

Os algoritmos, baseados em sistema nervoso artificial, fazem a leitura de um sensoriamento externo, tendo seus dados de entrada tratados quando utiliza-se as RNAs (Redes Neurais Artificiais). Assim, cria-se um aprendizado sobre as experiências passadas, criando soluções com uma maior quantificação para um problema futuro. Este algoritmo pode ser nomeado como algoritmo de aprendizado, e para um RNA é essencial para o funcionamento inteligente (LUDWIG e MONTGOMERY, 2007).

A RNA é uma das mais antigas técnicas de I.A (inteligência Artificial) usadas hoje. Esta ferramenta computacional foi desenvolvida em meados de 1940, por Walter Pitts, um matemático e McCulloch, um neurofisiologista. Essa técnica teve como ideia principal

fazer uma analogia entre neurônios biológicos e circuitos eletrônicos, assim, sendo capaz de simular conexões sinápticas pelo uso de resistências elétricas variáveis e amplificadores (BARRETO, 1997) ; (LUDWIG e MONTGOMERY, 2007).

Na década de 1940 também surge a arquitetura dos computadores utilizados até hoje, os computadores de processamento sequencial, proposto por John von Neumann. A arquitetura de von Neumann (Von Neumann, 1958) levou a computação a um comodismo, por mostrar resultados aceitáveis, e com isso deixando as RNAs em segundo plano, reduzindo grandiosamente os cientistas envolvidos com modelos de computadores capazes de aprender. Mesmo assim von Neumann nunca descartou a evolução da computação aliada a modelos cerebrais (LUDWIG e MONTGOMERY, 2007). Entretanto, ele também tinha plena consciência de que a tecnologia disponível nas décadas de 40 e 50 do século passado não possibilitavam o desenvolvimento de máquinas com essas arquiteturas.

Para o entendimento de uma rede neural biológica, tem-se na figura 2.6 um neurônio, e logo seguinte, a sua explicação.

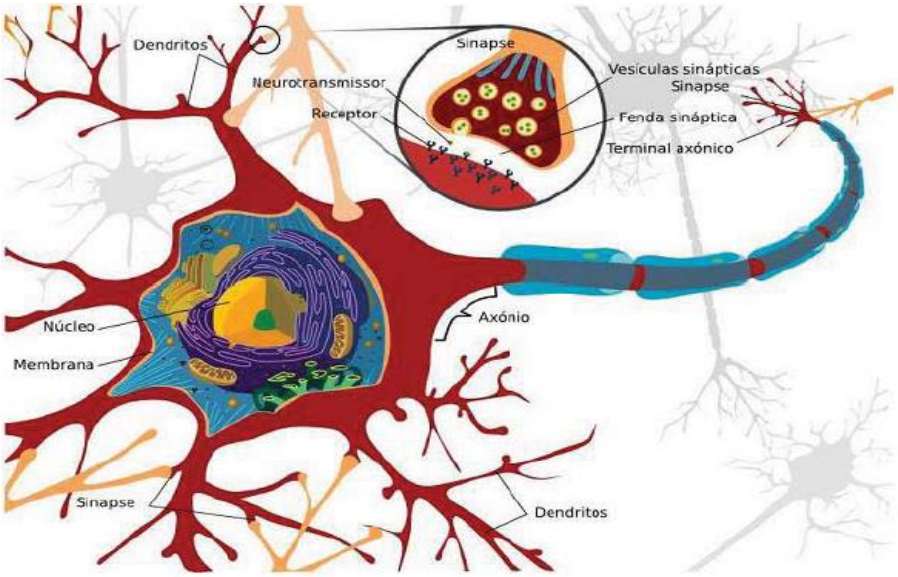


Figura 2.6: Modelo abstrato de um neurônio biológico, adaptado de (CHAVES, 2007)

Os principais componentes dos neurônios biológicos podem ser divididos em três partes (figura 2.6), e estas servem como base fundamental para a criação das RNA, sendo estas:

- Os dendritos têm como função receber estímulos (comunicação) de outros neurônios.

- O corpo do neurônio, que é mais conhecido como soma, que é responsável pela combinação de informações vindas de outros neurônios da rede.

- O axônio é formado por uma fibra tubular e pode alcançar grandiosas distâncias (metros) tendo como objetivo transmitir os estímulos para outras células(LUDWIG e MONTGOMERY, 2007).

Em um trabalho feito por Olivi (2005) foi utilizado RNA "*Back Propagation*" para o desvio de obstáculos em um ambiente dinâmico utilizando um robô *Khepera* (apenas em nível de simulação). Já no trabalho de Tripathi and V.Rihani (2012), foi utilizada uma rede do tipo *Hopfield* para modelar o ambiente *a priori* e, com isso, o robô é capaz de se movimentar de forma autônoma no ambiente para o qual foi treinado.

O trabalho de Pessin e Osório (2007) teve como objetivo principal criar um controlador robusto baseado em RNAs para rota de robôs móveis autônomos, utilizando um ambiente misto, com obstáculos estáticos e dinâmicos. O modelo proposto de RNA mostrou ser capaz de controlar os mecanismos (atuadores) do robô móvel em questão, usando apenas informações disponíveis localmente, obtidas pelos sensores em um ambiente dinâmico, com terreno irregular e a RNA mostrou por meio de simulações, ser capaz de controlar o robô móvel.

Outro estudo relacionado às RNAs e aos AGs deste trabalho foi desenvolvido por Heinen e Osório (2006), utilizando uma RNA para controlar o deslocamento de um robô móvel com pernas apresentando um problema, sendo esta a limitação de que não é possível se obter de antemão informações locais para o cálculo do gradiente e a correção dos erros e isto tem um problema potencial que é a não possibilidade de uso de algoritmos de aprendizado supervisionado tradicionais. Para a correção deste problema, foi utilizado um AG para a evolução dos pesos sinápticos, com a grande vantagem de que um AG não precisa de informações locais para a correção dos erros, em outras palavras, não necessitam de uma base de dados de treinamento.

O algoritmo de Heinen e Osório (2006) também é neurogenético, porém não tem a mesma funcionalidade do controlador desenvolvido neste trabalho, pois o AG é utilizado para o auxílio ao treinamento da RNA. Já no algoritmo neurogenético desenvolvido neste trabalho, o AG tem a função de gerar uma nova solução de desvio para o robô caso a RNA não consiga reconhecer o obstáculo e encontrar uma solução em seu banco de conhecimento. Para este caso, o AG trabalha de maneira secundária a RNA e só é chamado em segundo plano, pois seu custo de tempo é relativamente maior que a solução encontrada pela RNA.

O trabalho desenvolvido por Ayrosa (2011) teve como objetivo empregar uma rede neural no tratamento de colisões com obstáculos em um ambiente dinâmico e desconhecido. O objetivo do trabalho é utilizar a RNA para aprender e se adaptar a novas condições ambientais. O controlador mostrou-se eficiente para diversos ambientes após serem feitas simulações em quinze situações distintas.

O trabalho de Ayrosa (2011) teve como objetivo utilizar uma RNA como algoritmo único para o controle da navegação do robô, sendo que esta capacidade é dada a característica de reaprendizado da RNA utilizada. Se comparado ao algoritmo

neurogenético desenvolvido neste trabalho, uma diferença marcante seria a incapacidade da RNA utilizada em reaprender em tempo real, pois o algoritmo de aprendizado só é funcional no momento da troca de ambientes em que o robô tem a função de realizar seu trabalho.

Os algoritmos Genéticos (AGs) aqui apresentados, são algoritmos evolucionários inspirados na natureza, pois utilizam a lógica da evolução. Como exemplo, temos a seleção que o algoritmo faz de melhores indivíduos para se recombinarem e gerarem novos indivíduos “melhores” no futuro (SAMADI, OTHMAN, 2013).

O AG pode ser classificado como uma técnica evolucionária (bioinspirada) que tem como base a heurística global, onde as abordagens são dadas pela probabilidade, e não determinísticas (BECKMANN, 2010) ; (FERREIRA, 2015). Foi proposto por John Holland e se popularizou com o trabalho de David Edward Goldberg (GOLDBERG, 1989) ; (FERREIRA, 2015). Abstrair e explicar rigorosamente o processo adaptativo de sistemas naturais e desenvolver programas artificiais que possuíssem as importantes ferramentas e mecanismos de sistemas biológicos foi o foco das pesquisas de John Holland (GOLDBERG, 1989) ; (FERREIRA, 2015). Na figura 2.7 é apresentado um fluxograma de um AG clássico.

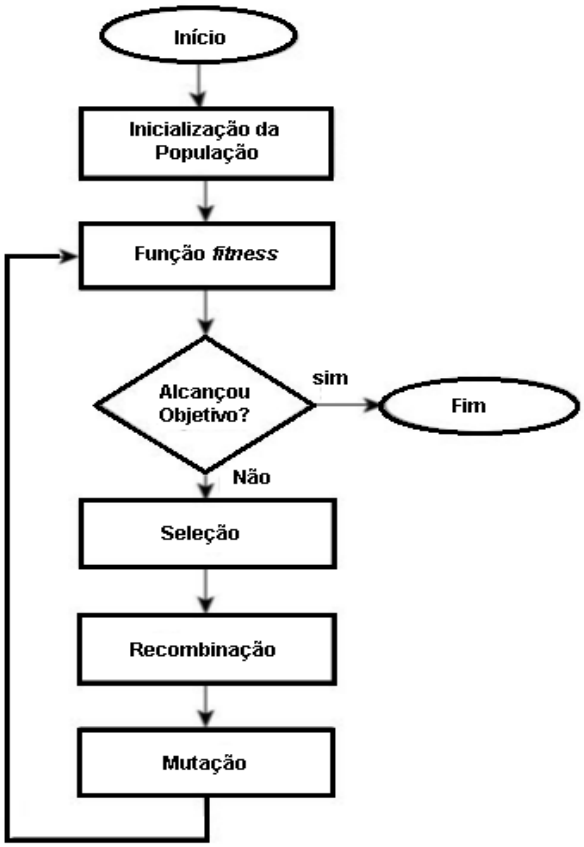


Figura 2.7: Fluxograma de um AG tradicional, adaptado de (QU; XING; ALEXANDER, 2013)

O diagrama de fluxo apresentado na figura 2.7 pode ser explicado da seguinte forma: A primeira etapa é gerar uma população definida dentro de uma região de interesse. O segundo passo é quantificar estes indivíduos com a função *fitness.*

Um critério de parada deve ser definido para que se encontre uma solução ótima global. O algoritmo utiliza os operadores genéticos até que uma solução seja encontrada e o critério de parada finalize as operações genéticas (SIERAKOWSKI, 2006) ; (FERREIRA, 2015).

Um trabalho feito por Purian e Farokhi (2013) faz uma comparação entre um AG e um algoritmo de colônia de formigas, onde ambos obtiveram sucesso em ambientes dinâmicos, encontrando um caminho viável utilizando seus critérios de otimização (FERREIRA, 2015).

Os resultados implicam que o algoritmo de colônia de formigas, em comparação com o AG, tem um tempo menor para encontrar a solução ótima, pois trabalha com um número menor de iterações. Em contrapartida, o AG apresenta melhor desempenho em ambientes com maior complexidade e dinâmica de obstáculos (PURIAN, FAROKHI, 2013).

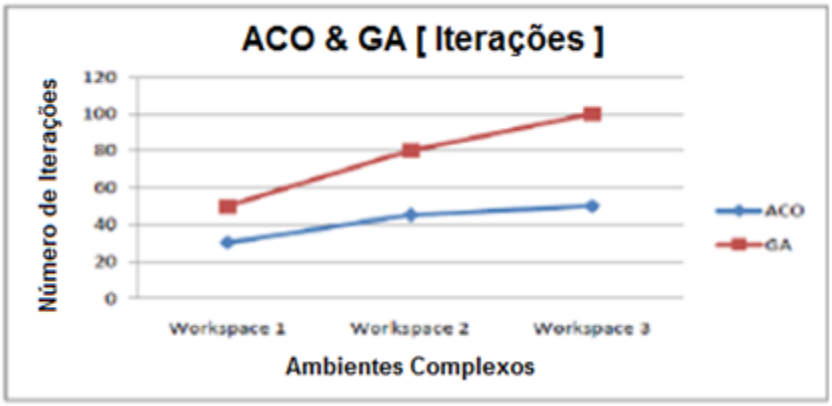


Figura 2.8: Comparação do número de iterações na colônia de formigas e algoritmos genéticos(PURIAN, FAROKHI, 2013).

Em outras palavras, em todo ambiente dinâmico para robôs, o algoritmo de colônia de formigas trabalha com menor número de iterações se comparado ao algoritmo genético.

No entanto, estudos com este tipo de algoritmo mostram que o método de otimização colônia de formigas não trata satisfatoriamente os obstáculos de maior problemática, na maioria dos casos obstáculos dinâmicos (PURIAN, FAROKHI, 2013).

Sendo assim, o algoritmo de colônia de formigas mostrou menor eficiência em ambientes complexos com maior dinâmica de obstáculos se comparado aos AGs. Estas características podem ser melhor observadas nos gráficos da figura 2.8 e 2.9 (PURIAN, FAROKHI, 2013) ; (FERREIRA, 2015).

No algoritmo neurogenético apresentado neste trabalho uma solução para a problemática de obstáculos dinâmicos foi utilizar uma RNA *perceptron* para tratar objetos de maior frequência encontrados pelo robô. A RNA aplicada à deficiência do algoritmo de colônia de formigas e do algoritmo genético em relação a obstáculos problemáticos foi um dos maiores objetivos deste trabalho.

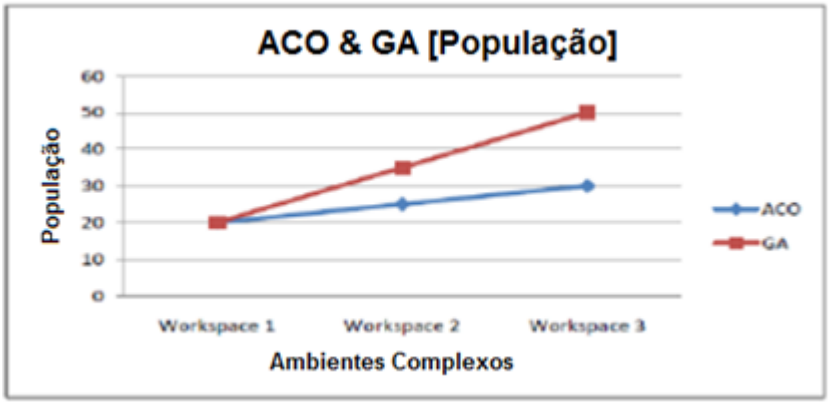


Figura 2.9: Comparação da população da colônia de formigas e algoritmos genéticos para diferentes ambientes dinâmicos (PURIAN, FAROKHI, 2013).

Nos últimos anos, estudos voltados à solução de problemas de planejamento de rota basearam-se em AGs com variação dos operadores genéticos, procurando sempre um desempenho de maior potencial (TUNCER; YILDIRIM, 2012) ; (FERREIRA, 2015).

Um trabalho desenvolvido por Ferreira (2015) propôs um AG clássico da literatura para a navegação autônoma de um robô móvel. O trabalho teve como objetivo maior a escolha dos melhores operadores genéticos para o planejamento de rota de robôs móveis, testando e comparando os métodos existentes, assim fazendo a escolha dos operadores por valores quantificados. O algoritmo para este caso teve como tarefa encontrar uma trajetória possível em um ambiente dinâmico, com a preocupação da minimização do custo computacional, percorrendo a menor rota possível (FERREIRA, 2015).

Em um trabalho desenvolvido por Panda e Choudhury (2015) também foi proposto um AG clássico da literatura para a navegação autônoma de um robô móvel. O trabalho teve como objetivo definir um novo método de representar um cromossomo. Este método de representação utiliza uma matriz binária para definir a estrutura de cada um dos cromossomos que serão manipulados por meio dos operados genéticos (PANDA ; CHOUDHURY, 2015).

Um método de recombinação diferenciado também é utilizado no trabalho de Panda e Choudhury (2015), este método é intitulado como recombinação sequencial, não utilizando, portanto, um método de seleção clássico para encontrar os melhores indivíduos para sofrerem a recombinação, pois, apenas são selecionados os melhores indivíduos em uma lista que foi classificada pela quantização de cada um deles, por meio

da função *fitness.* Os dois melhores indivíduos da lista sofrem a recombinação simples. Para a validação do algoritmo, são definidas rotas de navegação a serem cumpridas em um ambiente dinâmico e, para isso, são utilizados em alguns casos, mais de um robô ao mesmo tempo, para que as tarefas de navegação, sejam divididas para cada um deles (PANDA ; CHOUDHURY, 2015).

Se comparado o controlador desenvolvido por Panda e Choudhury (2015) com o controlador neurogenético desenvolvido neste trabalho, podem ser constadas duas potenciais diferenças: a primeira está ligada a otimização do algoritmo genético utilizando uma estrutura de cromossomo diferente e a segunda está ligada à desvantagem que o algoritmo de Panda e Choudhury (2015) tem em relação ao seu método de seleção, já que este método, apesar de ter uma grande simplicidade, não deve garantir um custo computacional e de tempo inferior ao método de seleção por torneio, método que foi utilizado no algoritmo neurogenético desenvolvido neste trabalho. O motivo destes maiores custos está ligado ao uso de uma estrutura de dados do tipo "fila" que deve ser reordenada toda vez que um novo indivíduo é inserido ou retirado. Já no método de seleção por torneio, este problema não existe, já que os indivíduos que não foram selecionados como os melhores, são descartados e não reclassificados.

Uma outra desvantagem deste algoritmo (PANDA ; CHOUDHURY, 2015) é que ele não possui um algoritmo auxiliar para solucionar obstáculos de maior frequência, assim, como foi utilizado no algoritmo híbrido desenvolvido neste trabalho. As melhorias obtidas por meio de um algoritmo auxiliar foram analisadas no decorrer desta dissertação, quando apresentado o desenvolvimento do trabalho. Um potencial significante deste trabalho foi o ajuste preciso da taxa de mutação do AG. Este ajuste serviu como base para o AG utilizado no algoritmo neurogenético desenvolvido neste trabalho.

Em um trabalho desenvolvido por Abinaya e Kumar (2014) também é proposto um controlador híbrido para a navegação autônoma de um robô móvel. O controlador utiliza um AG para gerar os pontos de desvio de rota, quando um obstáculo é encontrado e o algoritmo GJK (Gilbert-Johnson-Keerth) que é um método interativo de cálculo de distância entre objetos convexos (GILBERT, 1988), utilizado neste trabalho para calcular a distância dos obstáculos em relação ao robô. A proposta de Abinaya e Kumar (2014) é combinar o algoritmo GJK e o AG para trabalharem em conjunto em prol a navegação do robô, sendo o AG utilizado para o planejamento de rota global e o algoritmo GJK, para evitar obstáculos (ABINAYA ; KUMAR, 2014).

O controlador deve utilizar o AG para planejar as rotas auxiliares de desvio, usando para quantificar cada solução, uma função *fitness* baseada em distâncias, assim, calculando a distância do robô em relação aos seus pontos de desvio. O GJK deve calcular continuamente a posição dos obstáculos em relação ao movimento do robô, assim, alterando o planejamento de rota do AG. Por meio das distâncias de cada obstáculo em relação a posição atual do robô, o controlador deve encontrar um caminho ideal e livre de colisões com obstáculos (ABINAYA ; KUMAR, 2014).

Se comparado o controlador híbrido desenvolvido por Abinaya e Kumar (2014) com o controlador neurogenético desenvolvido neste trabalho, pode ser constatado que o controlador de Abinaya e Kumar (2014) utiliza o algoritmo GJK para auxiliar o AG, assim, fazendo uma abordagem sobre os obstáculos que estão impedindo a rota atual do robô. Então o algoritmo GJK deve ser utilizado como um sistema auxiliar, assim, trabalhando em prol da otimização das rotas auxiliares de navegação que devem ser geradas pelo AG. Já no controlador neurogenético proposto neste trabalho, o algoritmo neural, consegue gerar soluções independentes ao AG.

Em um trabalho desenvolvido por Fetanat e Haghzad (2015), são apresentados três métodos evolutivos para a otimização do planejamento de rota de um robô móvel em um ambiente dinâmico. Os três métodos evolutivos utilizados foram: Algoritmo de Pesquisa Padrão (APP), Otimização por Enxame de Partículas (OEP) e o Algoritmo Genético (AG). Estes três métodos foram comparados e utilizados para encontrar uma rota ideal nos ambientes em que se tem a necessidade de desviar de obstáculos para atingir os pontos de objetivo (FETANAT; HAGHZAD, 2015).

O APP mostrou ser mais rápido para encontrar uma solução de desvio em relação aos outros dois algoritmos. O algoritmo de OEP mostrou soluções de desvio com menores rotas, sendo este um grande potencial para a navegação autônoma, já que o menor caminho é sempre procurado, assim, reduzindo o custo de energia do sistema robótico. Em contrapartida este algoritmo é que tem o maior custo de tempo (FETANAT; HAGHZAD, 2015).

O AG foi o método que apresentou um balanceamento das vantagens apresentadas pelos outros dois métodos evolutivos, ou seja, este algoritmo gera soluções em um tempo aproximado ao APP e soluções com um nível de otimização próximo a OEP. Outra vantagem do AG está ligada à sua alta capacidade em explorar o espaço de buscas, assim, conseguindo encontrar potenciais soluções. Este potencial tem ligação com sua capacidade de otimização de rotas para desvio de obstáculos (FETANAT; HAGHZAD, 2015).

Se comparado o trabalho de Fetanat e Haghzad (2015) e seus métodos de controle com o controlador neurogenético desenvolvido neste trabalho, pode ser constatado que o principal objetivo do trabalho de Fetanat e Haghzad (2015) está ligado ao levantamento dos potencias de cada um dos algoritmos evolutivos utilizados, não levando em consideração nenhuma aplicação para otimizar estes algoritmos, assim, como foi feito por Panda e Ferreira (2015). Já no controlador neurogenético proposto neste trabalho, foram feitos estudos para ajustar tanto o AG como a RNA, assim, sendo possível conseguir o desempenho esperado para o tipo de problema a ser tratado.

Em um trabalho desenvolvido por Abbaspour e Alpour (2015), foi utilizado um conjunto de três robôs com rodas para a manipulação de um mesmo objeto, grande e pesado. O conjunto de robôs é utilizado para transportar o objeto a partir de uma posição inicial para uma posição de destino. Para cumprir esta função de transporte de objeto em conjunto, é

utilizado um sistema abrangendo duas técnicas bioinspiradas, sendo estas o AG e a OEP (ABBASPOUR; ALPOUR, 2015).

O funcionamento do sistema com múltiplos robôs utiliza uma estratégia de robô líder, onde um dos robôs é considerado como líder (mestre), enquanto os outros agem como seus seguidores (escravos). O robô mestre fornece aos robôs escravos as coordenadas de navegação que devem serem seguidas por eles, assim, sendo possível que os três robôs façam a mesma rota, em conjunto, em prol ao carregamento do objeto (ABBASPOUR; ALPOUR, 2015).

Para otimizar o posicionamento do conjunto de robôs em relação ao objeto, a melhor disposição deste conjunto deve ser encontrada. Para cada tipo de objeto a ser carregado pelo conjunto de robôs, uma nova configuração da disposição dos robôs deve ser otimizada. Para este fim, o AG e a OEP são utilizadas (ABBASPOUR ; ALPOUR , 2015). Os robôs devem trabalhar em conjunto para que consigam carregar o objeto, unindo suas forças.

Se comparado o trabalho de Abbaspour e Alpour (2015) ao algoritmo neurogenético desenvolvido neste trabalho, a aplicação final é a mesma, sendo esta, o carregamento de objetos para pontos pré-definidos, porém, a função dos algoritmos bioinspirados nestes dois trabalhos são diferentes. O trabalho de Abbaspour e Alpour (2015) não especificou um controlador para a navegação do conjunto de robôs, apenas definiu um método de otimização da disposição entre os robôs para que trabalhem em conjunto, sendo possível, por exemplo, o uso do controlador neurogenético desenvolvido neste trabalho para auxiliar o conjunto de robôs em sua navegação.

Em um trabalho desenvolvido por Santhosh (2014), também foi utilizado um AG para a otimização de rotas em prol a navegação de um robô móvel. O robô utilizado neste trabalho tem como objetivo navegar em ambientes de guerra para encontrar vítimas, assim, evitando que um humano seja utilizado para o resgate neste tipo de ambiente cheio de riscos (SANTHOSH, 2014).

O robô proposto por Santhosh (2014) utiliza um sistema de visão que trabalha com uma câmera de radiação para a detecção de corpos com vida. Este processo é feito de maneira automática e quando um sinal de vida é constatado, um sinal de alerta é enviado para um computador supervisório. Também é utilizado um sensoriamento para a detecção de vazamento de gás e altas temperaturas (SANTHOSH, 2014).

Se comparado o trabalho de Santhosh (2014) com o controlador neurogenético desenvolvido neste trabalho, pode ser constatado que o sistema de controle do robô de Santhosh (2014) utilizou um AG clássico da literatura para a navegação de um robô, visando o desenvolvimento de um sistema de sensoriamento equipado com uma câmera de radiação e sensores de detecção de altas temperaturas e de vazamento de gás em prol ao salvamento de vidas. Já o controlador neurogenético desenvolvido neste trabalho tem como maior objetivo auxiliar a navegação autônoma do robô móvel.

Como descrito acima, o trabalho de Santhosh (2014) não teve como maior objetivo o desenvolvimento de um sistema de navegação, assim, o seu sistema de sensoriamento pode ser implementado em conjunto ao controlador neurogenético desenvolvido neste trabalho, para que a união dos potenciais destes dois sistemas robóticos seja utilizada em prol ao salvamento de vidas em ambientes que são prejudiciais para um soldado de resgate.

Em um trabalho desenvolvido por Khelchandra e Huang (2014), foi apresentado um controlador híbrido baseado em algoritmos bioinspirados para o planejamento de rotas de robôs móveis. O controlador é aplicado na escolha de um caminho livre de colisões com obstáculos, a partir de um conjunto de *n* possíveis caminhos cadastrados em um mapa. O mapa de caminhos é gerado por meio de uma RNA (KHELCHANDRA ; HUANG, 2014).

Também é utilizada a lógica *fuzzy* para evitar caminhos sujeitos a colisões, quando os *n* caminhos cadastrados no mapa são bloqueados por obstáculos, sendo necessária a geração de um novo caminho, livre de obstáculos. Por fim, é utilizado um Algoritmo Genético (AG) que é aplicado para otimizar rotas alternativas livres de colisões com obstáculos, assim, o AG utiliza como base a lógica *fuzzy* (KHELCHANDRA ; HUANG, 2014).

O AG deve gerar um conjunto de soluções nebulosas para o sistema de lógica *fuzzy*, por meio de uma base de regras cadastrada em um banco de dados do controlador de navegação robótico (KHELCHANDRA ; HUANG, 2014). O sistema *fuzzy* trabalha em conjunto com o AG, sempre a partir de dois caminhos cadastrados no mapa neural, assim, gerando um terceiro caminho livre de obstáculos.

Por meio dos resultados do controlador de Khelchandra e Huang (2014), foi possível constatar que a combinação destas técnicas de inteligência artificial é computacionalmente eficiente para eliminar as limitações individuais de cada uma das técnicas utilizadas (KHELCHANDRA ; HUANG, 2014).

Se comparado o trabalho de Khelchandra e Huang (2014) com o controlador neurogenético desenvolvido neste trabalho, uma potencial desvantagem pode ser constatada, já que o algoritmo híbrido proposto por Khelchandra e Huang (2014) é aplicado somente em ambientes com obstáculos estáticos. Uma vantagem apresentada no trabalho de Khelchandra e Huang (2014), se aplicado em ambientes com obstáculos estáticos, é sua capacidade em tomar decisões de desvio de rotas baseadas em um mapa, assim, reduzindo o custo computacional e de tempo para encontrar uma rota alternativa.

Uma comparação com maior abrangência foi dispensada, já que o algoritmo neurogenético desenvolvido neste trabalho é aplicado em ambientes com obstáculos dinâmicos e, *a priori*, não utiliza um mapa para dar um conhecimento para o robô.

O algoritmo de Ferreira (2015) foi utilizado como base do AG aplicado ao controlador neurogenético deste trabalho. A RNA foi aplicada em conjunto com este algoritmo para

trazer algumas melhorias para o desempenho do controlador em alguns ambientes que utilizam objetos que se tornaram possivelmente obstáculos de maior problemática para o robô. Uma explicação detalhada para o algoritmo de Ferreira (2015) foi feita no capítulo 3.

Na maioria dos casos, ambientes dinâmicos possuem obstáculos de maior frequência, seja um ambiente natural com árvores e animais, ou até mesmo ambientes artificiais como por exemplo um estacionamento de supermercado com carrinhos de compras, pessoas e automóveis. Em ambientes com o conhecimento *a priori* do tipo de obstáculos que serão encontrados com maior frequência é que o algoritmo neurogenético desenvolvido neste trabalho deve ter um diferencial aos trabalhos citados neste texto.

Um trabalho feito por Shiltagh e Jalal (2013) apresenta uma modificação dos algoritmos genéticos, o estudo investiga o planejamento de caminhos para um robô móvel, aplicando uma variação dos AGs conhecida como AGM (Algoritmo Genético Modificado). O algoritmo tem como principal objetivo, tratar o problema da população inicial que contém muitos indivíduos que geram caminhos inviáveis. Para este caso, um algoritmo auxiliar, intitulado como AD (Algoritmo de Distinção) é utilizado para verificar se os indivíduos gerados inicialmente são viáveis ou inviáveis. Se inviável, o indivíduo é eliminado de imediato, evitando que este seja manipulado pelos operadores genéticos, assim reduzindo o custo e tempo computacional (SHILTAGH, JALAL, 2013).

O AGM proposto para este trabalho mostrou ser capaz de guiar um robô em movimento a partir da posição inicial até o seu objetivo, encontrando a solução ótima, o melhor caminho possível, sem sofrer colisões com obstáculos em ambientes declarados como complexos (SHILTAGH, JALAL, 2013).

O AG é uma técnica que tem grande potencial para o planejamento de rotas de robôs móveis autônomos, pois possui maior capacidade em explorar o ambiente preservando as melhores soluções já encontradas (ALLAIRE F.C.J.; LABONTÉ; FUSINA, 2009) ; (FERREIRA, 2015), e sendo de maior robustez para casos complexos, e teve tempo de convergência inferior as outras técnicas propostas (TUNCER; YILDIRIM, 2012).

Assim como o trabalho desenvolvido por Ferreira (2015), existem outros na literatura que implicam em trazer melhorias aos operadores genéticos, com vários objetivos, sendo os mais destacados para rota de robôs: Tempo de convergência, custo computacional e criar o menor caminho possível entre o ponto inicial e o destino, sendo este último o ponto de objetivo do robô.

Como exemplo, um trabalho desenvolvido por Shi e Cui (2010) teve como objetivo fazer um tratamento na função *fitness*, garantindo uma confiabilidade e também a escolha da melhor rota possível. Como para este caso trata-se de um caso de minimização, a menor rota é a melhor. O tratamento da função foi feito em partes (sub-funções): (1) cálculo do comprimento do caminho; (2) segurança; (3) e a última pela “suavidade” da trajetória. Sendo que a sub-função (2), tem uma maior relevância e é utilizada para analisar se é viável ou não a solução adotada para o obstáculo (SHI; CUI, 2010) ; (FERREIRA, 2015).

Outra forma de controle de sistemas robóticos é a lógica *fuzzy*. A ideia de desenvolver sistemas nebulosos aconteceu no início da década de 60 por Lofti A. Zadeh (SIERAKOWSKI, 2006). A ideia teve o propósito de desenvolver uma nova ferramenta matemática que não fosse totalmente precisa para os problemas complexos existentes no mundo real. Com esta ideia surgiu a lógica *fuzzy*, que tem grande potencial para o aprendizado de um sistema (SIERAKOWSKI, 2006).

A lógica *fuzzy* é aplicada a sistemas robóticos, cujas decisões serão tomadas a partir de uma base de dados que foi criada com o auxílio de um estudo feito por meio de uma experiência abstraída de maneira empírica do sistema a ser controlado. As inferências são feitas a partir de informações registradas neste banco de dados que é correspondente a base de regras do controlador robótico (SIERAKOWSKI, 2006).

O controlador *fuzzy* é bastante utilizado nas novas gerações de controladores inteligentes para robôs móveis autônomos. A lógica nebulosa, ou, como é mais conhecida, a lógica *fuzzy* é baseada em regras que trabalham com o propósito de transformar uma base de conhecimento adquirido ao longo de um tempo, geralmente por um humano especialista no sistema e posteriormente passado para o controlador automático (Khoury, et al., 2004).

Um trabalho desenvolvido por Cabreira (2012) utilizou um AG em conjunto com um controlador *fuzzy* para a rota autônoma de robôs. A função *fuzzy* aplicada neste algoritmo teve como objetivo tomar decisões prévias quando um obstáculo é encontrado, assim, antecipando o cálculo de rota alternativa para a solução do desvio do obstáculo.

O algoritmo é viável para ambientes pequenos e com poucos obstáculos, uma vez que, se o ambiente tiver alta complexidade, é inviável a construção de um banco de conhecimento com tanta informação. Para o caso de ambientes de alta complexidade é compensador utilizar o AG “puro”, pois o AG tem maior eficiência em ambientes com maior número de obstáculos exclusivos (CABREIRA; DIMURO; AGUIAR, 2012).

Este problema também tem ligação ao algoritmo neurogenético desenvolvido neste trabalho, já que a troca do ambiente dinâmico atual que o robô está "treinado" a trabalhar com sua RNA necessita de um novo treinamento da rede. Em contrapartida a esta desvantagem o AG deve operar com maior frequência quando o ambiente ficar "desconhecido" para o robô.

Se um novo treinamento para o reconhecimento de obstáculos de maior frequência não for feito, a RNA deve ficar impossibilitada de realizar sua função de maneira precisa, assim, o seu uso deve apenas aumentar o custo do algoritmo sem retorno algum, pois todas as soluções, neste caso, serão geradas pelo AG.

Os seres humanos utilizam potencialmente um grande número de funções mentais cerebrais em prol das suas atividades vivenciais sem a necessidade de cálculos de grande complexidade e/ou grande conhecimento *a priori* sobre as tarefas que está executando. Para implementar estas características de grande potencial existentes na inteligência humana, pode-se utilizar um controlador *neuro-fuzzy*, que fornecem meios

para se chegar a resultados aceitáveis para este caso (LUDWIG e MONTGOMERY, 2007) ; (MOHANTY; PARHI, 2013).

A lógica *fuzzy* oferece uma metodologia adequada para representar e implementar a especialidade humana, utilizando uma base de conhecimento (MOHANTY; PARHI, 2013). Com a lógica *fuzzy*, o raciocínio humano e suas tomadas de decisões podem ser criadas por um conjunto de regras do tipo SE-ENTÃO, unidas com as representações linguísticas cabíveis para cada caso (MOHANTY; PARHI, 2013). A RNA, trabalhando em conjunto com o controlador *fuzzy,* pode aliar a sua capacidade de reconhecimento e aprendizagem com a base de conhecimento da lógica *fuzzy* e suas regras (MOHANTY; PARHI, 2013).

Um controlador *neuro-fuzzy* para a navegação autônoma de um grupo de robôs móveis foi utilizado por Mohanty e Parhi (2013). A técnica utilizada, *neuro-fuzzy*, apresenta uma rede neural artificial trabalhando como um pré-processamento para um controlador de lógica *fuzzy* (Figura 2.10).

Na figura 2.10 é apresentada a estrutura de um controlador *neuro-fuzzy.* Este controlador tem como entrada uma RNA que deve desenvolver as funções de reconhecimento e aprendizagem deste algoritmo e enviar estes dados para um algoritmo *fuzzy,* que utilizará sua base de conhecimento e seus conjuntos de regras para tomar decisões com base em conhecimentos passados (MOHANTY; PARHI, 2013) .

Na figura 2.10 podem serem observadas as entradas da RNA para os sensores de visão frontal, lateral direito, lateral esquerdo e o ângulo utilizado pelo sensor.

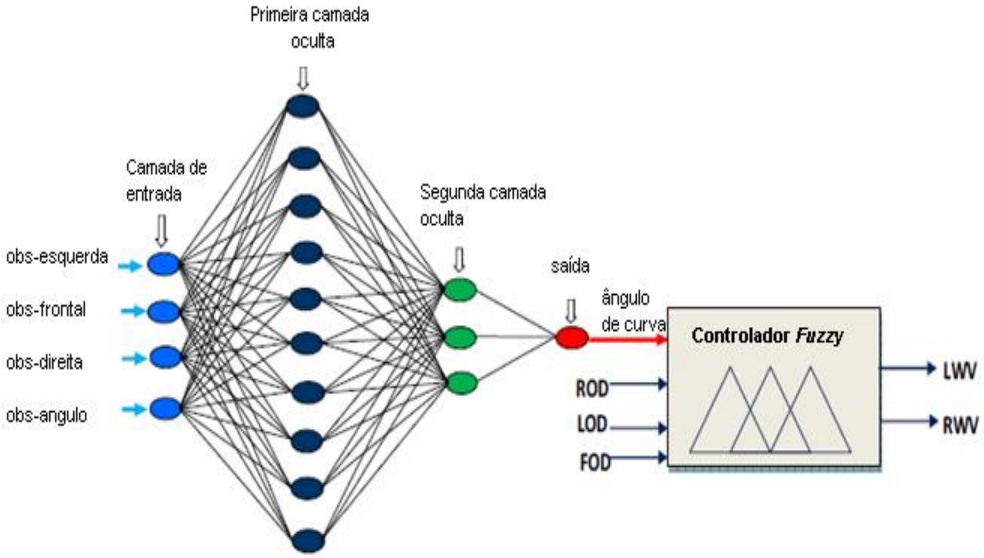


Figura 2.10: Controlador *neuro-fuzzy*, adaptado de (MOHANTY; PARHI, 2013)

A RNA tem, em sua estrutura, apenas duas camadas ocultas, e gera apenas uma saída que deve ser a entrada do controlador *fuzzy.*

Um trabalho desenvolvido por Mendonça (2013) utilizou esta lógica em seu controlador híbrido com o objetivo de conseguir uma navegação livre de obstáculos. Explicando de maneira simplificada, o controlador utiliza uma RNA em conjunto com um controlador *fuzzy*.

O controlador *fuzzy* foi utilizado para armazenar dados históricos de navegação. Os dados obtidos pelo controlador *fuzzy* são aplicados para o treinamento de uma RNA, tornando híbrido o controlador do robô (MENDONÇA, 2013). Este controlador tem a ordem de uso dos algoritmos de maneira inversa ao controlador proposto por MOHANTY e PARHI (2013).

Em um trabalho desenvolvido por Lyrio e Santos (2015), foi utilizado um sistema de mapeamento e localização baseado em um sistema de visão em conjunto com uma RNA. O sistema é análogo a um sistema de percepção humano, onde imagens (nomes de ruas, placas, árvores, etc) podem ser memorizadas para que se tornem informações que serão associadas a lugares onde o humano já esteve e possa criar uma trajetória em seu ambiente atual, com base em um conhecimento adquirido no passado (LYRIO ; SANTOS, 2015).

O sistema desenvolvido por Lyrio e Santos (2015), foi utilizado em um automóvel que fez um caminho de 3,75 quilômetros em torno do campus principal da Universidade Federal do Espírito Santo. O sistema gerou valores de posicionamento com erros abaixo de 1,5 metros (LYRIO ; SANTOS, 2015).

O sistema de navegação desenvolvido por Lyrio e Santos (2015), utiliza uma arquitetura neural para a construção de um mapa do ambiente, que é obtido por meio das imagens aplicadas a um banco de conhecimento da RNA (LYRIO ; SANTOS, 2015).

A geração do mapa tem como base as imagens capturadas *a priori* pelo sistema de visão, assim, classificando-as por meio da RNA e sendo possível o mapeamento e localização do robô móvel. Durante a navegação do robô, a RNA deve receber as imagens do ambiente e utilizar o conhecimento previamente adquirido (mapa neural) para recuperar a posição atual do robô. Isso é feito, por meio dos marcos associados a cada imagem referente ao ambiente, tendo as imagens sido capturadas durante a construção do mapa (LYRIO ; SANTOS, 2015).

Se comparado o controlador de Lyrio e Santos (2015), com o controlador neurogenético desenvolvido neste trabalho, o sistema de visão em conjunto com um algoritmo neural aplicado nos dois sistemas de navegação, desenvolveram suas funções em prol dos mesmos objetivos, ou seja, o sistema de visão deve capturar uma imagem que deve ser classificada por meio de um algoritmo neural.

O trabalho de Lyrio e Santos (2015), utiliza este mecanismo de classificação baseado em uma RNA para o mapeamento e localização de um robô em seu ambiente de navegação (LYRIO ; SANTOS, 2015), já o controlador neurogenético desenvolvido neste trabalho,

utiliza um mecanismo semelhante a este para reconhecer obstáculos e solucioná-los com rotinas de desvio previamente cadastradas em um banco de conhecimento.

O sistema de visão desenvolvido por Lyrio e Santos (2015) utiliza pontos de referência nas imagens capturadas para cadastrá-las como padrões para a RNA, assim, sendo possível um posterior reconhecimento de novas imagens, por meio do mapa neural (LYRIO ; SANTOS, 2015). Já no algoritmo neurogenético utilizado neste trabalho, a extração de características pelo sistema de visão é dada por meio das variáveis de cor e tamanho, assim, sendo possível um posterior reconhecimento de novos obstáculos encontrados pelo robô.

Em um trabalho desenvolvido por Bessa e Barroso (2015), foi feita uma comparação de algumas técnicas de visão omnidirecional utilizadas para o mapeamento de ambientes e localização de robôs móveis. O principal objetivo deste trabalho foi implementar e comparar extratores de características em prol do processamento em tempo real e a qualidade da descrição do ambiente, que é feita, por meio da extração de características nas imagens capturadas no ambiente em questão, assim, fornecendo estes dados de maneira textual para o robô (BESSA ; BARROSO, 2015). O principal método de extração de características utilizado neste trabalho é baseado no histograma RGB (*Red, Green e Blue*).

Assim como no trabalho de Lyrio e Santos (2015) e no algoritmo neurogenético desenvolvido neste trabalho, uma RNA é utilizada como classificador. O algoritmo neural deve avaliar as imagens para que a localização do robô seja feita com base em um banco de conhecimento formado por imagens e um mapa do ambiente (BESSA ; BARROSO, 2015).

O principal objetivo do trabalho de Bessa e Barroso (2015) está ligado ao sistema de visão robótico omnidirecional e o sistema classificador de imagens baseado em uma RNA. Um sistema de navegação não foi implementado efetivamente, porém, o sistema de visão proposto apresentou resultados satisfatórios (BESSA ; BARROSO, 2015). A proposta do trabalho de Bessa e Barroso (2015) é servir como base para um sistema de navegação robótico autônomo onde se necessita de um sistema de sensoriamento preciso.

Em um trabalho desenvolvido por Luo e Yang (2015), foi utilizado um algoritmo neural para a navegação autônoma de um robô móvel em um ambiente dinâmico. Uma RNA é aplicada para uma abordagem capaz de superar problemas ligados a desvios de obstáculos declarados "muito perto" ou "muito longe", assim, evitando colisões durante a navegação e também rotas auxiliares com desperdício de tempo e energia, já que fazer um desvio de rota maior que o necessário pode gerar estas deficiências no sistema de navegação (LUO, YANG, 2015).

O sistema de navegação é baseado em um método de campos potenciais, onde uma área não preenchida por obstáculos deve atrair o robô, inversamente a este caso uma área que contém obstáculos deve afastar o robô, assim, evitando uma colisão. O reajuste

dos pesos neurais deve ser dado por meio do potencial de desvio necessário para a não colisão com um obstáculo em específico. As informações referentes aos potencias exigidos para o desvio de obstáculos, são gravadas em um mapa neural que deve ser modificado constantemente, já que o ambiente é dinâmico (LUO ; YANG, 2015).

Os trabalhos relacionados a algoritmos neurais (LYRIO ; SANTOS, 2015); (BESSA ; BARROSO, 2015) ; (LUO ; YANG, 2015), apresentaram sistemas que são baseados em mapas, ou seja, a capacidade de aprendizado da RNA é utilizada em prol da construção de mapas que devem auxiliar o robô em sua navegação. O algoritmo neurogenético desenvolvido neste trabalho foge deste tipo de sistemática, assim, utilizando o algoritmo neural em conjunto com o sistema de visão robótico para o reconhecimento de obstáculos e interligando esta capacidade a rotinas de desvio previamente definidas em um banco de conhecimento da RNA.

2.2. Algoritmo híbrido

Entre os vários tipos de algoritmos inteligentes e bioinspirados para a robótica móvel, neste trabalho foram escolhidas duas técnicas bastante conhecidas na literatura para que sejam utilizadas de maneira híbrida, assim, fundindo o potencial de ambas as técnicas em prol ao problema a ser tratado. Os algoritmos escolhidos para este trabalho foram os algoritmos baseados em RNA (Redes Neurais Artificias) e AG (Algoritmos Genéticos).

O algoritmo de RNA tem como objetivo neste trabalho reconhecer obstáculos cadastrados em um banco de conhecimento do sistema de navegação. As entradas da rede neural trabalham com as variáveis de entrada: cor e tamanho.

Para avaliar um obstáculo conhecido ou não conhecido, o algoritmo utiliza como entrada a cor do objeto e o comprimento de uma de suas laterais (sendo que cada objeto pré-cadastrado neste ambiente tem quatro lados). Caso as duas informações codificadas na entrada (sensoriamento) gerem a combinação cadastrada no banco de conhecimento, então é declarado um obstáculo conhecido, e uma solução para este caso já é existente. Nas próximas seções são descritas estas duas técnicas inteligentes de maneira clara e independente.

2.2.1. Algoritmos Genéticos

O AG é um método de computação bioinspirada, classificada como uma técnica evolutiva que trabalha com técnicas heurísticas de busca global (BECKMANN, 2010); (FERREIRA, 2015). Para encontrar o ótimo global de um sistema, utiliza abordagens probabilísticas (onde uma região de busca é definida, e dentro desta é restrita a escolha do melhor individuo para a solução) e não determinísticas (BECKMANN, 2010). Essa técnica evolutiva foi proposta por John Holland e ficou conhecida por trabalhos de David Edward Goldberg (GOLDBERG, 1989) ; (FERREIRA, 2015). John Holland teve seus estudos com vistas para dois aspectos fundamentais: a abstração e a explicação rigorosa do processo

adaptativo de sistemas naturais (FERREIRA, 2015). Já os estudos de Goldberg, tiveram como principal objetivo o desenvolvimento de programas artificiais que utilizassem os mecanismos fundamentais de sistemas naturais (GOLDBERG, 1989) ; (FERREIRA, 2015).

Alguns pesquisadores da área de ciência da computação e engenharias desenvolveram controladores para sistemas robóticos com vistas para o controle de rotas inteligentes, e mostraram eficiência do algoritmo para este tipo de problema a ser tratado, assim como comparações com outros métodos inteligentes (PURIAN, FAROKHI, 2013);(FERREIRA, 2015);(TUNCER; YILDIRIM, 2012);(SHILTAGH, JALAL, 2013);(SHI; CUI, 2010); (SAMADI, OTHMAN, 2013).

2.2.1.1. Descrição dos operadores genéticos

Nas seções a seguir, são descritas as etapas fundamentais para que o comportamento evolutivo do AG tenha o funcionamento esperado e tenha suas aplicações validadas.

2.2.1.1.1. Codificação

O algoritmo genético tem seu início na codificação, que tem como objetivo principal transformar as informações características biológicas em uma cadeia de *bits* para sua representação e manipulação computacional, assim, mantendo um padrão para os dados a serem tratados (WATKINS, 2013) ; (FERREIRA, 2015), cada cadeia de *bits* é dada como cromossomo (figura 11), e cada gene de informação tem sua informação em uma posição fixa chamada lócus (TUNCER; YILDIRIM, 2012) ; (FERREIRA, 2015). Com esta característica, os operadores genéticos podem manipular os dados de maneira confiável. Cada cromossomo pode ser uma possível solução para o problema computacional a ser tratado (FERREIRA, 2015), sendo que cada gene do cromossomo faz parte desta solução (figura 2.11).

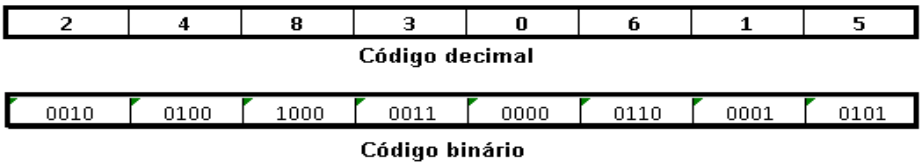


Figura 2.11: Codificação binária e decimal

2.2.1.1.2. Geração da população inicial

A ideia de pesquisar entre uma coleção de soluções candidatas para uma solução desejada é bastante comum nos métodos heurísticos da ciência da computação. Os algoritmos bioinspirados utilizam métodos heurísticos para trabalhar sobre o espaço de busca de interesse (GOLDBERG, 1989) ; (BUENO, 2009) ; (FERREIRA, 2015).

A geração da população inicial tem como maior objetivo gerar uma população inicial de indivíduos com a finalidade de explorar finitas situações no espaço de busca e também diversificar a população para que seja encontrada a solução ótima global com uma maior distribuição no espaço de busca (FERREIRA, 2015).

Existem diversas técnicas para a geração da população inicial (FERREIRA, 2015), sendo algumas delas: geração aleatória, geração uniforme por meio de uma grade, geração de maneira tendenciosa em regiões promissoras do espaço de busca (GOLDBERG, 1989) ; (FERREIRA, 2015). A ideia principal para este caso é trabalhar com a população inicial, usando operadores bioinspirados na manipulação genética natural e seleção natural (GOLDBERG, 1989) ; (MITCHELL, 1998) ; (FERREIRA, 2015).

2.2.1.1.3. Função *fitness*

A função *fitness* é quem deve valorar cada cromossomo, assim quantificando e qualificando cada indivíduo como uma solução para o problema a ser tratado. Os indivíduos com maior quantificação são habilitados como soluções de maior potencial e serão consequentemente escolhidos para serem manipulados por um conjunto de operados genéticos (MITCHELL, 1998) ; (FERREIRA, 2015).

Em um problema de rota de robôs móveis o caminho mais curto é procurado, sendo então um problema de minimização, ou seja, o menor valor quantificado para um conjunto de soluções (população inicial de indivíduos) deve ser a melhor solução. Como exemplo tem-se o algoritmo original de Holland (GOLDBERG, 1989) ; (FERREIRA, 2015), que utiliza a seleção por aptidão proporcional, ou seja, o valor esperado de um indivíduo (valor correspondente ao número de vezes que um indivíduo será selecionado para a base dos operadores genéticos) é a aptidão do indivíduo em questão, dividida pela aptidão média da população (MITCHELL, 1998) ; (FERREIRA,2015).

2.2.1.1.4. Seleção

Depois de definido o método de codificação, o segundo passo a ser definido é a decisão sobre o método de seleção a ser adotado, que tem como principal objetivo escolher os melhores indivíduos da população que irá fazer parte da criação dos próximos indivíduos (FERREIRA, 2015). Então de maneira simples, temos que a seleção deve selecionar os indivíduos com maior aptidão, com vistas em futuras gerações com indivíduos com maior grau de aptidão. O método de seleção deve ser equilibrado, para que os bons indivíduos, não dominem as futuras gerações, assim, comprometendo a diversidade necessária para mudanças e evolução dos indivíduos. Caso contrário o operador de mutação teria que fazer um trabalho muito complexo (MITCHELL, 1998) ; (FERREIRA, 2015).

Vários métodos de seleção são encontrados na literatura, mas a eficiência de cada um é altamente dependente do tipo de caso que o AG está sendo aplicado (BECKMANN,2010);(GOLDBERG, 1989). Para comparações entre diferentes técnicas,

alguns trabalhos devem ser consultados Goldberg e Deb (1991), Bäck and Hoffmeister (1991), De La Maza e Tidor (1991) e Ferreira (2015).

O AG original de Holland utiliza a seleção por aptidão proporcional que corresponde ao número esperado de vezes que um indivíduo é selecionado para fazer parte da nova reprodução de indivíduos (GOLDBERG, 1989) ; (FERREIRA, 2015).

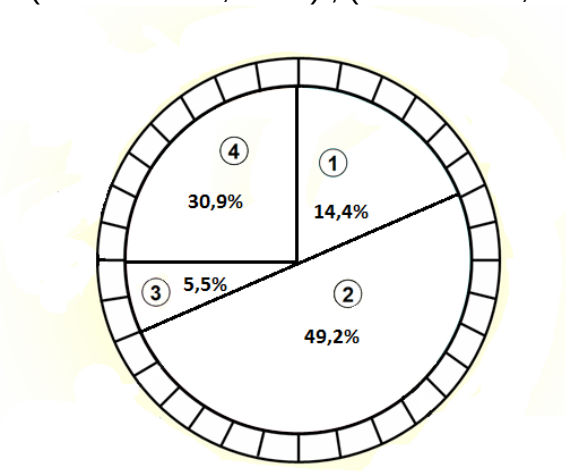


Figura 2.12: Método de seleção por roleta, adaptado de (GOLDBERG, 1989)

O método mais comum para implementar esta técnica, é o método de roleta. No método de seleção por giro da roleta os grupos de indivíduos da população têm um lugar definido na roleta (FERREIRA, 2015), o qual é quantificado de maneira dependente ao valor da função *fitness* (figura 2.12).

Uma outra decisão importante que deve ser tomada para bons resultados do AG é como aplicar os operadores genéticos de maneira eficiente e otimizada, assim, gerando resultados satisfatórios (MITCHELL, 1998) ; (FERREIRA, 2015). A aplicação dos operadores genéticos é altamente dependente do tipo de codificação aplicado neste algoritmo, pois, a recombinação e a mutação trabalham sobre cadeias de *bits* e estas devem ser bem definidas para que os dados sejam manipulados de maneira corretas (FERREIRA, 2015).

2.2.1.1.5. Recombinação

A recombinação é uma das principais características de um AG. A operação de recombinação é análoga ao processo da combinação de genes (QU; XING; ALEXANDER, 2013); (FERREIRA, 2015), que deve reproduzir computacionalmente a sua função. A recombinação mais simples e mais utilizada é a recombinação simples (figura 2.13), onde o ponto de troca de informações (ponto de corte) é escolhido aleatoriamente, e as informações (genes) do par de cromossomos pai é trocada (GOLDBERG, 1989) ; (FERREIRA, 2015).

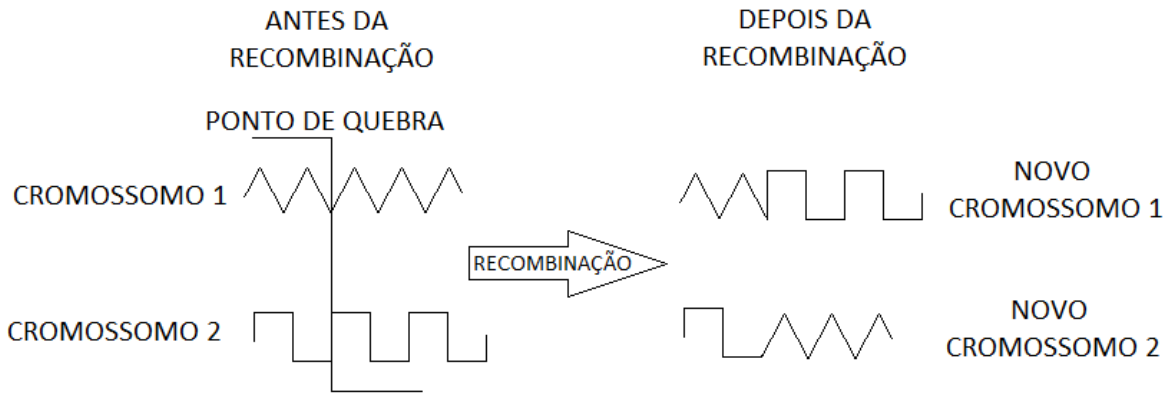


Figura 2.13: Operação de recombinação, adaptado de (GOLDBERG, 1989)

2.2.1.1.6. Mutação

O principal objetivo da mutação é evitar a convergência prematura e gerar indivíduos com características diferenciadas das gerações passadas (QU; XING; ALEXANDER, 2013) ; (TUNCER; YILDIRIM, 2012) ; (FERREIRA, 2015). O tipo de mutação mais utilizado é a mutação pontual (FERREIRA, 2015), onde um ponto fixo ou aleatório sofre modificação (figura 2.14).

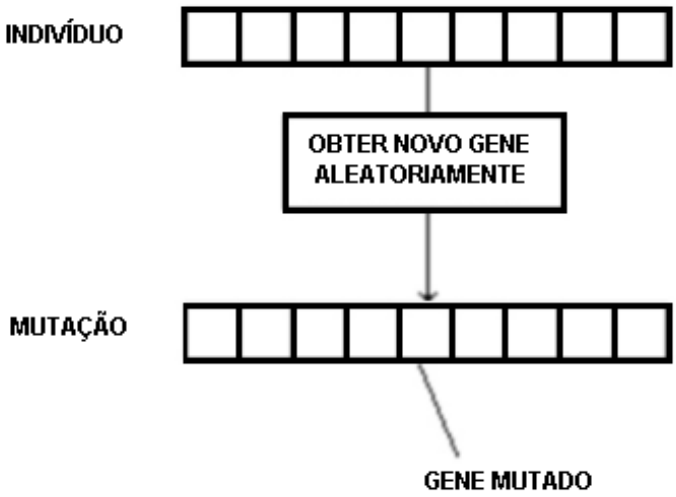


Figura 2.14: Operação de mutação pontual, adaptado de (BECKMANN, 2010)

2.3. Redes Neurais Artificiais

As RNAs são baseadas no comportamento biológico do cérebro humano que é composto por complexos circuitos neurais formados por várias conexões que possibilitam a comunicação entre seus neurônios, assim, possibilitando o comportamento inteligente da rede (LUDWIG e MONTGOMERY, 2007). Na figura 2.15 é apresentada a RNA proposta por Walter Pitts e McCulloch (1943).

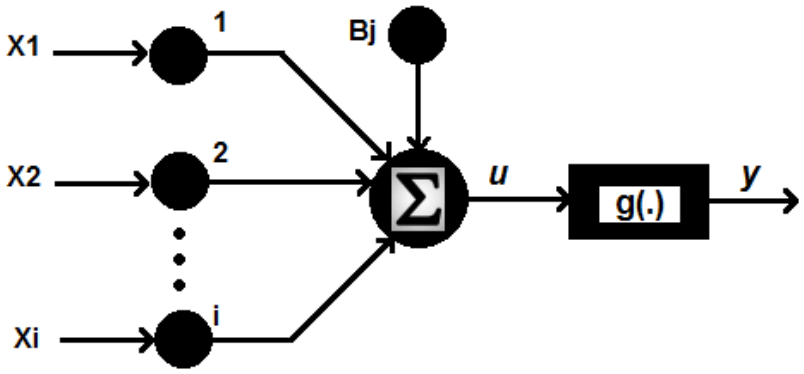


Figura 2.15: O Neurônio Artificial de McCulloch e Pitts, adaptado de (McCULLLOCH; PITTS, 1943) e (LUDWIG e MONTGOMERY, 2007).

Segundo LUDWIG e MONTGOMERY(2007), as RNAs se caracterizam como sistemas de processamento paralelo e distribuído, assim como o cérebro biológico, cada região do cérebro é especializada em uma função especifica, como: processamento de sinais visuais, auditivos, pensamentos, desejos, prazer e etc. O processamento destas múltiplas funções acontece de maneira paralela, assim sendo possível serem satisfeitas ao mesmo tempo.

A RNA tem a habilidade de generalização, ou seja, pode gerar saídas adequadas para entradas que não estavam presentes durante o treinamento, isso faz a diferença dela entre os demais algoritmos, que apenas processam o que lhes foi ensinado. Esta característica é altamente ligada à inteligência da RNA (LUDWIG e MONTGOMERY, 2007). Entretanto, uma RNA não consegue resolver sozinha problemas de grande complexidade (LUDWIG e MONTGOMERY, 2007).

Uma solução potencial para esta deficiência relacionada a problemas complexos é fracionar o problema em questão em tarefas menores e mais simples, assim, aplicando às RNAs pertencentes ao mesmo sistema um particionamento de subproblemas que sejam compatíveis com a capacidade da rede que está sendo utilizada para cada caso (LUDWIG e MONTGOMERY, 2007).

Um benefício das RNAs, está ligado ao tratamento de um problema clássico de I.A (Inteligência Artificial) que é a representação de um universo onde as estatísticas não são

constantes, ou seja, mudam com o passar do tempo. Para a correção deste problema, uma RNA pode modificar seus pesos sinápticos em tempo real (LUDWIG e MONTGOMERY, 2007). Uma desvantagem potencial das RNAs é que elas são uma “caixa preta”, ou seja, é impossível saber como a rede chegou a tal resultado e avaliar seu comportamento interno, como por exemplo, as alterações dos pesos sinápticos (LUDWIG e MONTGOMERY, 2007).

O neurônio de McCulloch e Pitt (figura 2.15) foi o primeiro modelo de neurônio proposto e aceito. Este neurônio é uma abstração do entendimento da época sobre um neurônio natural (LUDWIG e MONTGOMERY, 2007). A descrição de cada elemento da unidade de processamento, proposta por McCulloch e Pitt (1943) é apresentada a seguir (LUDWIG e MONTGOMERY, 2007) ; (CARVALHO, 2015):

- **Conexões de entrada:** por meio destas conexões os neurônios recebem os sinais que são apresentados nas entradas (estímulos) de outros neurônios ou do sistema que fornece estes dados externamente (sensoriamento disponível no sistema).

- **Conexões de saída:** por meio destas conexões conhecidas como axônios, os sinais de saída são levados às entradas de ativação de outros neurônios da rede ou para outros sistemas dependentes das respostas da rede (LUDWIG e MONTGOMERY, 2007).

- **Função de soma:** faz a aplicação da soma global por meio da combinação das entradas, assim, reproduzindo um nível de atividade que representa esta somatória (LUDWIG e MONTGOMERY, 2007) ; (CARVALHO, 2015).

- **Função de transferência**: tem como objetivo transformar a soma de ativação (função de ativação) das entradas da rede em um sinal ativo na saída. A aplicação da resposta da rede na saída está ligada com a resultante obtida por esta função (LUDWIG e MONTGOMERY, 2007) ; (CARVALHO, 2015).

Um exemplo de RNA bastante utilizada na literatura é descrito a seguir, dando base para o entendimento do algoritmo neurogenético desenvolvido neste trabalho.

2.3.1. Projeto de uma rede neural artificial

A arquitetura de uma RNA deve ser definida a partir do problema a ser tratado, porém existem algumas especificações que devem ser seguidas para que a rede seja bem definida e apresente resultados satisfatórios (LUDWIG e MONTGOMERY, 2007). As cinco fases de projeto são descritas nas próximas seções, segundo LUDWIG e MONTGOMERY(2007), com o objetivo de esclarecer como foi implementada a RNA utilizada neste trabalho.

2.3.1.1. Coleta e seleção de dados

A precisão na coleta de dados relativos ao problema é de grande importância para o bom funcionamento da rede (LUDWIG e MONTGOMERY, 2007). Os dados devem ser divididos em dois seguimentos, sendo estes: dados de treinamento e dados de validação. Os dados de treinamento serão utilizados para o treinamento da rede. Já os dados de validação serão utilizados com o objetivo de verificação do desempenho de funcionamento da rede (LUDWIG e MONTGOMERY, 2007).

O número de dados essencial para o treinamento de uma RNA está em função do número de sinapses e bias (neurônio especial localizado na camada de entrada da rede) da rede a ser desenvolvida. Sendo desta forma quanto mais sinapses e bias uma RNA possuir, mais exemplificação de treinamento é necessária (LUDWIG e MONTGOMERY, 2007).

Algumas ferramentas matemáticas podem auxiliar a escolha dos dados que farão parte do vetor de entrada, sendo as mais conhecidas: Análise do nível de entropia e a análise dos componentes principais (LUDWIG e MONTGOMERY, 2007).

2.3.1.2. Configuração da rede

O segundo passo está ligado com a configuração da rede, e para melhor explicação da metodologia adotada, é explicado a seguir em quatro etapas distintas (LUDWIG e MONTGOMERY, 2007).

- O primeiro passo é fazer a seleção do paradigma neural que é viável para o problema a ser tratado. Em outras palavras, a configuração da RNA a ser utilizada como exemplo é uma RNA *perceptron* simples, que será utilizada neste trabalho.

- Determinação da topologia a ser utilizada na rede: o número de camadas, o número de neurônios, o número de bias e nós em cada camada da rede.

- Definição do tipo de algoritmo de treinamento, taxa de aprendizagem e outros parâmetros de treinamento. Este passo tem grande potencial no desempenho final do sistema.

- O último passo é definir o tipo de função de transferência a ser utilizado na rede (LUDWIG e MONTGOMERY, 2007).

Porém, existem conhecimentos que fazem parte da experiência do projetista e podem potencializar a configuração da rede. Estes conhecimentos podem ser obtidos com as experiências passadas em aplicações variadas (LUDWIG e MONTGOMERY, 2007).

2.3.1.3. Treinamento

Na fase de treinamento, os pesos das conexões devem ser ajustados e a tarefa do projetista da rede está ligada a determinação dos valores que inicializarão a rede, sendo estes valores correspondentes aos pesos sinápticos, e qual o algoritmo de aprendizagem e tempo de treinamento para a rede “aprender” (LUDWIG e MONTGOMERY, 2007). Os valores iniciais da rede podem ser gerados por um gerador de números aleatórios uniformemente em um intervalo bem definido. O algoritmo de treinamento deve ser escolhido a partir do conhecimento do problema a ser tratado (LUDWIG e MONTGOMERY, 2007).

Em relação ao tempo de treinamento, vários fatores podem interferir e influenciar a sua duração, mas é preciso utilizar um critério de parada, como por exemplo, o número de ciclos da rede (LUDWIG e MONTGOMERY, 2007).

O treinamento da rede deve ser interrompido quando a rede apresentar uma potencial capacidade de generalização (diferenciar diferentes “problemas”) e quando a taxa de erro for aceitável (LUDWIG e MONTGOMERY, 2007).

2.3.1.4. Teste

A fase de teste de uma RNA é que vai verificar se a rede não memorizou os dados de entrada e validou a rede para a aplicação em que se deseja trabalhar (LUDWIG e MONTGOMERY, 2007). A fase de teste deve, durante o seu procedimento, utilizar o conjunto de validação para determinar o desempenho da rede com novos dados, sendo estes dados nunca apresentados à rede anteriormente. O conjunto de validação é dependente do problema que está sendo tratado (LUDWIG e MONTGOMERY, 2007).

2.3.1.5. Integração

A fase de integração é o processo final e está ligada a integração da rede a um sistema de interesse (para o qual foi projetada). O sistema integrador deve possuir facilidades, como: uma interface conveniente e facilidade de aquisição de dados por meio de unidades de processamento de sinais, arquivos padronizados e planilhas. O sistema ainda deve verificar constantemente o desempenho da rede e fazer manutenção se for necessário, utilizando, por exemplo, novas seções de treinamento (LUDWIG e MONTGOMERY, 2007).

2.3.2. Rede *Perceptron*

A RNA *perceptron* é a arquitetura de redes mais simples, apresentando apenas um conjunto de neurônios de entrada e um conjunto de neurônios de saída (figura 2.16). Este tipo de rede não possui camadas de neurônios intermediarias (LUDWIG e

MONTGOMERY, 2007). Esta arquitetura de rede mesmo sendo simples, pode tratar problemas específicos, não sendo indicada apenas em aplicações mais avançadas, assim, sendo indicada em aplicações de decisão simples (LUDWIG e MONTGOMERY, 2007).

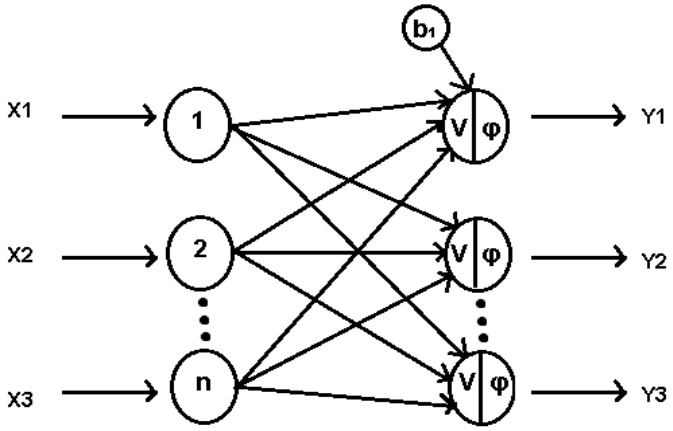


Figura 2.16: RNA *perceptron*, adaptada de (LUDWIG e MONTGOMERY, 2007)

Os elementos da camada de entrada devem ter a mesma quantidade de elementos da camada de saída, e os sinais de entrada, são distribuídos para todos os neurônios da camada seguinte, sendo esta a camada de saída, já que não existe camada intermediaria neste tipo de rede (LUDWIG e MONTGOMERY, 2007). Os neurônios são constituídos por duas funções, sendo estas a função de ativação e a função de transferência (estas funções foram descritas neste mesmo capítulo, nas fases de projeto) ; (LUDWIG e MONTGOMERY, 2007).

Um exemplo dado por LUDWIG e MONTGOMERY (2007) é exemplificado pela situação real de reação do organismo quando um humano se encosta a um objeto com temperatura alta: a entrada da temperatura pelos sensores (sistema nervoso) indica a possível queimadura (função de ativação), que rapidamente ativa o cérebro a enviar um comando (a saída) de se afastar imediatamente (função de transferência), pois existe perigo (LUDWIG e MONTGOMERY, 2007).

2.3.2.1. Algoritmo de aprendizagem *Perceptron*

Segundo LUDWIG e MONTGOMERY (2007), o algoritmo de aprendizagem *perceptron* trabalha com o objetivo de ajustar os pesos sinápticos de uma rede, assim, permitindo que, a partir de um conjunto de sinais de entrada, estes sejam processados e apresentem o conjunto de sinais desejados na saída. O ajuste feito nos pesos sinápticos de uma rede é feito de maneira iterativa. Em uma rede *perceptron* de uma única camada, a regra utilizada é chamada de regra *delta*. Já o elemento de processamento que utiliza esta regra, é conhecido como ADALINE *(Adaptative Linear Neuron ou Adaptative Linear Element)* (LUDWIG e MONTGOMERY, 2007).

Inicialmente, são atribuídos valores aleatórios aos pesos e apresentado um conjunto de valores de entrada, e calcula-se a resposta da rede. A resposta da rede é comparada com o valor desejado, este processo de comparação é dado como treinamento supervisionado, e se o erro não for aceitável, é feito o ajuste dos pesos sinápticos. Na figura 2.17 pode ser observado este algoritmo de maneira mais clara utilizando um fluxograma (LUDWIG e MONTGOMERY, 2007).

No fluxograma da figura 2.17 pode ser observado de uma maneira simples o algoritmo de aprendizado do tipo *perceptron*.

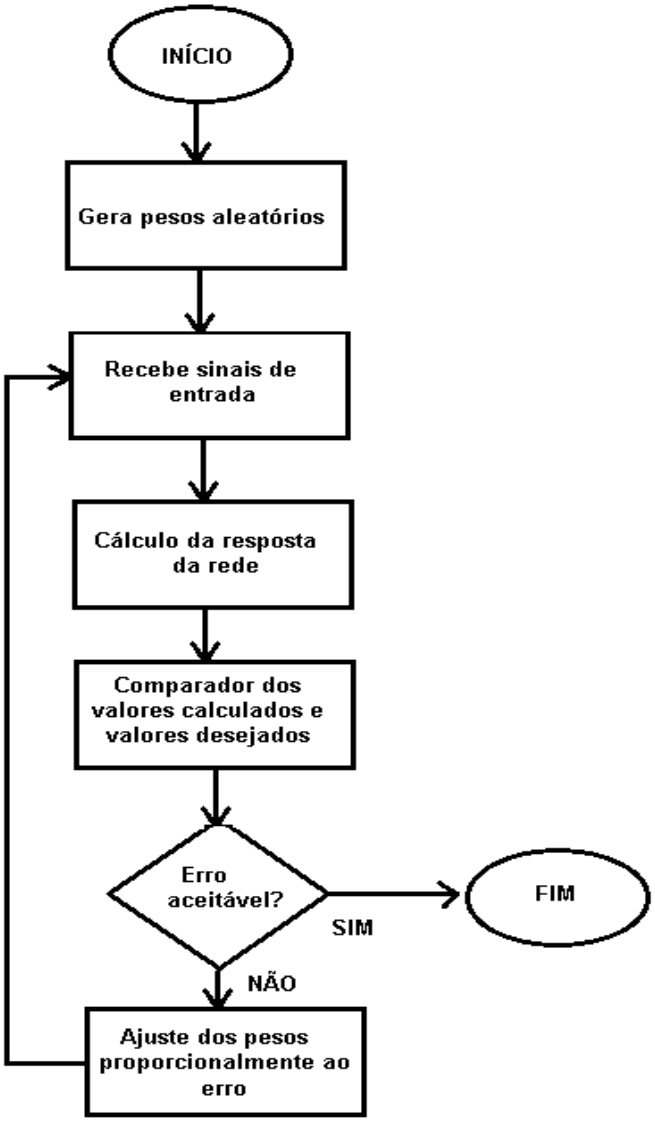


Figura 2.17:Fluxograma para o algoritmo de aprendizado do tipo *perceptron*

O primeiro passo do algoritmo é a geração de pesos aleatórios para o treinamento inicial da RNA. Estes valores devem ser bem baixos para que o ajuste correto não seja

ultrapassado por uma geração de pesos que começou de um conjunto de valores muito alto (LUDWIG e MONTGOMERY, 2007).

O segundo passo é receber os sinais de entrada e enviá-los para o cálculo da resposta da rede, utilizando a função de ativação. Feito isso, o cálculo da resposta da rede é obtido.

Um comparador é utilizado para comparar os valores calculados com os valores desejados. Se o erro for aceitável o algoritmo de treinamento é finalizado. Caso contrário um reajuste nos pesos é feito e continua-se o processo de treinamento até que o erro seja aceitável e termine o treinamento da rede (LUDWIG e MONTGOMERY, 2007).

Capítulo 3

Descrição e desenvolvimento do projeto

Neste capítulo, são apresentadas as técnicas utilizadas neste trabalho para a implementação do controlador de movimento do robô móvel autônomo. Também é apresentada a metodologia para o desenvolvimento de cada um dos algoritmos de controle e as estratégias que cada um utiliza para a solução de um obstáculo. Neste capítulo também são apresentados o ambiente de simulação para robôs e a modelagem do ambiente dinâmico utilizado neste trabalho.

3.1. Controlador

Nesta seção é descrito o funcionamento do algoritmo híbrido desenvolvido, assim como as técnicas que foram utilizadas para se conseguir o comportamento esperado do controlador. Também é descrito como e quando cada algoritmo entra em funcionamento quando encontra um obstáculo.

3.1.1. Algoritmo neural – *Perceptron* (RNA – *perceptron*)

O algoritmo baseado em RNA – *perceptron*, tem como objetivo um aprendizado inicial sobre os obstáculos de maior frequência neste ambiente e também cadastrar um conhecimento inicial em um banco de dados, sendo este conhecimento inicial, as possíveis soluções para o desvio de obstáculos existentes no ambiente de trabalho do robô que aparecem com maior frequência.

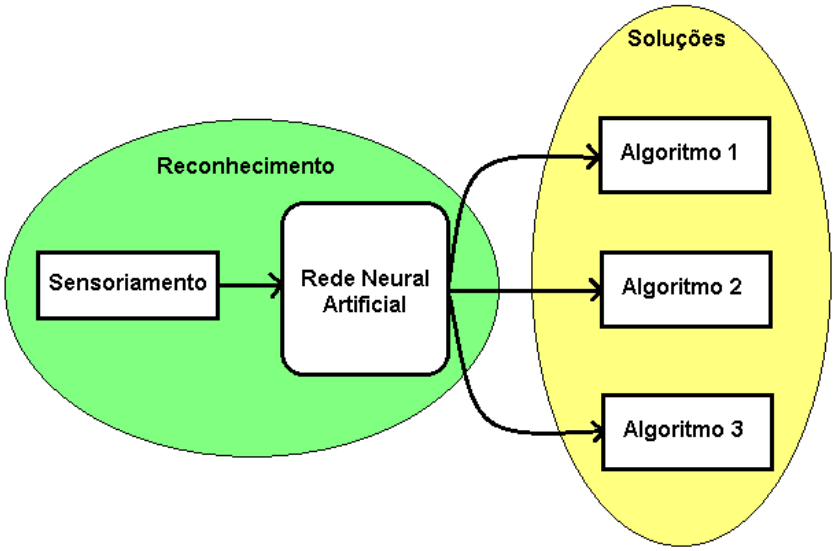


Figura 3.1: Processo de reconhecimento de obstáculos e seus algoritmos de soluções

Quando o robô encontrar um obstáculo, ele deve utilizar seus sensores de proximidade e visão para reconhecer o tamanho de uma das bordas do obstáculo e sua cor, respectivamente. Estes dados servem para o conjunto de dados de entrada da RNA, e serão manipulados para que seja feito ou não o reconhecimento do obstáculo. Se o obstáculo for conhecido, existe uma solução para este no banco de dados. Caso contrário, o controlador deve utilizar o AG para gerar uma nova solução.

O algoritmo de aprendizado utilizado para esta rede foi o algoritmo *perceptron* (descrito no capítulo 2). O algoritmo *perceptron* utilizou uma RNA sem camada oculta. Na figura 3.1 é apresentado o processo de reconhecimento do obstáculo pela RNA, tendo como entradas o sensoriamento do ambiente, e um banco de soluções para cada tipo de obstáculo reconhecido pela rede. As soluções são algoritmos para o desvio de obstáculos reconhecidos pela RNA.

Na figura 3.2 é apresentado um caso em que o robô está seguindo sua melhor rota possível entre o ponto inicial e final (linha reta vermelha) e encontra um obstáculo conhecido pela RNA.

O ponto intermediário é definido pelo banco de conhecimento de soluções para obstáculos, a partir da posição atual do robô, ponto que detectou o obstáculo, assim, são geradas as distâncias d1 e d2 e o ângulo de rotação necessário para a mudança da rota. A linha azul é o novo trajeto livre de obstáculos para o robô. As variáveis DLA e DLO são descritas juntamente com a explicação da equação *flat-earth* (equação 3.5) (FERREIRA, 2015).

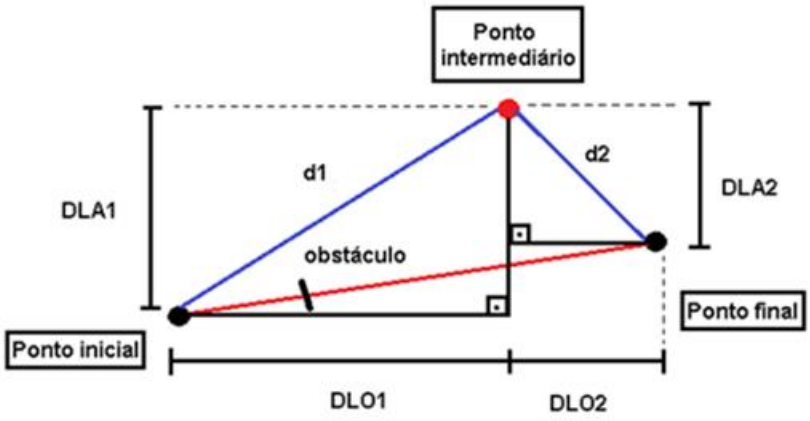


Figura 3.2: Triangulação feita pelo algoritmo de solução para não colidir com o obstáculo cadastrado (FERREIRA, 2015)

Para a definição do ponto intermediário é utilizada a fórmula da distância (FERREIRA, 2015), que é a mesma utilizada neste trabalho para o cálculo do *fitness* (equação 3.5).

Para o cálculo das distâncias d1 e d2 e o ângulo de rotação é aplicado à mesma equação utilizada para o cálculo do *fitness.* Para isso, basta que as variáveis *P.(intermediário_Lat)*

e *P.(intermediário_Long)* sejam isoladas e se consiga obter os seus valores. Um melhor entendimento pode ser obtido na descrição feita a seguir.

Se **DLA1** e **DLO1** estão armazenadas no banco de conhecimento para cada obstáculo, e são variáveis conhecidas quando se encontra um novo obstáculo, então:

Se **DLA1** = P.(atual_Lat) – P. (intermediário_Lat)

então:

P.(intermediário_Lat) = P(atual_Lat) + DLA1

O cálculo acima é feito para se obter a posição intermediária em latitude para o desvio da rota. O mesmo cálculo é feito para longitude, utilizando DLO1 e DLO2.

O algoritmo de solução de desvio de rota para obstáculos conhecidos utiliza estes cálculos em seu comportamento básico.

3.1.2. Funcionamento da rede

No gráfico da figura 3.3 é apresentada a reta de classificação (vermelha) de obstáculos conhecidos e não conhecidos pela RNA *perceptron*. O obstáculo que for declarado desconhecido pela RNA deve ser tratado pelo AG.

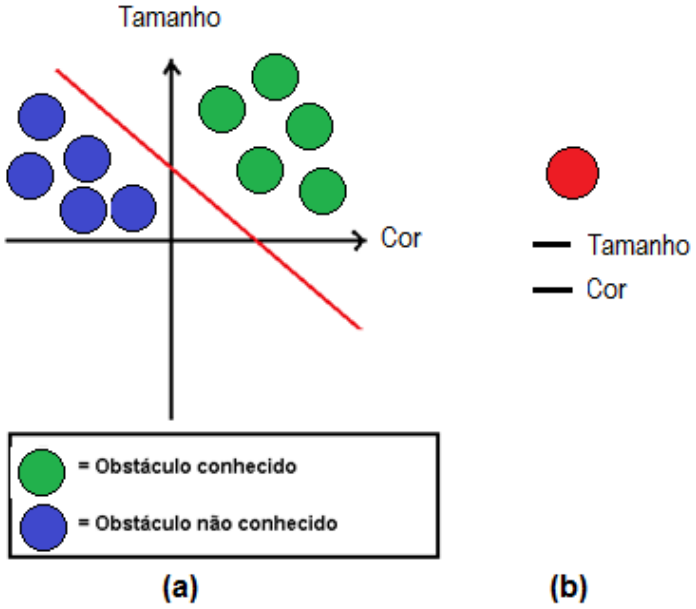


Figura 3.3: (a) gráfico de aprendizagem da RNA *Perceptron* (b) nova amostra para ser classificada pela rede, adaptado de (SILVA, 2010)

O uso do *Bias*, se igual a zero, a reta de classificação ficará estática, assim, não será possivel o deslocamento da reta de classificação e, consequentemente, não será possível a classificação. Então o *Bias* é utililizado para deslocar a reta para um ponto de interesse (forçando a reta para um ponto de interesse), assim, sendo possível o ajuste buscado.

Equação de treinamento (3.1) segundo LUDWIG e MONTGOMERY (2007).

$$W(i,j)_{T+1} = W(i,j)_{T} + \eta E(j)_T x(j) \quad (3.1)$$

As variáveis utilizadas na equação 3.1 são descritas a seguir.

$\boldsymbol{W(i,j)_{T+1}}$ = valor do peso corrigido;

$\boldsymbol{W(i,j)_{T}}$ = valor do peso na iteração anterior;

$\boldsymbol{E(j)_{T}}$ = valor do erro para o neurônio j;

i = indice do sinal de entrada;

T= iteração;

j = indice do neurônio;

η= taxa de aprendizado;

x(i) = sinal de entrada;

O erro *E(j)* é a diferença entre o sinal de saída desejado para o neurônio *j*, dado como *d(j)* e o sinal de saída calculado pela rede y(j). Com esta característica temos que a RNA-*perceptron* utilizada neste trabalho é supervisionada (LUDWIG e MONTGOMERY, 2007).

O treinamento da RNA é supervisionado para que trabalhe com o reajuste dos erros dos pesos de suas *sinapses*, permitindo o alcance de uma classificação desejada (equação 3.2).

$$E(j) = d(j) - y(j) \quad (3.2)$$

Para este treinamento, um padrão é apresentado a rede. Para este trabalho em específico, os padrões a serem apresentados são a cor e o tamanho do obstáculo.

Quando um padrão é apresentado a rede, ela produz uma saída. Com o sinal de erro, que é a diferença entre a resposta atual e a resposta desejada, a rede deve reajustar seus pesos das entradas (equação 3.2). Assim, reduzindo o erro. Para este caso, temos que esta regra de aprendizado supervisionado é chamada de regra *Delta* (LUDWIG e MONTGOMERY, 2007).

Na figura 3.4 é apresentada a estrutura da RNA para um melhor entendimento.

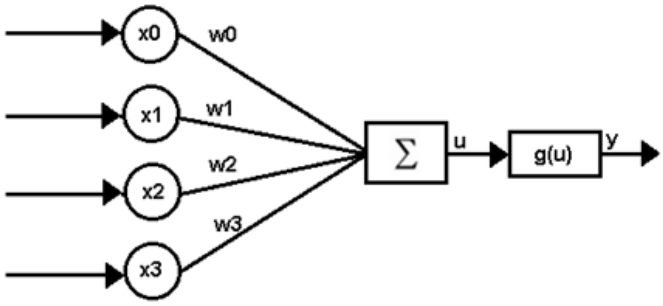


Figura 3.4: RNA *Perceptron* e suas funções

A função de ativação (3.3) está representada pela letra *u*. A função de transferência (3.4) é dada pelo símbolo y e são utilizadas funções de limite, como, por exemplo, a função degrau, onde *y* pode assumir dois valores (LUDWIG e MONTGOMERY, 2007). Uma explicação da aplicação destas funções foi feita no capítulo 2.

$$u_j = \sum_{i=1}^{n} w_{ij}\, x_i \qquad (3.3)$$

$$y(v) = \begin{cases} 1, \forall\ v \geq 0 \\ 0, \forall\ v < 0 \end{cases} \qquad (3.4)$$

As funções de limite ríspido (3.4) devem determinar a veracidade de uma entrada definida (LUDWIG e MONTGOMERY, 2007). Para o caso deste trabalho, a função deve determinar se uma dada entrada é conhecida e associá-la a uma solução para o obstáculo encontrado.

3.1.3. Algoritmo para treinamento da RNA

Os algoritmos de treinamento ou aprendizagem, como são mais conhecidos, são utilizados com o objetivo de ajustar os valores de peso (Slowik e Bialko, 2008). O algoritmo de aprendizado deste trabalho é supervisionado, uma vez que existe uma comparação entre os dados reais de saída (calculados) e os dados de saída desejados, assim, retornando o valor de erro para a rede. O pseudocódigo foi adaptado de Slowik e Bialko (2008) e Silva (2010). Uma descrição detalhada do funcionamento da rede foi feita na seção anterior para que seja possível um melhor entendimento dos algoritmos apresentados a seguir.

Como descrito no capítulo 2, na fase de treinamento, os pesos das conexões devem ser ajustados e uma das tarefas do projetista da rede está ligada a determinação dos valores iniciais da rede, sendo estes correspondentes aos pesos sinápticos .

Uma descrição deste sistema de treinamento é feita no algoritmo 3.1 e uma explicação detalhada sobre o comportamento do mesmo pode ser acompanhada a seguir.

Primeiramente, é apresentada à rede um conjunto de amostras (X^K) que devem ser cadastradas no banco de conhecimento. No caso deste trabalho, as amostras são referentes às características dos obstáculos que estão sendo apresentadas como padrões a rede. O próximo passo é associar estas amostras com suas respectivas saídas desejadas (D^K).

Os valores iniciais (w) dos pesos foram gerados por um gerador de números aleatórios uniformemente em um intervalo bem definido. Um valor para a taxa de aprendizado também deve ser definido (η). Esta taxa não deve ser muito alta para que o ponto desejado para um bom treinamento não seja ultrapassado. Um valor muito baixo para η deve deixar lento o processo de treinamento.

Em relação ao tempo de treinamento, vários fatores podem interferir e influenciar a sua duração, mas é preciso utilizar um critério de parada, como, por exemplo, o número de ciclos da rede. Para isso foi utilizado um contador de "épocas". A cada nova "época" incrementada uma nova comparação do sinal (y $\leftarrow W^t * X^K$) com a saída desejada (D^K) é feita para que seja aplicado o reajuste nos pesos.

O treinamento da rede deve ser interrompido quando a rede apresentar uma capacidade de generalização (diferenciar “problemas”) e quando a taxa de erro for aceitável. O limite de "épocas", caso utilizado, quando atingido deve finalizar o processo de treinamento independentemente da situação da capacidade de generalização e da taxa de erro.

Algoritmo 3.1: Algoritmo *Perceptron* – Fase de Treinamento, adaptado de Slowik e Bialko (2008) e Silva (2010)

```
//==[{Inicio { Algoritmo Peceptron – Fase de Treinamento} ]====//

<1> Obter o conjunto de amostras de treinamento { x^(k)};
<2> Associar a saída desejada { d^(k)} para cada amostra obtida;
<3> Iniciar o vetor w com valores aleatórios pequenos
<4> Especificar a taxa de aprendizagem {η};
<5> iniciar o contador de número de épocas { época <- 0}
     <6> Repetir as instruções
     <6.1> erro  ← “inexistente”.
// a rede está treinada= Até que erro ← “inexiste”
     <6.2> Para amostras de treinamento {x^(k), d^(k)}, fazer
          <6.2.1> u ←w^t * x^(k);
          <6.2.2>y ← sinal (u);
          <6.2.3> se y ≠ d^(k);
               <6.2.3.1> Então { w ← w + η* (d^(k) – y) * x^(k)}
               <6.2.3.2>{erro <-“existe” // sim, existe um erro
          <6.3> época <- época + 1;
              Até que erro ← “inexiste”
Fim {Algoritmo Perceptron – Fase de Treinamento}
```

3.1.4. Algoritmo para fase de operação

O algoritmo operacional tem a função de classificar as amostras, uma vez que a rede foi treinada pelo algoritmo de treinamento. O algoritmo 3.2 foi adaptado de Slowik e Bialko (2008) e Silva (2010).

Uma descrição do comportamento operacional da RNA é feita no algoritmo 3.2 e uma explicação detalhada deste pode ser acompanhada a seguir.

A primeira etapa do algoritmo operacional da RNA é receber em suas entradas uma amostra a ser classificada (x). Esta amostra deve representar um obstáculo que está sendo avaliado. Para declarar este obstáculo reconhecido ou não reconhecido deve-se utilizar os pesos (w) que foram gerados durante a fase de treinamento, assim, gerando o sinal (y) na saída da rede.

Se o sinal de saída gerar um valor 1 (positivo), o obstáculo em questão é declarado reconhecido (Classe A) pela RNA. Caso o sinal gerado na saída seja igual a -1 (negativo), então o obstáculo é declarado como desconhecido (Classe B) ; (SILVA, 2010).

Algoritmo 3.2: Algoritmo Perceptron – Fase de operação, adaptado de Slowik e Bialko (2008) e Silva (2010)

```
//==[ {Inicio { Algoritmo Perceptron – Fase de operação}]======//

// uma vez a rede treinada qualquer amostra pode ser classificada
<1> Obter uma amostra a ser classificada{ x};
<2> Utilizar o vetor w ajustado durante o treinamento;
<3> Executar as seguintes instruções;
     <3.1> u←w^T * x;
     <3.2> y ← sinal(u);
     <3.3> Se y = 1
          <3.3.1> Então: amostra X ∈ {Classe A}
     <3.4> Se y = -1
          <3.4.1> Então: amostra X ∈ {Classe B}

Fim {Algoritmo Perceptron – Fase de operação}
```

3.1.5. Modelagem da RNA

A modelagem de RNA *Perceptron* foi desenvolvida no Scilab, um *software* para computação numérica que fornece um ambiente computacional para aplicações cientificas. O *Scilab* teve seu desenvolvimento iniciado em 1990 pelos cientistas do INRIA (*Institut National de Recherche en Informatique et en Automatique*), e da ENPC (*École Nationale des Ponts et Chaussées*), então pelo consórcio *Scilab* desde Maio de 2003,

Scilab é agora mantido e desenvolvido pelo *Scilab Enterprises* desde Julho de 2012 (CAMPBELL , 2006). O *software* é distribuído gratuitamente para as plataformas: Linux, Windows e MAC OS.

No gráfico da figura 3.4.1, é apresentado um conjunto de 20 amostras classificas pela RNA. A partir de um limiar, as amostras são classificadas se são obstáculos conhecidos ou desconhecidos. Depois de treinada a rede, a reta de classificação deve permanecer fixa, para diferenciar os obstáculos que não são conhecidos pela rede e tratá-los com o AG. Por meio desta modelagem e simulação da RNA, um algoritmo em linguagem *C* foi desenvolvido para que seja a RNA efetiva do controlador híbrido.

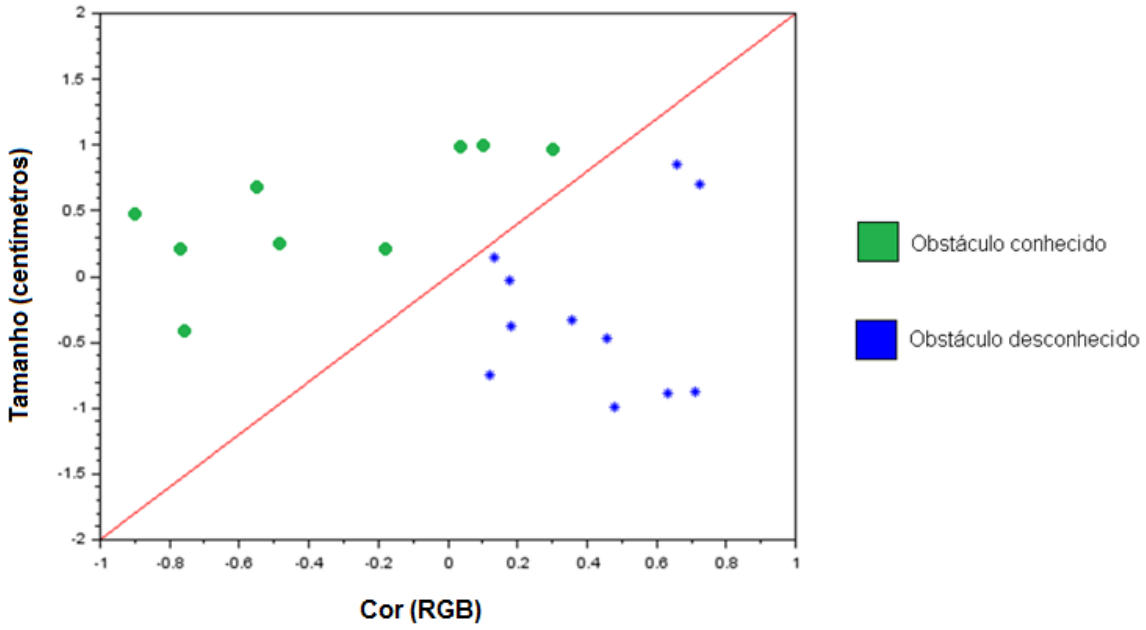


Figura 3.4.1: Gráfico de aprendizado da RNA *Perceptron* com 20 amostras (*software Scilab*)

O gráfico foi gerado a partir da simulação do algoritmo de treinamento da RNA. Esta simulação no *Scilab* serviu como base para a escolha da função de ativação e transferência da RNA implementada em linguagem *C*. O resultado das simulações foi positivo, mostrando que a RNA pode diferenciar os obstáculos que são cadastrados no seu banco de conhecimento dos infinitos outros tipos de obstáculos que podem ser encontrados pelo robô.

3.2. Algoritmo Genético (AG)

O AG é o algoritmo de segundo plano deste algoritmo híbrido, devendo ser chamado apenas quando a RNA não encontrar uma solução *a priori*, assim, reduzindo de maneira significativa o custo computacional e tempo, já que o custo do AG é mais alto. O AG deve

gerar uma população inicial de possíveis soluções para o obstáculo desconhecido e, logo seguinte, utilizar os operadores genéticos para quantificar os indivíduos de maior aptidão.

A aptidão de cada indivíduo (possível solução) deve ser calculada pela função *fitness*. A função *fitness* para o caso de rotas de robôs móveis deve ser um caso de minimização, já que o melhor caminho é o caminho mais curto livre de obstáculos. Após o cálculo do *fitness* de cada indivíduo, deve ser feita a seleção dos melhores deles para a próxima etapa do AG (FERREIRA, 2015). Na figura 3.5 é apresentada a sequência dos operadores genéticos para gerar uma solução ótima

Encontrado os melhores indivíduos pela seleção, o AG deve recombinar os melhores pais e gerar possíveis melhores filhos. Neste trabalho, o método de recombinação utilizado é o simples de único ponto. Após a recombinação, deve ser feita a mutação, operador genético que evita a convergência prematura, utilizando o método aleatório de apenas um ponto (FERREIRA, 2015).

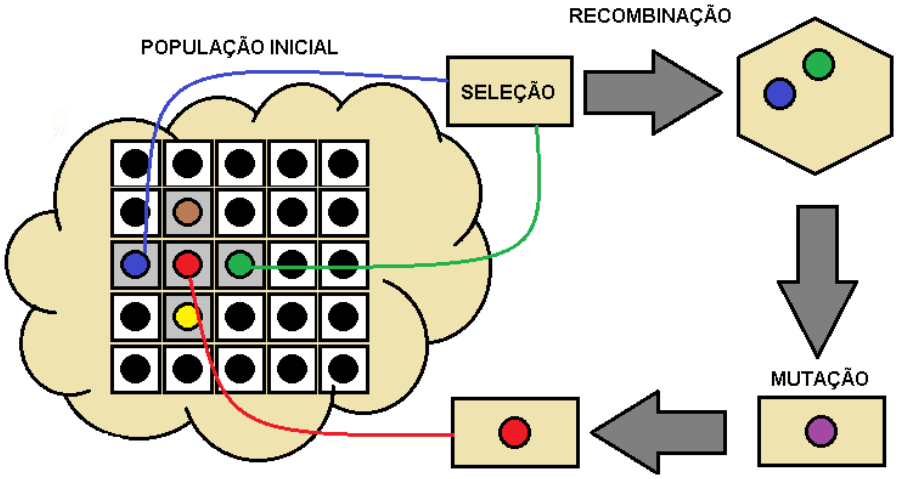


Figura 3.5: Procedimento do AG, adaptado de (BOTASSOLI, 2015)

As comparações de dois métodos de seleção foram feitas neste trabalho, assim escolhendo o melhor método para ser utilizado. Os métodos comparados foram: torneio e roleta. Um vetor contendo os possíveis *fitness* tratados pelo AG, seis indivíduos foram considerados, sendo eles: 7, 3, 2, 5, 10, 1. O índice deste valor tem início em zero, que para este caso representa o valor 7 e vai até o índice 5 que representa o último valor, sendo o valor 1 (FERREIRA, 2015).

Essa etapa foi submetida a mil seleções para cada um dos dois métodos testados. Os resultados podem ser visualizados no gráfico da figura 3.6. A diferença entre cada método de seleção é potencialmente diferente. O índice detentor do melhor *fitness* é o 5. No método de torneio ele foi selecionado 314 vezes contra 205 vezes pelo método de roleta. O pior índice está localizado no índice 4. No torneio, ele foi selecionado 27 vezes contra 112 no método de seleção por roleta (FERREIRA, 2015).

Diante desses dados o potencial do método de seleção por torneio é mais vantajoso para esta aplicação, por conta da maior tendência da escolha de melhores indivíduos para dar continuidade ao AG (FERREIRA, 2015).

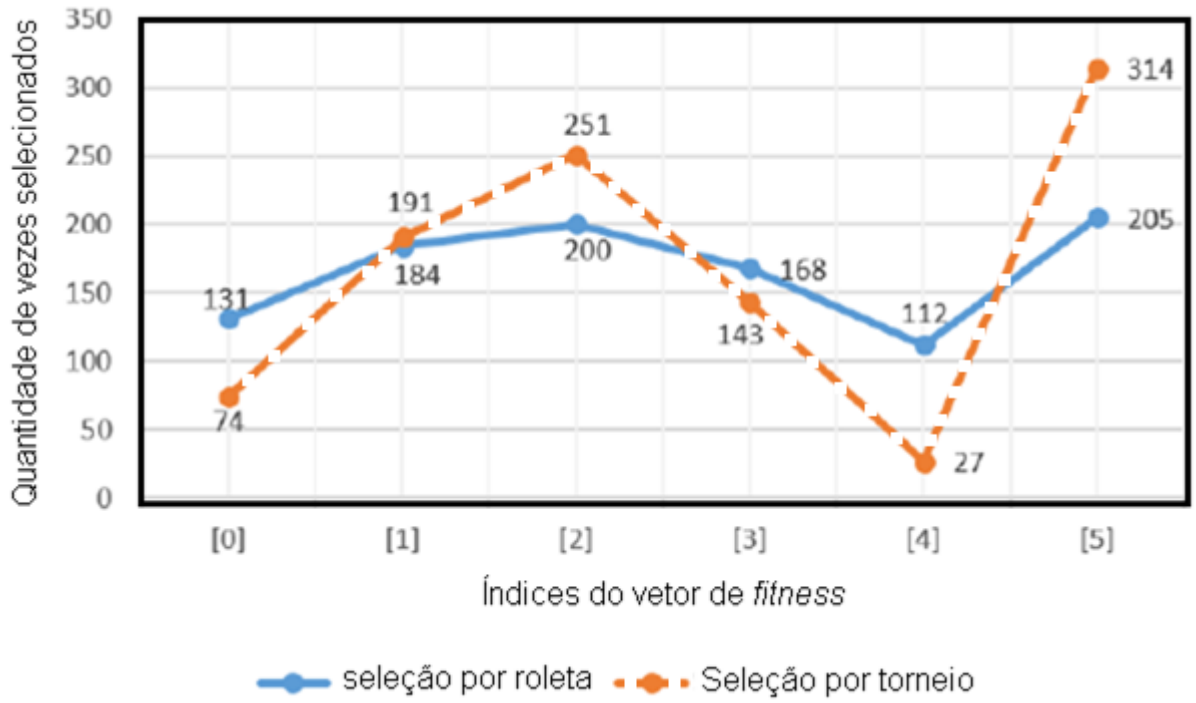


Figura 3.6: Comparação entre os métodos de seleção por torneio e roleta, adaptado de (FERREIRA, 2015)

O tamanho do torneio adotado foi dois devido à simplicidade no tratamento dos dados, ou seja, para a seleção de um pai dois indivíduos são sorteados e posteriormente seus *fitness* são comparados, vencendo a disputa aquele que possuir o menor valor (já que o caso de rota de robôs móveis autônomos é um caso de minimização). O número de pais foi dois, então são necessários dois torneios para a definição dos pais para a etapa de recombinação (FERREIRA, 2015).

O cálculo feito para o *fitness* utiliza uma equação que é a combinação de dois cálculos utilizados na técnica *flat-earth* (figura 3.7), gerando a equação (3.5) (FERREIRA, 2015). A FCC (*Federal Communications Commision*) defende esta técnica, sendo uma técnica que admite que a distância entre dois pontos do globo é a hipotenusa do triângulo retângulo formado pelas diferenças de latitude e de longitude entre esses dois pontos multiplicada pelo comprimento em graus que essa linha se estende (FERREIRA, 2015). A representação deste cálculo pode ser observada na figura 3.7 para um melhor entendimento da equação (3.5).

$$fitness = \sqrt{(\boldsymbol{DLA1})^2 - (\boldsymbol{DLO1})^2} + \sqrt{(\boldsymbol{DLA2})^2 - (\boldsymbol{DLO2})^2} \quad (3.5)$$

As diferenças são calculadas a partir de dois pontos de interesse, sendo estes o ponto atual (ponto em que encontrou um obstáculo) e o intermediário (ponto de desvio para solução do obstáculo encontrado), e entre o ponto intermediário e o final (ponto de objetivo) ; (FERREIRA, 2015).

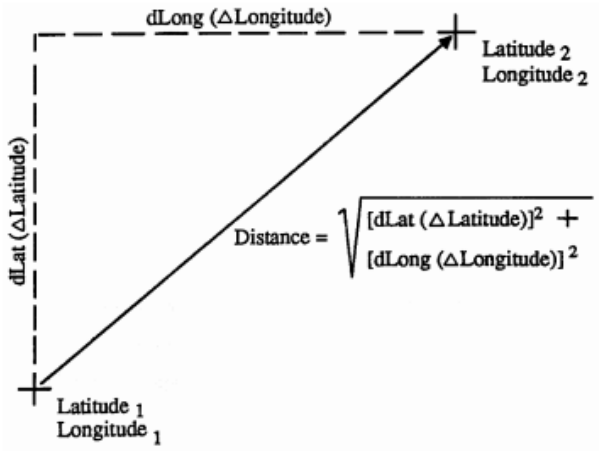


Figura 3.7: Equação da técnica *flat-earth* (JONES; LAYER; OSENKOWSKY, 2007)

- **DLA1** é à distância da latitude entre o ponto atual e intermediário
- **DLO1** é à distância da longitude entre o ponto atual e intermediário
- **DLA2** é à distância da latitude entre o ponto intermediário e o final
- **DLO2** é à distância da longitude entre o ponto intermediário e final (FERREIRA, 2015).

A taxa de mutação utilizada para este AG foi de 5%. Esta taxa foi definida após serem feitas algumas simulações no algoritmo e observado melhores resultados para esta taxa.

Um valor baixo para a taxa de mutação deve garantir que uma posição aleatória do cromossomo não fique travada em um valor, pelo motivo de sofrer mutação. Em outras palavras uma baixa taxa de mutação garante que todo o espaço de busca heurístico do AG seja explorado.

3.2.1. Estrutura do cromossomo

A estrutura do cromossomo utilizado para armazenar as informações referentes as coordenadas de navegação do robô móvel é dividida em seis partes, sendo elas: grau, minuto e segundo para latitude e longitude. A estrutura tem as características mostradas na figura 3.8.

Esta estrutura do cromossomo apresentada na figura 3.8 é reconhecida pelo código do AG para que se possa fazer todas as operações genéticas, já que todos operadores

genéticos trabalham com base nessa estrutura de dados. As informações de latitude e longitude devem ser fornecidas pelo GPS.

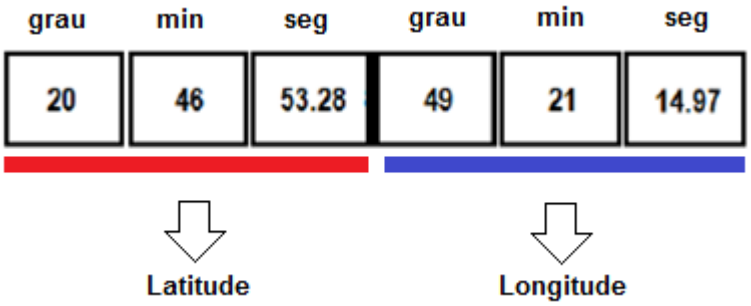


Figura 3.8: Estrutura de um cromossomo

O algoritmo neurogenético também deve utilizar esta estrutura de cromossomo, já que a posição atual do robô e seu ponto de objetivo devem ser tratados pelas coordenadas do GPS que são armazenadas nesta estrutura.

3.3. Algoritmo híbrido (AG + RNA)

O propósito do uso da RNA é economizar tempo e custo computacional para gerar uma solução para o mesmo tipo de obstáculo infinitas vezes (obstáculos de maior frequência no ambiente que o robô está navegando). Caso o obstáculo não seja conhecido pela RNA e sua base de conhecimento, então o AG entra em funcionamento.

A solução ótima global deve ser uma solução para o desvio do obstáculo (ponto intermediário entre o início e o ponto de objetivo) e assim conseguir completar o caminho com o menor custo possível, encontrando uma trajetória possível no ambiente dinâmico em que o robô está navegando. Na figura 3.9(a) é apresentado o fluxograma do algoritmo híbrido para um entendimento simplificado do seu funcionamento.

Quando um obstáculo é encontrado pelo robô, a RNA com sua base de conhecimento avalia o obstáculo que está impedindo a rota traçada inicialmente pelo controlador, se ele for conhecido uma solução está pré-definida no banco de dados. Se o obstáculo não for conhecido o AG entra em funcionamento gerando uma população finita de indivíduos (possíveis soluções). Estes indivíduos sofrem alterações com os operadores genéticos com vistas para a busca da solução ótima global. Uma melhor explicação do AG foi feita no capítulo 2.

Na figura 3.9(b) é apresentada uma Rede de Petri lugar/transição para a modelagem do algoritmo híbrido.

Um material sugerido para o entendimento desta técnica de modelagem é encontrado no elo: http://www.dcce.ibilce.unesp.br/~norian/cursos/mds/ApostilaRdP-CA.pdf.

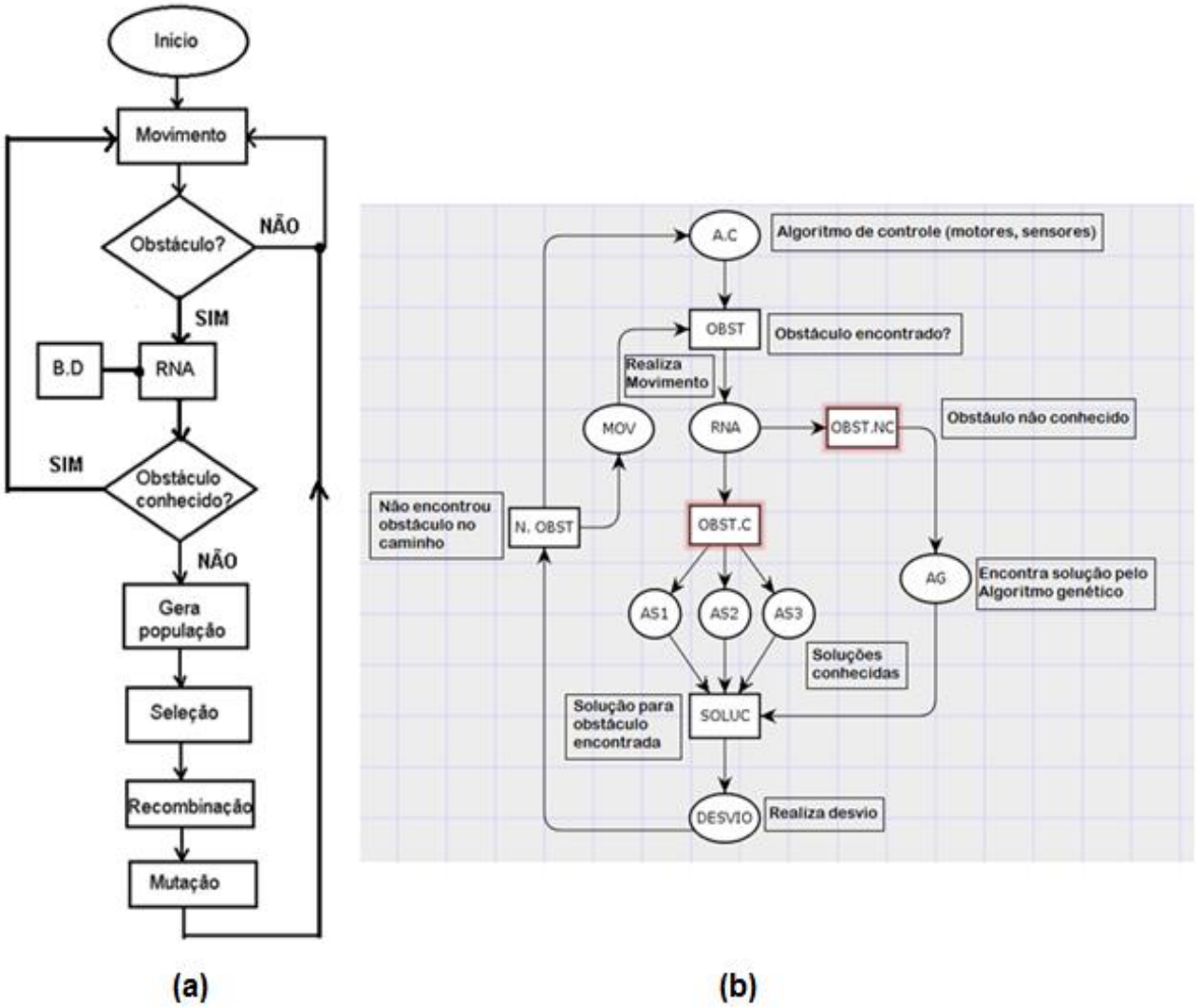


Figura 3.9: (a) Fluxograma do algoritmo híbrido (RNA + AG) (b) Rede de Petri lugar/ transição do algoritmo híbrido

As transições que controlam o tipo de algoritmo que será utilizado para encontrar uma solução para o obstáculo são: *OBST,* que indica se um obstáculo foi encontrado; *OBS.C,* que é ativada se o obstáculo for conhecido pela RNA, levando o problema para uma solução pré-definida. A transição *OBST.NC* é ativada se o obstáculo não for conhecido, então, entrando em funcionamento o AG.

Quando uma solução é encontrada, a transição *SOLUC* deve ser ativada continuando a movimentação do robô pela rota de DESVIO até que um obstáculo seja encontrado novamente e ative a transição *OBST* novamente. Enquanto nenhum obstáculo for

encontrado a transição *N.OBST* permanece ativa e o algoritmo de controle do robô referente a sensoriamento e controle dos motores representado pelo lugar *AC* permanece ativo.

3.4. Modelagem do robô e do seu ambiente dinâmico

Nesta seção é apresentado o ambiente dinâmico proposto e utilizado como ambiente de base neste trabalho, assim como as funções que o robô móvel deve desenvolver neste ambiente. Também é apresentado nesta seção à modelagem do robô e seu ambiente, assim como o ambiente de simulações para robôs que será utilizado para que o controlador neurogenético seja simulado e validado.

O ambiente dinâmico modelado neste trabalho é um "chão de fábrica" industrial, composto por quatro células de manufatura (figura 3.10). Este ambiente será explicado de maneira detalhada nas próximas seções.

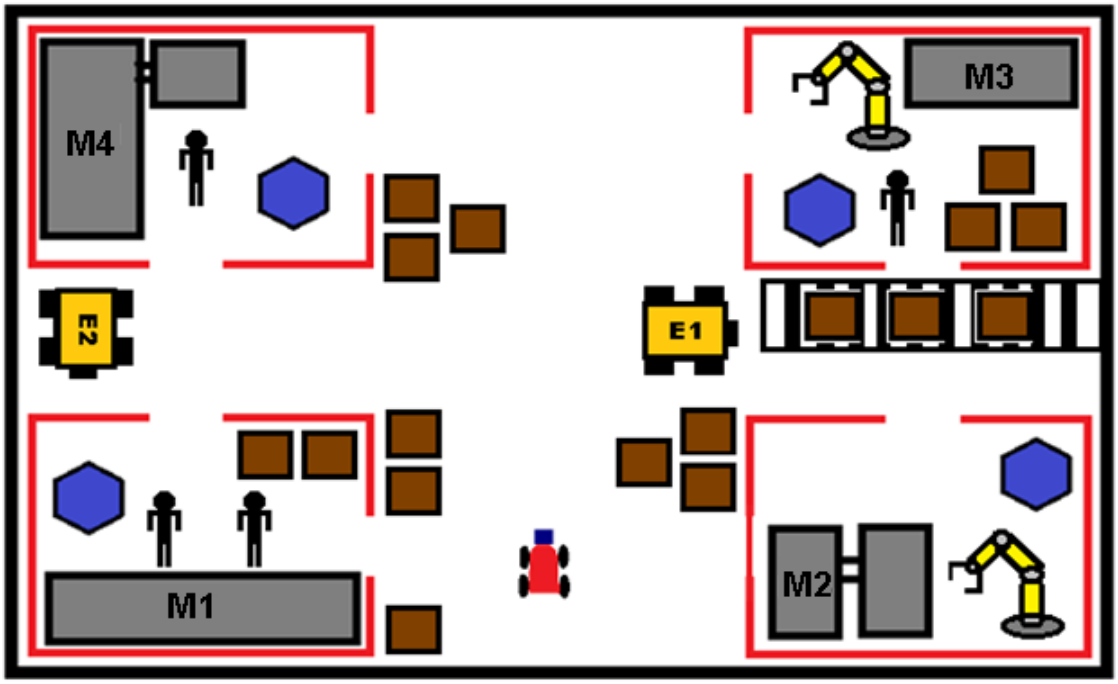


Figura 3.10: Ambiente dinâmico proposto

3.4.1. Ambiente de simulações robóticas (V- REP)

O simulador de robôs V- REP (*Virtual Robot Experimentation Platform*) tem grande flexibilidade se comparado a outros simuladores para esta mesma finalidade (COPELIA ROBOTS, 2015). Este simulador possui um grande número de funções específicas para robótica móvel e não móvel, e APIs *(Application Programming Interface)* mais elaboradas.

O simulador de robôs V-REP, conta com ambiente de desenvolvimento integrado, sendo baseado em uma arquitetura de controle distribuído, ou seja, cada objeto / modelo pode ser controlado individualmente por meio de um script incorporado, um plug-in, um nó ROS, um cliente API remoto ou uma solução personalizada. Isso torna o V-REP muito versátil e ideal para aplicações de múltiplos robôs (COPELIA ROBOTS, 2015).

Controladores podem ser implementados em *C / C ++, Python, Java, Lua, Matlab, Octave* ou *Urbi*, o que trouxe grande liberdade na escolha da linguagem de programação a ser utilizada neste trabalho (COPELIA ROBOTS, 2015).

3.4.2. Ambiente dinâmico proposto

O ambiente proposto neste trabalho é um ambiente dinâmico industrial, ou seja, com obstáculos móveis. O ambiente industrial é composto por quatro células robóticas e estas devem ser "servidas" com peças (parafusos, porcas, arrebite, engrenagens) pelo robô móvel utilizado neste trabalho, em pontos pré-definidos dentro de cada uma das quatro células.

A motivação para a função do robô neste trabalho é que os robôs manipuladores e os humanos que atuam nas células robóticas não percam tempo, e evitando também o perigo de um humano interagir com um ambiente cheio de riscos para buscar as peças no depósito, riscos (maquinas, robôs, empilhadeiras, químicos) existentes em cada uma destas diferentes células e no trajeto de cada uma delas (figura 3.10).

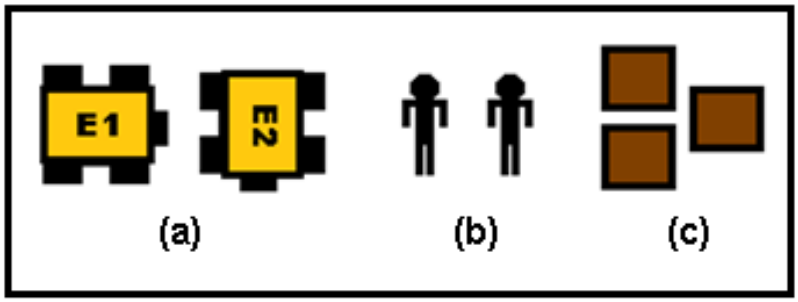


Figura 3.11: Obstáculos cadastrados (a) empilhadeira; (b) humanos e (c) caixas

O ambiente possui, no caminho de ligação, das células robóticas, alguns tipos de obstáculos que são mais frequentes, e que serão cadastrados em um banco de conhecimento da RNA. Estes obstáculos são: empilhadeiras, caixas e humanos (figura 3.11).

Para o robô, existem infinitas possibilidades de encontrar um caminho possível, livre de colisões, entre as quatro diferentes células, porém como o menor caminho deve ser levado em conta, o algoritmo deve ser capaz de encontrar uma solução para cada caso em particular, levando em consideração que os dois pontos (início e final) e a linha reta entre estes dois pontos é o menor caminho possível.

O algoritmo deve procurar uma solução mais próxima desta linha reta quando um obstáculo impedir este caminho. Para a minimização de rota, o acesso a cada célula por diferentes portas pode influenciar a escolha do melhor caminho (figura 3.12).

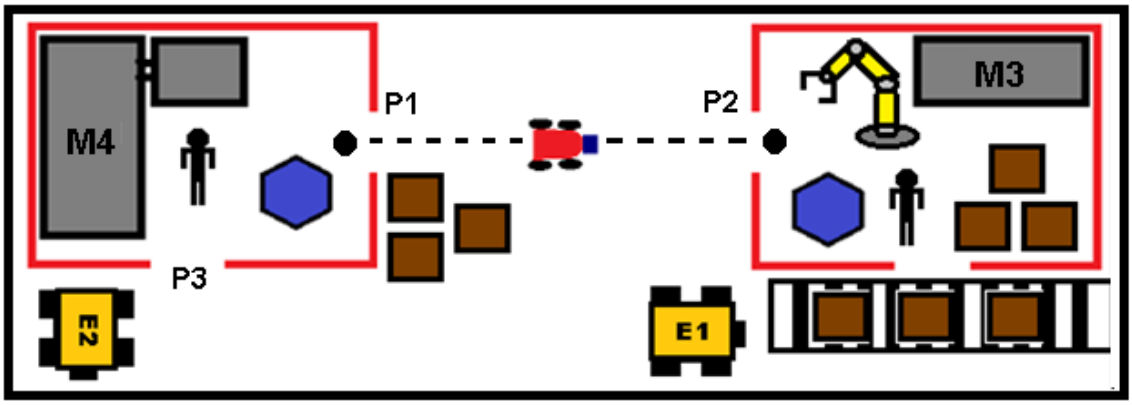


Figura 3.12: Caminho feito entre a célula M4 e M3

Na figura 3.12 observa-se o robô móvel que está se locomovendo entre a célula M4 e M3, e, para isso, foram utilizadas as portas P1 e P2, sendo, assim, possível uma rota consideravelmente menor se comparada ao caminho feito entre as portas P3 e P2.

Os obstáculos conhecidos também podem se tornar obstáculos desconhecidos, se a configuração do ambiente sofrer alterações. Para este caso, temos, por exemplo, a caixa, um obstáculo conhecido pela RNA, porém, se um conjunto de caixas estiver reunido em um mesmo local, estas formarão um obstáculo maior e desconhecido pela rede, sendo, assim, necessária a busca de uma solução pelo AG. Isso pode acontecer em infinitas possibilidades, pois obstáculos conhecidos de maneira combinada podem gerar diferentes obstáculos não conhecidos pela RNA. Na seção 3.4.5 é exemplificado com mais clareza estes casos.

Estes casos mostram que mesmo o ambiente parecendo ser "pseudo dinâmico" (por possuir células robóticas e caminhos entre estas células em locais estáticos e obstáculos dinâmicos), as alterações são feitas constantemente no ambiente tornando-o dinâmico, por conta dos obstáculos móveis e da reconfiguração repentina do ambiente.

As células possuem pontos fixos para serem servidas pelo robô, sendo possível declarar estes pontos para que sejam servidos constantemente (O1*[objetivo 1]...*O4 *[objetivo 4]).* Para declarar um ponto de objetivo a ser alcançado pelo robô, neste trabalho foi utilizado o GPS (*Global Positioning System*). Uma alteração destes pontos pode ser feita sem grandes problemas.

O robô tem capacidade para servir as quatro células cada vez que sair do depósito de componentes. Depois disso, deve voltar ao depósito e ser carregado por um humano com uma nova carga de componentes (uma otimização de carregamento diferente a esta pode ser feita sem grandes dificuldades).

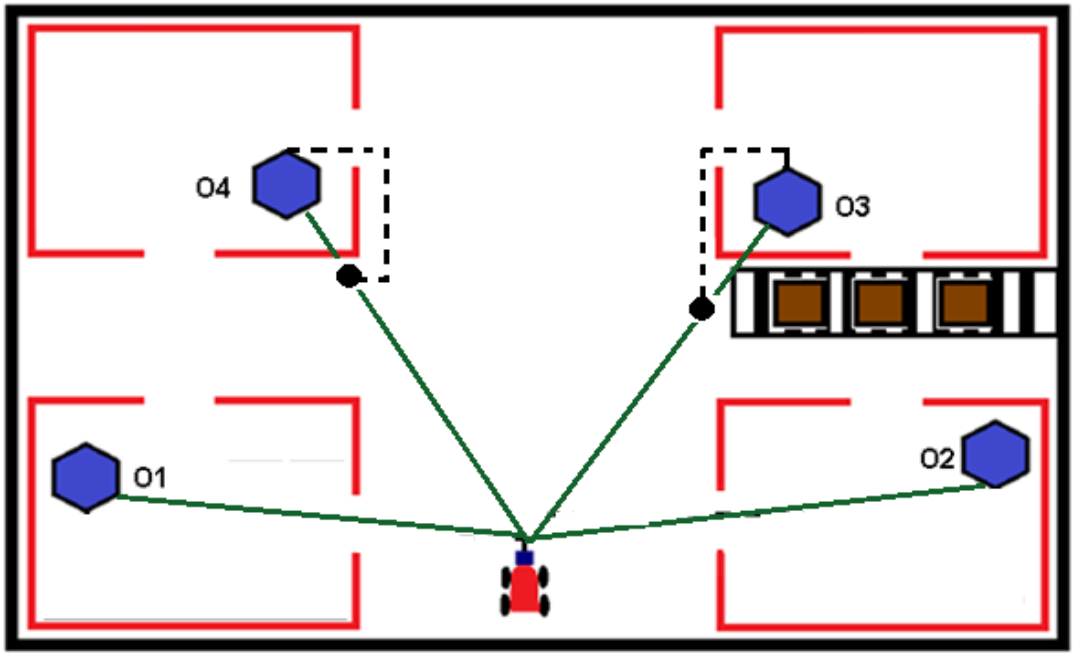


Figura 3.13: Pontos que o robô deve servir (menor rota possível)

Na figura 3.13 é apresentado os pontos de objetivo, em azul, que devem ser servidos pelo robô móvel. Os caminhos em verde são os menores trajetos até os pontos de objetivo, mas são impossíveis de serem feitos em alguns casos (O3 e O4) por conta dos obstáculos. Os caminhos pontilhados são as possíveis soluções *a priori* (para servir O3 e O4), já que obstáculos móveis ou estáticos podem impedi-los e seja necessária uma mudança na rota.

As caixas devem ser transportadas por empilhadeiras até a esteira, em que o produto final é entregue para o depósito de produto final. O trânsito de empilhadeiras é constante, sendo o obstáculo de maior problemática para este sistema.

Se um novo ambiente for escolhido para que o robô utilize este mesmo controlador híbrido, um novo aprendizado inicial dos obstáculos mais frequentes neste novo ambiente deve ser feito utilizando a RNA, assim atualizando o banco de conhecimento para este novo ambiente. Este procedimento deve ser feito sem grandes dificuldades.

3.4.3. Robô utilizado

A plataforma robótica utilizada como base para as simulações neste trabalho, *Kihon*, foi desenvolvido por MATSUMURA (2014). Este robô foi desenvolvido para servir de plataforma didática. A plataforma tem como maior objetivo a facilitação e flexibilidade de implementações de aplicações robóticas e estudos por meio de uma interface de comunicação de simples uso (MATSUMURA, 2014). A implementação do algoritmo híbrido em um ambiente de simulação, tem vistas para uma futura implementação do

algoritmo em *hardware*, na plataforma *Kihon*. Na figura 3.14 visualiza-se a plataforma *Kihon*.

O robô possui sensoriamento ultrassônico (figura 3.14-b) e para este trabalho foi adicionado (em simulações) um sensor de imagem (câmera) para verificação das cores dos objetos.

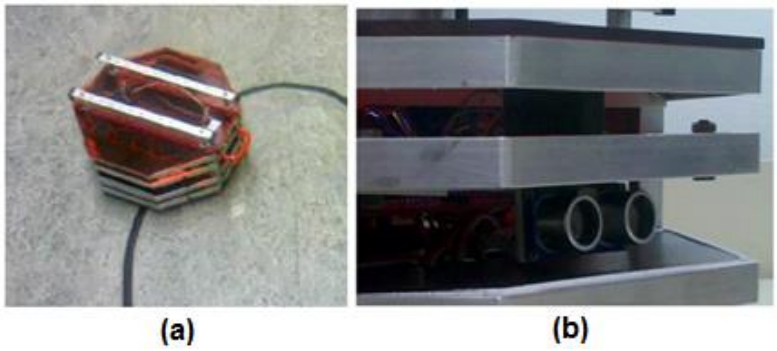


Figura 3.14: (a) Robô utilizado para o sistema de transporte de material automatizado (AMTS sigla em inglês) - (MATSUMURA, 2014) (b) disposição do sensor ultrassônico

O robô utilizado no simulador V-REP para representar o robô proposto por MATSUMURA (2014), foi o *Pioneer* (figura 3.15), por ter uma configuração mais próxima da plataforma robótica *Kihon* (MATSUMURA, 2014). Na figura 3.15 (b) é apresentada a plataforma *Pioneer* sendo utilizada em conjunto com um "braço" manipulador da empresa *Adept Mobile Robots* (SECCHI, 2008).

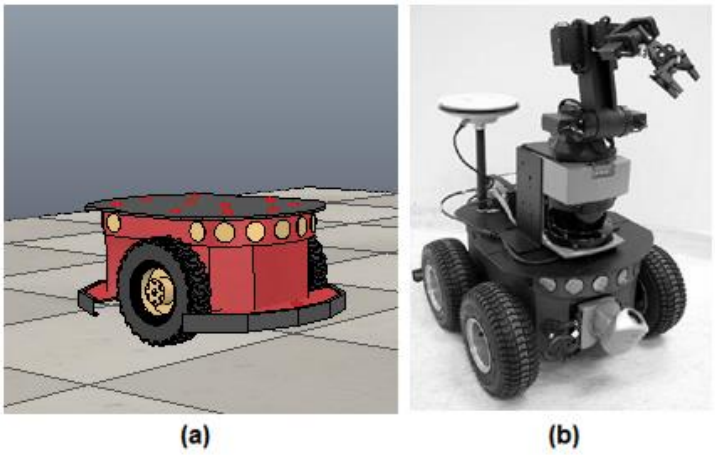


Figura 3.15: (a) robô móvel *Pioneer* no simulador V-REP; (b) robô *Pioneer* real (SECCHI, 2008)

O robô *Pioneer* possui características de grande semelhança ao robô *Kihon* e grande disponibilidade para o uso de sensores, seja ultrassônico, óptico ou câmera. Sendo

assim, não foi necessária a modelagem do robô *Kihon*, tendo sido utilizado o robô *Pioneer* para todas as simulações deste trabalho.

O controlador híbrido desenvolvido neste trabalho pode ser implementado na plataforma *Kihon* sem grandes dificuldades desde que toda a implementação feita em simulações, como inclusão de sensores para maior precisão, seja feita também em *hardware*, na plataforma *Kihon.*

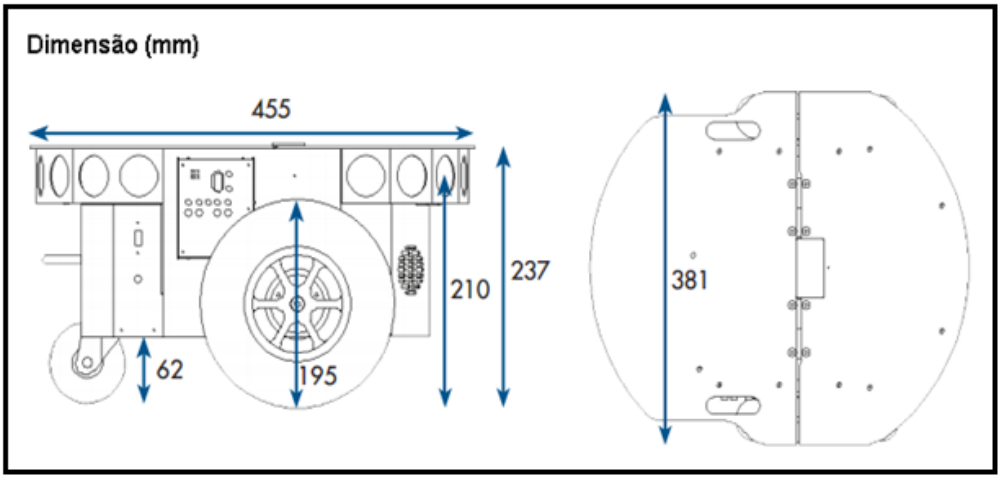


Figura 3.16: Dimensão do robô *Pionner* (PIONEER, 2011)

O robô *Pioneer* conta com as dimensões apresentadas na figura 3.16, e possui as seguintes características físicas que podem ser visualizadas na figura 3.17:

Caracteristicas físicas do robô Pioneer 3 - DX	
Operação	
peso	9 kg
carga suportada	19 kg
Comando diferencial	
Raio de Curva	0 cm
Raio de rotação	26,7 centímetros
Max. Velocidade frente / trás	1,2 m / s
Velocidade de rotação	300 ° / s
Energia	
Tempo de execução	8-10 horas w / 3 baterias

Figura 3.17: Características físicas do robô *Pioneer,* adaptado de (PIONEER, 2011)

A carga suportada é uma característica de grande importância já que o robô tem a função de servir peças para as células do “chão de fábrica”. Um melhor aproveitamento e otimização do carregamento no robô, levando em consideração o peso máximo de carga,

pode evitar desperdício de tempo e energia do robô. Uma otimização da carga suportada também pode dispensar uma extensão do robô para carregar as peças a serem servidas.

Como exemplo, o robô da *Carnegie Mellon University* (figura 2.5 - a), que utiliza duas plataformas anexadas em série ao robô (estilo vagão) para transporte de uma quantidade maior de peças (desde que a capacidade em carga suportada não ultrapasse os limites físicos do robô) ; (SECCHI, 2008).

3.4.4. Disponibilidade de sensoriamento

O sensoriamento ultrassônico, disponível no robô *Kihon* é restrito a uma área de cobertura de 8 a 30 graus, para cada sensor, sendo, assim, impossível qualquer tipo de compreensão do ambiente quando o ângulo de percepção ultrassônica estiver fora desta faixa (MATSUMURA, 2014). Além do sensor ultrassônico, o robô possui três sensores de proximidade por contato (figura 3.18), que para este trabalho não serão utilizados.

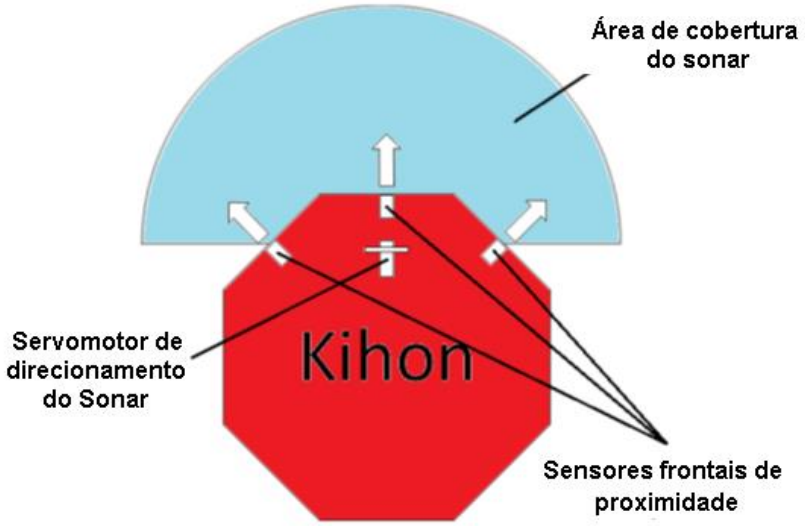


Figura 3.18: Disposição dos sensores do *Kihon* (MATSUMURA, 2014)

O sensor ultrassônico tem como função identificar o impedimento no caminho do robô (obstáculo). O robô *Kihon* possui apenas três sensores ultrassônicos e estes ficam localizados na parte frontal do robô, o que impossibilita uma melhor “varredura” dos obstáculos no ambiente em questão (MATSUMURA, 2014).

Uma proposta para solução deste problema é o aumento na quantidade de sensores ultrassônicos para uma cobertura de 180 graus no seu ângulo de percepção. O robô *Pioneer* possui 8 sensores ultrassônicos frontais e 8 traseiros, o que garante uma melhor varredura do ambiente. Este conjunto de sensores pode ser implementado na plataforma *Kihon* para melhores resultados do controlador neurogenético.

Uma outra solução para uma melhor percepção do ambiente é rotacionar o robô quando um obstáculo for encontrado, assim, sendo possível aumentar a dimensão de percepção do robô de 8 a 30 graus para até 180 graus. O problema para este caso é o custo de tempo elevado e o custo de energia para a realização destas manobras.

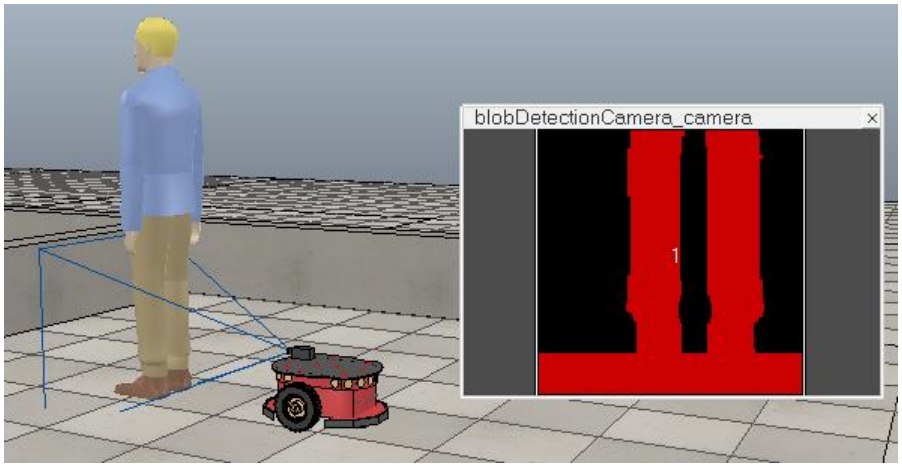


Figura 3.19: Sistema de visão do robô móvel *Pioneer*

E, por fim, um sensor de imagem (câmera) deve identificar a cor do objeto e seu tamanho (figura 3.19), sendo duas das entradas de dados da RNA utilizada neste trabalho. A câmera deve codificar a cor que foi captada para que sirva como um conjunto de dados para o treinamento e operação da RNA.

A câmera só deve ser acionada quando um obstáculo for encontrado pelo sensoriamento ultrassônico, assim, reduzindo o custo computacional para o sensoriamento. Logo seguinte deve-se captar a cor e o tamanho do obstáculo, assim, servindo como entrada de dados para a RNA.

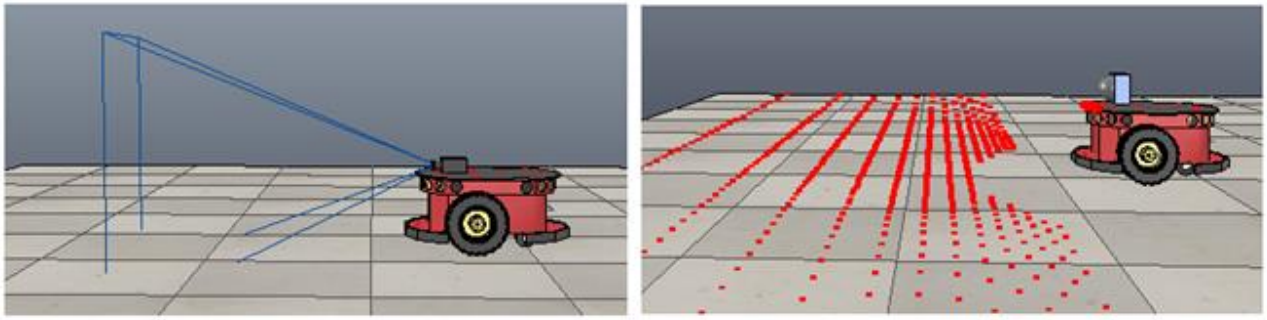


Figura 3.20: (a) sensoriamento por câmera (b) sensoriamento óptico de 180 graus

Uma outra alternativa para a solução do problema do ângulo de visão é utilizar apenas um sensor óptico (figura 3.20 - b), ao invés de um conjunto de sensores ultrassônicos, o que pode trazer uma maior simplicidade na captura dos dados de entrada e também uma

maior precisão na percepção do ambiente e seus obstáculos, já que apenas um sensor deve ser tratado e não um conjunto, diminuindo ruídos e outros problemas ligados ao sensoriamento. Na seção 3.4.6.2., este procedimento é descrito e não apresentou bons resultados para a captura de tamanho dos obstáculos.

3.4.5. Detecção de obstáculos frequentes

Nesta seção são apresentados os casos em que o robô encontra um obstáculo de maior frequência no ambiente dinâmico. Os obstáculos aqui apresentados são reconhecidos pela RNA caso sejam encontrados pelo robô móvel, pois, seus padrões foram cadastrados em um banco de conhecimento.

Se o obstáculo for realmente declarado reconhecido pela rede, existe uma solução pronta para desviar deste obstáculo e continuar a rota até o ponto de objetivo.

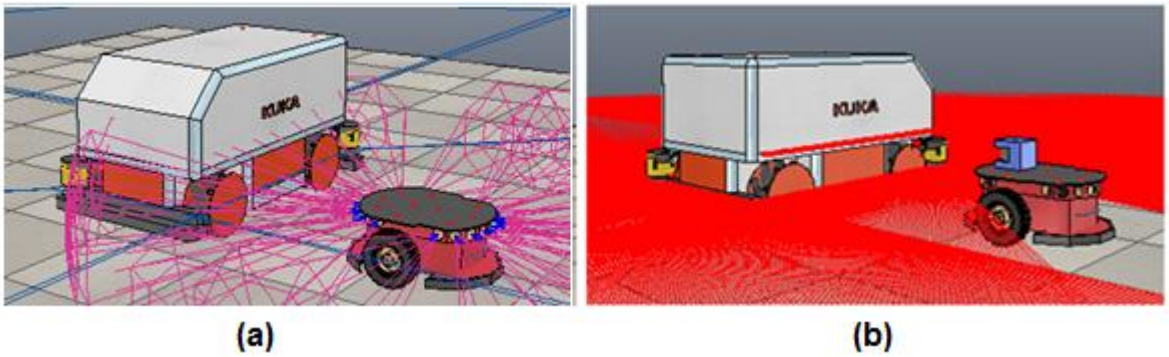


Figura 3.21: Obstáculo detectado (empilhadeira) (a) sensor ultrassônico; (b) sensor óptico

O sensor ultrassônico deve “varrer” (figura 3.21) o ambiente até que encontre um obstáculo ou até que o ponto de objetivo seja alcançado.

Caso um obstáculo seja encontrado, o robô deve utilizar a visão computacional para verificar a dimensão do obstáculo e também deve-se utilizar a câmera para captar a cor do obstáculo.

Caso as variáveis de entrada cor e tamanho do obstáculo sejam equivalentes às variáveis de um obstáculo conhecido pela RNA, sendo eles: caixa, humano ou empilhadeira, então o obstáculo é declarado como reconhecido, e a RNA direciona este reconhecimento para uma rotina de solução pré-definida. O algoritmo de solução que deve fazer o robô contornar este obstáculo. Uma descrição deste algoritmo foi feita na seção 3.1.1.

O algoritmo de desvio estará disponível no banco de conhecimento do controlador neurogenético.

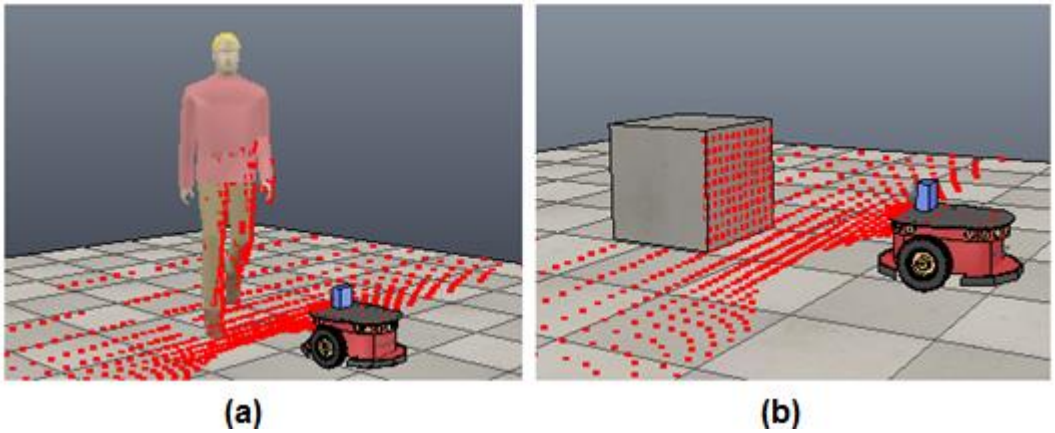


Figura 3.22: Outros dois tipos de obstáculos também reconhecidos pela RNA (a) Obstáculo detectado (humano); (b) obstáculo detectado (caixa)

Outro obstáculo frequente é o humano (figura 3.22- a), assim como a empilhadeira (figura 3.21), é um obstáculo móvel. Neste trabalho a empilhadeira foi representada pelo robô móvel *KUKA* (figura 3.21), pelo motivo de que este robô pode representar a movimentação da empilhadeira com um algoritmo de controle simples.

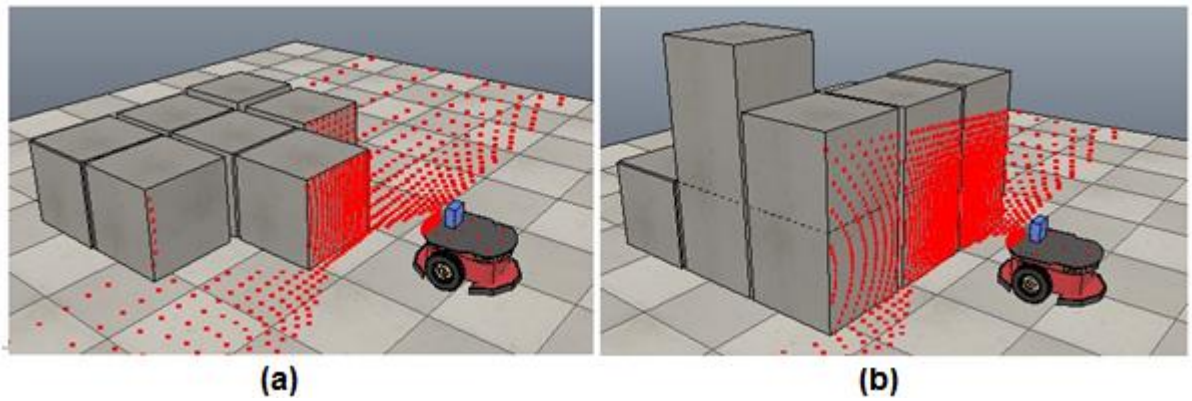


Figura 3.23: Obstáculos formados por caixas (obstáculo conhecido) (a e b) diferentes configurações para obstáculos formados por caixas

Gerar apenas uma solução para obstáculos móveis pode não ser suficiente, pois, encontrada uma solução para desviar do obstáculo, seja uma solução existente no banco de dados da RNA ou uma nova solução gerada pelo AG, e este se mover, pode entrar novamente na rota do robô sendo necessária uma nova solução para o desvio do mesmo obstáculo.

O obstáculo, caixa (figura 3.22 – b), é o obstáculo de maior simplicidade em seu tratamento, pois não tem mobilidade. A característica de ser um objeto que será unido a outros do mesmo tipo, podendo se transformar em um grupo de caixas, que por sua dimensão modificada, se torna um obstáculo irreconhecível pela RNA, é que pode gerar um maior custo na sua solução, pois será necessária a solução pelo AG (figura 3.23 – a e b).

Com a apresentação dos tipos de obstáculos neste ambiente e suas características, pode-se entender a dimensão da problemática de solução destes obstáculos, tanto pela mobilidade dos humanos e das empilhadeiras quanto pela flexibilidade de reconfiguração das caixas e outros obstáculos, se tornando obstáculos irreconhecíveis pela RNA.

Um agrupamento dos outros dois tipos de obstáculos reconhecidos pela RNA também pode ser feito (humanos e empilhadeiras), mas estes casos devem ser de baixa frequência neste ambiente.

O objetivo de todo o trabalho do robô em sua rota autônoma neste ambiente é servir as células com peças, evitando a colisão com os obstáculos e fazendo a menor rota possível. Uma descrição mais detalhada do objetivo do robô foi apresentada na seção 3.4.2.

Na figura 3.24 são apresentadas duas células, com os pontos em amarelo para serem servidos. O robô deve otimizar sua carga de peças e aproveitar o limite máximo de peso de carga para evitar movimentações desnecessárias, economizando, assim, tempo e energia.

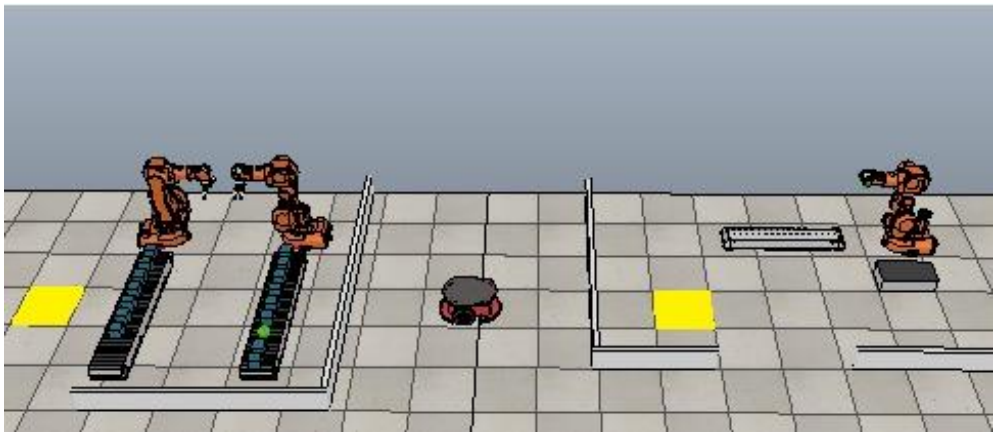


Figura 3.24: Células a serem servidas pelo robô móvel (pontos em amarelo são os lugares que o robô deve deixar as peças)

As células deste ambiente são compostas basicamente por esteiras, robôs manipuladores e humanos. Uma “barreira” deve impedir o robô móvel de interagir com os robôs manipuladores deste ambiente, evitando assim grandes problemas.

3.4.6. Testes no ambiente dinâmico e do sensoriamento robótico

Nesta seção serão apresentados os testes relacionados ao sensoriamento do robô feitos no ambiente modelado. Os testes tiveram como maior objetivo a validação do sensoriamento utilizado, assim, analisando variados tipos de sensores para a navegação do robô móvel em diferentes situações no ambiente.

O sensoriamento aplicado neste trabalho utilizou três tipos de sensores, sendo cada um aplicado para objetivos diferentes.

3.4.6.1. Sensoriamento ultrassônico

O sensoriamento ultrassônico para este trabalho foi utilizado para detectar um obstáculo na rota do robô. Durante a "varredura" do ambiente o sensoriamento ultrassônico deve detectar um obstáculo com a distância máxima de noventa centímetros em relação à base do robô. Foram utilizados oito sensores ultrassônicos para uma melhor cobertura do ambiente que está sendo explorado.

3.4.6.2. Sensoriamento óptico

O sensoriamento óptico foi utilizado para fazer a "varredura" do ambiente e gerar um gráfico de campo de visão do robô. Esta funcionalidade está ligada a casos em que deseja-se assistir o robô em tempo real por meio de um cliente instalado em uma máquina secundária. Para isso ser possível, o computador cliente deve acessar o computador servidor onde está sendo processado o controlador neurogenético em tempo real.

Uma vantagem de acompanhar a rota do robô por meio deste gráfico de visibilidade se comparado a uma câmera de visão, é que, o custo de se enviar dados para gerar este gráfico é muito menor que enviar imagens em tempo real. Sendo assim, a aplicação de monitoramento do robô se torna mais rápida e menos custosa.

Na figura 3.25 é apresentado o gráfico de visibilidade do sensor óptico em duas diferentes situações. Na figura 3.25 (a) pode ser observado o robô detectando a presença da empilhadeira como seu obstáculo atual.

Para um melhor entendimento do gráfico gerado pelo sistema de sensoriamento, uma análise estrutural do ambiente deve ser feita por meio da figura em questão. Na figura 3.25 pode ser observada a estrutura do ambiente, sendo esta, formada por duas paredes laterais ao robô *Pioneer,* e também, o obstáculo que está impedindo a navegação do robô, seja uma empilhadeira (figura 3.25 - a) ou um humano (figura 3.25 - b). Outros objetos não podem ser vistos no gráfico (figura 3.25 - a e b) por conta da oclusão gerada pelo impedimento do feixe do sensor quando detectadas as paredes e o obstáculo atual.

O obstáculo, em amarelo, pode ser visto claramente no gráfico de visibilidade do sensor óptico. As duas paredes laterais ao robô também podem ser vistas neste gráfico, já que este sensoriamento tem cobertura de 180 graus.

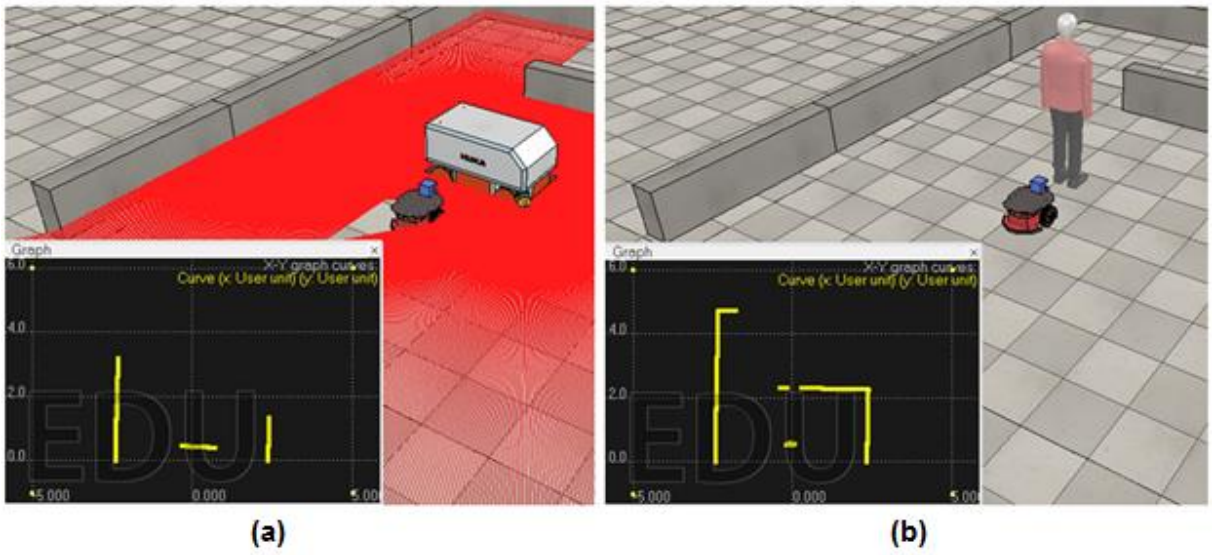


Figura 3.25: Gráfico de cobertura gerado pelo sensoriamento óptico (a) robô encontra a empilhadeira como obstáculo (a) robô encontra um humano como obstáculo

Na figura 3.25 (b) pode ser observado o obstáculo encontrado, sendo este um humano. Este obstáculo também pode ser visto claramente no gráfico de visibilidade (que pode ser visualizado na "janela" de campo de visão do sensor nesta mesma imagem). Neste caso o robô consegue “ver” a parede que está atrás do humano com um pequeno corte que foi gerado pela interferência do seu corpo.

Pode-se observar com maior simplicidade nesta figura o campo de visibilidade do sensor óptico sendo obstruído pelo humano, assim, gerando um corte no segmento da parede que está atrás do obstáculo detectado. Esta área de oclusão no sensoriamento óptico pode trazer problemas quando se deseja por exemplo calcular o tamanho de alguns obstáculos.

Foram feitos alguns testes para analisar o tamanho de obstáculos utilizando sensoriamento óptico, porém, neste trabalho, se tornou ineficiente este tipo de sensor para extrair às características de tamanho dos obstáculos.

Um *laser* com varredura de 180 graus, usual do modelo de sensor *Sick*ou e outro de 240 graus, usual da *Hokuyo* (COPELIA ROBOTICS, 2015) foram testados para a medição de obstáculos, gerando em seu campo de visão um semicírculo. O campo de visão é recortado pelos objetos que estão dentro dele. Assim fica mais difícil calcular o tamanho de todo o objeto, pois não se sabe qual dos “cortes” no feixe de visão deve ser tratado como obstáculo atual, ou seja, o obstáculo que está interferindo na rota do robô (BERRI, 2015).

Na figura 3.26 observa-se um problema referente à medição de obstáculos observado nas simulações. Neste caso é deficiente calcular o tamanho de um obstáculo apenas pelo contorno sem saber qual “dente” (o "dente" é o recorte gerado no feixe de *laser* do sensor) no sensoriamento óptico corresponde ao obstáculo detectado inicialmente pelo sensor de ultrassom e está impedindo a rota do robô. Para este caso a caixa está interferindo na rota do robô, porém outros 2 obstáculos estão sendo detectados pelo sensor óptico.

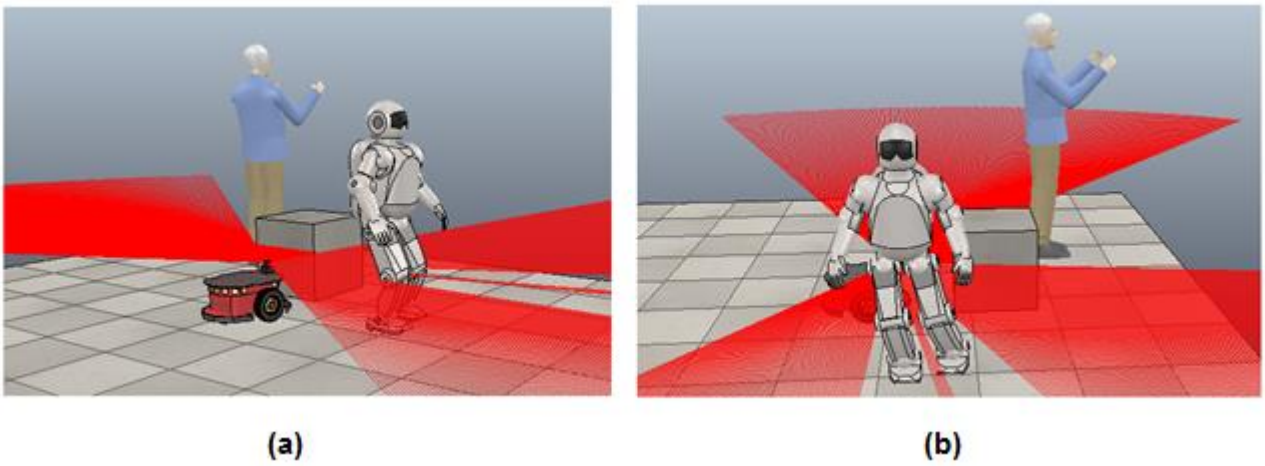


Figura 3.26 :(a,b) Problemática no sensoriamento óptico gerado por vários obstáculos detectados ao mesmo tempo

Não é de grande dificuldade calcular a distância da abertura do "dente" em termos métricos e consequentemente descobrir o tamanho do obstáculo. Porém, outro problema existe neste tipo de sensoriamento (a deficiência de não poder "ver" atrás do obstáculo) e, como consequência, o perímetro total do obstáculo não pode ser calculado com um sensor embarcado na plataforma do robô (BERRI, 2015). Para a medição de obstáculos, foi escolhido um método de visão computacional chamado de “bolha” e será descrito na próxima seção.

3.4.6.3. Sensoriamento por câmera

Para a medição dos obstáculos encontrados no caminho do robô, foi utilizada uma câmera para a visão computacional. A câmera utilizada foi a *blob detection*, a qual tem seu funcionamento baseado em detecção de regiões de uma imagem onde existem mudanças de propriedades como brilho e cor se comparado a regiões vizinhas da que se tem interesse (COPELIA ROBOTICS, 2015).

A câmera *blob detection* processa regiões de interesse, estas são chamadas de "bolhas", onde nessas regiões as propriedades de cor e brilho devem ser diferenciadas do restante da imagem. Neste trabalho a tarefa inicial da câmera é utilizar a lógica de "bolha" para obter as regiões de interesse na imagem e fazendo a “binarização” (a "binarização" é o

processo que deve deixar a imagem em apenas duas cores por meio de um limiar pré-definido) de cores entre o objeto e o fundo para um posterior processamento digital nesta região. Este processo deve diferenciar o objeto de interesse do seu fundo.

Antes de ser feita a "binarização" é preciso capturar a cor RGB da região de interesse. A variável cor em RGB será servida como a primeira entrada da RNA para o reconhecimento do obstáculo. Isto é feito antes da binarização, uma vez que a imagem ficará apenas vermelha e preta.

A região de interesse deve ser representada pela cor vermelha, e seu fundo que é constituído por todo o resto da imagem deve ser representado pela cor preta. Na figura 3.27 observa-se em suas “janelas” de campo de visão da câmera *blob detection* a "binarização" gerada pelo pré-processamento "bolha".

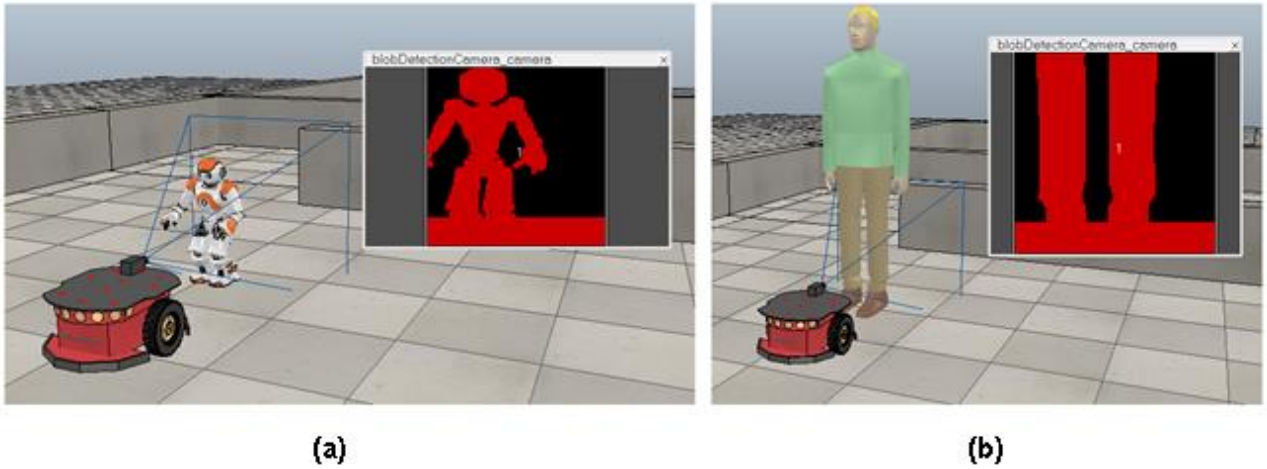


Figura 3.27: Detecção de obstáculos por "região bolha" (a) robô NAO sendo diferenciado por "região bolha" do fundo da imagem (b) pernas do humano sendo diferenciado por "região bolha" do fundo da imagem

A segunda etapa da visão computacional com este método é utilizar a região de interesse que já foi bem definida pelo pré-processamento e calcular o tamanho deste objeto, servindo como a segunda entrada para a RNA escolhida neste trabalho para reconhecer os obstáculos de maior frequência no ambiente dinâmico.

Na figura 3.27 pode-se observar dois casos em que diferentes obstáculos na rota do robô são pré-processados pelo método de "bolha". O primeiro caso, figura 3.27(a), a câmera *blob detection* detecta o robô NAO como obstáculo. Por ter um tamanho pequeno, a câmera consegue detectar e criar a "região bolha" para o obstáculo de forma total.

Já na figura 3.27(b) pode ser observado outro obstáculo, sendo este, um humano, e por ter a altura maior que a área de alcance em que está sendo submetida a visão da câmera, está criando a "região bolha" apenas para as pernas do humano. Neste caso a

medição de tamanho de obstáculos implicará apenas na medida das pernas e seu intervalo.

Com estas informações já é possível passar para a segunda fase de processamento da imagem e calcular o tamanho do obstáculo atual.

3.4.6.4. Proposta de aplicação do robô *YouBot*

Nesta seção é apresentada a proposta do robô *YouBot* para o carregamento de materiais entre as células do ambiente dinâmico, utilizando o robô da fabricante *KUKA*, que utiliza um braço robótico em sua plataforma para o carregamento e descarregamento das peças que serão transportadas sobre sua plataforma (COPELIA ROBOTICS, 2015).

O robô *YouBot* foi utilizado em algumas simulações para mostrar o funcionamento esperado quando as células devem ser servidas por peças. Estas simulações são apresentadas no capítulo 8. Na figura 3.28(a) é apresentado o robô *YouBot* pegando uma peça no estoque. Na figura 3.28(b) pode ser observado o robô *YouBot* carregando uma peça em sua plataforma.

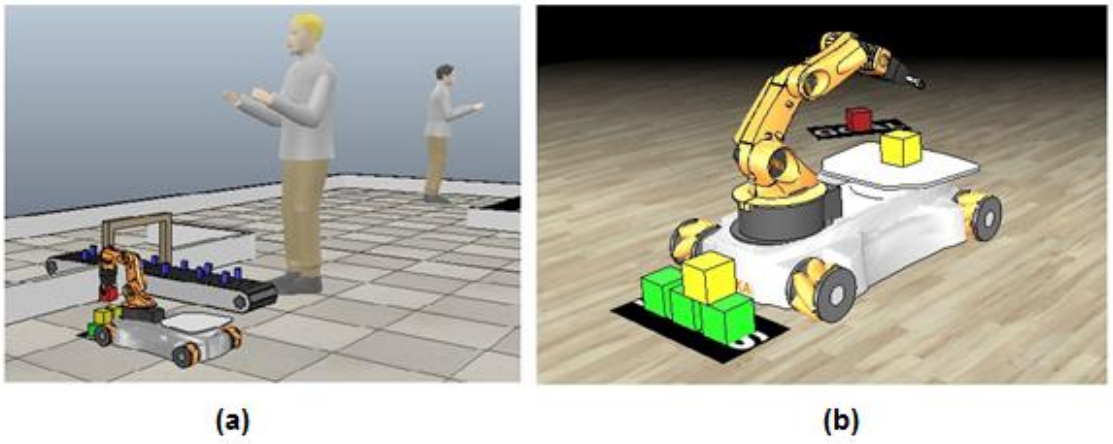


Figura 3.28: Robô *YouBot* fazendo seu trabalho de carregamento de peças (a) robô pegando uma peça no estoque (b) robô colocando uma peça em sua plataforma

O robô *Pioneer* não foi utilizado neste caso por não possuir um braço robótico, tendo então como consequência a necessidade de um humano ficar no estoque para carregar o robô e outros humanos para descarregar o robô em cada ponto que deve ser servido.

Porém, tratando-se de mobilidade e sensoriamento, o robô *Pioneer* tem um desempenho muito melhor e tem similaridade com o *Kihon*, plataforma robótica que foi proposta para servir como base para os testes do controlador neurogenético desenvolvido neste trabalho, com vistas para uma futura implementação deste controlador no *hardware* deste robô.

Para unir as características positivas dos robôs *Pioneer* e *YouBot,* foi criado um robô híbrido que pode ser visto na figura 3.29(b). Porém, a plataforma do *Pioneer* (figura 3.29 - a) não suporta fisicamente este braço, gerando instabilidade no robô e consequentemente o seu tombamento. Mesmo com o uso do grupo de rodas do *YouBot* na plataforma *Pioneer,* o problema se manteve.

Com esta deficiência conclui-se que o simulador apresenta instabilidade na reconstrução de robôs sem que seja feita as devidas alterações nas propriedades físicas internas do simulador. Estas alterações físicas não são disponíveis para os usuários do simulador.

Para os testes com o robô *YouBot,* foram definidos os pontos de objetivo do robô, ou seja, os pontos a serem servidos com peças. Foram utilizados blocos de diferentes cores para representar diferentes tipos de peças.

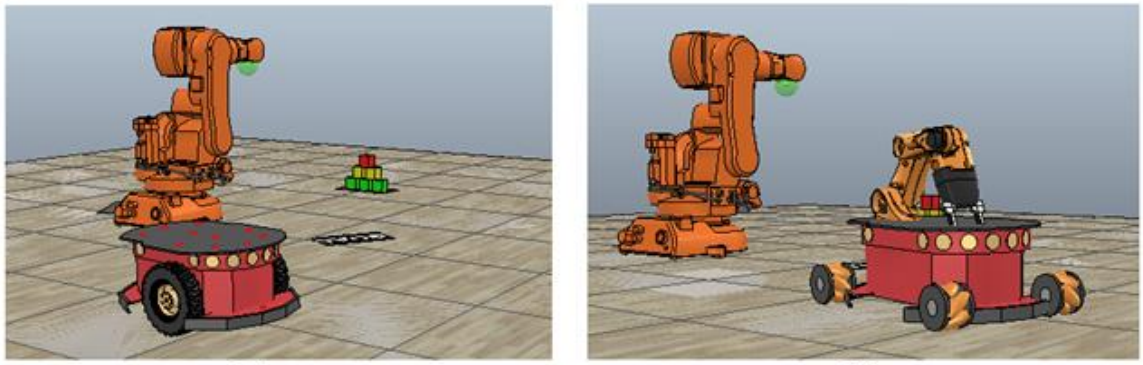


Figura 3.29: (a) robô *Pioneer* e (b) robô híbrido

Para definir os pontos de objetivo do robô foram criados "tapetes", e a estes são atribuídos os valores de coordenadas cartesianas do GPS para o qual o robô deve se orientar para conseguir chegar até este ponto pré-definido e servi-lo com as peças. No capítulo 6 é apresentado com melhor clareza as funções de navegação autônoma do robô.

Capítulo 4

Visão Computacional aplicada na análise e reconhecimento de obstáculos frequentes com vistas para o tratamento de dados para a RNA

Este capítulo foi dedicado exclusivamente ao Processamento Digital de Imagens(PDI) e à Visão Computacional(VC) utilizados para analisar os obstáculos encontrados na rota do robô móvel. Para um melhor entendimento das técnicas apresentas neste capítulo é indicado o livro de Processamento Digital de Imagens de Gonzalez (2011).

Em resumo os robôs móveis são altamente dependentes de sistemas de sensoriamento que os auxiliem na percepção do ambiente em que estão desenvolvendo seu trabalho, uma vez que eles precisam tomar decisões baseando-se no conhecimento do ambiente, assim, sendo possível navegar de maneira segura com o potencial de solucionar obstáculos encontrados em sua rota. Neste trabalho a VC deve auxiliar o robô na sua navegação. O objetivo principal do sistema de visão é avaliar cada obstáculo e extrair as características que são relevantes para o tratamento dos obstáculos em prol à RNA.

Os sistemas de VC aplicados a robótica móvel devem ser precisos e com custo de tempo relativamente baixo, pois este sistema deve trabalhar em tempo real, assim como todo sensoriamento e tomada de decisão do robô. Por este motivo, neste trabalho foi utilizado um sistema de VC que trabalha de forma otimizada gerando menor carga de processamento computacional. Algumas técnicas foram estudadas até que fosse possível encontrar o melhor conjunto de algoritmos para realizar a funcionalidade esperada da VC para o reconhecimento de obstáculos.

As variáveis analisadas em cada obstáculo foram as cores em RGB e o tamanho. O objetivo da análise destas variáveis em específico, usando algoritmos baseados em duas vertentes da área cientifica de imagens digitais, é que estas variáveis são as entradas da RNA *perceptron* que deve avaliar um obstáculo e declarar o mesmo reconhecido ou não por meio destas variáveis.

O sistema de VC aplicado no controlador neurogenético utilizou como sensor de captura de imagens a câmera *blob detection.* Uma descrição detalhada deste sensor foi feita no capítulo 3.

A VC foi utilizada com vistas para a extração das características dos obstáculos do ambiente dinâmico, gerando dados textuais a serem aplicados nas entradas da RNA. Quando a rede reconhece um obstáculo, uma busca em sua base de conhecimento é feita para que a solução para este obstáculo seja encontrada e aplicada para o desvio da rota do robô.

O sistema de VC tem como entrada várias imagens obtidas pela câmera *blob detection* e que são pré-processadas utilizando a lógica de "bolha". Este tratamento inicial utilizando

PDI tem como objetivo filtrar as imagens e eliminar delas informações que são irrelevantes para a VC do robô.

A lógica de "bolha" utilizada pela câmera *blob detection* nada mais é que uma "binarização" (processamento que deixa a imagem com apenas duas cores) da imagem para diferenciar o objeto detectado do seu fundo. A região que representa o obstáculo é chamada de "região bolha".

A literatura básica de VC aplicada à robótica móvel costuma chamar este tipo de lógica de extração de fundo, onde o fundo da imagem em que se encontra o obstáculo é eliminado evidenciando apenas o obstáculo em questão.

Um algoritmo bastante conhecido para extração de fundo é o EGMM *(Enhanced Gaussian Mixture Model)*, que foi desenvolvido por Zivkovic et.al (2006). Este algoritmo é largamente utilizado na robótica móvel para identificação de regiões navegáveis. A função deste algoritmo é eliminar da imagem o que não deve ser analisado pelo sistema de VC.

No sistema de visão desenvolvido para este trabalho, o algoritmo de "binarização" aplicado ao PDI deixou a região de interesse da imagem evidenciado com a "região bolha". A região de interesse deve representar o obstáculo. Com isso o algoritmo de "binarização" contribuiu potencialmente com o sistema de VC para a extração de características, já que a binarização da imagem deve segmentar o obstáculo e criar a "região bolha".

Explicando de maneira simplificada, o PDI teve como objetivo fazer o pré-processamento da imagem criando a "região bolha" para a região de interesse na imagem. Na região de interesse deve estar o obstáculo detectado. Para a definição da região de interesse a imagem é "binarizada" diferenciando o obstáculo do seu fundo.

Como a VC receberá as imagens binarizadas, a extração das cores do obstáculo detectado na "região bolha" deve ser obtido pelo algoritmo de extração de cores utilizando a imagem real. Obviamente este processo seria impossível na imagem binarizada. O diagrama de blocos da interconexão da funcionalidade destes algoritmos pode ser visto na figura 4.1.

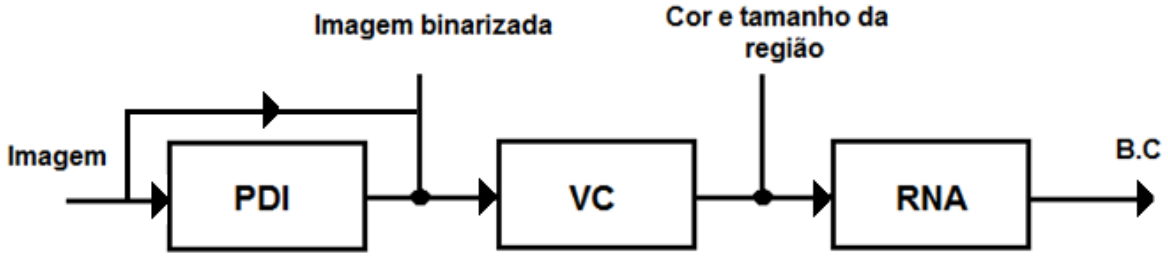


Figura 4.1: Sistema de visão do robô

O primeiro passo está ligado ao fornecimento da imagem por meio do sensor de visão *blob detection* ao algoritmo de PDI. O pré-processamento é feito, a "região bolha" é detectada pela variação de cor e intensidade de brilho da imagem gerando a imagem "binarizada". A imagem "binarizada" e a imagem real são enviadas para o sistema de VC que utilizará apenas a região de interesse na extração de características.

O segundo passo está ligado ao tratamento da imagem "binarizada" pelos algoritmos de VC. A extração da cor é feita na imagem real (colorida) em função da imagem "binarizada" e sua "região bolha" que representa o obstáculo. O algoritmo para o cálculo do tamanho do obstáculo também deve utilizar a "região bolha" para obter o tamanho do obstáculo.

A cor e o tamanho devem ser enviados para RNA e serem avaliados para que o obstáculo seja reconhecido ou não. A última etapa é ligar à saída da RNA ao Banco de Conhecimento (B.C). Uma descrição detalhada deste processo foi feita no capítulo 3.

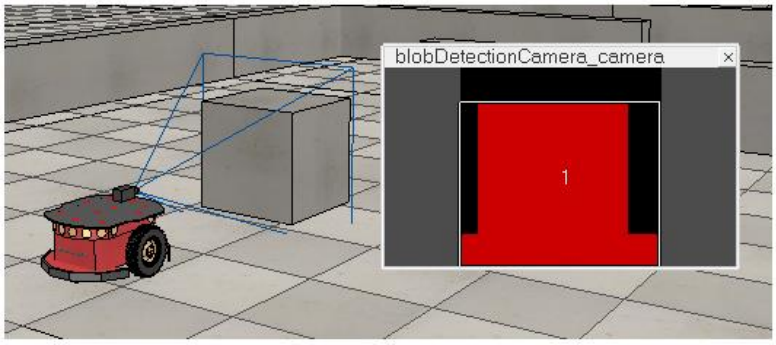


Figura 4.2: PDI da visão do robô

Na figura 4.2 pode ser observado o obstáculo caixa sendo detectado por meio do algoritmo de "binarização". A binarização entre o obstáculo e seu fundo pode ser observada na "região bolha" entre as cores: preto e vermelho, em que a cor vermelha representa o obstáculo e a cor preta o seu fundo. Uma melhor descrição desta câmera e seu pré-processamento foi feita no capítulo 3.

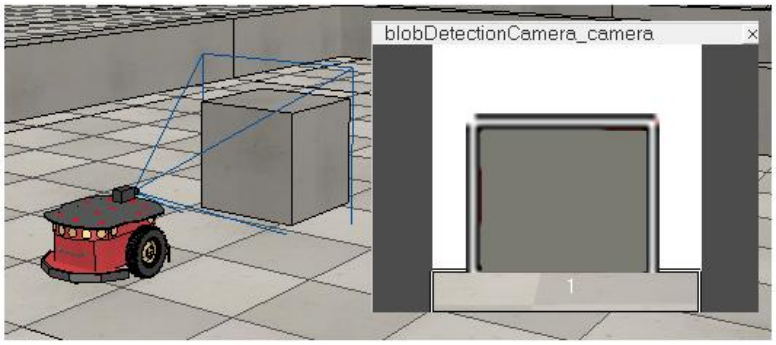


Figura 4.3: VC do robô

Na figura 4.3 pode ser observada a "região bolha" criada pelo algoritmo de "binarização" sendo aplicada sobre a imagem real. A análise feita pela VC sobre esta imagem está ligada às bordas da região de interesse que foram bem definidas na imagem "binarizada". Pode-se observar que o fundo foi eliminado. O fundo não deve ser eliminado totalmente em obstáculos não uniformes, assim, não sendo possível avaliar especificamente o obstáculo em questão por meio da "região bolha". Por este motivo foi criado um algoritmo que "cria" uma região de interesse (máscara) sobre a "região bolha". A máscara deve conter o obstáculo definido pela "região bolha" e também alguns "pedaços" do fundo do obstáculo. Uma melhor explicação sobre este processo é feita ao longo deste capítulo.

O algoritmo de "binarização" foi o único utilizado para a obtenção das bordas dos obstáculos. Um filtro de detecção de bordas poderia ser aplicado para uma melhor definição da "região bolha". Contudo, o custo computacional de um algoritmo para um filtro destes seria muito alto.

A função de reconhecimento de um obstáculo deve ser feita potencialmente pela RNA, então uma região definida robustamente pelo algoritmo de binarização foi o suficiente para este sistema de VC.

O filtro de detecção de bordas de Sobel foi implementado porém seu custo computacional ultrapassou o custo total do algoritmo genético. O custo máximo de tempo do AG é igual a 30 milissegundos, já o custo do filtro de Sobel gerou um custo de 3000 milissegundos. Por este motivo foi eliminado este filtro do sistema de VC. A análise de tempo foi feita utilizando o mesmo método apresentado no capítulo 9. Método que foi aplicado para a análise de custo de tempo do controlador neurogenético.

4.1. Algoritmos utilizados no sistema de visão e reconhecimento de imagens

Esta seção tem como objetivo apresentar os algoritmos utilizados no sistema de visão robótica com vistas para o reconhecimento de obstáculos pela RNA *perceptron.* Todos os algoritmos utilizados neste sistema foram modelados e simulados no *software* matemático *Scilab* antes que fossem implementados efetivamente para o auxílio do controlador neurogenético. As simulações e resultados são apresentados nas próximas seções.

4.1.1. Algoritmos de Processamento Digital de Imagens

O PDI foi feito por dois algoritmos, sendo eles: algoritmo de quantização de cores e o algoritmo de "binarização" da imagem. O algoritmo de quantização de cores é aplicado juntamente com os algoritmos de VC antes mesmo que o algoritmo de extração de cores seja aplicado. Já o algoritmo de "binarização" deve ser aplicado no momento que o sensor de imagem fornecer a imagem real. A imagem "binarizada" deve ser enviada para a etapa de VC.

4.1.1.1. Algoritmo de binarização de imagens

O algoritmo de "binarização" deve ser a base para o reconhecimento da região de interesse e diferenciar o obstáculo em relação ao fundo da imagem. A "binarização" é feita em relação à região que representa o possível obstáculo e o fundo da imagem. Essa imagem "binarizada" será utilizada potencialmente pelo algoritmo de VC.

Para a binarização da imagem foi utilizado o algoritmo de Sauvola (SAUVOLA, 2000). O algoritmo de Sauvola utiliza um valor de limiar referente a 128. O valor de limiar é aplicado a todos os *pixels* da imagem. Sauvola (2000) chegou aos melhores resultados utilizando este valor de limiar.

Quando o valor do *pixel* é superior (ou inferior) ao limiar é provável pertencer ao obstáculo. Caso contrário pertence ao fundo.

O código conforme apresentado na janela a seguir, é referente ao algoritmo para "binarização" de imagens proposto por Sauvola (2000).

```
F = imread('C:\PDI\humano_4.bmp');

[L C] = size(F);
for i = 2:L
   for j = 2:C
       if (F(i,j) >= 0 & F(i,j) <= 128) then
           G(i,j) = 0;
       else
           G(i,j) = 255;
        end
    end
end

imshow(G);
```

Na figura 4.4 é apresentado o resultado da simulação do algoritmo de "binarização" que foi modelado no ambiente *Scilab.* Este algoritmo é utilizado pela câmera para a detecção da "região bolha". Na figura pode ser observada a imagem real à esquerda e a imagem "binarizada" à direita.

Nesta mesma figura pode ser observado o obstáculo humano sendo detectado pela "região bolha" (cor branca) e sendo diferenciado do fundo da imagem (cor preta) pelo algoritmo de Sauvola (2000). Esta imagem deve passar por um filtro para eliminação dos ruídos, pois caso contrário a região que representa o obstáculo pode não ficar bem definida.

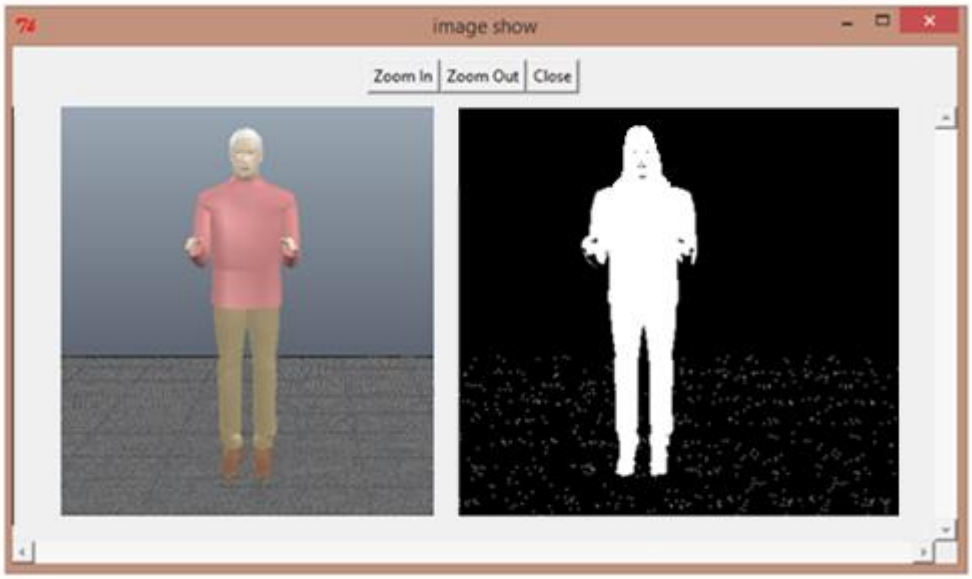


Figura 4.4: Imagem do humano sendo binarizada pelo algoritmo de Sauvola (2000)

Estes ruídos são facilmente observados no chão do ambiente em que o humano está pisando (figura 4.4). Porém, neste trabalho estes ruídos não foram constatados nas imagens geradas pelo PDI da câmera *blob detection* (figura 4.5 - b) no ambiente de simulações V-REP, assim, foi dispensado o uso de um filtro para a eliminação dos ruídos que foram identificados nas imagens geradas nas simulações por meio do *software Scilab.*

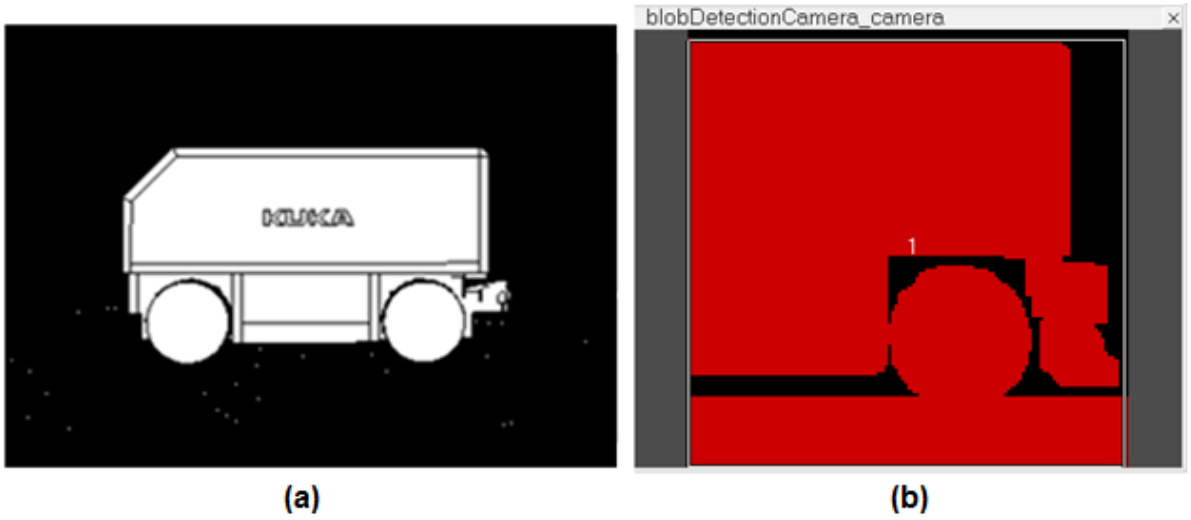


Figura 4.5: (a) Imagem da empilhadeira "binarizada" pelo algoritmo de Sauvola (2000); (b) "região bolha" gerada pelo simulador V-REP

Um reajuste no valor do limiar proposto por Sauvola (2000) no seu algoritmo de "binarização" pode mudar a imagem final e diminuir o nível de ruídos na imagem.

Na figura 4.5 (a), pode ser observada uma imagem "binarizada" com limiar de 150 e, neste caso, o nível de ruído foi bem menor se comparada a imagem "binarizada" da figura 4.4 que utilizou um limiar de 128. No entanto, a média do valor de limiar para bons resultados, na maioria das imagens, permaneceu no valor de 128, assim como proposto por Sauvola (2000).

No caso de uma implementação deste PDI em um ambiente real seria necessário o uso de um filtro morfológico, pois as simulações do algoritmo de "binarização" via *Scilab* apresentaram ruídos que devem gerar problemas relevantes para a identificação da região que representa o obstáculo. Para esta problemática seria necessário um filtro morfológico para suavização e eliminação de ruídos (Gonzalez, 2011).

Neste trabalho não foi aplicado este filtro, pois o simulador de robôs desconsidera os ruídos gerados no pré-processamento da câmera *blob detection.* Na figura 4.5 (b) pode ser observada a imagem "binarizada" e livre de ruídos gerada pelo sistema de PDI implementado em linguagem LUA no *software* V-REP.

4.1.2. Algoritmos de Visão Computacional

A VC foi feita por 3 algoritmos, sendo eles: o algoritmo para detecção de região de interesse, o algoritmo para quantização e extração de cores e o algoritmo para calcular o tamanho da área de interesse.

O algoritmo de detecção de região de interesse deve trabalhar sobre a imagem "binarizada" e sua "região bolha".

Para definir à região a ser feita a extração de cores e ser calculado o tamanho da área é utilizada a imagem real e a imagem "binarizada" respectivamente.

A medição do obstáculo é feita por meio da contagem de *pixels* na região de interesse (máscara) que representa o obstáculo nesta imagem. A extração de cores é feita por meio da região de interesse representada na imagem real.

4.1.2.1. Algoritmo para detecção da região de interesse

O algoritmo utilizado para detecção de região de interesse deve detectar todos os *pixels* brancos agrupados em maior proporção que representam o possível obstáculo e definir uma máscara para representar o seu tamanho.

Este algoritmo deve definir a região de interesse (máscara) para que o algoritmo de extração de cores e o algoritmo de cálculo de tamanho trabalhem sobre esta região. A

região que representa o obstáculo pode ser observada na figura 4.6 onde o obstáculo empilhadeira e o humano são bem definidos. Esta região de interesse será chamada de máscara nos algoritmos de VC.

O tamanho da máscara (o retângulo vermelho sobre cada obstáculo) foi definido como padrão de entrada para treinamento da RNA, ou seja, a região de interesse deve sempre ter o mesmo valor de tamanho para cada tipo de obstáculo. Sendo assim o tamanho do obstáculo é declarado como reconhecido pela rede. O tamanho é definido pela contagem de *pixels* na região de interesse.

Uma margem de tolerância a erros foi definida para o cálculo do tamanho do obstáculo. Ou seja, o tamanho da região não precisa ser exato para que o obstáculo seja reconhecido. O treinamento da RNA deve ser feito em função das variações de tamanho para o mesmo tipo de obstáculo. A rede deve trabalhar como um classificador e definir este obstáculo como reconhecido ou não reconhecido.

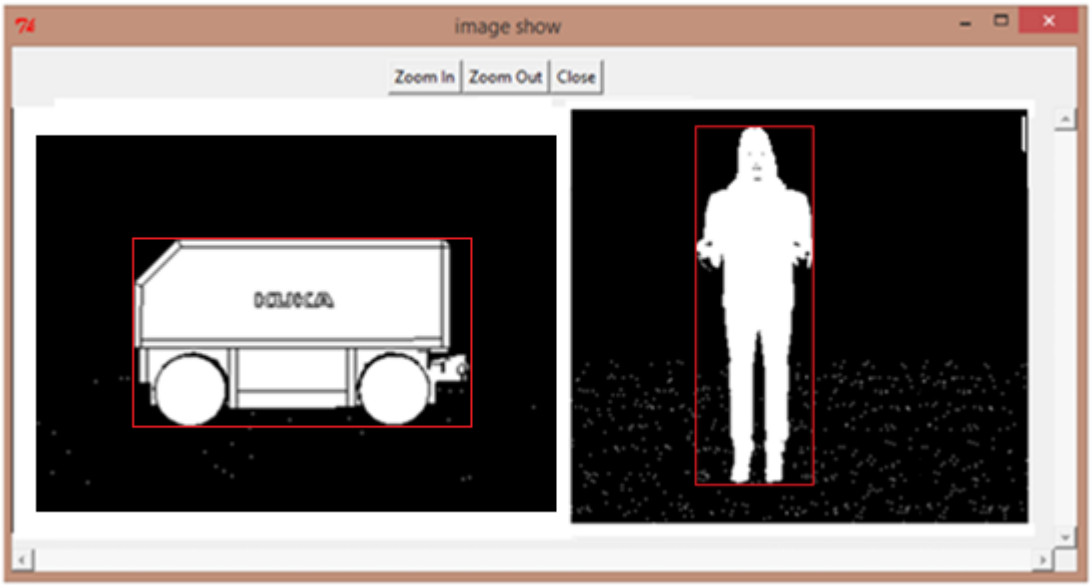


Figura 4.6: Definição da região que representa o obstáculo

Na figura 4.6 pode ser observado apenas um ângulo de visão do robô para detectar um obstáculo, mas outros ângulos foram definidos e apresentados como padrões à RNA em sua fase de treinamento para que outros tamanhos de região de um mesmo obstáculo pudessem ser declarados reconhecidos pela RNA.

O robô pode encontrar o obstáculo em diferentes posições, como exemplo: frontal ou lateral. Na figura 4.7 pode ser observado o obstáculo humano e a empilhadeira sendo detectados por outros ângulos de visão.

Na figura 4.7 pode-se observar também outras duas regiões de interesse que são apresentados como padrão ao treinamento da RNA. Estas imagens são geradas por

diferentes ângulos de visão e captura do sensoriamento por câmera, desempenhando um papel fundamental para bons resultados de treinamento e reconhecimento do algoritmo neural. Pois, nem sempre o obstáculo será detectado pelo robô com o mesmo ângulo de visão.

Vários tamanhos são definidos para cada tipo de obstáculo em diferentes ângulos de visão. Estas informações são utilizadas como padrões de dados de entrada para o algoritmo de treinamento da RNA em prol de uma maior robustez do sistema neural .

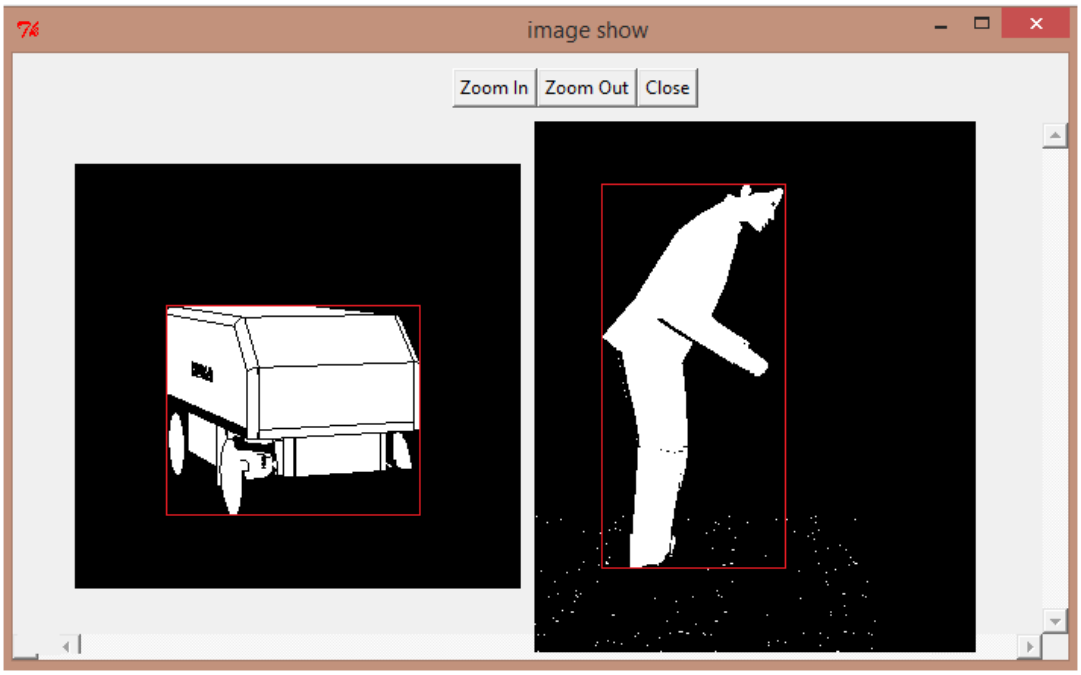


Figura 4.7: Definição da região que representa o obstáculo (visão em outros ângulos)

O trecho de código apresentado a seguir tem como objetivo comparar qual obstáculo foi detectado de acordo com sua dimensão. A sequência de obstáculos em função de seu tamanho menor para o maior é : caixa[1], empilhadeira[2] e humano[3]. Porém este código é apenas um comparador, e não deve ser utilizado como um classificador preciso.

O comparador apresentado analisa o tamanho de 3 obstáculos cadastrados em uma base de dados. Poderia ser criado um algoritmo baseado em lógica *fuzzy* para o tratamento destes obstáculos por meio de um conjunto de regras e uma base de dados.

Porém neste trabalho as funções de comparação foram todas aplicadas a RNA. O motivo está ligado ao reaprendizado da rede e seu potencial em reconhecer obstáculos com variações de características. Um comparador de características utilizando lógica *fuzzy* com este mesmo potencial teria uma maior complexidade se comparado ao controlador neural.

O trecho de código para comparação de três obstáculos de tamanhos diferentes é apresentado na janela a seguir (COPELIA ROBOTICS, 2015).

```
 -- Comparador para o tamanho dos obstáculos

-- Avaliação do contorno dos obstáculos (para 3 obstáculos diferentes)

if(Blob_Dim[1]>Blob_Dim[2]) and (Blob_Dim [1]>Blob_Dim[3])
then
     tmp=blobBoxDimensions[1] --objeto 1 [caixa]
ifelse(Blob_Dim[2]>Blob_Dim[1]) and (Blob_Dim[2]>Blob_Dim[3])
then
     tmp=blobBoxDimensions[2] --objeto 2 [empilhadeira]

ifelse(Blob_Dim[3]>Blob_Dim[1]) and (Blob_Dim[3]>Blob_Dim[2])
then

     tmp=blobBoxDimensions[3] --objeto 3 [humano]

     end
end
```

A distância que o robô faz a captura da imagem é sempre a mesma, pois a câmera é acionada apenas no momento em que o sensoriamento ultrassônico detecta um obstáculo. A distância máxima que o sensor ultrassônico detecta um obstáculo na rota do robô é constante.

Pelo fato do sensoriamento ultrassônico se manter com uma distância de captura de obstáculos constante, a medida da dimensão para os obstáculos capturados pela câmera devem manter uma margem de tolerância a variação de tamanho.

Os cinco sensores ultrassônicos (S1, S2, S3, S4, e S5) de maior potencial para a detecção de obstáculos são representados na figura 4.8.

Os sensores ultrassônicos tem uma área de cobertura de 90 centímetros (cm) em linha reta. Um obstáculo não pode ser detectado em uma distância superior a esta. A câmera

deve capturar a imagem no momento em que o sensor ultrassônico detectar um obstáculo, assim, evitando grandes variações no tamanho dos obstáculos detectados.

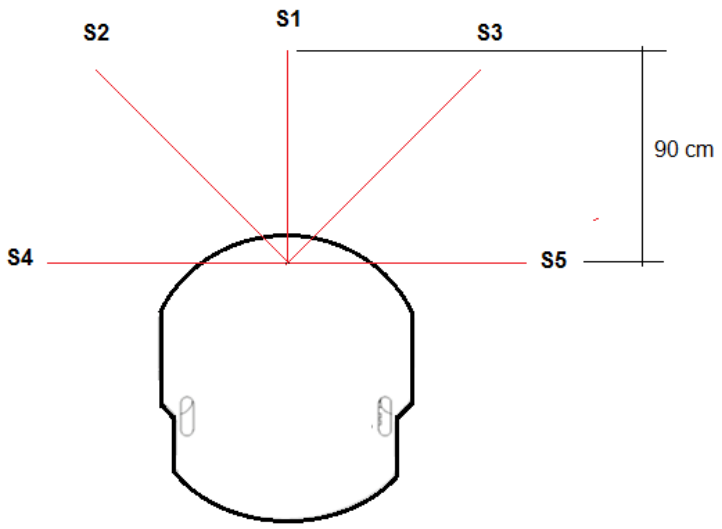


Figura 4.8: Sensoriamento ultrassônico com distância de detecção de obstáculos constante

É apresentado na janela, a seguir, um trecho do código que define a região de interesse em uma imagem qualquer por meio de uma imagem já “binarizada”.

Para início do procedimento, é feita uma "varredura" na imagem e, quando um *pixel* branco é encontrado, este é pintado de vermelho. Se um *pixel* preto é encontrado, ele é pintado de preto novamente garantindo que fique preto. Este procedimento é feito ainda na fase de PDI para criar a "região bolha" e diferenciar o obstáculo do fundo.

```
int l, c;

     for (l = 0; l < linhas; l++){
               for (c = 0; c < colunas; c++){

          if pixel == branco
               then pixel = pinta_vermelho
          else
                  then pixel = pinta_preto

// guarda valor da cor na matriz

região_bolha = matriz_pontos(M, l, c); //guarda região
```

Uma nova imagem binária é criada, desta vez, com a informação dos *pixels* que representam o obstáculo.

Um outro laço é utilizado para criar uma nova imagem que deve conter apenas os *pixels* que representam a região do obstáculo (máscara). Nesta imagem posteriormente será feita a extração de características para a RNA. É apresentado na janela, a seguir, um trecho do código que define a região de interesse em uma imagem qualquer por meio de uma imagem já “binarizada”. Este processo deve auxiliar o algoritmo de extração de cores e o algoritmo para o cálculo de tamanho, assim, guardando as coordenadas da "região bolha" em uma matriz.

```
// faz a varredura na imagem binarizada

for (l = 0; l < linhas; l++) {
     for (c = 0; c < colunas; c++){

//  cria  nova  imagem  apenas  com  os  pixels  que  representam  o
obstáculo

          if pixel == vermelho

               then [M] (L,C) = 1 // obstáculo
          else
               then [M] (L,C) = 0 // fundo

     posição_bolha = matriz_posição(M, l, c)
```

A nova imagem criada é uma máscara que contém informações de grande importância que, por sua vez, estão ligadas às coordenadas dos *pixels* representando um obstáculo da imagem real. Posteriormente, a nova imagem será, então, utilizada para extrair as características de cor e tamanho dos obstáculos. Na variável *matriz_posição* devem ser guardadas as posições que representam o obstáculo na imagem real. Uma melhor explicação do método de extração de cores foi feita na próxima seção

4.1.2.2. Algoritmo de extração de cores

O algoritmo de extração de cores deve extrair as cores da imagem real e enviá-las para a RNA. O algoritmo utiliza a máscara gerada como região de interesse pelo algoritmo de detecção dessas regiões e por meio das coordenadas dos *pixels* que definem esta máscara deve ser aplicada sobre a imagem real para extrair as cores apenas desta região bem definida.

A máscara "binarizada" é utilizada para detectar a região de interesse na imagem real com suas cores quantizadas (a quantização de cores é utilizada para reduzir a carga de informação referente as cores da imagem). Para isso a máscara deve ser colocada sobre a imagem real e "recortar" a região de interesse que representa o obstáculo detectado.

Por meio deste "recorte", devem ser extraídas as cores de interesse a serem enviadas para a RNA.

Na região de interesse da imagem real deve-se quantizar suas cores para que o tratamento pela RNA seja facilitado.

A rede, neste caso, é utilizada como um classificador, onde declara uma cor como conhecida ou não. Lembrando que, apenas o reconhecimento da cor do obstáculo não o declara reconhecido, pois o reconhecimento é dependente também do tamanho do obstáculo.

A quantização de cores deve trabalhar sobre os três canais, sendo estes canais representados em uma variação de 0 à 255: R(*Red),* G (*Green*) e B(*Blue).*

Para a quantização das cores foram definidos 8 intervalos para representar os níveis de cores de 0 a 255. A quantização foi feita nos 3 canais (RGB). Na figura 4.9 pode ser observada a representação da quantização nos 3 canais utilizando 8 intervalos pré-definidos.

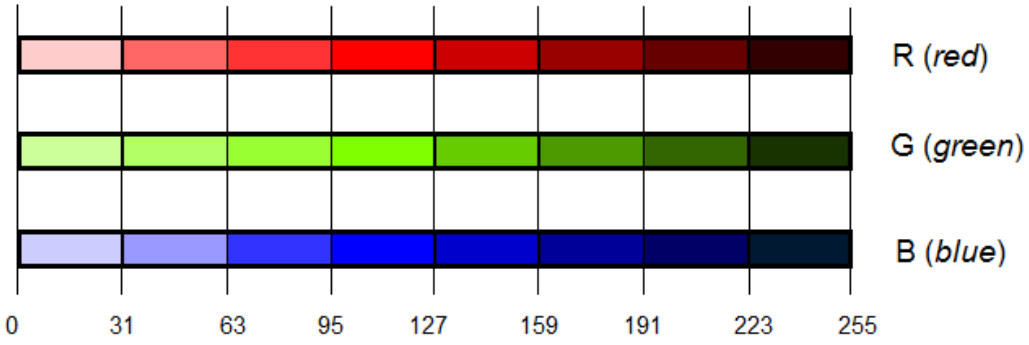


Figura 4.9: Quantização das cores RGB

A quantização das cores RGB teve como objetivo reduzir o custo computacional da RNA. O número de tipos de cores utilizando o modelo RGB real teria uma variação de 0 a 255 para cada canal, totalizando para as 3 cores 255^3. Com a quantização este valor cai drasticamente para 8^3.

Na figura 4.10(a) pode ser observada a máscara de região de interesse "binarizada" sendo sobreposta na imagem real. A região que foi definida por meio do algoritmo para detecção de região de interesse deve ser utilizada como máscara.

A máscara deve ser sobreposta na imagem real (colorida) para extração das cores em sua posição original e enviá-las para a RNA. Obviamente a extração de cores não seria possível na imagem binarizada.

Com este método, a região de interesse é "recortada" da imagem real. Por meio deste recorte da região de interesse é que será feita a análise de cores pelas RNA (figura 4.10 - b).

Lembrando que, antes de serem enviadas as cores deste "recorte" da imagem para a RNA deve ser feita a quantização de cores.

Existem vários métodos de extração de cor na VC, porém neste trabalho foi utilizado o método baseado em regiões fixas. Para este algoritmo a região fixa é representada pelo "recorte" que foi feito na imagem real quantizada (figura 4.10 - b).

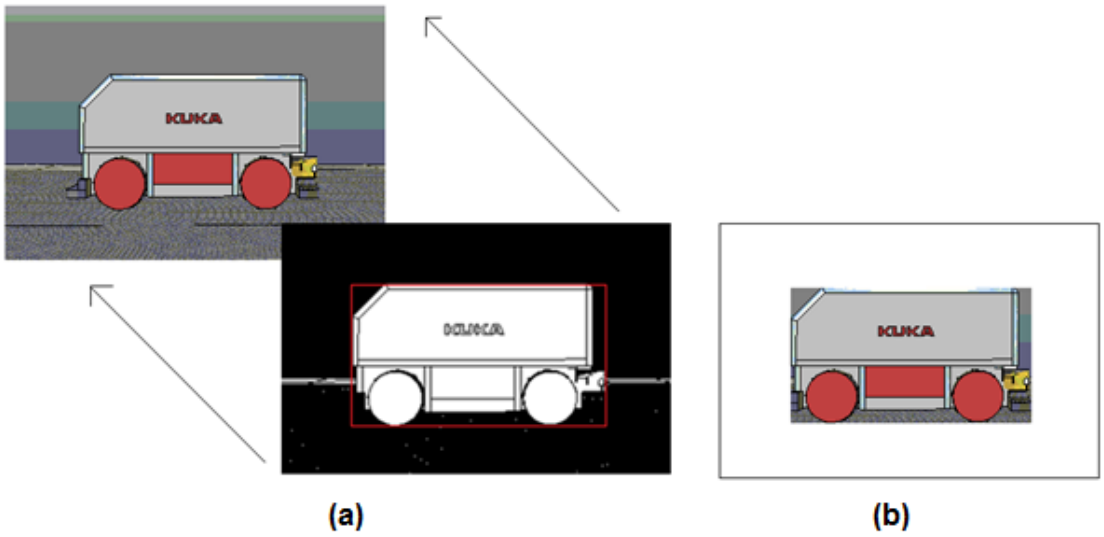


Figura 4.10: Recorte de região de interesse (a) máscara sobreposta sobre a imagem real e (b) recorte da região de interesse na imagem real

A função da RNA é reconhecer a média de cores nesta região de interesse avaliando cada *pixel* da imagem. Com a RNA treinada, por meio dos padrões de cores referentes a cada obstáculo e apresentados à rede na fase de treinamento, deve-se declarar esta cor reconhecida ou não pela rede.

O trecho de código conforme apresentado na janela a seguir, tem como objetivo, indicar para a função de extração de cores onde está a região de interesse. Definida a região de interesse as cores podem ser extraídas de alguns pontos *(x,y)* pré-definidos ou extrair as cores de todos os *pixels* desta região fazendo uma varredura nas linhas (L) e colunas (C).

```
matriz_posição = [p1;p2;p3;p4]; //quatro pontos que definem o
retângulo da região de interesse
```

Este código trata das posições *(L,C)* da imagem "binarizada", pois deve analisar a "região bolha" primeiramente e depois exportar estas posições para a imagem real e gerar o recorte. As posições para gerar o recorte são fornecidas pelo algoritmo de detecção de região de interesse.

O código que define a máscara trabalha apenas com valores 0 e 1 por, evidentemente, tratar-se de uma análise de imagem binária. Isso gera uma simplicidade se comparada ao reconhecimento da imagem colorida que tem uma variação de 0 à 255 cores para cada canal.

Quando um *pixel* referente ao obstáculo é analisado, sua posição é obtida por meio da máscara (binária) e aplicada na imagem colorida (com as cores quantizadas) e sua cor é armazenada em uma matriz (*matriz_cores[M]*).

A RNA utiliza a *matriz_cores[M]* para seus dados de entrada. As cores são enviadas para as entradas da rede no formato RGB. Uma melhor descrição do funcionamento da RNA foi feita no capítulo 3.

O trecho de código conforme apresentado na janela a seguir faz a leitura de todos os *pixels*. É feita uma varredura dos *pixels* na horizontal (linhas) e na vertical (colunas). Tanto na horizontal como na vertical é incrementada uma posição na leitura a cada vez que é feita a leitura do laço *for*. Este procedimento é repedido até que a região de interesse (máscara) termine.

```
int l, c;
     for (l = 0; l < linhas; l++){
           for (c = 0; c < colunas; c++){

cores_bolha = matriz_cores(M, l, c); // guarda valor da cor na
matriz
```

A variável *cores_bolha* guarda as cores dos *pixels* extraídos da região de interesse nas posições da matriz que representam cada *pixel* que está sendo avaliado na "região bolha". A RNA deve utilizar como entrada as cores que estão armazenadas nesta matriz. O funcionamento da rede foi melhor explicado no capítulo 3.

Na figura 4.11 pode ser observado o processo de extração de cores da imagem e envio dos dados de forma textual para a RNA.

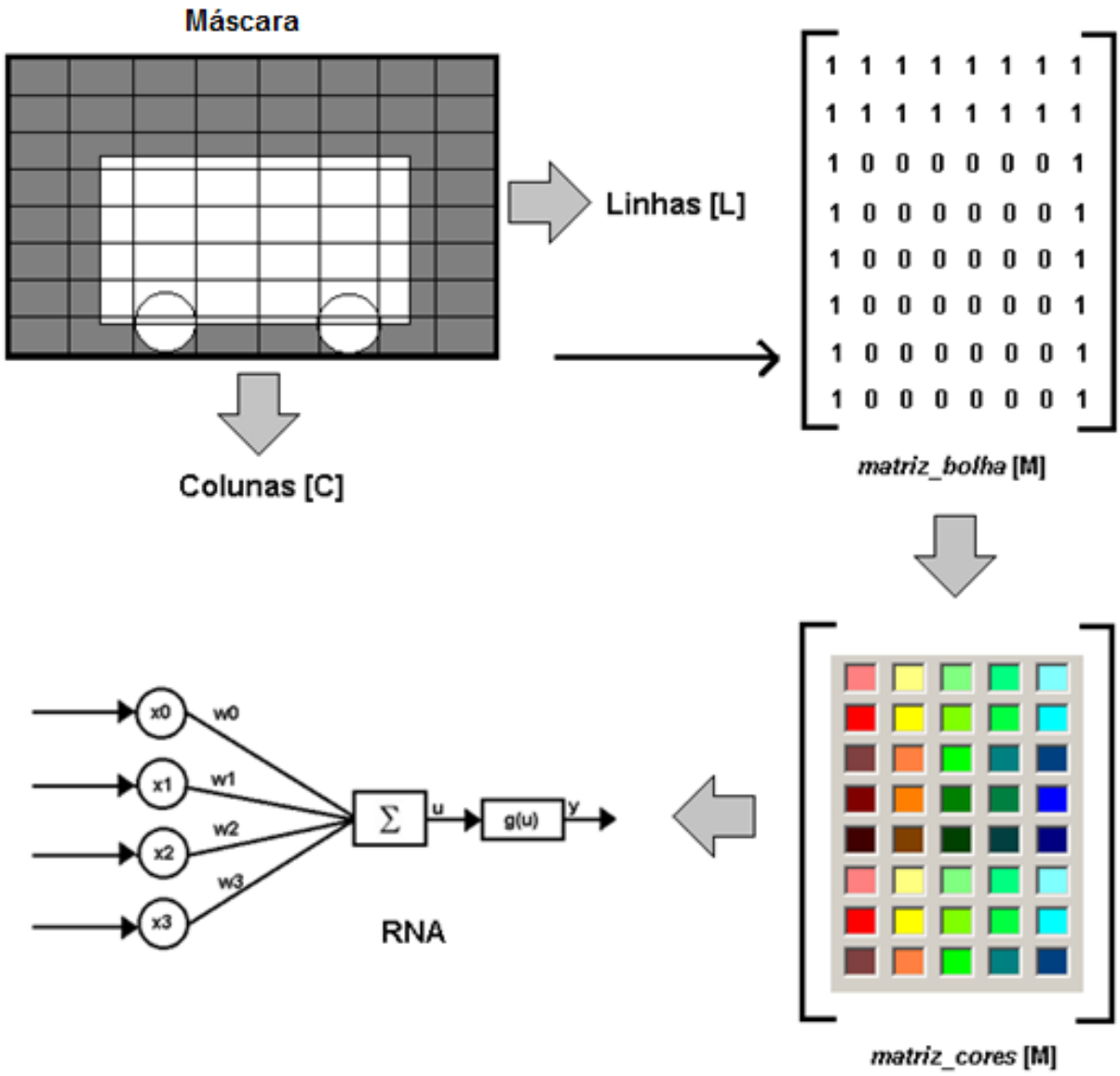


Figura 4.11: Processo de extração e representação de cores

Na figura 4.11 é apresentada a máscara que foi recortada da imagem real por meio do algoritmo de detecção de região de interesse. Por meio desta máscara deve-se endereçar os *pixels* que serão analisados pelo algoritmo de análise de cores na imagem colorida e quantizada.

4.1.2.3. Algoritmo para o cálculo de tamanho da área do obstáculo

Depois de detectada a região de interesse na imagem e feita a extração de cores, o algoritmo para cálculo da área do obstáculo deve ser ativado. O algoritmo para cálculo da área de interesse deve trabalhar em função da quantidade de *pixels* que representam a região de interesse. O cálculo é feito por meio da quantidade de *pixels* horizontais e verticais que representam a máscara.

A quantidade de *pixels* que existem nas regiões que representam os diferentes tipos de obstáculos em diferentes ângulos de visão foram apresentadas como padrão para a fase de treinamento da RNA. O objetivo foi aumentar o nível de aprendizado da rede com diversos padrões de tamanho para o mesmo tipo de obstáculo.

Na figura 4.12 pode ser observada a região detectada como obstáculo sendo utilizada para a contagem de *pixels* na horizontal (H) e vertical (V). A quantidade de *pixels* na horizontal e na vertical devem gerar um valor que será utilizado para a análise de tamanho da imagem. A máscara utilizada para a contagem de pixels foi a mesma utilizada para a extração de cores.

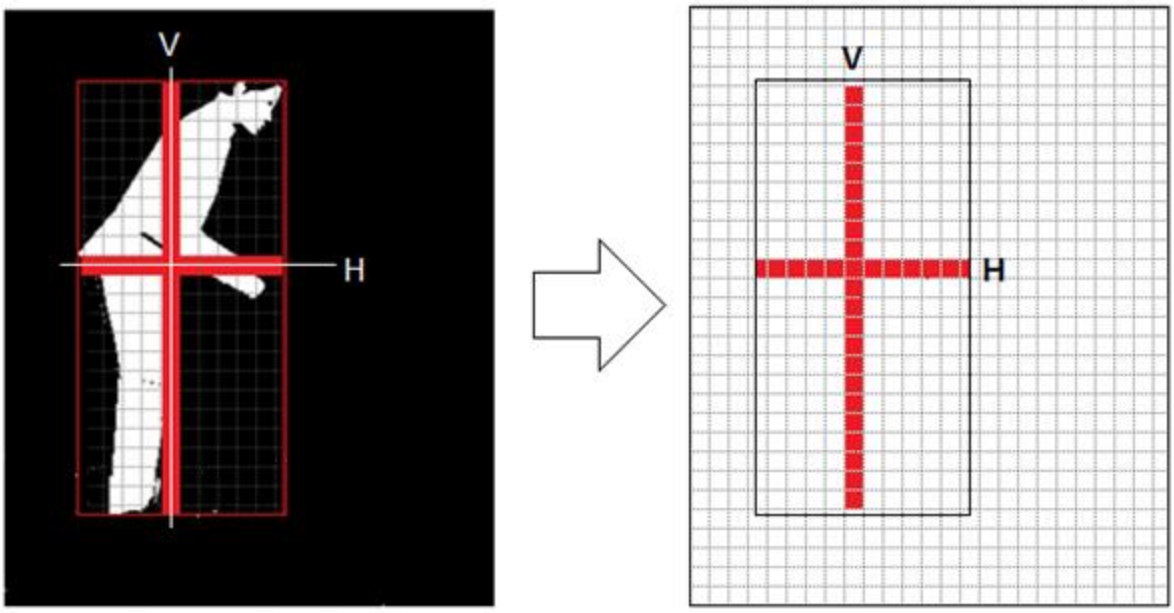


Figura 4.12: Contagem de *pixel* na horizontal (H) e vertical (V)

A quantidade de *pixels* de imagens que representam o mesmo obstáculo pode variar pela distância e pelo ângulo de captura da imagem pela câmera. Mesmo tendo a distância pré-definida para capturar a imagem, variações devem acontecer e uma tolerância a erros foi definida no treinamento da RNA por meio dos padrões apresentados. A contagem de *pixels* na horizontal e vertical deve declarar o tamanho da máscara, consequentemente podendo ser feita uma estimativa do tamanho do obstáculo detectado. O valor da contagem de *pixels* foi utilizado como um dado textual aplicado na entrada da RNA.

Outra análise feita sobre a máscara é a contagem de pixels pretos e brancos. Esta variável ajudou à RNA a reconhecer um obstáculo. O trecho de código conforme apresentado na janela a seguir deve fazer a contagem de *pixels* em toda a região da máscara que representa o obstáculo. A contagem é dada em número de *pixels* brancos e pretos. Para esta contagem também foram utilizadas várias imagens representando o mesmo obstáculo com diferentes ângulos de visão em prol ao treinamento da RNA.

```
Image = imread('C:\empilhadeira.bmp');

// faz a contagem de pixels

//contagem de pixels brancos e pretos

conta_pixel_branco = 0;
conta_pixel_preto = 0;

for j=1:(xmax)-1
   for i=1:(ymax)-1

         if ponto(i,j)==0
             conta_ pixel_branco = branco_pixel+1;
        else
             conta_pixel_preto = preto_pixel+1;
        end
   end
end
```

O tratamento destes dados e a tolerância a erros na medição se aplicará à função da RNA em reconhecer ou não este obstáculo em vistas a essas condições e vulnerabilidade a erros de medida, sendo este processo altamente dependente de um bom treinamento da rede.

4.2. Sistema de VC implementado no simulador V-REP

As seções anteriores tiveram como objetivo apresentar os algoritmos de PDI e VC modelados e simulados no ambiente *Scilab.* Para a implementação dos algoritmos em linguagem LUA foram utilizados estes mesmos algoritmos apresentados nas seções anteriores.

A modelagem e simulação foram feitas com vistas para a análise de comportamento de cada algoritmo e consequentemente a validação dos mesmos.

Nesta seção será apresentando o sistema de VC implementado efetivamente no controlador do robô utilizando linguagem LUA.

O motivo da implementação do sistema de VC em linguagem LUA está ligado as vantagens que esta linguagem apresenta no sensoriamento. Uma das vantagens está ligada a simplicidade do tratamento dos sensores no simulador V-REP utilizando esta linguagem, o motivo é que todos os objetos do ambiente são programados nesta linguagem.

Outro motivo potencial para o uso da linguagem LUA está ligado ao custo computacional. Os sensores estão programados nesta linguagem internamente ao simulador V-REP, e para utilizar outra linguagem seria necessária a utilização da comunicação cliente/servidor.

A comunicação cliente servidor já existe para o controlador neurogenético, mas para tratar todo o sensoriamento externamente ao simulador apresentaria um grande custo de transmissão de dados, já que tanto o sensoriamento de visão como também o sensoriamento ultrassônico, acontecem constantemente, trabalhando com uma grande carga de dados.

Na figura 4.13 é apresentado alguns sensores disponíveis no simulador e o código em LUA que corresponde ao acionamento da câmera *blob detection* em função do "mundo" em que está sendo utilizada (COPELIA ROBOTICS, 2015).

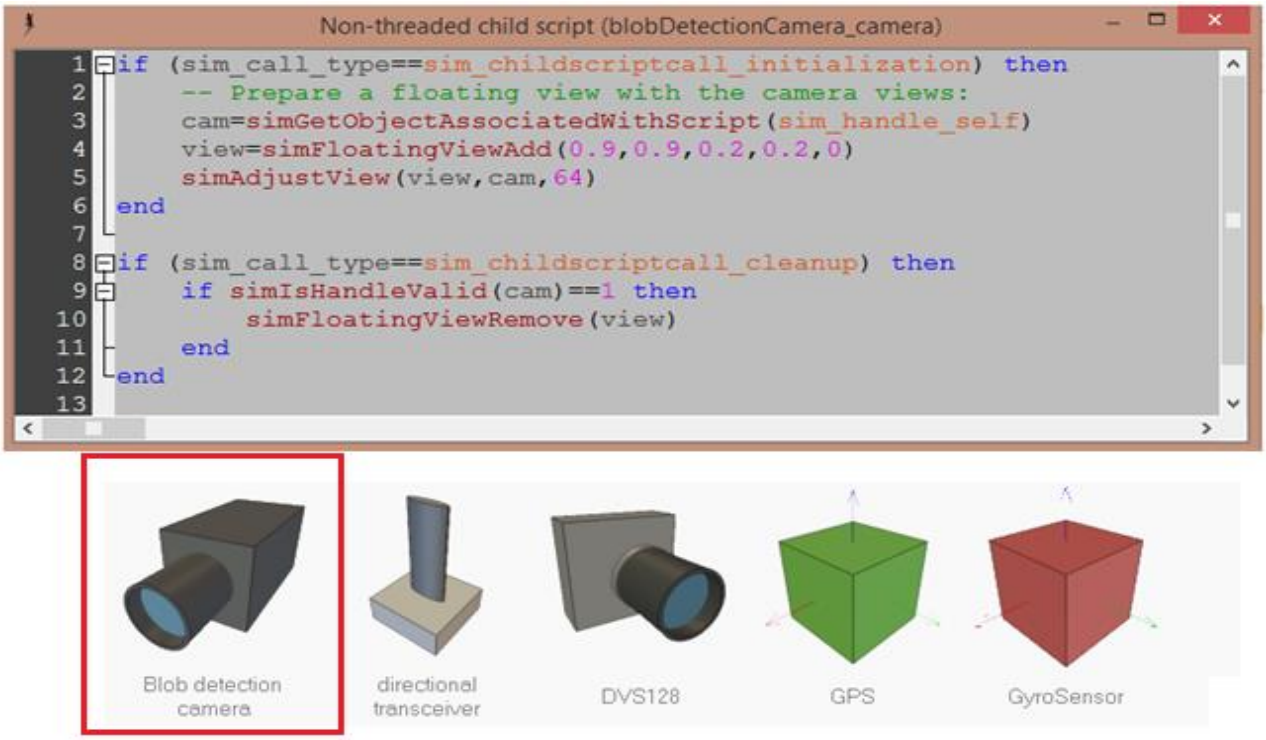


Figura 4.13: sensores disponíveis no simulador e o código correspondente a leitura da câmera *blob detection*

Tabela 4.1: funções de visão para a câmera *blob detection* (COPELIA ROBOTICS,2015)

simGetVisionSensorImage *remote API* equivalente:*simxGetVisionSensorImage* ROS API equivalente:*simRosGetVisionSensorImage*	
Descrição	Recupera a imagem RGB (ou uma parte dela) de um sensor de visão. Use *simGetVisionSensorResolution* para conhecer a resolução da imagem completa.
C sinopse	*simFloat* simGetVisionSensorImage(simIntsensorHandle)*
Parâmetros em C	*sensorHandle: handle* de visão do sensor
Valor de retorno	*Buffer* de Imagem (tamanho do *buffer* é a resolução *X*resolution*3)* ou *NULL* em caso de erro. O usuário é responsável por liberar o *buffer* retornando com a função *simReleaseBuffe*r. Valores de retorno estão dentro da faixa de 0 a 1 (0 = min. Intensidade, 1 = máx. Intensidade)
Parâmetros em LUA	*sensorHandle: handle* de visão do sensor; *posX / posY:* posição da imagem que se deseja recuperar; *sizeX / sizeY:* tamanho da região da imagem que se deseja recuperar. Zero é o definido como padrão inicial, o que significa que a imagem completa deve ser recuperada; *returnType:* tipo de *buffer* devolvido. Retorna uma tabela preenchida com valores RGB na faixa de 0 a 1. A saída deve ser uma *string* preenchida com valores RGB no intervalo 0 a 255
Retorno de valores em LUA	*imageBuffer:* retorna NULL no caso de um erro. Caso contrário, uma tabela contendo os valores RGB (o tamanho da tabela é sizeX * sizeY * 3, os valores RGB na faixa de 0 a 1) ou uma *string* contendo valores RGB (tamanho da tabela é sizeX * sizeY * 3, os valores RGB na faixa de 0 a 255)

Todos os sensores são programados em linguagem LUA (figura 4.13) internamente ao simulador e, para a chamada de um sensor no controlador externo é necessário realizar a chamada do mesmo a cada vez que for utilizado. Na figura 4.14 é apresentada a função de ativação da leitura da câmera *blob detection.*

Figura 4.14: Função de ativação da leitura da câmera *blob detection*

Na tabela 4.1 pode-se observar as funções de visão para a câmera *blob detection* (COPELIA ROBOTICS, 2015). O objetivo da apresentação desta tabela é mostrar, de maneira simplificada as características de acionamento, captura e tratamento de imagens utilizando esta câmera.

4.2.1. Algoritmo de Visão Computacional (VC) implementado em linguagem LUA com vistas para o sensoriamento de obstáculos

O algoritmo para a VC foi explicado detalhadamente nas seções anteriores. Por meio do ambiente *Scilab,* a modelagem e a simulação foram feitas para auxiliar a análise e a apresentação das funções de cada etapa do algoritmo de VC.

Com esta seção, objetiva-se mostrar o funcionamento do algoritmo de VC em linguagem LUA utilizando as funcionalidades e ferramentas disponíveis no simulador V-REP e que foram utilizadas neste trabalho.

A principal ferramenta utilizada neste ambiente para à VC foi a câmera *blob detection,* sendo esta utilizada como elemento de captura dos sinais de entrada para todo o sistema de visão do robô.

Uma explicação mais detalhada deste sensor de imagem já está feita no capítulo 3. Nesta seção, o funcionamento da câmera será explicado em função dos algoritmos que foram aplicados a esse tipo de sensor de imagem.

O algoritmo em LUA e as ferramentas para VC disponíveis no simulador V-REP serão descritos a seguir com o auxílio do diagrama de blocos da figura 4.15.

A primeira etapa do sistema de VC está ligada à detecção de um obstáculo pelo sensoriamento ultrassônico. A câmera só é ativada quando o sensor ultrassônico detecta um obstáculo na rota do robô. Neste momento a função *simGetVisionSensorImage* (tabela 4.1) deve acionar a leitura da câmera para capturar a imagem e enviá-la para o PDI.

A câmera *blob detection* deve capturar uma imagem em sua resolução máxima e enviar para o próximo passo que deve quantizar as cores. A quantização de cores deve ser feita para a redução de custo computacional da RNA *perceptron.*

O próximo passo é fazer a "binarização" da imagem com suas cores quantizadas. A binarização deve segmentar a imagem e gerar a "região bolha" que representa o obstáculo detectado na rota do robô (COPELIA ROBOTICS, 2015).

Na figura 4.15 pode ser observado o robô *NAO* sendo detectado pelo sistema de visão e sua imagem "binarizada", gerando a região na cor vermelha, que corresponde a "região bolha". A "região bolha" deve representar o obstáculo detectado.

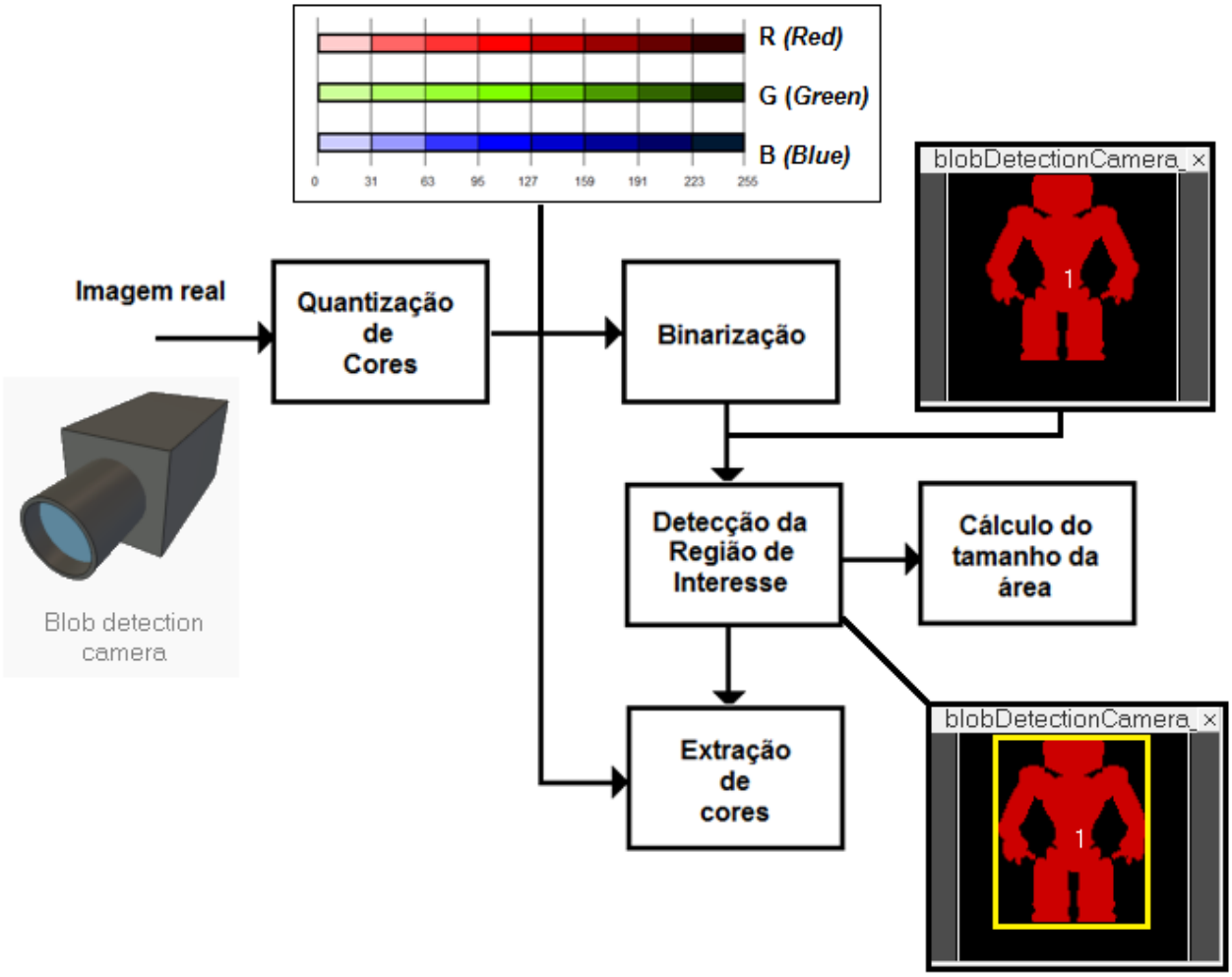


Figura 4.15: Diagrama de blocos do sistema de VC implementado em linguagem LUA

O terceiro passo está ligado à definição da máscara. Este processo consiste em deixar evidenciada a região onde os *pixels* representam o possível obstáculo. Para esta definição é avaliada a "região bolha" e seus pixels que se diferenciam do fundo da

imagem. Na figura 4.15 esta região está contornando na cor amarela o obstáculo detectado pela "região bolha".

O quarto passo está ligado à extração de características da imagem. Todos os passos anteriores serviram como auxílio a esta etapa. O algoritmo de extração de características é dividido em duas partes, sendo elas: o algoritmo de extração de cores e o algoritmo de cálculo de tamanho.

O algoritmo de extração de cores deve utilizar a máscara binária de região de interesse e "recortar" da imagem colorida a região que representa o obstáculo com as cores já quantizadas e enviar as cores dessa região para a RNA.

A rede deve classificar as cores com base em seus padrões de treinamento. Com este método a RNA deve declarar por meio do seu treinamento e seus padrões se este obstáculo tem uma cor reconhecida ou não.

O algoritmo de cálculo de tamanho deve utilizar a máscara da região de interesse criada no terceiro passo para calcular o tamanho do obstáculo. O tamanho do obstáculo é analisado pela contagem de *pixels* que representam essa região.

A contagem de *pixels* para um mesmo tipo de obstáculo deve variar por conta do ângulo de captura da imagem. Porém a distância da câmera em relação ao obstáculo no momento da captura da imagem deve permanecer constante. Por este motivo, a variação de contagem de *pixels* é controlada por uma margem de erro que foi apresentada ao treinamento da RNA.

Toda a variação deve ser tratada pelo treinamento e reconhecimento da RNA que deve declarar o tamanho do obstáculo reconhecido ou não. Se a cor e o tamanho forem reconhecidos pela rede o obstáculo é declarado reconhecido.

4.2.2. Análise comportamental do sistema de VC implementado em linguagem LUA

Nesta seção serão apresentados os testes feitos com o sistema de VC. Nesta seção o sistema de PDI e VC serão chamados apenas por VC. O sistema de visão utilizado mostrou-se eficiente para a análise das variáveis de entrada para a RNA.

Os algoritmos implementados no ambiente *Scilab* e apresentados nas seções anteriores tiveram como objetivo a validação e análise comportamental de cada algoritmo utilizado no sistema de visão do robô. Com a modelagem e simulação de cada algoritmo de maneira independente foi possível validar cada um dos métodos que integram o sistema de VC.

O sistema de VC foi implementado efetivamente na linguagem LUA internamente ao simulador V-REP. Os motivos para a escolha desta linguagem e as consequentes vantagens apresentadas foram descritas na seção anterior.

4.2.3. Simulações do sensoriamento por câmera para a análise comportamental do sistema de VC no ambiente V-REP

Nesta seção, serão apresentados os testes comportamentais do sistema de visão quando encontra obstáculos no caminho do robô. A câmera *blob detection* tem como objetivo a captura e envio da imagem para o sistema de VC para que seja feito o processamento e extração das características para o envio de dados de forma textual para as entradas da RNA.

Na figura 4.16, observa-se a fase de detecção de obstáculo por meio do sensoriamento de visão.

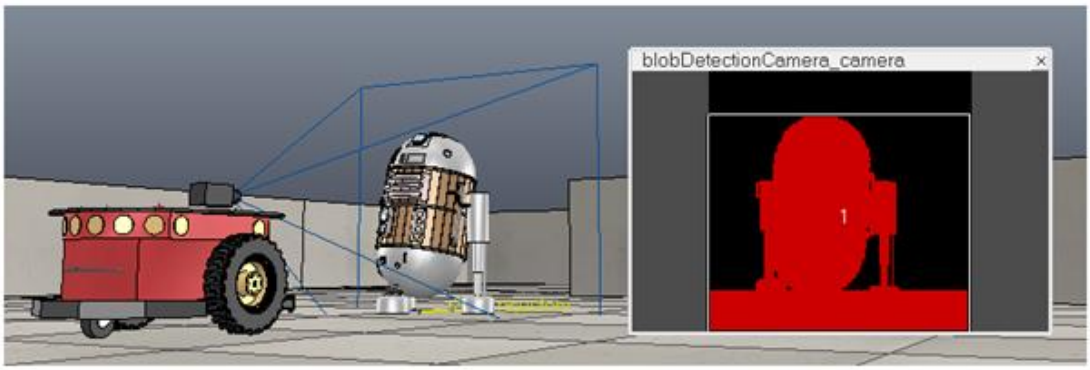


Figura 4.16: Obstáculo robô R2-D2 sendo detectado

O robô R2-D2 (figura 4.16), está como um obstáculo e, por isso, é detectado pelo sistema de VC. O obstáculo em questão tem suas características de cor e tamanho próprias que o diferencia de qualquer outro obstáculo.

Estas características serão extraídas pelo sistema de VC e enviadas textualmente para a RNA, que deve declarar este obstáculo reconhecido ou não. Neste caso este obstáculo não deve ser reconhecido, pois o robô R2-D2 não é um obstáculo problemático que foi cadastrado no banco de conhecimento da RNA.

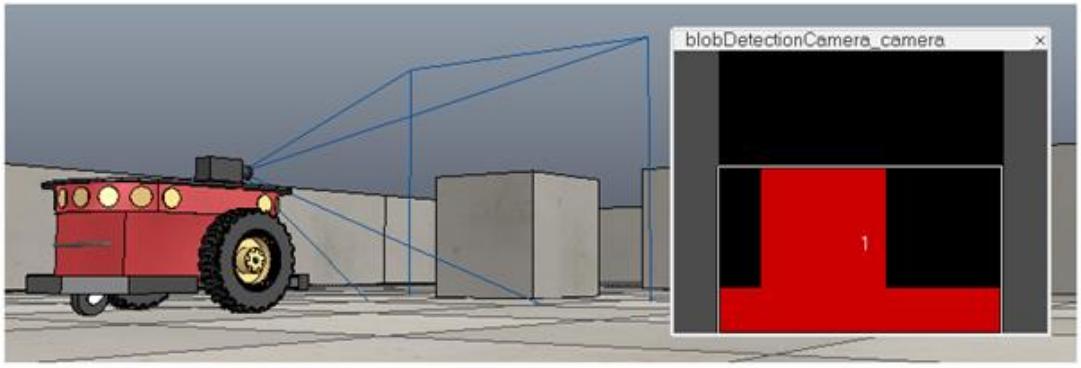


Figura 4.17: obstáculo caixa sendo detectado

Conforme pode ser observado na figura 4.17, o sensoriamento detecta a caixa como um obstáculo, cujas características deste obstáculo devem ser extraídas da mesma forma em que foi feito para um outro obstáculo não reconhecido pela RNA, pelo fato de que o sistema de VC não tem potencial para distinguir um obstáculo desconhecido de um conhecido. Toda esta tarefa deve ser aplicada a RNA.

Em resumo, o sistema de VC utilizado neste trabalho teve como único objetivo extrair as variáveis de cor e tamanho para que servissem de entrada para a RNA.

Para o teste da VC foram utilizados dois tipos de obstáculos, diferenciados por tamanho e formato. Um deles tem o formato de "T" e o outro tem o formato de "I".

Esta segunda fase de processamento é utilizada como exemplo no simulador V-REP e foi usada como base na programação da visão computacional do robô. Na figura 4.18 pode-se observar alguns objetos capturados pelo sistema de visão.

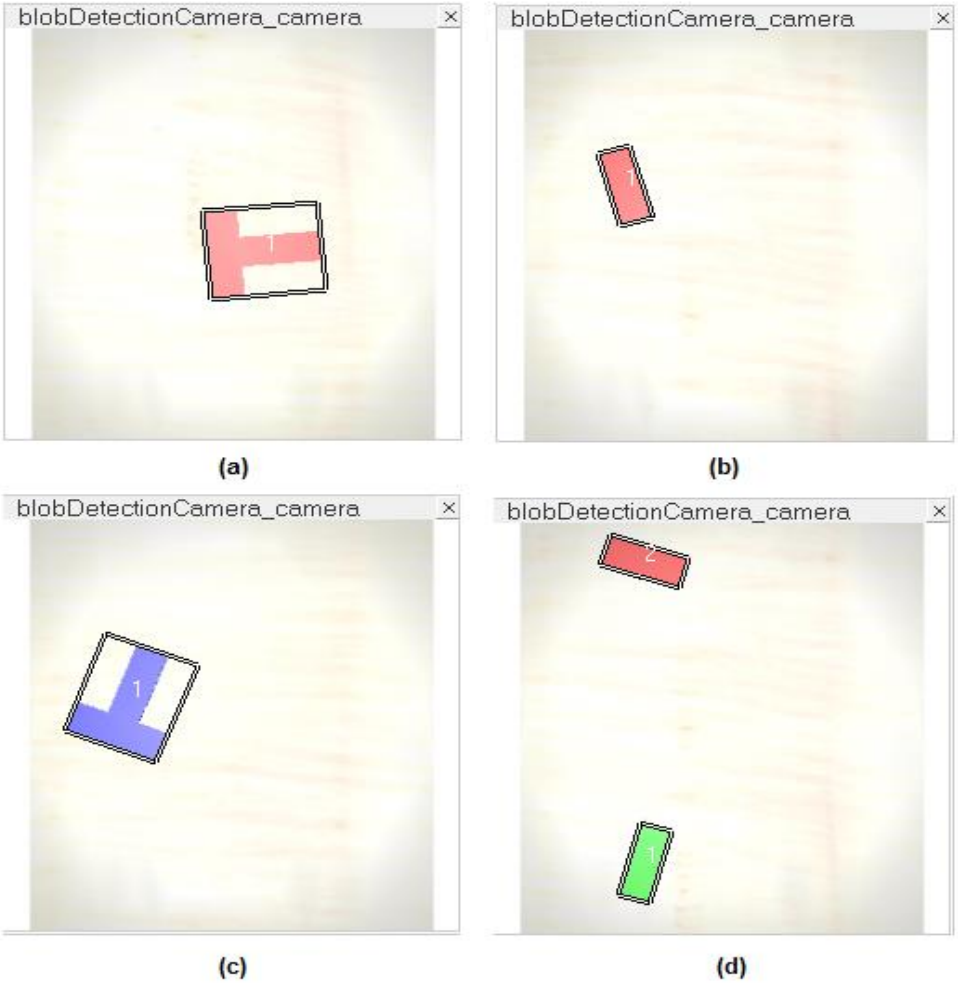


Figura 4.18: Testes feitos para dois tipos de objetos diferentes; (a) detecção do objeto em formato de "T", (b) detecção do objeto em formato de "I", (c) detecção de um objeto em formato de "T" na cor azul e (d) dois objetos em formato de "I" com cores diferentes

Outra variação nestes objetos foi feita em relação a suas cores. Foi aplicada a variação de três cores, sendo elas: vermelho, verde e azul. A fase de VC deve extrair as cores e o tamanho dos objetos. O objetivo da extração destas variáveis foi melhor descrito no capítulo 3 e está ligado ao reconhecimento dos obstáculos presentes na rota do robô.

Algumas imagens (figura 4.18), foram obtidas pelo sensoriamento da câmera *blob detection.* Nestas imagens, é possível constatar a segunda fase de processamento, que foi utilizado para o reconhecimento de padrões dos obstáculos detectados na rota do robô.

Na figura 4.18(a) pode ser observado o objeto em formato de "T", este objeto pode ser diferenciado do objeto em formato de "I" (figura 4.18 - b) por suas "abas" que caracterizam um objeto em formato de "T" e não estão presentes obviamente em um objeto em formato de "I".

A consequência nesta alteração de formato é uma quantidade de *pixels* maior para o obstáculo de formato de "T". Obviamente o obstáculo de formato de "T" tem mais *pixels* que o objeto de formato de "I". A RNA deve trabalhar nesta lógica para reconhecer diferentes obstáculos neste ambiente. Lembrando que os padrões de cada obstáculo foram apresentados para a RNA na fase de treinamento.

O objeto em formato de "I", como na figura 4.18 (b), foi detectado pelo sensoriamento e diferenciado do objeto em forma de "T" pela ausência das abas laterais.

Nas figuras 4.18 (c e d) podem ser observados os objetos em formado de "T" ou "I" sendo diferenciados pela mudança de tamanho e cores em RGB. A RNA recebe em suas entradas os valores das cores em três entradas, sendo elas: R, G e B (COPELIA ROBOTICS, 2015).

Este sistema de visão foi implementado com vistas para o reconhecimento de obstáculos de maior problemática nos ambientes dinâmicos em que o robô foi utilizado. A lógica apresentada para o reconhecimento destes dois tipos de obstáculos foi utilizada para todos os outros obstáculos presentes nos ambientes modelados para simulações do controlador neurogenético. Um fator de extrema importância que foi observado neste trabalho está ligado a necessidade de um bom treinamento da RNA.

O custo máximo de tempo para este sistema de visão ficou em 6 milissegundos. O motivo de um custo tão baixo está ligado a câmera utilizada, que possui um pré-processamento implementado em *hardware* que gera a imagem "binarizada" e sua "região bolha" (esta implementação em *hardware* é simulada e nativa do simulador de robôs V-REP e foi modelada em linguagem LUA). Outro motivo relacionado ao baixo custo de tempo está ligado a preocupação que se teve no desenvolvimento de um sistema de visão o mais simples possível. Os algoritmos de extração de características também foram implementados em linguagem LUA, junto ao sistema de percepção ultrassônico do robô, assim, reduzindo o custo de processamento e tempo do sistema de visão. Uma explicação detalhada deste sistema de visão foi feita nas seções anteriores.

Capítulo 5

Análise e levantamento de dados em ambientes reais

Para as simulações do algoritmo neurogenético, foram analisados dois ambientes industriais para servirem como base para a modelagem e simulação do ambiente dinâmico desenvolvido neste trabalho, sendo um deles um ambiente industrial didático, e o outro um ambiente industrial com sua produção de peças em pleno funcionamento.

Os ambientes analisados são considerados "chão de fábrica" assim como o ambiente modelado para as simulações e validação do controlador.

O maior objetivo da análise destes ambientes reais foi trazer a problemática do mundo real para o ambiente de simulações robóticas virtual. Com isso, foi possível fazer algumas estimativas sobre as vantagens do algoritmo em variadas situações nestes ambientes com base em dados coletados e aplicados ao ambiente modelado para simulações e validação do controlador.

Os ambientes analisados não possuem sistema robótico, e para os fins do robô utilizado neste trabalho, foi mostrado grande interesse neste tipo de automatização de processos, tanto para fins didáticos e de redução de riscos no ambiente 1 ["chão de fábrica" didático], tanto para fins de aumento de produtividade e redução de riscos no ambiente 2 ["chão de fábrica" de uma linha de produção real].

Foram analisadas as características relacionadas às estruturas dos ambientes que estão ligadas à navegação do robô nestes locais. Foram analisadas criticamente as seguintes características: (a) tipo de obstáculos de maior frequência, (b) mudança de objetos utilizados no ambiente (uniformes, caixas, empilhadeira), (c) possibilidade de um novo obstáculo entrar no ambiente e a (d) disposição das máquinas.

Uma tabela foi gerada para mostrar a dinâmica dos obstáculos nestes ambientes, baseando-se no comportamento real dos obstáculos e configuração da estrutura do ambiente. Esta tabela contém a estimativa de interferência dos obstáculos de maior frequência sobre a rota do robô móvel. A interferência gerada por um obstáculo na rota do robô está ligada ao grau de movimentação e a taxa de ocupação de cada obstáculo.

Falando de uma maneira clara e objetiva, o algoritmo neurogenético tem vantagens quando a RNA está bem treinada e os obstáculos conhecidos são os de maior frequência.

Para um bom treinamento da RNA, deve-se caracterizar o ambiente em que o robô deve realizar suas funções e consequentemente gerar uma base de dados para o treinamento da RNA. Um novo treinamento da RNA deve diminuir a deficiência apresentada para reconhecer novos obstáculos frequentes, seja em momentos em que esses obstáculos

são inseridos após o último treinamento da RNA ou quando o robô é destinado a realizar seu trabalho em um novo ambiente.

Quando se deseja trocar o robô de ambiente com grande frequência e o treinamento da RNA é inviável a cada troca, o AG "puro" deve apresentar um melhor desempenho, pois, no pior caso, nenhum obstáculo será conhecido neste novo ambiente, já que os obstáculos cadastrados no banco de conhecimento provavelmente não farão parte deste novo ambiente.

As próximas seções descrevem as variáveis quantitativas para a avaliação do potencial dos obstáculos frequentes e que serviram de base para o treinamento da RNA *perceptron.*

5.1. Grau de movimentação e taxa de ocupação

Nesta seção é feita a descrição das variáveis quantitativas e de influência para o nível de problemática de cada obstáculo. Duas variáveis apresentam maior peso, sendo elas o grau de movimentação de um obstáculo e taxa de ocupação deste obstáculo em relação ao seu tempo de permanência no ambiente. Nas próximas seções são descritas as quantificações destas variáveis.

5.1.1. Grau de movimentação

O grau de movimentação (GM) é dado pela quantidade de vezes que um determinado obstáculo se movimenta em uma unidade de tempo pré-definida. Para este tipo de informação, deve-se observar o ambiente em uma faixa de tempo e contabilizar quantas vezes cada tipo de obstáculo se movimentou neste intervalo de tempo. A equação 5.1 é utilizada para o cálculo desta variável.

$$\mathrm{GM} = \frac{\boldsymbol{MOV}}{\boldsymbol{Tempo}\ (\boldsymbol{H})} \qquad (5.1)$$

Sendo:

GM = Grau de Movimentação

MOV = A quantidade de vezes que um obstáculo se movimenta em uma certa faixa de tempo

Tempo(H) = A unidade de tempo utilizada para relacionar a quantidade de vezes que um obstáculo se moveu por um tempo pré-definido.

Com esta equação é possível calcular e quantificar o grau de mobilidade de cada obstáculo presente no ambiente.

Para o cálculo de *GM,* foi feita a observação do ambiente por um intervalo de tempo de 1 hora, sendo este valor de tempo aplicado a variável *Tempo(H).* Para cada obstáculo com maior taxa de ocupação, foi contabilizado o número de vezes que cada obstáculo se moveu no ambiente dentro do tempo definido em *Tempo(H)*, sendo este valor representado pela variável *MOV.* Quanto maior o valor de *GM* maior o nível de interferência de um obstáculo.

5.1.2. Taxa de ocupação

A taxa de ocupação de um determinado obstáculo é quantificada com o objetivo de identificar o quanto um obstáculo deve estar presente no ambiente de navegação no robô. O valor gerado deve ser um diferencial na escolha de obstáculos a serem cadastrados no banco de conhecimento da RNA.

O cálculo de *GM* só foi feito para os obstáculos que apresentaram maior Taxa de Ocupação no ambiente (TO). Esta variável será descrita na próxima seção.

A taxa de ocupação é dada pelo número de obstáculos do mesmo tipo que estão presentes no ambiente em relação ao seu tempo de permanência no ambiente. Esta taxa não apresenta grande variação para os obstáculos permanentes no ambiente, como empilhadeiras e carro de transporte manual por exemplo. Porém, no caso do humano, existe uma grande variação, que torna esta variável bastante relevante na problemática de obstáculos. O cálculo desta taxa é feito pela equação 5.2.

$$\mathrm{TO} = \frac{\boldsymbol{N.OBST}}{\boldsymbol{Tempo.P\ (H)}} \qquad (5.2)$$

Sendo:

TO = Taxa de ocupação de cada obstáculo

N.OBST = número de obstáculos do mesmo tipo e presentes no mesmo intervalo de tempo em que está sendo avaliado a taxa de ocupação.

Tempo.P(H) = A unidade de tempo utilizada para relacionar a quantidade de obstáculos iguais e sua ocupação no ambiente dinâmico por uma faixa de tempo pré-definida.

A taxa de ocupação de *TO* é calculada em função do número de um determinado tipo de obstáculo de maior frequência *N.OBST* em função do *Tempo.P(H)* deste conjunto de obstáculos no ambiente estudado (equação 5.2).

O controlador não deve reajustar estes dados quando o robô estiver em funcionamento, pois todo esse conhecimento será potencial para o seu treinamento inicial, consequentemente devem ser passados para o controlador neurogenético antes da primeira navegação do robô. Todos os dados devem ser obtidos de maneira empírica e com auxílio da experiência humana dos responsáveis por cada um dos ambientes dinâmicos reais propostos neste trabalho.

A observação do ambiente para que seja feita uma caracterização de todas estas variáveis tem grande custo de tempo. Se o ambiente não tiver nenhum banco de dados fornecidos a *priori* de maneira empírica por um trabalhador que trabalha no local este custo deve aumentar. Neste caso o ambiente deve ser explorado para que seja feita a sua caracterização.

O nível de interferência (*NI)* de cada obstáculo é dado pelo valor de *GM e TO.* Quanto maior o valor de *GM* e de *TO* maior o *NI* de um obstáculo. O *NI* deve ser calculado pela equação 5.3:

$$\mathbf{NI} = (\boldsymbol{TO} * \mathbf{30} + \boldsymbol{GM} * \mathbf{70}) / \mathbf{100} \qquad (5.3)$$

Sendo:

NI = nível de interferência do obstáculo na navegação do robô

TO = Taxa de ocupação

GM = Grau de movimentação

Os pesos dados às variáveis quantitativas *TO e GM* estão ligados ao potencial de interferência na navegação do robô, que é gerado por cada uma delas. O peso de 70% é dado para o *GM* de um obstáculo, pois quanto maior este nível, mais difícil será a solução para este obstáculo. O peso de 30% da *TO* tem um valor baixo se comparado a *GM,* esta diferença de peso está ligada a mobilidade do obstáculo.

Um obstáculo pode ter uma grande *TO* e não ter um *GM* tão alto, isso resulta em um obstáculo menos problemático para o desvio de rota do robô. Uma melhor explicação sobre obstáculos dinâmicos foi feita no capítulo 3.

Com base nestas variáveis quantitativas foram feitas as análises sobre os ambientes descritos nas próximas seções. Um banco de dados sobre cada ambiente dinâmico e seus obstáculos de maior frequência foi criado para a base do treinamento da RNA.

5.2. Ambiente dinâmico 1: Ambiente industrial didático

O ambiente dinâmico 1 é um ambiente industrial didático utilizado pelo Instituto Federal de São Paulo (IFSP), campus de Catanduva para os cursos de tecnologia em mecatrônica e o curso técnico em fabricação mecânica para as disciplinas de Fabricação Mecânica e Controle Numérico Computadorizado.

O laboratório tem como objetivo a fabricação de peças mecânicas para auxiliar no ensino das disciplinas ministradas neste laboratório. Este ambiente dinâmico apresenta todas as características propostas no ambiente modelado para simulações, então, por este motivo, ele serviu com um grande potencial para a análise e coleta de dados a serem utilizadas nas simulações. Na figura 5.1 pode ser feita uma visão geral sobre a construção do ambiente e a disposição das maquinas configuradas para este local de trabalho.

Uma característica interessante neste ambiente são as "trilhas", caminhos em azul contornados de amarelo. Estes caminhos são utilizados para a passagem de pessoas, carros de transporte materiais e empilhadeiras.

Figura 5.1: Ambiente industrial didático (IFSP Catanduva)

Estes caminhos devem garantir a segurança do humano, assim evitando o contato com as maquinas que são utilizadas nas células. De maneira resumida, os caminhos indicam onde é seguro se locomover neste ambiente dinâmico.

O ambiente possui um conjunto de objetos que são utilizados com grande frequência para auxiliar no processo de produção de peças mecânicas. Os objetos utilizados com

maior frequência neste ambiente são apresentados na figura 5.2. Na figura pode ser observado o humano uniformizado, o carro manual de transporte, a empilhadeira e o carro de transporte de cilindros de oxigênio e acetileno.

Na figura 5.2(a) é apresentado outro ponto importante neste ambiente,sendo este a uniformização dos professores e alunos. Os uniformes são utilizados por motivos de segurança, e são do tipo jaleco e calça de proteção. Estes uniformes são confeccionados em uma cor padrão.

Na figura 5.2(b) é apresentado o carro manual de transporte de peças e materiais. Este carro tem grande uso neste local, sendo assim um obstáculo de grande frequência principalmente nas trilhas de segurança.

Na figura 5.2(c) é apresentada a empilhadeira, objeto de grande frequência neste ambiente, utilizado para o carregamento de objetos que o carro manual não tem capacidade de transportar.

Na figura 5.2 (d) é apresentado o carro de transporte de cilindros de oxigênio e acetileno. Esta ferramenta não tem grande movimentação no ambiente se comparado aos outros meios de transporte de materiais e peças usados neste ambiente. Porém possui uma grande quantidade deste objeto neste ambiente.

Figura 5.2: Obstáculos de maior frequência no ambiente dinâmico 1: (a) humano uniformizado ; (b) carro para transporte de materiais e peças; (c) empilhadeira manual e (d) carro de transporte de cilindros de oxigênio e acetileno

Os objetos apresentados na figura 5.2 são os obstáculos com maior frequência neste ambiente. Os obstáculos de maior problemática devem ser cadastrados no banco de conhecimento da RNA, pois estes devem interferir na navegação do robô com grande frequência. O reconhecimento do obstáculo pela RNA e desvio por solução pré-definida no banco de conhecimento deve evitar a chamada do AG que tem um maior custo para obter uma nova solução.

O ambiente manteve esta configuração desde sua construção e está em funcionamento há 5 anos sem nenhuma reconfiguração nos objetos principais encontrados neste local, como: cor, tamanho, inclusão de novos equipamentos. As mudanças que acontecem com frequência estão ligadas à reconfiguração da disposição do maquinário e suas células de trabalho.

Na tabela 5.1 são apresentados os obstáculos com maior potencial de interferência na navegação do robô. Com ela, pode-se ter uma noção de como estes obstáculos de maior frequência devem interferir no ambiente. Todos os testes foram feitos com um intervalo de tempo de 1 hora.

Tabela 5.1: Obstáculos com maior potencial de interferência na navegação do robô - [Ambiente 1]

Obstáculos com maior potencial de interferência na navegação do robô - [Ambiente 1]			
Tipo de obstáculo	**Taxa de ocupação (TO)**	**Grau de movimentação (GM)**	**Nível de interferência (NI)**
Humano	20	60	48
Carro de transporte	8	20	16,4
Empilhadeira manual	3	5	4,4

O nível de interferência é dado de 0 a 100, sendo 100 o nível mais alto de problemática para um obstáculo. Os obstáculos móveis têm um maior custo para o algoritmo neurogenético e por este motivo o *GM* é o que mais influência no nível de interferência de um obstáculo. Uma descrição melhor para obstáculos móveis pode ser vista no capítulo 3.

O obstáculo que apresentou maior interferência foi o humano, com nível de interferência em 48. Já o obstáculo com menor interferência foi a empilhadeira, nível 4,4. O motivo destes extremos de nível de interferência é dado pelos seguintes motivos:

- Os humanos (alunos e professores) durante a produção de uma peça devem estabelecer uma dinâmica neste ambiente para visualizar os trabalhos feitos por colegas de classe, e assim ter um maior aprendizado prático.

- As empilhadeiras manuais são movimentadas neste ambiente apenas quando é necessário o transporte de uma peça de grande peso e que está sendo utilizada como experimento na aula , ou quando for transportada uma caixa com grande volume de peças.

O ambiente apresentou problemática na uniformização dos humanos (alunos e professores), já que foi constatado que em alguns casos, alunos que não participam das aulas práticas não são obrigados a vestir o uniforme de segurança. Com isso podem

estar vestindo qualquer cor de roupa, o que dificulta o reconhecimento do obstáculo humano.

Como conclusão, tem-se que este ambiente é de grande potencial para base das simulações e validação do algoritmo híbrido, pois utiliza com frequência um conjunto de obstáculos. Com isso foi criado uma modelagem no simulador V-REP baseada na configuração deste ambiente. Os testes podem ser vistos no capítulo 7.

5.3. Ambiente dinâmico 2: Ambiente industrial real

O ambiente dinâmico 2, é um ambiente de chão de fábrica da indústria DDS Industrial limitada, que tem suas instalações na cidade de Catanduva-SP. Este ambiente é utilizado para a fabricação de peças mecânicas para hidráulica e pneumática.

Este ambiente dinâmico também apresenta todas as características propostas no ambiente modelado para simulações deste trabalho. Se comparado ao ambiente dinâmico 1, este ambiente apresenta algumas características diferenciadas. Então por estes motivos o ambiente real 2 serviu com grande potencial para a análise e coleta de dados a serem utilizadas nas simulações.

Na figura 5.3 pode ser feita uma visão geral sobre a construção do ambiente e a disposição das maquinas configuradas para este local de trabalho. Neste ambiente também são utilizadas trilhas, mas estas são obrigatórias apenas para o transporte de materiais utilizando uma empilhadeira. Os humanos têm liberdade de locomoção dentro ou fora das trilhas.

Figura 5.3: Ambiente industrial real (indústria DDS)

Este ambiente dinâmico se comparado com o ambiente dinâmico 1, também tem objetos que estão sendo utilizados com grande frequência para auxiliar no processo de produção de peças mecânicas. Na figura 5.4 é apresentado os objetos que são encontrados com maior frequência neste ambiente.

Os obstáculos de maior problemática devem ser cadastrados no banco de conhecimento da RNA, pois estes devem interferir na navegação do robô com grande frequência.

Na figura 5.4(a) é apresentado o carro de transporte de caixas. Este carro utiliza como base para o transporte de carga para *pallets* de madeira. Então para o reconhecimento deste obstáculo seria necessário o reconhecimento deste conjunto de objetos mencionados, formando um único obstáculo (caixas, *pallets e* carro de transporte).

O cadastro de apenas uma caixa como obstáculo também seria de grande importância, pois o robô pode encontrar em sua rota apenas uma caixa, e não um conjunto delas sobre o carro de transporte.

Na figura 5.4(b) é apresentado o carro manual de transporte de ferramentas para manutenção das maquinas. Este carro tem grande uso neste local, sendo assim, um obstáculo de grande frequência. Este carro deve circular entre as células levando as ferramentas necessárias para a manutenção preditiva, preventiva e corretiva.

Na figura 5.4(c) é apresentada a empilhadeira, objeto de frequência moderada neste ambiente, utilizado para o carregamento de materiais que o carro manual para transporte de caixas não tem capacidade de transportar.

Na figura 5.4(d) é apresentado o modelo de caixa utilizado para embalar o produto final. Este tipo de obstáculo só deve ser encontrado no setor de produto final, sendo, portanto, diferente dos outros objetos que podem ser encontrados em qualquer ponto do chão de fábrica.

Este objeto é cadastrado de maneira unitária na RNA, quando várias unidades destas caixas estiverem reunidas, se tornarão um obstáculo desconhecido. Uma melhor descrição deste problema pode ser vista no capítulo 3.

Figura 5.4: Obstáculos de maior frequência no ambiente dinâmico 1: (a) carro de transporte de caixas; (b) carro de transporte de ferramentas; (c) empilhadeira manual e (d) caixas para produto final

Diferente do ambiente dinâmico 1, este ambiente está em constante reconfiguração para otimizar a linha de produção. Quando um produto novo é projetado, as maquinas devem ser reconfiguradas para conseguir atender a demanda. Possivelmente será necessária uma nova caracterização do ambiente para o treinamento da RNA.

Um novo tipo de objeto é inserido neste ambiente de acordo com as necessidades do plano de produção. Na maioria dos casos, são inseridos no ambiente novas maquinas, novo tipo de caixa para produto final, alteração dos uniformes de acordo com o clima e inserção de novos carros para transporte de materiais e peças.

Na tabela 5.2 são apresentados os obstáculos com maior potencial de interferência na navegação do robô.Com ela pode-se ter uma noção de como estes obstáculos de maior frequência devem interferir na navegação do robô móvel. Todos os testes foram feitos com um intervalo de tempo de 1 hora.

Tabela 5.2: Obstáculos com maior potencial de interferência na navegação do robô - [Ambiente 2]

Obstáculos com maior potencial de interferência na navegação do robô - [Ambiente 2]			
Tipo de obstáculo	**Taxa de ocupação (TO)**	**Grau de movimentação (GM)**	**Nível de interferência (NI)**
Humanos	40	20	26
Carro de transporte	5	12	9,9
Caixas de transporte	80	15	34,5

Assim como no ambiente dinâmico 1, o nível de interferência é dado de 0 a 100, sendo 100 o nível mais alto de problemática para um obstáculo. Já que os obstáculos móveis tem um maior custo para o algoritmo neurogenético.

O obstáculo que apresentou maior nível de movimentação foram as caixas de transporte, com nível 34,5. Já os obstáculos com menor nível de interferência foram os carros manuais de transporte, com o nível de 9,9. O motivo destes extremos de nível de interferência é dado pelos seguintes motivos:

- Os carros de transporte manual tiveram baixa taxa de interferência no ambiente pois sua *TO* é muito baixa, consequência do fato de existirem apenas 5 carros no ambiente. Outro fator é o *GM* muito baixo, menor ainda que do humano. O motivo é que às vezes as caixas estão pouco carregadas e o humano que as carrega não usa o carro de transporte para levá-las.

- As caixas de transporte são movidas com grande frequência. Seu *GM* é alto pelo fato de que elas devem servir de base para o carregamento das peças entre as células de produção, já que as peças são bem pequenas. O humano é sempre o mesmo para carregar estas caixas. A *TO* também é alta, pois existem 80 caixas desta no ambiente. O motivo de tanta caixa é que elas servem também como armazenamento primário.

O *NI* dos obstáculos neste ambiente foi estimado pela *TO* do mesmo tipo de obstáculo e seu *GM* apresentado em um intervalo de tempo de 1 hora.

O ambiente apresentou problemática na uniformização dos humanos, já que foi constatado que em alguns casos outras pessoas frequentam este ambiente, sendo estas não participantes do setor de produção e sim de outras áreas como engenharia e gestão. Neste caso, a RNA não reconhece o obstáculo e a solução deve ser gerada pelo AG.

O ambiente também apresentou grande problemática na constante reconfiguração do chão de fábrica. Consequentemente, é necessário que a RNA seja treinada novamente.

5.4. Conclusões sobre os ambientes dinâmicos reais

Como conclusão, tem-se que estes ambientes são de grande potencial para as simulações e validação do algoritmo híbrido, pois utilizam com frequência um conjunto de obstáculos pré-definidos e apresentam reconfiguração constante. Estes fatores são potenciais para o uso do algoritmo neurogenético.

Se comparado com o ambiente dinâmico 1, o ambiente dinâmico 2 apresenta um nível de dificuldade maior, pois utiliza objetos com uma variação mais elevada e menos controlada. Algumas alterações no ambiente de simulação modelado no V-REP foram feitas para analisar cada um destes ambientes.

Algumas das alterações foram: mudança nas cores dos uniformes dos humanos, mudança nas cores das caixas e inclusão de objetos nunca cadastrados no banco de conhecimento da RNA.

Os resultados apresentados sobre a avaliação destes ambientes dinâmicos foram positivos para este trabalho, pois a dinâmica de obstáculos em um ambiente pré-definido foi constatada e serviu de base para as simulações e validação do algoritmo neurogenético.

O interesse do professor consultado para avaliação do ambiente dinâmico 1 ficou ligado ao transporte de peças a serem utilizadas nas maquinas em cada célula e também para o ensino da robótica móvel nas disciplinas que utilizam este laboratório.

Já no ambiente dinâmico 2, o interesse do gerente industrial está ligado ao aumento do nível de produção e segurança dos funcionários. A proposta do gerente industrial seria utilizar o robô entre o ciclo de produção, terminando sua tarefa ao levar as peças finalizadas ao setor de estoque de produto final. Os testes podem ser vistos no capítulo 6 e 7.

Capítulo 6

Metodologia aplicada na implementação do controlador híbrido

Neste capítulo é apresentado os métodos e ferramentas utilizados para a implementação do controlador neurogenético, assim como os testes e resultados obtidos por meio de simulações feitas no ambiente modelado. As simulações aqui apresentadas tiveram como objetivo a validação do ambiente dinâmico modelado. Na figura 6.1 pode ser observado o ambiente dinâmico modelado para os testes.

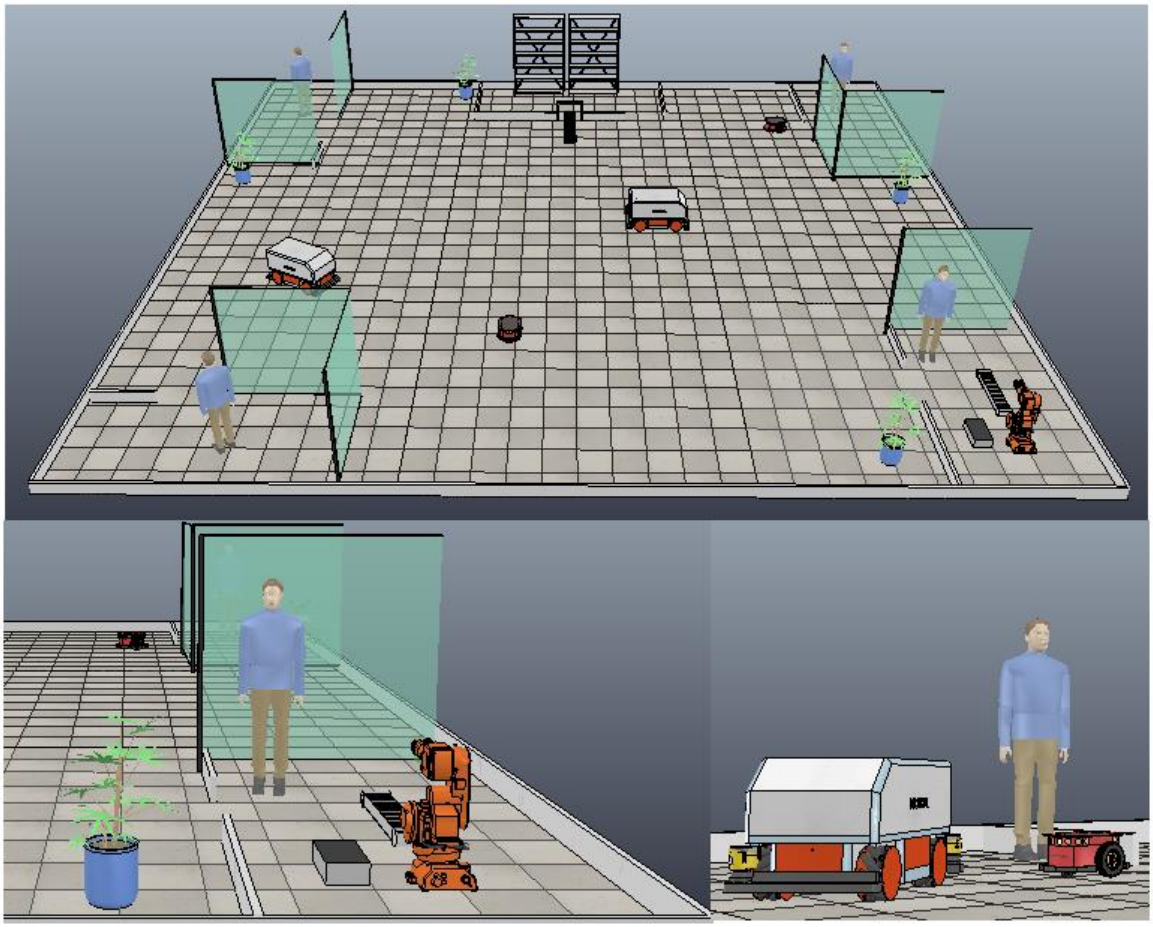


Figura 6.1: Ambiente final modelado no simulador de robôs V-REP

Para a validação foram feitas simulações do ambiente utilizando o robô *Pioneer* com um algoritmo simples de controle. O algoritmo utilizado para este fim utiliza lógica simples, por exemplo, quando um obstáculo é detectado, o robô faz um desvio escolhendo uma posição aleatória.

A validação do ambiente foi feita pelo conjunto de testes e simulações descritos nas próximas seções. O ambiente modelado teve sua validação baseada nas simulações de comunicação cliente/servidor, sensoriamento, e disposição das células. Estes itens são fundamentais para o funcionamento do controlador neurogenético neste ambiente.

Como descrito no capítulo 3, este ambiente não é mapeado, ou seja, o robô é colocado no ambiente sem conhecimento algum sobre sua disposição. No caso de usar o robô sempre neste mesmo ambiente, seria viável utilizar um mapa. Porém, um dos objetivos deste trabalho é que o robô tenha flexibilidade em relação a troca de ambientes, ou seja, que tenha capacidade de navegar em diversos tipos de ambientes dinâmicos.

6.1. Flexibilidade na reconfiguração do ambiente modelado para simulações

Para uma troca de ambiente não é necessário nenhum reajuste físico no robô para que ele consiga navegar de maneira segura, porém, para um melhor desempenho do controlador, é necessário o treinamento da RNA novamente. Este treinamento é necessário para que a RNA reaprenda os obstáculos de maior frequência neste tipo de ambiente.

Uma troca de objetos no ambiente atual já exige um novo treinamento da RNA. Como exemplo, a troca de uniformes dos humanos da fábrica por uma nova cor, ou uma nova compra de diferentes tipos de caixas, com diferentes tamanhos e/ou cores.

Na figura 6.2 é apresentado um caso onde a cor do uniforme do humano é trocada. Para este caso a RNA deve ser treinada novamente. Caso contrário, o humano não será mais um obstáculo conhecido pela RNA, e consequentemente não terá na base de dados uma solução para desvio.

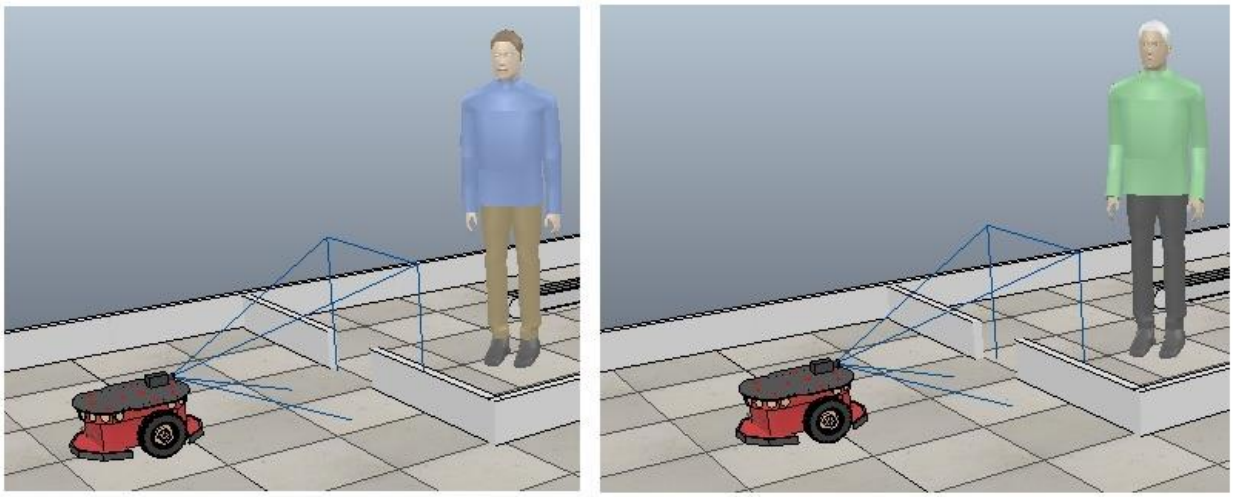


Figura 6.2: (a,b) Mudança de cores no uniforme do humano

Como a RNA trabalha com as variáveis de entrada cor e tamanho dos objetos que impedem a rota planejada, inicialmente será necessário um reaprendizado da rede. Caso contrário o algoritmo híbrido perde todo seu potencial apresentado neste trabalho. Pois os obstáculos de maior frequência não serão reconhecidos e será necessária uma nova solução pelo AG infinitas vezes para os mesmos tipos de obstáculos.

Para o novo treinamento da RNA deve ser feito a parada do robô, e mostrar os novos padrões de cores e/ou tamanhos que o robô deve reconhecer a partir desta reconfiguração no ambiente em que ele está trabalhando. Na tabela 6.1 é apresentado o processo de treinamento para um conjunto de cores.

Tabela 6.1: Tabela de treinamento para a RNA

Tabela de treinamento de cor da RNA			
Cores	*Entradas*	*Saídas*	*Reconhecimento*
Vermelho	10001010	[1 -1 -1 -1]	Não conhecido
Verde	10011010	[-1 1 -1 -1]	Conhecido
Azul	10101010	[-1 -1 1 -1]	Conhecido
Amarelo	10111010	[-1 -1 -1 1]	Conhecido

A RNA deve gerar saída o valor [1 -1 -1 -1] quando um obstáculo não for reconhecido. A saída [1 -1 -1 -1] liga a saída da RNA à entrada do algoritmo genético, gerando, portanto, uma nova solução para o obstáculo. Qualquer outra saída liga a saída da RNA ao banco de conhecimento para obstáculos previamente conhecidos.

O banco de conhecimento de obstáculos conhecidos possui os algoritmos de soluções para cada um dos três obstáculos conhecidos. Sendo que o algoritmo para os três casos é o mesmo, no entanto, as distâncias e ângulos são distintos para cada caso.

6.2. Ferramentas utilizadas para a implementação do código

A implementação do algoritmo híbrido foi desenvolvida na linguagem de programação *C.* Todo o controlador utilizou esta linguagem de programação. Foi utilizado o compilador *Kdevelop* em conjunto com o *CMake* para a implementação dos códigos de maneira independente e de maneira global para fundir os algoritmos (RNA+AG) e gerar o algoritmo híbrido.

O compilador *Kdevelop* é um *software* livre, para *Linux*. Suporta as linguagens de programação C/C++ utilizadas neste trabalho (KDevelop, 2015).

O *CMake* é um sistema que trabalha com um pré-processamento para o compilador, gerando de maneira automatizada arquivos auxiliares para algoritmos. Desta forma pode-se criar um projeto de maneira completa (CMake, 2015).

O simulador de robôs V-REP utilizado para as simulações robóticas deste trabalho, teve como objetivo a validação do controlador neurogenético. O simulador robótico é gratuito para estudantes (COPELIA ROBOTS, 2015). O simulador está disponível para *Windows, Linux e OS X.* Uma melhor descrição deste simulador foi feita no capítulo 3.

Como exemplo, este trabalho possui alguns arquivos extras para leitura (população.txt, geração.txt entre outros). Esses arquivos podem ser unidos em um único projeto com o auxílio do *CMake*.

Os controles básicos do robô como sensoriamento, controle de motores, orientação foram implementados dentro do simulador (ver descrição do simulador no capítulo 3) em linguagem LUA.

A linguagem LUA é inteiramente projetada, implementada e desenvolvida no Brasil, por uma equipe na PUC - Rio de Janeiro (Pontifícia Universidade Católica do Rio de Janeiro). Esta linguagem de programação é poderosa, rápida e leve, projetada principalmente para estender aplicações (LUA, 2015).

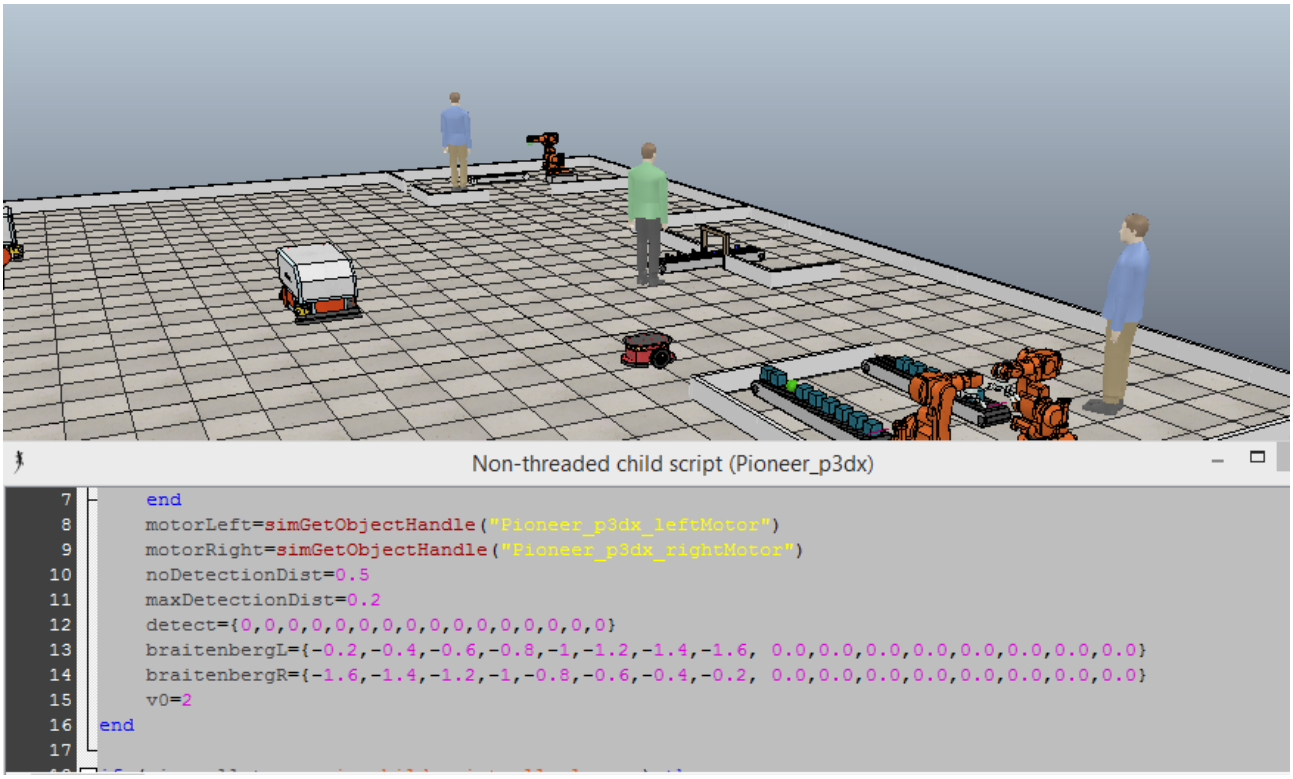


Figura 6.3: Leitura dos sensores em linguagem Lua (controles básicos do robô)

Um conceito fundamental no projeto de LUA é fornecer *meta-mecanismos* para a implementação de construções, em vez de fornecer uma exaustiva quantidade de construções diretamente na linguagem. Embora Lua não seja uma linguagem puramente orientada a objetos, ela fornece meta-mecanismos para a implementação de classes e herança (LUA, 2015).

Os meta-mecanismos de Lua trazem uma economia de conceitos e mantêm a linguagem pequena, ao mesmo tempo que permitem que a semântica seja estendida de maneiras não convencionais (LUA, 2015). Na figura 6.3 pode ser observada a programação em linguagem LUA para à leitura dos sensores ultrassônicos e acionamento dos dois motores.

A função *simGetObjectHandle* faz a leitura de um objeto dentro do simulador, seja este um motor, um sensor, para que o código em LUA possa ter acesso às informações do ambiente em tempo real e poder inferir de maneira controlada (COPELIA ROBOTS, 2015).

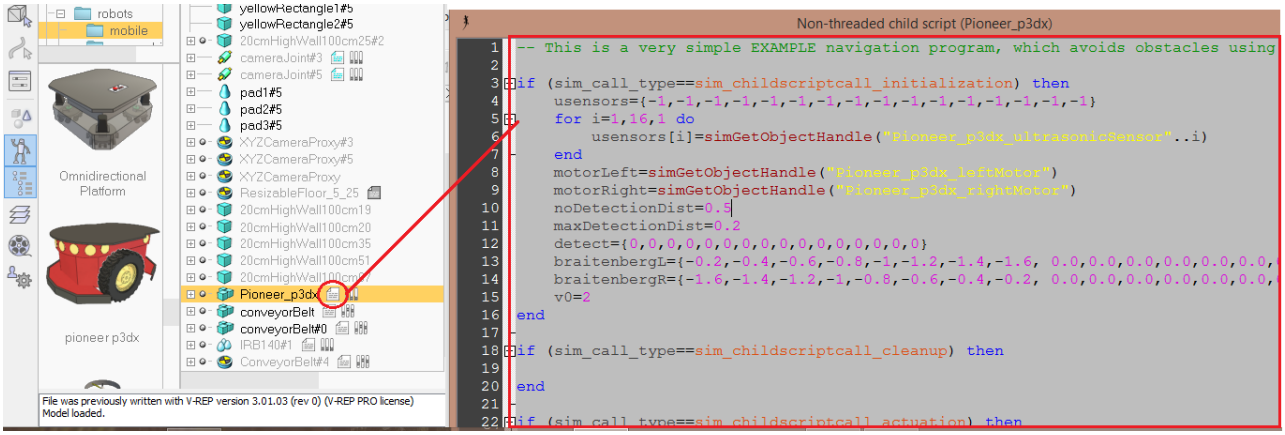


Figura 6.4: Código Lua correspondente ao controle básico do robô *Pioneer*

Estas funções de controle interno podem ser utilizadas via cliente/servidor, assim possibilitando uma programação do controlador do robô em uma outra linguagem suportada pelo simulador (COPELIA ROBOTS, 2015). Na figura 6.4 pode ser observado na lista de objetos da simulação, o robô *Pioneer* e seu código de controle básico anexado, circulado em vermelho. Ao lado da lista de objetos, o código correspondente ao controle do robô em linguagem LUA é mostrado.

Para este trabalho, foi utilizada uma comunicação cliente/servidor em linguagem *C* a fim de se tornar possível a implementação do controlador híbrido também nesta linguagem. O controle básico de sensores e motores foi mantido em linguagem LUA, dentro do próprio simulador. Na próxima seção é descrito o sistema de comunicação cliente/servidor utilizado neste controlador.

6.3. Comunicação cliente/servidor

A comunicação cliente servidor utilizada neste trabalho foi desenvolvida em linguagem *C* juntamente com o controlador híbrido. Para a comunicação cliente/servidor foram utilizadas funções *remote API* (BERRI, 2015).

As funções *remote API* são utilizadas para que um programa desenvolvido em outra linguagem tenha acesso a dados do simulador, como por exemplo: leitura de sensores, orientação do robô e acionamento de motores. Resumidamente as funções *remote API* são aplicações que podem ser acessadas e fazer alterações a partir de outras linguagens (COPELIA ROBOTS, 2015).

Com o cliente/servidor trabalhando em conjunto com as funções *remote API* o controlador pode ser desenvolvido em qualquer uma das linguagens suportadas por este simulador. Neste trabalho, a linguagem utilizada foi C/C++, utilizando consequentemente para a comunicação do controlador híbrido com o simulador de robôs as funções *remote API* (C/C++).

Dentro do *script* principal da cena, é possível encontrar todas as configurações automáticas do simulador. Tudo o que for criado ou alterado dentro do "mundo" utilizado como cena de simulação gera uma alteração automática dentro deste código principal.

Alterações manuais pelo programador não são bem-vindas, desde que seja feita com o reconhecimento do risco de que a cena poderá parar de funcionar se for corrompido alguma parte do código que foi criado automaticamente. Na figura 6.5 é apresentada uma alteração no *script* principal da cena (BERRI, 2015), para a inclusão da função *simExtRemoteApiStart(19999).*

```
12
13  -- =====[Inicialização]====(executada apenas uma vez, no início de simulação) ---------
14  if (sim_call_type==sim_mainscriptcall_initialization) then
15      simOpenModule(sim_handle_all)
16      simHandleGraph(sim_handle_all_except_explicit,0)
17      simExtRemoteApiStart(19999)
18  end
19  ------------------------------------------------------------------------------
```

Figura 6.5: Inicialização do servidor

O comando *simExtRemoteApiStart(19999)* inicia o servidor na porta 1999. Este comando inicia o servidor toda vez que o simulador estiver rodando. A conexão com o servidor é finalizada quando é parada a simulação (BERRI, 2015).

Para o caso da leitura do sensoriamento ultrassônico, a informação recebida pelo controlador é apenas se um obstáculo foi encontrado ou não. Ou seja, o sensor ultrassônico tem uma resposta binária. Se um obstáculo for encontrado, o algoritmo neurogenético entra em funcionamento para encontrar uma solução de desvio para o obstáculo.

Na figura 6.6 observa-se o terminal aberto com a comunicação cliente/servidor ativa. A tela mostrada na figura está exibindo a leitura dos oito sensores frontais do robô *Pioneer*.

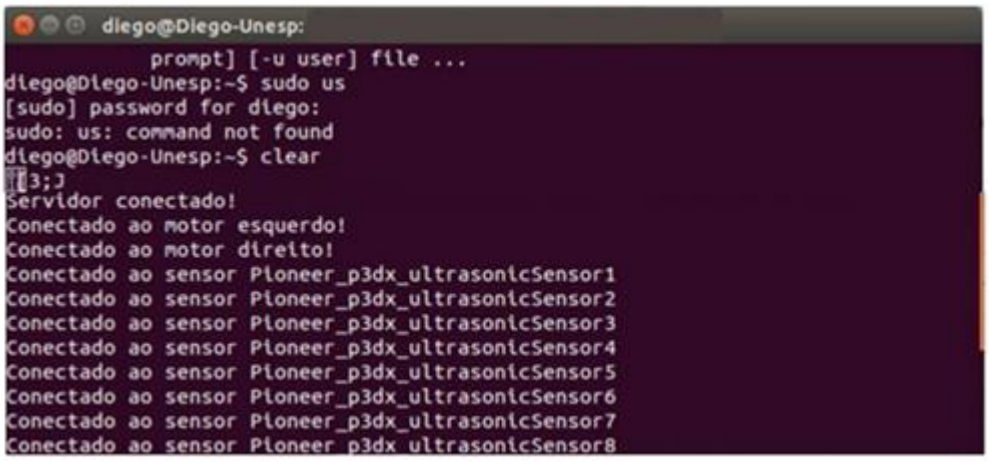


Figura 6.6: Cliente/servidor em execução

Uma das funções da comunicação cliente servidor neste trabalho é enviar para o controlador a posição que o robô encontra um obstáculo e a posição que o robô deve alcançar como objetivo. A partir destas informações, o algoritmo neurogenético poderá gerar uma solução de desvio para o obstáculo encontrado. Na figura 6.7 é apresentada uma situação em que o robô encontra um obstáculo em sua rota. A posição do GPS mostrada na caixa deve ser enviada para o controlador.

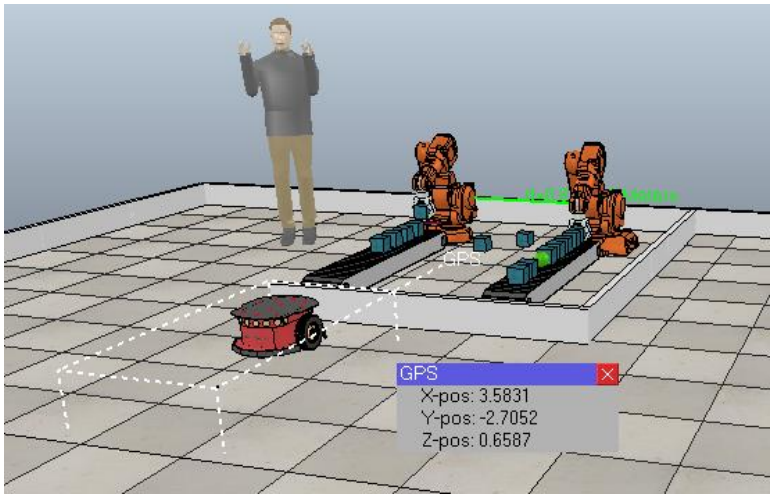


Figura 6.7: Coordenadas do robô via GPS no momento que encontrou um obstáculo

Outra função da comunicação cliente/servidor é enviar os dados de sensoriamento para entrada da RNA, para que seja possível a análise de reconhecimento do obstáculo. As variáveis de entrada para a RNA são: cor e tamanho. Uma descrição mais detalhada deste processo pode ser vista no capítulo 3.

Para enviar e receber estes dados pela comunicação cliente servidor, devem ser utilizados os comandos das funções *remote API* (C/C++). Para a funcionalidade destes comandos ser ativa, é necessária a integração de três bibliotecas nativas do simulador V-REP junto ao projeto cliente/servidor. As bibliotecas utilizadas são: *extAPI.h, extApiInternal.h* e *extApiPlatform.h* (BERRI, 2015). Um dos comandos mais utilizados nesta comunicação cliente/servidor é: *simGetObjectHandle.* Suas características essenciais são apresentadas na tabela 6.2.

Tabela 6.2: Funções *remote API* para o comando *simGetObjectHandle* (COPELIA ROBOTICS,2015)

simxGetObjectHandle regular API equivalente: *simGetObjectHandle* ROS *service* equivalente: *simRosGetObjectHandle*	
Descrição	Recupera um identificador de objeto com base em seu nome. Pode enviar e receber informações apenas com o nome do identificador.
C sinopse	*simxInt simxGetObjectHandle(simxInt clientID, const simxChar* objectName,simxInt* handle,simxInt operationMode)*
Parâmetros em C	*clientID:* a identificação do cliente. Referem-se a simxStart. *objectName:*o nome do objeto. Se possível, não contam com o mecanismo de ajuste automático de nome, e sempre especificam o nome do objeto completo, incluindo o #: se o objeto é *"myJoint",* especifique *"myJoint #",* se o objeto é "myJoint # 0", especifique "myJoint # 0", etc. *Handle:* ponteiro para um valor que vai receber o identificador *operationMode:* um modo de operação função API remoto. Um modo de operação recomendada para esta função é *simx_opmode_oneshot_wait*

Na tabela 6.2 é apresentada uma simples descrição para à função *simGetObjectHandle.* Esta função pode ser utilizada em outras linguagens como: *Python, Java, Matlab, Octave, Urbi, Lua* (COPELIA ROBOTS, 2015). Um comando gerado por essa função retorna ou

envia um valor a partir do nome de identificador do objeto, seja este um sensor, motor ou outro objeto no "mundo" criado para simulações robóticas.

A aplicação cliente/servidor utilizada neste trabalho foi desenvolvida na distribuição Ubuntu, que utiliza uma arquitetura de sistema operacional *Linux* (BERRI, 2015). Esta escolha foi baseada nas escolhas de ambientes de desenvolvimento de códigos e projetos descritos na seção 5.2 deste trabalho. Sendo que um destes ambientes, o *Kdevelop* só é disponível para a arquitetura *Linux*, e por trabalhar com uma configuração de criação de projetos que engloba vários códigos criados no CMake, foi escolhido para este trabalho (BERRI, 2015).

Para a comunicação do GPS com o *remote API* também foi utilizada a função *simxGetObjectHandle.* Essa função será chamada toda vez que um obstáculo for detectado pelo sensoriamento.

```
simxGetObjectPosition(clientID, gpsHandle, -1, (simxFloat *) posicao,
simx_opmode_buffer);
```

Esta função também deve auxiliar o controlador no algoritmo de ponto a ponto. Para este caso, é gerada uma nova coordenada de desvio pelo algoritmo neurogenético, sendo o valor posteriormente repassado ao algoritmo de ponto a ponto fazer o desvio do obstáculo e continuar seguindo até seu ponto de objetivo final. Caso um novo obstáculo seja encontrado é necessário que o controlador gere um novo ponto intermediário e este procedimento recomece.

Capítulo 7

Testes comportamentais e de custo computacional do algoritmo neurogenético

Neste capítulo são apresentados os testes comportamentais e de custo computacional feitos no conjunto de algoritmos que formam o algoritmo híbrido desenvolvido neste trabalho, intitulado como algoritmo neurogenético. Para a validação do algoritmo, neste capítulo foram apresentadas as análises de comportamento e custo computacional que foram feitas em cada um dos códigos a partir de simulações no ambiente dinâmico modelado com base em dados coletados nos ambientes reais.

A análise de comportamento do algoritmo teve como objetivo avaliar o controle do robô em situações diversas que devem ser encontradas por ele em plena navegação. Esta análise foi focada em situações que o controlador teve que encontrar uma solução de desvio, tanto para obstáculos conhecidos pela RNA, quanto para soluções inéditas que foram solucionadas pelo AG.

A análise de custo computacional de cada algoritmo foi focada em verificar o tempo que o robô demora para encontrar uma solução usando cada um dos códigos. Com essa análise pode-se indicar em quais situações o algoritmo desenvolvido neste trabalho apresenta vantagens, já que o custo de tempo é um dos fatores mais importantes quando se espera encontrar uma solução para o desvio da rota.

Para as simulações no ambiente dinâmico modelado, foram utilizados os dados coletados nos ambientes industriais reais e aplicados em várias situações para a avaliação de comportamento e flexibilidade do algoritmo neurogenético e consequentemente a sua validação.

7.1. Testes comportamentais do algoritmo genético

Nesta seção é mostrado o funcionamento do AG a partir de simulações no ambiente dinâmico modelado. Em todos os testes aqui feitos o AG foi utilizado de maneira independente, ou seja, nestes testes não foi utilizada a RNA. Na figura 7.1(a) pode ser observado um caso em que a melhor rota do robô móvel foi impedida por uma empilhadeira (robô KUKA) impossibilitando o robô de chegar ao seu ponto de objetivo.

O AG gerou um novo ponto para o desvio para que o robô continuasse sua rota até o ponto de objetivo. Para este caso a solução foi satisfatória, pois o ponto de desvio não gerou um novo conflito com obstáculos.

Na figura 7.1(b) pode ser observada a rota alternativa gerada pelo ponto de desvio do AG. Para este caso o AG conseguiu uma boa solução. Porém, em alguns casos, o AG pode gerar uma solução que pode levar o robô para um desvio que resultaria em

problemas, como por exemplo: o robô poderia desviar deste obstáculo e acabar encontrando outro obstáculo não esperado.

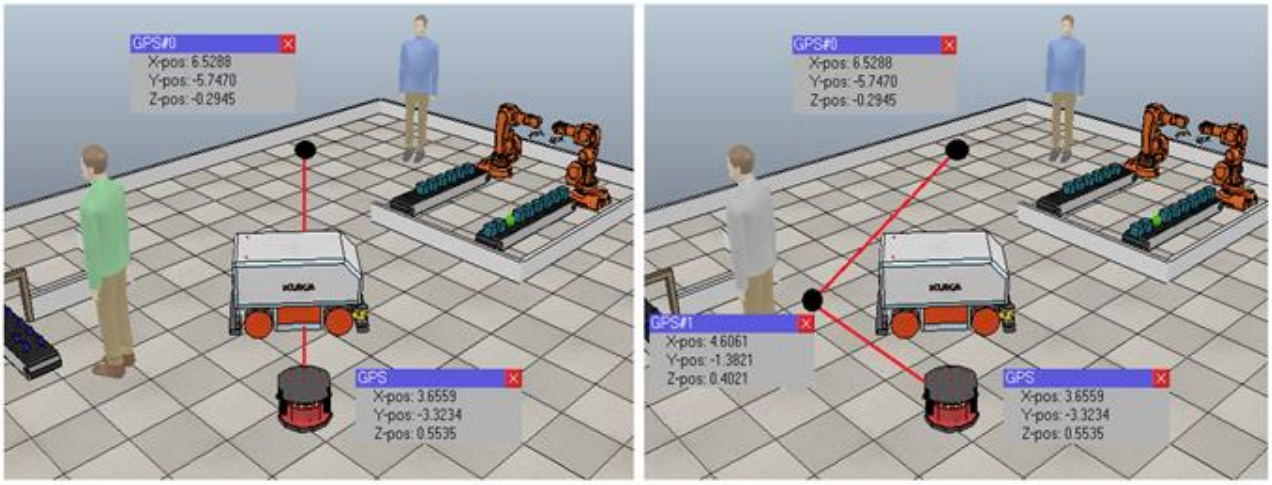


Figura 7.1: Solução gerada pelo AG. (a) melhor rota sendo impedida por uma empilhadeira; (b) solução encontrada pelo AG para o desvio do obstáculo

Isto deve acontecer com frequência, uma vez que o robô não tem um conhecimento do ambiente *a priori.* É inevitável também que o AG gere uma solução e em alguns casos não consiga desviar do obstáculo e tenha que gerar uma nova solução. Este caso é bastante frequente em obstáculos móveis, pois pode-se desviar do obstáculo e ele entrar de novo na rota do robô. Várias soluções podem ser geradas para solucionar este caso.

Na figura 7.2(a) é apresentado um caso em que a solução gerada pelo AG para o desvio da empilhadeira implica em outro obstáculo na rota do robô de mesmo nível de problemática, pois ambos são obstáculos móveis.

Uma solução poderia ter sido gerada para o lado oposto (lado esquerdo) assim evitando que o AG tivesse que encontrar uma segunda solução para um novo obstáculo, porém como o ambiente é desconhecido, tudo que tiver fora do alcance do sensoriamento é novo para o robô. Os humanos não seguem um modelo de uniforme pois o AG "puro" não deve utilizar este tipo de informação, pois neste caso não está utilizando a RNA.

Na figura 7.2(b) é apresentado um caso que a solução gerada pelo AG para o desvio de um obstáculo, no caso, uma caixa, gera um problema conhecido como obstáculo infinito, onde a dimensão do obstáculo não é possível de ser captada pelo sensoriamento.

No caso apresentado na figura 7.2(b) o robô deve desviar das caixas que não são conhecidas pela RNA por estarem em grupo e encontrar a parede em seu caminho. Se a solução tivesse sido gerada pelo lado inverso, o caminho estaria livre.

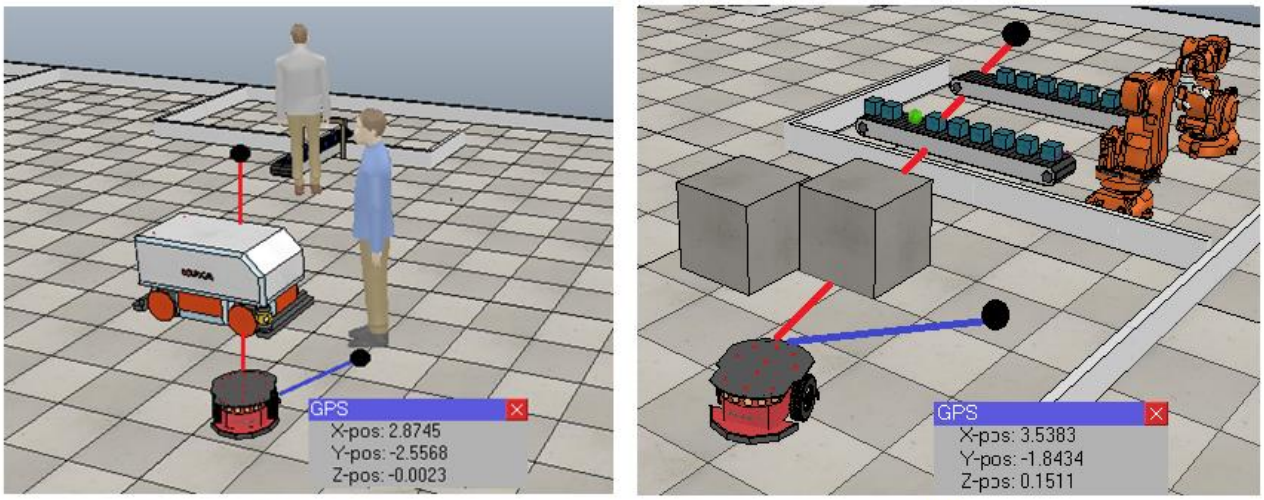


Figura 7.2: Obstáculo encontrado na rota de desvio (a) humano detectado na rota de desvio (b) parede encontrada na rota de desvio

Obstáculos infinitos são um problema de grande relevância na robótica móvel. Este problema está presente principalmente em ambientes totalmente desconhecidos, ou seja, onde o robô não possui um mapa do ambiente *a priori.*

Sobre a análise comportamental do AG para um controlador, cuja aplicação tem finalidade de auxiliar a navegação do robô em um ambiente dinâmico e em contextos em que ele nunca realizou um trabalho antes e sem o auxílio de um mapa, notou-se que é possível fazer com que o robô consiga realizar sua função por atingir os pontos de objetivo utilizando suas soluções heurísticas sem grandes problemas.

Mesmo com a problemática de obstáculos infinitos o robô conseguiu realizar seu trabalho, sendo que este tipo de situação é normal neste tipo de navegação e (onde o ambiente é dinâmico e não existe um mapa *a priori*) foi encontrado neste trabalho em diversos casos.

A RNA deve reduzir consideravelmente o problema de obstáculos infinitos, utilizando o conhecimento sobre alguns obstáculos cadastrados em seu banco de conhecimento. A pesquisa científica sobre este assunto aposta fortemente em estudos de Inteligência Artificial para solucionar tais problemas. Alguns trabalhos sobre Inteligência Artificial foram apresentados no capítulo 2.

7.2. Testes comportamentais da RNA e suas rotinas de solução

Para o treinamento da RNA, foram apresentados padrões na entrada da RNA e feitos os testes para avaliar o reconhecimento de obstáculos diversificados e, portanto, validar o ajuste dos pesos sinápticos, fundamentais para o bom funcionamento do controlador. O algoritmo de aprendizado da RNA apresentou comportamento satisfatório para o reconhecimento de obstáculos de maior frequência no ambiente.

A RNA possui entradas de dados textuais, dados gerados pelo sistema de VC. Para o reconhecimento da cor, sendo um canal para a cor vermelha (R), um canal para a cor verde (G), e um canal para a cor azul (B). Para o reconhecimento do tamanho são utilizadas as entradas referentes a contagem de *pixels.* O último neurônio é o de *BIAS.* Uma descrição detalhada sobre a estrutura desta rede foi apresentada no capítulo 3.

Os dados são enviados de maneira paralela para a função de ativação da rede, sendo esta uma característica marcante das RNAs. Para a aplicação da RNA neste trabalho, sua velocidade de processamento está muito ligada ao conjunto de entradas em paralelo, que neste caso, é superior a um processamento serial. Uma descrição detalhada da RNA que foi utilizada no algoritmo neurogenético pode ser vista no capítulo 3.

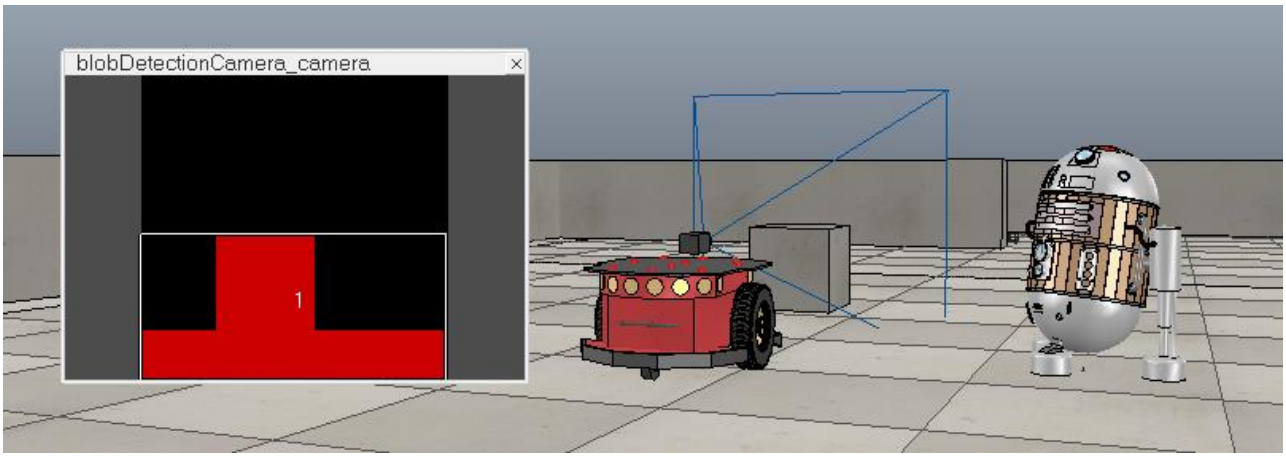


Figura 7.3: Entrada de cor e tamanho para a RNA quando o obstáculo caixa é detectado

Na figura 7.3 pode ser observado o funcionamento da câmera *blob detection* acionada quando um obstáculo, neste caso, uma caixa foi detectada pelo sensoriamento ultrassônico. A "janela" de reprodução da visão da câmera mostra a caixa sendo detectada e sendo "binarizada" (processo que destaca o obstáculo) em relação ao fundo da imagem (COPELIA ROBOTS, 2015). Esta característica de processamento de imagens digitais da câmera utilizada facilitou bastante a obtenção das variáveis de entrada para a RNA.

O sistema de visão aplicado no reconhecimento de obstáculos cadastrados no banco de conhecimento da RNA mostrou-se eficiente na maioria dos casos. Problemas foram

apresentados, mas nenhum deles foi relevante a ponto de neutralizar as vantagens da RNA utilizada no controlador neurogenético.

Um dos maiores problemas é relacionado ao agrupamento de obstáculos (caixas, humanos e empilhadeiras) conhecidos que consequentemente se tornam obstáculos desconhecidos. Um processamento de imagem mais apurado poderia resolver este problema segmentando a imagem e extraindo as características de interesse da imagem, utilizando como base os padrões que foram apresentados e cadastrados no banco de conhecimento da RNA.

Porém, para este caso, um poder de processamento de maior custo seria necessário, e toda a técnica de resolução de problemas para obstáculos frequentes estudados e aplicados para este trabalho precisariam ser reavaliados.

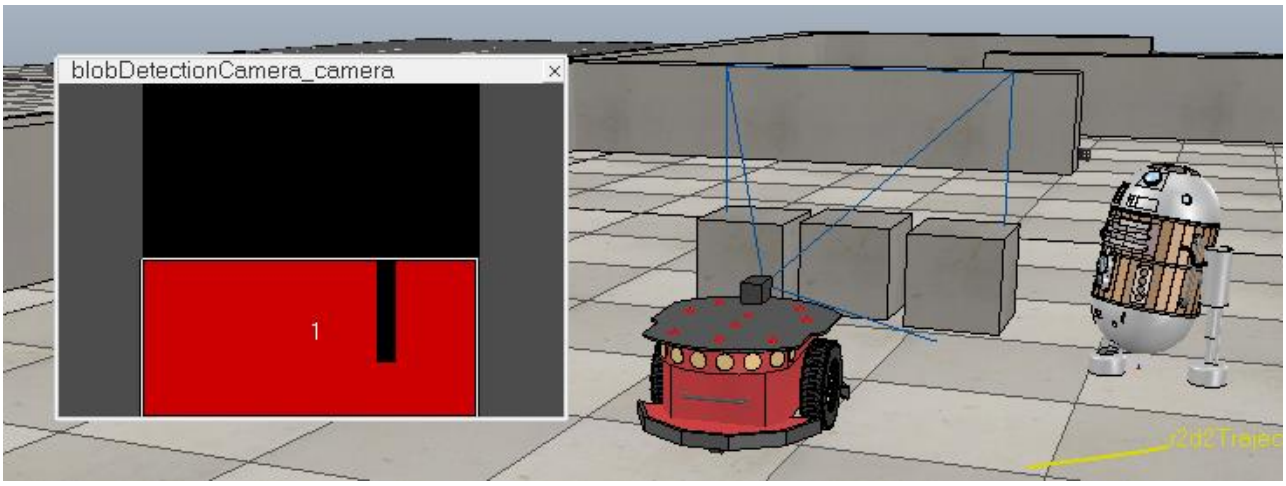


Figura 7.4: Entrada de cor e tamanho para a RNA quando um conjunto do obstáculo caixa é detectado

Na figura 7.4 é apresentado o problema descrito anteriormente. Quando a câmera *blob detection* detecta um conjunto de caixas como um único obstáculo. Para este caso não existe solução cadastrada no banco de conhecimento, pois a dimensão de apenas uma caixa é cadastrada como obstáculo conhecido.

Quando várias caixas se unem formando um único obstáculo as dimensões são modificadas e consequentemente o AG deve ser chamado para encontrar uma solução de desvio de rota.

Mesmo apresentando esta problemática para o uso da RNA, a rede *perceptron* se mostrou bastante eficiente para o reconhecimento de obstáculos frequentes e as rotinas prévias de solução foram chamadas para realizar o desvio de rota sem maiores problemas.

7.2.1. Dados utilizados para o treinamento da RNA

Na Figura 7.5, visualiza-se o arquivo "treino", criado para o treinamento da RNA. Neste arquivo, devem ser apresentados todos os padrões para que os obstáculos sejam reconhecidos pelo algoritmo operacional da RNA. Toda vez que o robô for trocado de ambiente e consequentemente os obstáculos de maior problemática forem mudados, novos padrões devem ser cadastrados a esse arquivo.

Este arquivo de treinamento possui informações de padrões relacionados a cor e tamanho de cada obstáculo cadastrado no banco de conhecimento da RNA. Os dois padrões apresentados à rede devem permanecer na linha *in.*

As três primeiras informações deste vetor são relacionadas às cores RGB, sendo um valor numérico para cada canal [R], [G] e [B]. A saída esperada é visualizada na linha *out* no vetor de saída.

A última informação do vetor é referente ao tamanho do obstáculo, dado pela contagem de *pixels* na horizontal. Lembrando que para o reconhecimento do tamanho de um obstáculo é necessário também a contagem de *pixels* na vertical. Uma informação extra foi utilizada em prol ao reconhecimento de obstáculos, sendo esta, a contagem de *pixels* brancos e pretos na região que representa o obstáculo detectado. Esta variável de entrada foi explicada no capítulo 4 e apresentou melhorias significativas para o sistema de reconhecimento de obstáculos. Então, a RNA utiliza 7 entradas de dados, sendo estas: R, G, B, *pixels* na horizontal, *pixels* na vertical, *pixels* pretos e *pixels* brancos.

Outra observação importante é que quanto melhor estudado e apresentado os padrões ao treinamento da rede, melhores serão os resultados de reconhecimento de obstáculos na fase operacional da rede. Então, um bom treinamento com a apresentação de padrões quantitativos referentes a cada tipo de obstáculo cadastrado no banco de conhecimento da RNA é diretamente proporcional a um bom resultado do controlador neurogenético.

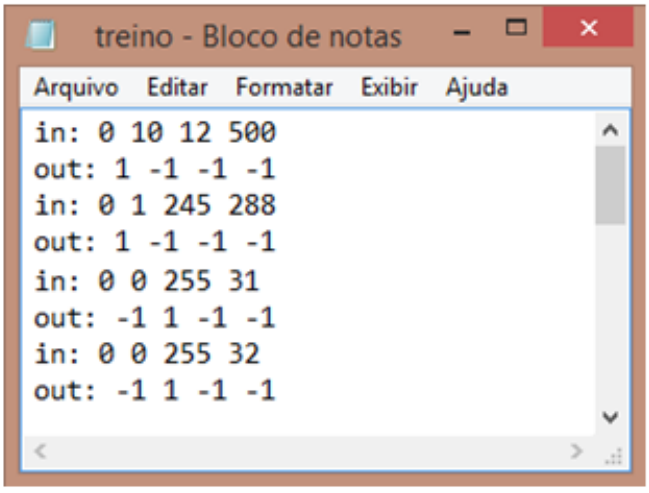


Figura 7.5: Arquivo com os padrões para o treinamento da RNA

Na Figura 7.6, visualiza-se o arquivo de "pesos", gerado pelo algoritmo de treinamento para ser aplicado no algoritmo operacional da RNA. O objetivo do algoritmo de treinamento é gerar este arquivo para que todo obstáculo que seja avaliado pelo sistema neurogenético seja reconhecido ou não pelo algoritmo operacional com base nestes pesos.

Os pesos estão dados na ordem para os neurônios da RNA. O arquivo de *pesos* deve ser lido pelo algoritmo operacional da rede toda vez que um novo obstáculo for encontrado. Lembrando que estes pesos não serão alterados até que um novo treinamento da RNA seja feito.

Outro ponto importante a ser relembrado é que o algoritmo de treinamento da rede só é executado quando o robô for parado para um novo treinamento, já que o algoritmo neural não aprende em tempo real, ou seja, enquanto o robô estiver navegando, ele não aprenderá "coisas" novas.

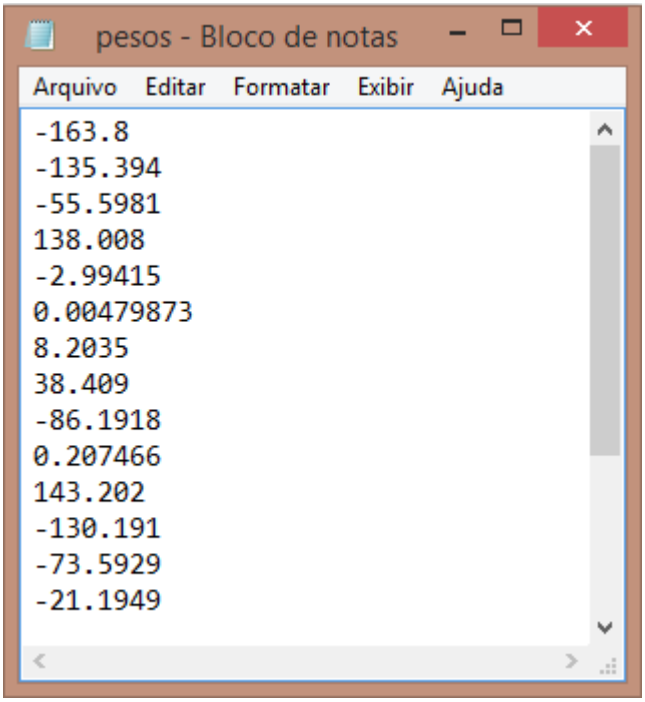


Figura 7.6: Arquivo com os pesos gerados pelo algoritmo de treinamento

Na Figura 7.7, visualiza-se o arquivo "saída". Neste arquivo, é gerado a saída do algoritmo operacional. O vetor de saída informa se o obstáculo foi reconhecido ou não. Se o obstáculo for reconhecido, a posição do vetor deve informar qual é o obstáculo que foi reconhecido e interligá-lo ao banco de conhecimento do algoritmo neural para que seja aplicada uma solução de desvio para este obstáculo.

Um exemplo é dado para o vetor [-1 -1 1 -1], neste caso, o obstáculo reconhecido pela RNA é uma caixa, pois o único valor positivo está na terceira posição do vetor. O vetor [-1 -1 -1 1] representa o não reconhecimento de nenhum obstáculo, já que a última posição do vetor é que está positiva. Uma melhor explicação desta lógica foi feita no capítulo 6.

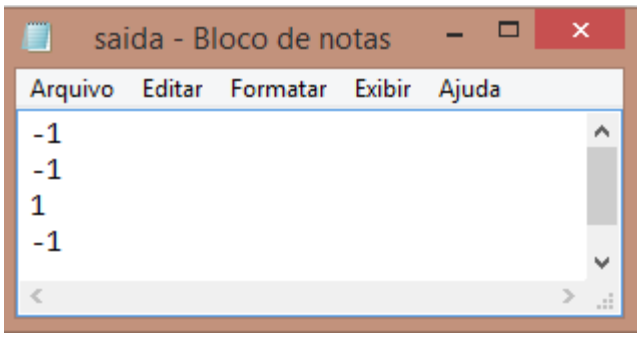


Figura 7.7: Arquivo com a saída gerada pelo algoritmo operacional

Estes arquivos são importantes para o funcionamento do algoritmo neurogenético e devem estar incluídos na mesma pasta que o controlador implementado em linguagem C está armazenado.

7.3. Obstáculos dinâmicos e sua problemática

Os obstáculos podem gerar uma variação muito grande de problemas relacionados à navegação do robô. Algumas aplicações foram feitas em prol da redução de problemas relacionados aos obstáculos dinâmicos e serão descritos nesta seção.

Para os obstáculos dinâmicos que tem a capacidade de se movimentar com comportamentos não esperados no ambiente dinâmico, por exemplo, os humanos e empilhadeiras foram definidas algumas características potenciais ao controlador.

Para o caso do mesmo obstáculo interferir na rota do robô, mais de uma vez, caso que acontece se o robô desvia do obstáculo móvel e novamente o obstáculo infere na rota do robô, uma solução aplicada foi esperar um tempo, cerca de 1 segundo (figura 7.5).

Provavelmente neste tempo o obstáculo móvel poderia já ter saído da rota do robô e assim não seria necessário um desvio, consequentemente reduzindo o custo computacional para esta solução. Esta aplicação pode gerar uma potencial redução de custo computacional, pois o mesmo obstáculo dinâmico pode exigir várias soluções de desvio para um mesmo instante.

Um exemplo análogo a este caso seria quando estamos andando na rua e encontramos um obstáculo dinâmico, este pode ser outro humano, por exemplo. Se estivermos nos movendo contra este humano, ou seja, na mesma reta, o desvio será necessário para evitar uma colisão frontal. Como não conhecemos o comportamento da outra pessoa

decidimos um lado aleatório de desvio, e se este lado for o mesmo escolhido pelo outro humano, teremos novamente uma interferência na rota atual. Este problema pode acontecer infinitas vezes caso uma medida de solução não seja tomada. Na figura 7.8(a) pode ser observado este caso com maior clareza.

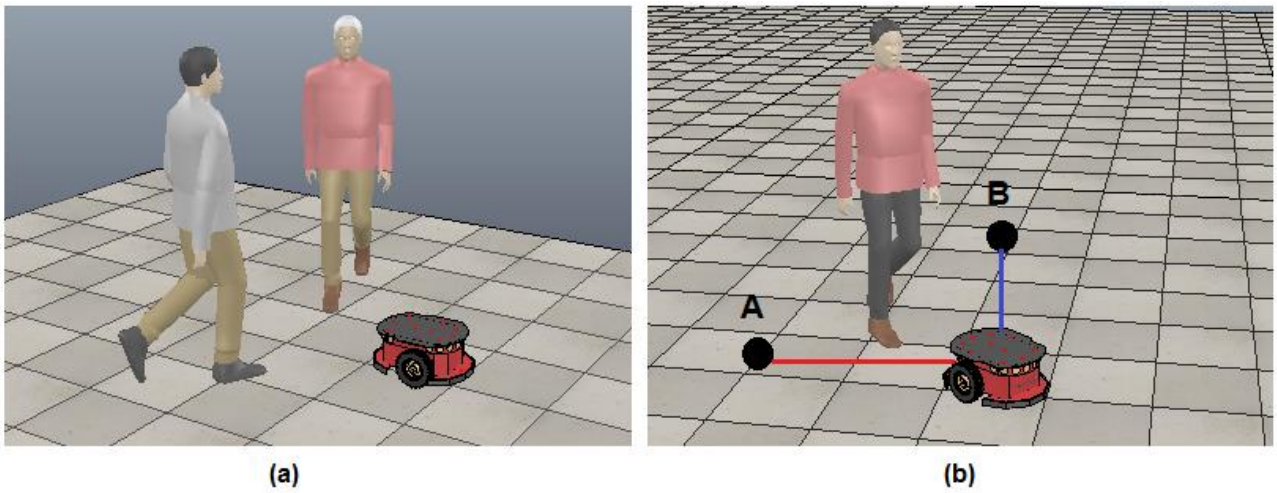


Figura 7.8: Obstáculos dinâmicos (a) humanos se movimentando de encontro um contra o outro (b) robô *Pioneer* e suas opções de desvio aleatórias, a e b

Na figura 7.8(a) é apresentada uma situação em que dois humanos estão se movendo um de encontro ao outro, este caso foi descrito anteriormente.

Na figura 7.8(b) é apresentada uma outra situação em que o robô *Pioneer* deve escolher o lado A ou B de maneira aleatória por não ter, *a priori,* a informação de qual o lado que o humano deve se mover para desviar do robô. Essa problemática deve acontecer com todos os obstáculos dinâmicos.

A escolha aleatória entre A ou B é aplicada ao comportamento deste robô em casos deste tipo. O controlador neurogenético deve gerar uma solução aleatória (sempre com vistas para o menor caminho) se relacionada à direção de desvio, pois ele também desconhece qual região poderá ter um novo obstáculo, ou até mesmo para que lado o obstáculo atual deve se mover.

Se compararmos o caso da navegação do robô, com o caso análogo que foi apresentado na figura 7.8(a), podemos dizer que isso pode acontecer infinitas vezes em ambos os casos. Então, o melhor seria esperarmos o outro humano tomar sua decisão e sair de nossa rota. Enquanto isso, permanecemos parados e consequentemente nos livramos do obstáculo sem qualquer alteração em nossa rota atual. Esta lógica foi aplicada ao controlador do robô.

O robô pode usar esta lógica para solucionar este tipo de obstáculo problemático definindo um número máximo de soluções consecutivas para o mesmo obstáculo. Para o

controlador neurogenético deste trabalho, foram definidas 24 soluções consecutivas como limite. Esta escolha foi baseada no custo de soluções para o pior caso, em que o obstáculo não é reconhecido pela RNA e deve ser solucionado pelo AG. O algoritmo neurogenético tem um custo máximo de tempo igual a 40 milissegundos, sendo 10 milissegundos para a RNA e 30 milissegundos para o AG. Este custo máximo é referente ao pior caso, ou seja, quando o obstáculo não é reconhecido pela RNA e deve ser solucionado pelo AG.

O tempo de 1 segundo foi utilizado como comparativo para este tipo de caso. O motivo é que este é o tempo médio que um obstáculo móvel demora para sair da frente do robô.

Para 25 soluções do algoritmo neurogenético o custo é de 1000 milissegundos, e para 26 soluções o custo deve ser 1040 milissegundos. No caso de 26 soluções consecutivas para o mesmo obstáculo é vantajoso esperar e parar o robô durante 1 segundo. Depois de contado este tempo o robô faz a leitura dos sensores ultrassônicos novamente e se não encontrar o obstáculo novamente, segue sua rota normalmente. Caso contrário o algoritmo neurogenético faz novamente o processo de gerar uma nova solução de desvio.

Uma outra aplicação para a navegação do robô em relação a obstáculos dinâmicos foi implementar um disparo sonoro toda vez que o robô encontra um obstáculo. Este sinal serve para alertar um humano no chão de fábrica para que tome cuidado com o robô. Isso pode evitar a destruição do robô caso o obstáculo seja uma empilhadeira, ou algum dano físico ao humano.

7.4. Orientação do robô para navegação de ponto a ponto

O algoritmo de ponto a ponto é utilizado para seguir do ponto inicial, até o ponto final. O ponto final é alternado para cada caso, pois este ponto seria cada uma das células que o robô deve servir com as peças buscadas no estoque.

O ponto inicial também sofre alterações, pois a cada instante o ponto inicial passa a ser uma nova partida do robô para chegar a um novo objetivo.

Para a orientação do robô foi utilizada uma bússola virtual que se orienta pelos ângulos de Euler (os ângulos de Euler foram formulados para a representação da orientação de um corpo rígido em movimento em um espaço euclidiano tridimensional) ; (BERRI, 2015).

A orientação do robô é dada em radianos. A frente do robô é declarada por um eixo virtual e este é utilizado para saber qual a rotação necessária para o robô se alinhar com o ponto de objetivo e se movimentar em linha reta (BERRI, 2015).

Na figura 7.9 são apresentados os ângulos de Euler X,Y e Z sendo utilizados para a orientação dos movimentos de rotação de aeronaves em torno dos seus eixos principais (ALSINA, 2008).

Os movimentos também podem ser chamados de rolamento (φ), lançamento (θ) e guinada (ψ). Devem ser utilizados para a orientação da aeronave em relação da sua posição no espaço (ALSINA, 2008).

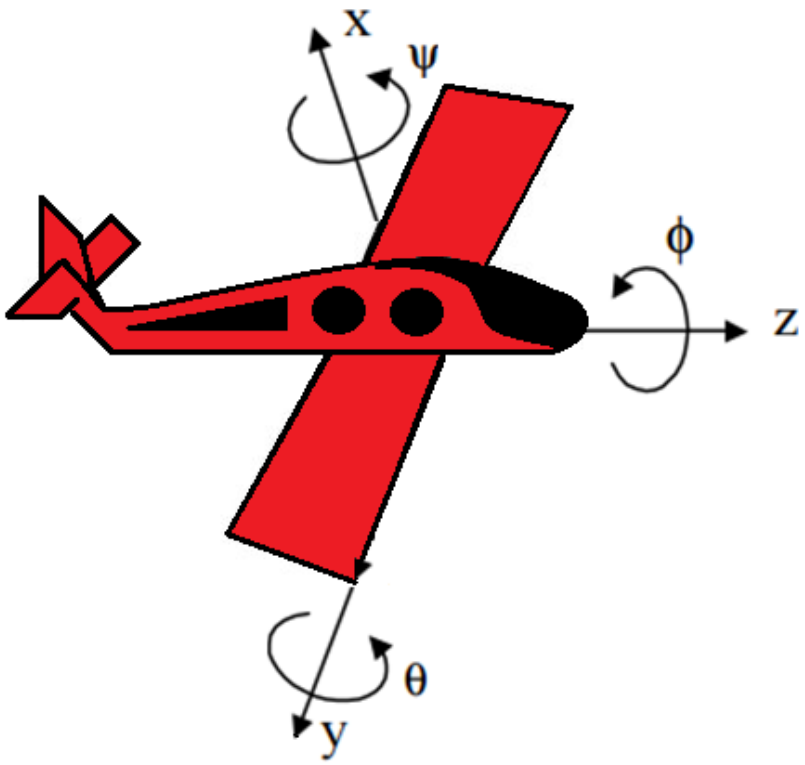


Figura 7.9: Aeronave com seus ângulos de rolamento, lançamento e guinada φ, θ, ψ. Imagem adaptada de (ALSINA, 2008)

Para o caso do robô móvel é utilizado apenas um eixo, e este orienta a frente da plataforma robótica em relação ao universo. Com este eixo é possível utilizar os ângulos de Euler e orientar o robô utilizando a bússola virtual da plataforma *Pioneer* (BERRI, 2015).

A função abaixo retorna a orientação do robô em radianos:

```
simAddStatusbarMessage(string.format("AngEuler: %.4f %.4f %.4f
Orientacao:
%d",AngEuler[1],AngEuler[2],AngEuler[3],valorangulo))
```

Na figura 7.10 é apresentado o robô *Pioneer* em uma situação que encontra um obstáculo. Os ângulos de Euler e a orientação é impressa na tela utilizando a função *simAddStatusbarMessage* que pertence ao *remote API* (BERRI, 2015).

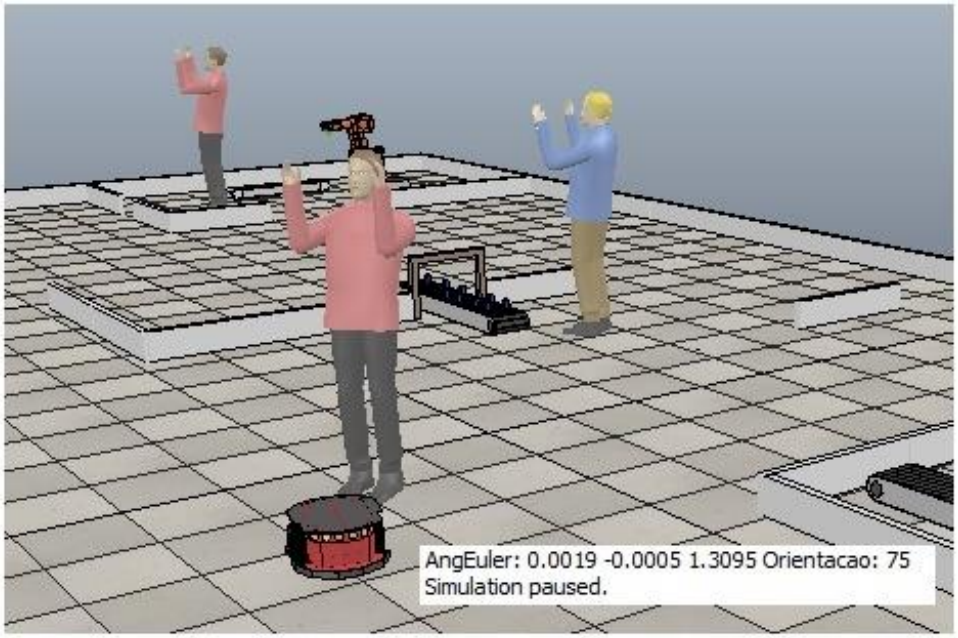


Figura 7.10: Ângulos e Euler e orientação do robô em radianos

Com esta orientação o robô deve girar em torno da própria base e se alinhar ao ponto de objetivo para logo depois seguir em linha reta até alcançar o seu objetivo ou encontrar novamente um obstáculo. Na próxima seção é descrito o algoritmo utilizado para um robô navegar entre dois pontos pré-definidos.

7.5. Algoritmo de ponto a ponto

O algoritmo de ponto a ponto deve controlar o robô para que ele consiga sair do ponto inicial seguir até o seu ponto de objetivo. Este algoritmo controla o robô baseando-se na bússola virtual descrita na seção anterior.

O controle do algoritmo de ponto a ponto é interrompido quando o robô encontra um obstáculo e a chamada do algoritmo híbrido é feita para encontrar uma solução de desvio de rota. O algoritmo de ponto a ponto é chamado novamente quando a posição de desvio for encontrada.

O algoritmo de controle para movimentar o robô de ponto a ponto deve ler o sinal gerado pela bússola virtual e também as coordenadas dos pontos de objetivo e assim orientar o robô para seguir até o seu destino. O pseudocódigo para este algoritmo é apresentado na janela a seguir.

Algoritmo 7.1: Algoritmo de ponto a ponto

Pseudocódigo do algoritmo de ponto a ponto

Leia os pontos inicial e final
Orienta-se o robô com sua bússola

SE o robô está alinhado com o ponto de objetivo
ENTÃO
Aciona os dois motores com igualdade de sinal
SE NÃO rotacionar robô invertendo o sinal dos motores até se alinhar com o objetivo
SE alinhado em linha reta
ENTÃO
Segue em linha reta comparando o ponto atual com o ponto de destino
SE ponto de objetivo é igual ao ponto atual

ENTÃO
Desliga os motores
Fim

SE obstáculo encontrado
Chama algoritmo neurogenético para a solução
RETORNA o novo ponto de objetivo intermediário

Fim do procedimento

Se o robô não estiver alinhado deve-se rotacionar sua plataforma até que o alinhamento se efetive. Uma inversão no sinal dos motores deve girar o robô até que o alinhamento aconteça.

Após o alinhamento o robô segue em linha reta até o ponto de objetivo. Um comparador deve comparar o ponto atual em que o robô se encontra com o ponto de objetivo. Se o ponto atual for igual ao ponto de objetivo significa que o robô já chegou no seu destino e deve parar os motores. Caso contrário o robô deve permanecer com os motores ligados em igual proporção até atingir o ponto de objetivo ou encontrar um novo obstáculo.

Se um obstáculo é encontrado o algoritmo de ponto a ponto é abortado, os motores são desligados, a câmera é ligada e o algoritmo neurogenético é chamado para gerar uma solução de desvio.

Quando o ponto intermediário for gerado o algoritmo de ponto a ponto recebe a coordenada desta nova posição e reinicia-se o procedimento do algoritmo descrito anteriormente.

7.6. Análise de viabilidade das soluções geradas pelo algoritmo neurogenético em ambientes reais

Nesta seção é feita uma comparação de custos computacionais e vantagens do uso de cada algoritmo em diferentes casos encontrados no ambiente dinâmico modelado neste trabalho. Os testes foram baseados nos ambientes reais analisados no capítulo 5 e aplicados ao ambiente modelado no ambiente V-REP.

O principal objetivo é analisar a viabilidade do algoritmo em diversas situações reais. Na figura 7.11 é apresentado os tempos para diferentes situações do algoritmo neurogenético.

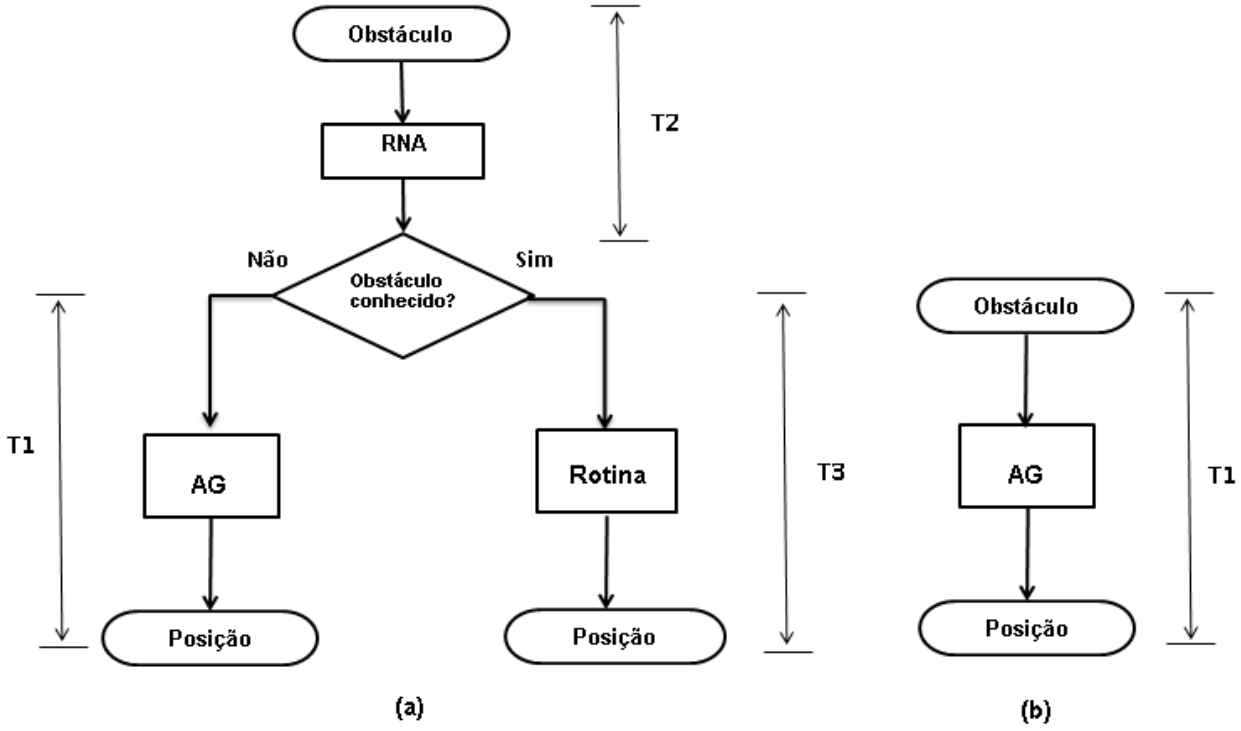


Figura 7.11 : Análise de custo dos algoritmos (a) algoritmo híbrido; (b) algoritmo genético

Na figura 7.11 podem ser observados os tempos gerados para a execução de cada situação do algoritmo neurogenético. O objetivo é mostrar onde este algoritmo deve trazer vantagens. Para isso o tempo de cada algoritmo é comparado em diferentes situações, tendo como objetivo apresentar em quais casos o controlador neurogenético apresenta um desempenho vantajoso.

Os valores reais de tempo de cada algoritmo pertencente ao controlador neurogenético podem ser vistos no capítulo 9, onde foram avaliados o custo de tempo para a RNA e o AG.

Pode-se observar também na figura 7.11, o T1 (tempo 1) é referente ao tempo de execução do AG quando este gera uma posição intermediária para o desvio do obstáculo. Esta solução não é aplicada de maneira independente a este controlador, pois para todo obstáculo encontrado pelo robô a RNA deve entrar em funcionamento a *priori* para verificar se o obstáculo é conhecido. Então a análise de tempo deste algoritmo foi feita apenas para efeitos comparativos com os outros meios que o algoritmo híbrido utiliza para solucionar o problema de desvio de um obstáculo.

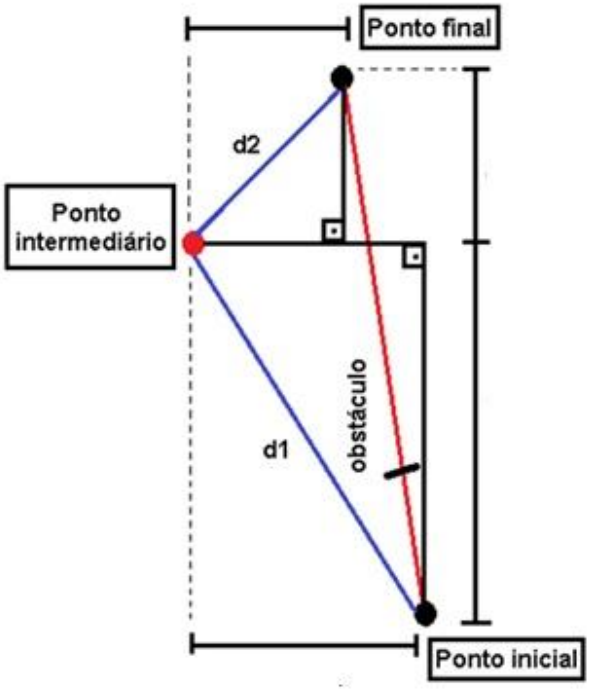


Figura 7.12: Rotina para desvio de obstáculo reconhecido pela RNA, adaptado de (FERREIRA, 2015)

Conforme observa-se também na figura 7.11, o T2 (tempo 2) é referente ao tempo de execução da RNA quando o robô encontra um obstáculo. Este custo de tempo está relacionado ao algoritmo operacional da RNA, já que o algoritmo de aprendizado nunca é utilizado quando o robô encontra um obstáculo. O algoritmo operacional deve verificar as características do obstáculo e comparar com os padrões que foram armazenados no banco de conhecimento. Uma melhor descrição deste algoritmo pode ser vista no capítulo 3.

O T2 deve ser um tempo muito menor que T1. Por este motivo é relativamente vantajoso analisar o obstáculo utilizando a RNA e se o mesmo for reconhecido utilizar a rotina de desvio (T3) que existe para este obstáculo e fazer o desvio da rota.

O T3 (tempo 3) é referente a rotina que o algoritmo neurogenético utiliza para desviar de um obstáculo que foi declarado conhecido pela RNA. O custo de tempo deste algoritmo de desvio é relativamente baixo, pois sua função é apenas armazenar valores de distâncias e o ângulo que o robô deve rotacionar para desviar do obstáculo.

Na figura 7.12 é apresentado o resultado da rotina de solução que gerou um ponto intermediário para o desvio do obstáculo. O caminho em vermelho é feito em linha reta, pois é o menor caminho possível.

O melhor caminho é impedido por um obstáculo e o ponto intermediário gerado pela rotina de solução deve servir como base para gerar a rota alternativa representada na cor azul. Independentemente de onde o ponto intermediário foi gerado o ponto final deve ser atingido.

A soma de tempo T2+T3, sendo estes tempos referentes ao algoritmo de operação da RNA e o tempo de execução da rotina de desvio pré-cadastrada, respectivamente, deve ser sempre menor que o tempo T1 referente ao tempo de execução do AG. Caso isso não aconteça o AG se torna mais viável trabalhando sozinho.

$$(T2+T3) \leq (T1)$$

A relação de custo computacional do algoritmo neurogenético também é dependente do ambiente em que o robô está desenvolvendo seu trabalho, pois se o robô encontrar neste ambiente obstáculos desconhecidos na maioria dos casos, a RNA só deve aumentar o custo computacional das soluções, consequentemente não apresentando um retorno positivo para a navegação do robô. Esta problemática é melhor descrita no capítulo 3. No capítulo 9 é apresentada a comparação de tempo dos algoritmos quando aplicados nos ambientes reais estudados no capítulo 5.

No capítulo 9 são apresentados os resultados referentes à análise de custo computacional dos algoritmos utilizados no controlador neurogenético de maneira independente. Os testes do controlador foram feitos com base nos dados coletados nos dois ambientes industriais apresentados no capítulo 5 e aplicados na modelagem e simulação do ambiente utilizando o *software* de simulações robóticas V-REP.

Capítulo 8

Validação do controlador neurogenético com vistas para as funções propostas ao robô neste trabalho

Neste capítulo será apresentado o controlador neurogenético com vistas a auxiliar o robô no desenvolvimento de suas funções propostas neste trabalho. O objetivo maior para este capítulo foi mostrar diferentes situações para que o controlador pudesse apresentar sua eficiência. Também é apresentado neste capítulo, a sistemática adotada em prol a análise e coleta de dados

Dentre estas situações estão a mudança na configuração dos objetos do ambiente de acordo com a análise feita em ambientes reais, mudança nos pontos de objetivo do robô e mudança na disposição dos obstáculos estáticos.

A tarefa de servir peças entre as células foi estudada em diversos casos e para esta função, foram utilizados dois tipos diferentes de plataformas robóticas, sendo que cada uma das plataformas apresentou suas vantagens e desvantagens descritas no capítulo 3. O controlador neurogenético foi aplicado apenas na plataforma *Pioneer* .

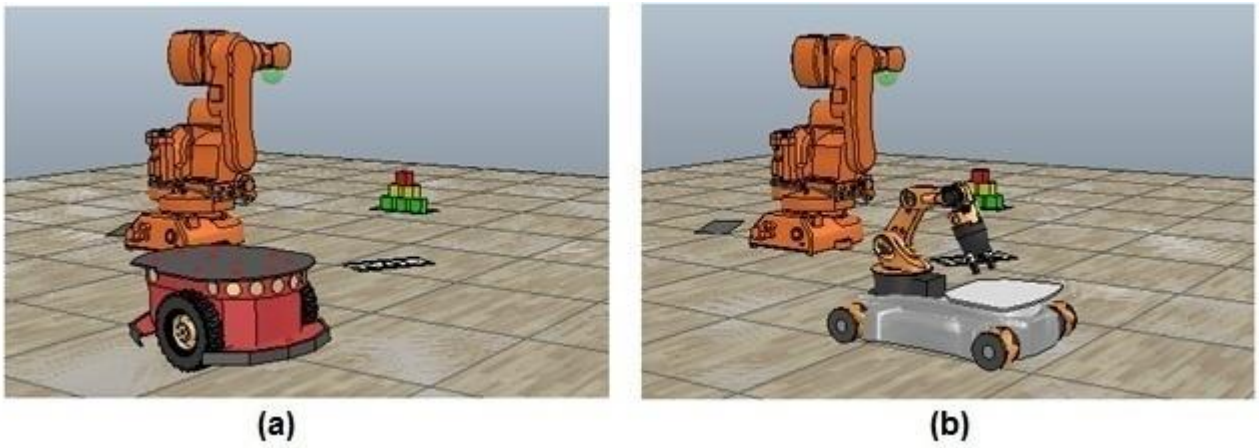


Figura 8.1: Variações de plataformas utilizadas (a) robô *Pioneer* ;(b) robô *YouBot*

O robô *Pioneer* apresentado na figura 8.1 (a) manteve-se em sua forma original para a maioria dos testes. As alterações foram feitas apenas em seu sensoriamento. O motivo da plataforma *Pioneer* ser utilizada para a maioria dos casos é que este tipo de robô é um dos mais indicados para pesquisas com vistas em navegação robótica autônoma.

Seu potencial está ligado ao seu sensoriamento preciso e, também, sua disponibilidade em aceitar novos tipos de sensores.

Outras vantagens desta plataforma robótica para aplicações de navegação autônoma estão ligadas ao seu grau de liberdade, seja pela disposição das rodas, mas também pelo fato de que esta plataforma tem similaridades com a plataforma *Kihon*.

Contudo, uma desvantagem deste robô é a sua incapacidade quando se necessita de um braço robótico para auxiliá-lo em sua tarefa de transporte de peças e, consequentemente, sua plataforma não apresenta bons resultados para este tipo de aplicação. O problema em questão está ligado à forma que o conjunto de rodas se movimenta. Além disso, outro problema está relacionado à capacidade física da plataforma que não suporta um braço robótico. Uma descrição detalhada deste robô foi apresentada no capítulo 3.

Uma proposta para a função de carregamento de peças de forma automática é a utilização do robô *YouBot* que utiliza um braço robótico com vistas ao carregamento de peças (figura 8.1 - b). Esta função será abordada nas próximas seções.

8.1. Navegação do robô entre as células de produção industrial

Para a definição dos pontos a serem servidos pelo robô foi utilizado um mapa com as coordenadas dos 4 pontos. Como o objetivo deste trabalho é um controlador que funcione em ambientes desconhecidos, nenhuma outra informação sobre o ambiente foi inserida neste mapa. Para uma mudança de ambiente deve-se apenas alterar as coordenadas dos pontos a serem servidos pelo robô e um novo treinamento da RNA.

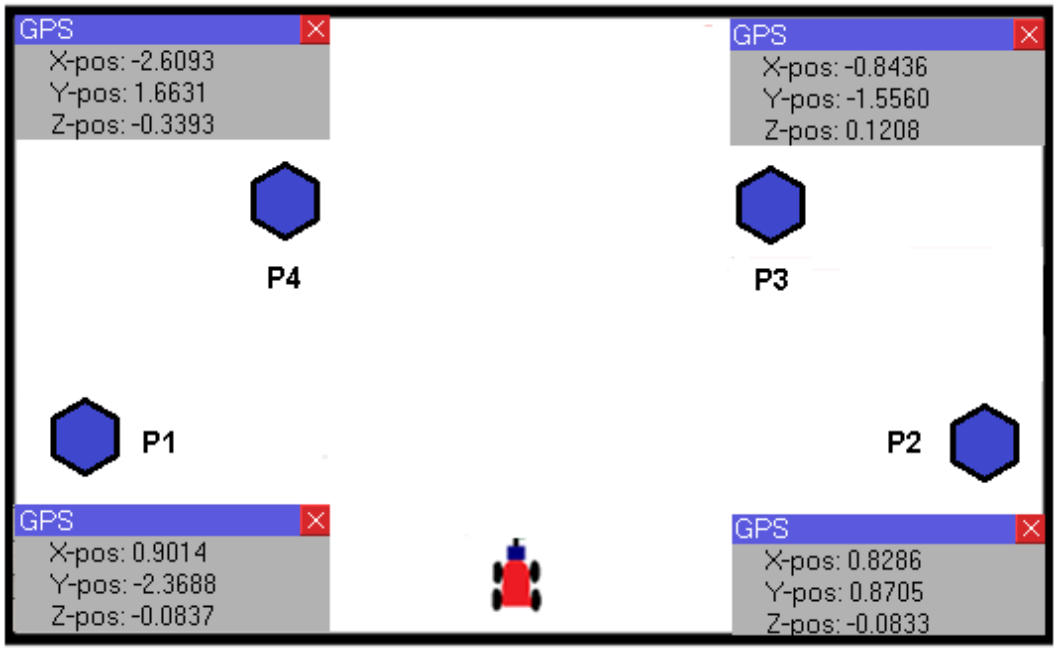


Figura 8.2: mapeamento dos pontos de objetivo do robô

Na figura 8.2 é apresentado o mapa de pontos a serem servidos pelo robô. Para orientar o robô a chegar até estes pontos foram utilizadas as coordenadas cartesianas do GPS, X, Y. O eixo Z deve ser desconsiderado.

O controlador utiliza o algoritmo de ponto a ponto para navegar até estes pontos. Quando um obstáculo é encontrado o algoritmo neurogenético entra em funcionamento e gera uma posição intermediária para o desvio da rota. O controlador deve usar novamente o algoritmo de ponto a ponto para navegar do ponto intermediário até o ponto de objetivo. Se um novo obstáculo for encontrado reinicia-se este processo.

O algoritmo de ponto a ponto utiliza a posição relativa do robô. Esta posição é obtida pela bússola virtual e enviada pela comunicação cliente/servidor por meio da função do *remote API* apresentada na janela de código a seguir (COPELIA ROBOTS, 2015).

```
local
    relativePosition=simGetObjectPosition(targetObjectHandle,robo
    tBaseHandle)
```

O algoritmo de ponto a ponto deve utilizar a função *simSetJointTargetVelocity* para controlar a velocidade dos motores e rotacionar o robô até que aconteça o seu alinhamento com o eixo frontal em relação ao seu ponto de objetivo. Para saber mais sobre os valores corretos a serem aplicados nos motores , é necessário saber a *priori* a orientação atual do robô e sua posição em relação à posição de destino.

Uma inversão nos sinais dos motores esquerdo e direito deve fazer com que o robô rotacione sobre sua base, assim, facilitando o seu posicionamento. Para o tratamento da orientação e posição relativa do robô ao seu ponto de objetivo é utilizada a bússola virtual que foi descrita no capítulo 6.

Para relacionar a posição atual do robô com os pontos de objetivo deve-se utilizar uma matriz de pontos, sendo esta responsável pela criação do mapeamento de pontos de objetivo da navegação. As funções para a leitura do mapa de pontos são apresentadas na janela de código a seguir.

```
robotMatrix=simGetObjectMatrix(robotBaseHandle,-1)
     robotMatrixInv=simGetInvertedMatrix(robotMatrix)
     targetPositionRelativeToRobot=simMultiplyVector(robotMatrixIn
     v,targetPosition)
```

As funções matriciais utilizam as coordenadas absolutas dos pontos de objetivo. Estas coordenadas devem ser declaradas no mapa de pontos de objetivo.

Para as alterações nos pontos de objetivo quando é feita uma reconfiguração no ambiente atual, ou quando é feita a troca de ambiente de trabalho do robô, é necessário apenas a alteração das coordenadas absolutas na matriz, que gera o mapa de pontos de objetivo. Neste trabalho, foram feitas, no mapa, três alterações, tendo elas sido baseadas na análise dos ambientes dinâmicos reais.

O robô *Pioneer* mostrou ser eficiente para a navegação entre os pontos de objetivo, sendo auxiliado pelo controlador neurogenético. Variações nas posições de objetivo foram geradas para os ambientes dinâmicos utilizados neste trabalho, em todos os casos o controlador mostrou ser eficiente na realização da navegação.

Na próxima seção é descrito o comportamento de carregamento de peças, usando os robôs *YouBot* e o robô híbrido.

8.2. Carregamento de peças entre as células de produção industrial

O robô *YouBot* foi proposto a ser utilizado para automatizar o processo de carregamento de peças sem a necessidade do auxílio de um humano ou outro robô. Este robô apresentou eficiência para o carregamento de peças utilizando o braço robótico que o auxilia. Na figura 8.3 é apresentado uma situação em que o robô *YouBot* está fazendo o carregamento de peças utilizando seu braço robótico.

A tarefa deste robô foi pegar peças no depósito e transportá-la até as células que devem ser servidas. Quando as peças transportadas terminarem, o robô deve voltar ao depósito para um novo carregamento (COPELIA ROBOTS, 2015). Uma melhor explicação do funcionamento do transporte de peças foi feita no capítulo 3.

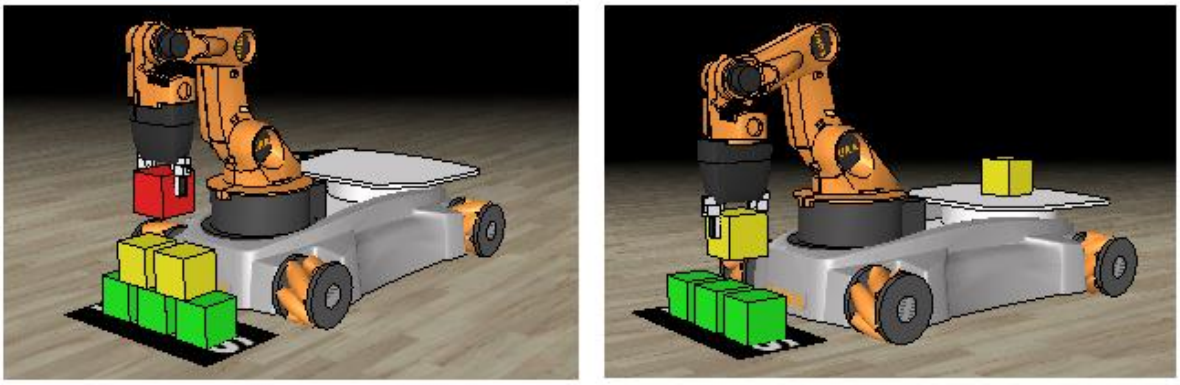


Figura 8.3: robô youbot (a) *YouBot* utilizando seu braço para carregar peças (b) *YouBot* utilizando sua plataforma para carregar peças

Algumas deficiências foram apresentadas para este tipo de plataforma robótica, sendo elas: baixa capacidade da plataforma para carregar peças, baixa velocidade de

locomoção e baixo nível de sensoriamento. Para resolver estes problemas foi criado o robô híbrido, porém não foi possível utilizá-lo por problemas físicos descritos no capítulo 3.

O controlador neurogenético não foi aplicado a esta plataforma por conta das deficiências descritas anteriormente. Para a navegação do robô *YouBot* foi utilizado apenas o algoritmo de ponto a ponto, o que implica na incapacidade de desviar de obstáculos.

Esta característica negativa não trouxe grandes problemas para estes testes, pois o objetivo do uso deste robô é apenas mostrar como deve ser feito um processo automático do carregamento de peças utilizando um braço robótico. De maneira resumida, o único ponto positivo do *YouBot* é seu braço robótico.

Com a base robótica *Pioneer* foi possível aumentar o nível de carga de peças no transporte, consequentemente reduzindo o custo de tempo e energia do robô, pois deve-se consequentemente reduzir o acesso ao depósito. Na base do *Pioneer* também foi possível utilizar os oito sensores ultrassônicos frontais com uma melhor disposição se comparado ao sensoriamento deficiente do *YouBot.*

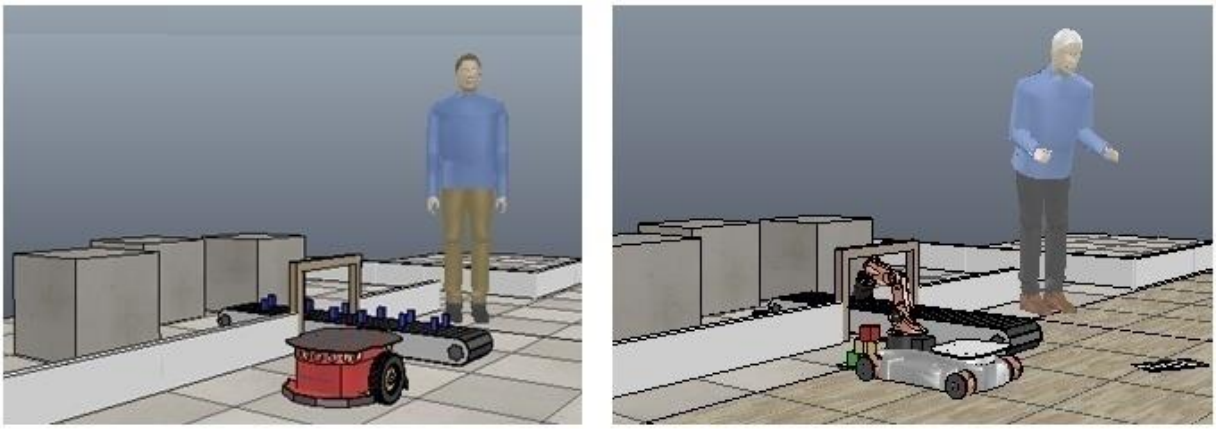


Figura 8.4: Depósito de peças (a) robô *Pioneer* acessando o depósito e (b) robô *YouBot* buscando uma nova peça

Na figura 8.4(a) é apresentada uma situação em que o robô *Pioneer* está acessando o depósito de peças. Para esta plataforma é necessário que um humano recarregue o robô com peças. Um tempo de espera de 5 segundos é dado antes que o robô siga para os pontos de objetivo.

Na figura 8.4(b) é apresentada uma outra situação em que o robô *YouBot* está acessando a célula para buscar uma nova peça. Neste caso é dispensável o auxílio de um humano.

A característica do robô *YouBot* andar de lado é possível graças à inversão do sinal dos motores direito e esquerdo. Porém este conjunto de rodas implica em baixa velocidade de locomoção do robô. Esta característica de mobilidade contribui com grande potencial ao braço robótico quando é necessário pegar uma peça, pois diminui o grau de deslocamento do braço.

Para a validação da navegação entre os pontos de objetivo descrito na seção anterior foi utilizada a plataforma *Pioneer.* Para este caso não foi feito o carregamento de peças de maneira automática .

O robô *Youbot* mostrou ser eficiente para o carregamento de peças utilizando o seu braço robótico. O robô híbrido proposto foi desenvolvido com o objetivo de corrigir as deficiências apresentadas nas plataformas *YouBot e Pioneer*. Porém a limitação física da plataforma *Pioneer* não permitiu o uso do braço robótico. Para cada caso um robô diferente foi utilizado de acordo com seu potencial para a validação das funções do controlador.

O robô *YouBot* não recebeu a implementação do controlador neurogenético, apenas o controle de ponto a ponto. Este ponto negativo é consequente das deficiências encontradas nesta plataforma e que foram apresentadas neste capítulo.

Com os testes feitos e apresentados neste capítulo foi possível validar as funções do robô *Pioneer* utilizado neste trabalho e também potencializar o uso de outros tipos de robôs.

O maior objetivo para esta análise de comportamento das funções robóticas aplicadas aos robôs apresentados nesta seção foi mostrar que mesmo sendo escolhida a plataforma *Pioneer* para o desenvolvimento deste trabalho é possível utilizar outros robôs para desempenhar as funções propostas neste trabalho. Foi apresentado também as vantagens e desvantagens para as 3 variações de robôs em cada caso.

Para o robô *Pioneer* fica a proposta de implementação de um braço robótico em sua plataforma, já que o robô híbrido não teve sucesso. O braço deve ser mais leve e menor, já que a limitação de carga foi ultrapassada com o braço robótico utilizado pela fabricante *KUKA robotics*.

8.3. Sistemática adotada em prol da análise e coleta de dados

Nesta seção é feita uma apresentação da sistemática que foi adotada para a análise e coleta de dados do controlador neurogenético e seu sistema de sensoriamento, desenvolvidos neste trabalho. Os resultados foram obtidos por meio de simulações computacionais nos *softwares* V-REP e *Scilab*. Para a análise de tempo de cada algoritmo foi utilizado o algoritmo de contagem de tempo apresentado no capítulo 9.

Uma síntese de todos os experimentos é apresentada nesta seção, cujos resultados serão apresentados e discutidos no capítulo 9.

Na tabela 8.1 é apresentada uma síntese dos experimentos realizados para as simulações, em prol, a análise e coleta de dados do controlador neurogenético e seu sistema de sensoriamento. Esta tabela deve facilitar a compreensão do raciocínio que foi utilizado nas simulações para se chegar aos resultados e conclusões apresentados nos capítulos 9 e 10, respectivamente.

Lembrando que uma análise qualitativa do controlador neurogenético e seu sistema de sensoriamento foi apresentado no capítulo 7. Esta análise foi focada em avaliar o comportamento do algoritmo neurogenético em diversos casos que o controlador robótico desenvolvido neste trabalho deve encontrar durante seu funcionamento.

Uma breve descrição de cada um dos sistemas que foram apresentados (tabela 8.1), é feita a seguir.

Tabela 8.1: Sistemática utilizada para a realização das simulações do controlador neurogenético

Síntese dos experimentos realizados				
	Sistema	Tipo de análise	Número de iterações	Tipo de resultado
1	RNA	tempo	100	Tempo (ms)
2	AG	tempo	100	Tempo (ms)
3	Sensoriamento por visão e RNA	obstáculos falso - negativos e falso - positivos	100	Obstáculos reconhecidos (%)
4	Sensoriamento ultrassônico	detecção de obstáculos	100	Obstáculos detectados (%)

1) - O primeiro experimento realizado está ligado à análise de tempo do algoritmo neural em conjunto com seu banco de conhecimento. Esta análise foi feita por meio do algoritmo de contagem de tempo. Os testes foram feitos para 100 iterações do algoritmo neural. O resultado é dado em milissegundos.

2) - O segundo experimento realizado está ligado à análise de tempo do AG. Esta análise foi feita por meio do algoritmo de contagem de tempo. Os testes foram feitos para 100 iterações do AG. O resultado é dado em milissegundos.

3) - O terceiro experimento realizado está ligado à análise de obstáculos falso -positivos e falso - negativos. Este experimento foi feito para 100 amostras de obstáculos no ambiente modelado. O objetivo deste experimento é avaliar a capacidade de reconhecimento de obstáculos do sistema de sensoriamento por câmera em conjunto com a RNA utilizada. O resultado deste experimento é dado em porcentagem de obstáculos falso - positivos e falso - negativos.

4) - O quarto experimento realizado está ligado à análise de obstáculos detectados por meio do sistema de sensoriamento ultrassônico. Este experimento também foi feito para 100 amostras de obstáculos no ambiente modelado. O objetivo deste experimento é avaliar a capacidade de detecção de obstáculos encontrados na rota de navegação do robô. O resultado deste experimento é dado em porcentagem (%) de obstáculos detectados e não detectados na rota de navegação do robô. Lembrando que um obstáculo não detectado por este sensoriamento deve gerar uma colisão.

Para uma melhor compreensão sobre a análise comportamental do sistema de visão robótico utilizado em conjunto com uma RNA neste trabalho, duas situações devem ser compreendidas, sendo estas:

- **Obstáculos falso - positivo:** obstáculos que não foram cadastrados no banco de conhecimento da RNA, porém, por uma imprecisão do sensoriamento e/ou treinamento da RNA, o algoritmo neural o declara como um obstáculo reconhecido. Isso deve gerar um problema, como por exemplo, a incapacidade de desvio de rota, assim, sendo necessária uma nova solução de desvio de rota.

- **Obstáculos falso - negativo:** obstáculos que foram cadastrados no banco de conhecimento da RNA, porém, por uma imprecisão do sensoriamento e/ou treinamento da RNA, o algoritmo neural o declara como um obstáculo desconhecido. Neste caso, o algoritmo genético deve gerar uma solução, assim, gerando um maior custo de tempo e computacional para o desvio da rota.

O sistema de controle robótico deve ter um menor rendimento, se obstáculos falso - positivos e/ou falso negativos forem detectados com frequência, pois, o AG, deve gerar uma solução para o obstáculo nestes casos, assim, demorando mais e gerando soluções de menor potencial para o caso de otimização de rotas auxiliares, se comparado ao algoritmo neural. As soluções de desvio de obstáculos geradas pelo algoritmo neural são de maior potencial pelo fato de que suas rotinas de desvio são fixas, assim, evitando uma rota auxiliar maior que a necessária.

Para a solução do problema de obstáculos falso - positivos e/ou falso negativos, uma melhor otimização do sistema de visão e um melhor treinamento da RNA é exigido.

Capítulo 9

Resultados

Neste capítulo são apresentados os resultados quantitativos do algoritmo neurogenético que foram obtidos com base na análise referente ao custo computacional do AG e da RNA de maneira independente. Esta avaliação tem como objetivo analisar o custo de tempo dos algoritmos pertencentes ao controlador neurogenético para gerar uma nova solução de desvio para obstáculos.

A análise de tempo feita nesta seção será de grande importância para a comparação dos códigos e indicar em quais situações cada algoritmo apresenta melhor comportamento e viabilidade de aplicação em diversos ambientes dinâmicos.

É necessário ressaltar também que a análise em questão foi baseada na modelagem e simulação apresentadas no capítulo 7, onde se apresentou o comportamento de cada um dos algoritmos.

Para a contagem de tempo do algoritmo foi utilizada a biblioteca *windows.h,* biblioteca nativa da maior parte dos compiladores *C/C++.* O seguinte código foi utilizado para analisar o tempo e realizar em sua saída uma medição em milissegundos.

```
#include <stdio.h>
#include <windows.h>

//Retorno é em milisegundos!

int main() {
    int i, j;
    int inicio, final, tmili;

    inicio = GetTickCount();
    printf("tempo decorrido: %d\n", inicio);

    // código cujo tempo de execução deverá ser medido.

    for (j = 0; j < 10; j ++)
        for (i = 0; i < 1000000000; i ++);

    final = GetTickCount();
    printf("tempo fim: %d\n", final);
    tmili = final - inicio;

    printf("tempo decorrido: %d\n", tmili);
    return 0;
}
```

O cálculo do tempo é feito a partir da diferença das variáveis de tempo:*final = GetTickCount( );* e tempo inicial *inicio = GetTickCount( );* O trecho onde está a indicação de código em execução é onde foi inserido o código para o cálculo do tempo.

O *hardware* do computador utilizado para medir o custo de tempo de um algoritmo deve ser levado em consideração neste caso. O computador utilizado em todos os testes deste trabalho manteve a seguinte configuração apresentada na tabela 9.1 para todos os algoritmos:

Tabela 9.1: Configurações de *hardware* do computador utilizado

Configurações de *hardware* do computador utilizado	
Processador	Intel(R) Core(TM) i7-5500U CPU @2.40GHz
Memória (RAM)	8 GB
Sistema Operacional	Windows 8.1 (64 bits)

Para maior precisão, todos os testes para esta análise de custo computacional foram feitos seguindo o mesmo procedimento para todos os algoritmos pertencentes ao controlador neurogenético. O algoritmo de base para o cálculo de tempo também foi o mesmo utilizado em todos os casos.

9.1. Análise de custo do AG

O AG manteve o comportamento esperado para as simulações feitas nos ambientes dinâmicos utilizados neste trabalho, e teve um custo maior que a solução gerada pela rotina da RNA, já que o algoritmo genético deste trabalho é independente de qualquer conhecimento do ambiente *a priori.*

A característica fundamental de um AG é sua capacidade de se adaptar a novos ambientes sem grandes problemas, pois suas soluções heurísticas devem trabalhar sobre regiões de interesse bem definidas. Sendo assim, este algoritmo teve sua função aplicada a casos que o robô encontra um obstáculo que precisa de uma solução inédita e nenhum conhecimento prévio existe para o auxílio desta solução.

Para o desenvolvimento deste trabalho, a região de interesse a fim de gerar soluções foi definida em 50 metros quadrados. Esta região de interesse foi definida a partir do conhecimento do tamanho do ambiente em que o robô deve trabalhar.

Explicando sucintamente, o AG apresentou um custo de tempo superior à solução da RNA e seu algoritmo de solução cadastrado em um banco de conhecimento.

Toda vez que um obstáculo conhecido é encontrado e a solução é realizada pela RNA, o custo de tempo se torna muito menor que uma solução exclusiva gerada pelo AG para o problema.

Se a RNA não fosse aplicada para as soluções de desvio de rota, toda vez que o mesmo obstáculo fosse encontrado, uma nova solução teria que ser gerada pelo AG para o mesmo tipo de problema, o que aumentaria consideravelmente o tempo para encontrar uma solução ótima.

O ambiente dinâmico 1 apresenta uma área de 324 metros quadrados, já o ambiente dinâmico 2 possui uma área de 530 metros quadrados. Para gerar a população de indivíduos apenas 50 metros quadrados em volta do centro do robô são de interesse para o AG.

O custo máximo de tempo do AG para uma nova solução quando o robô encontra um obstáculo e não o reconhece é de 30 milissegundos.

Este custo de tempo é bastante superior se comparado a uma solução realizada pelo reconhecimento do obstáculo pela RNA e direcionado a uma rotina pré-definida para o desvio de rota.

Porém, quando o robô não encontra mais obstáculos conhecidos no ambiente que está trabalhando é necessário um novo treinamento da RNA, pois certamente o ambiente sofreu alterações e, o AG está trabalhando no seu nível máximo para encontrar soluções. O que não é bem-vindo para o algoritmo neurogenético, pois é o pior caso deste algoritmo. Uma melhor descrição deste problema foi apresentada no capítulo 3.

Caso não seja feito um novo treinamento da RNA, o algoritmo neurogenético perde sua eficiência, no entanto, caso não seja possível fazer o treinamento da rede, o AG "puro" seria mais eficiente, pois a RNA não treinada deve apenas aumentar o custo computacional das soluções.

Na figura 9.1 pode ser observado o gráfico gerado a partir da análise do número de iterações e o tempo do AG. Este algoritmo mostra o mesmo nível de desempenho para diferentes tipos de obstáculos, já que toda solução gerada por ele é inédita.

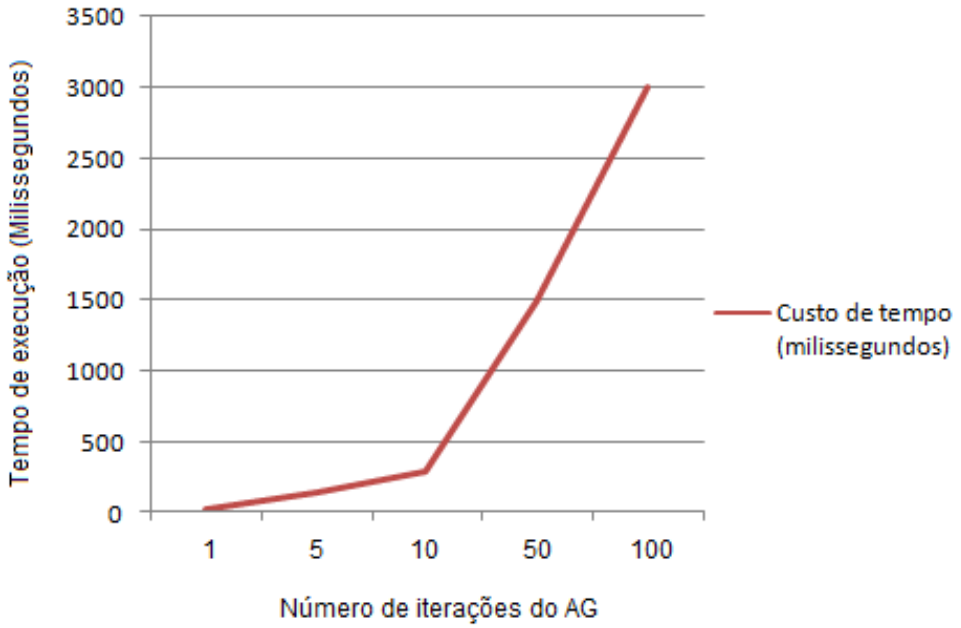


Figura 9.1 : Gráfico de custo de tempo para *n* iterações do AG

Os ajustes necessários para o AG estão ligados apenas às seguintes variáveis: taxa de mutação, região de interesse, na qual devem ser gerados os indivíduos e a solução ótima global. Os ajustes são altamente dependentes do ambiente em que o robô deve realizar sua função. Estes ajustes foram melhores explicados no capítulo 3.

9.2. Análise de custo da RNA e suas rotinas de solução

A RNA aplicada neste trabalho utiliza dois algoritmos que são a base da RNA *perceptron.* O primeiro algoritmo é utilizado para o treinamento da RNA, ou seja, só deve ser utilizado quando os obstáculos de maior frequência forem apresentados para a RNA como padrões a serem reconhecidos.

Os valores de custo de tempo apresentados nesta seção foram obtidos por meio do algoritmo de contagem de tempo de execução que foi aplicado em cada um dos algoritmos analisados, mantendo sempre a mesma configuração de *hardware* (tabela 9.1). Os valores de custo de tempo obtidos foram calculados em milissegundos.

O algoritmo de treinamento tem um alto custo de tempo, custo máximo de 1.8 segundos.

Porém ele só deve ser utilizado quando a rede for treinada, ou seja, quando o robô for colocado em um novo ambiente dinâmico para realizar o seu trabalho, ou quando o ambiente atual recebe novos objetos. Então deve ser desconsiderado este tempo quando o robô estiver realizando efetivamente sua função no seu ambiente de trabalho.

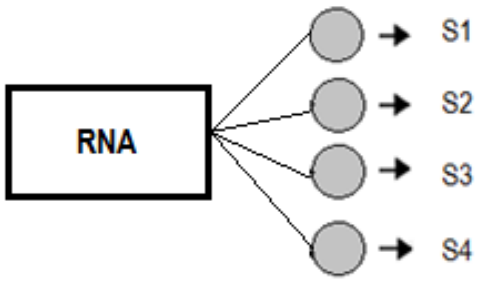
O segundo algoritmo é utilizado para realizar as operações efetivas de reconhecimento de obstáculos. Este algoritmo utiliza os pesos gerados pelo algoritmo de treinamento, pesos que não devem mudar durante a execução do algoritmo operacional da RNA, já que esta rede não deve reaprender de maneira automática. Uma melhor descrição deste problema pode ser vista no capítulo 3 deste trabalho.

O custo máximo de tempo do algoritmo operacional da RNA tem um tempo de 10 milissegundos

Porém não basta um obstáculo ser reconhecido, a saída da RNA deve ser ligada a uma solução pré-definida no banco de conhecimento do algoritmo neurogenético. Assim sendo necessário a soma do tempo deste algoritmo de solução pré-definida para que se possa ser comparada a uma solução gerada pelo AG. Espera-se que sempre a soma de custo deste dois algoritmos seja menor que a do AG, independentemente do número de obstáculos cadastrados.

A rotina de solução tem um custo máximo de tempo de 4 milissegundos.

Este custo é bastante pequeno pelo fato que o único trabalho deste algoritmo é utilizar um banco de conhecimento com dados do tipo: ângulo e distâncias e gerar uma saída para o robô desviar do obstáculo. O algoritmo utiliza como base as distâncias DLO e DLA que são a base para o cálculo do *fitness* de cada indivíduo gerado pelo AG. Uma melhor descrição deste algoritmo pode ser vista no capítulo 3 deste trabalho.



Reconhecimento da RNA	
ENTRADAS	**SAÍDAS**
NÃO RECONHECIDO	[1 -1 -1 -1]
HUMANO	[-1 1 -1 -1]
CAIXA	[-1 -1 1 -1]
EMPILHADEIRA	[-1 -1 -1 1]

Figura 9.2: Representação dos valores de saída da RNA

Na figura 9.2 são apresentadas as saídas do algoritmo operacional da RNA onde os padrões de cor e tamanho foram apresentados nas entradas da RNA para o reconhecimento de obstáculos que foram cadastrados no banco de conhecimento. Para cada ambiente dinâmico real um conjunto de obstáculos foi cadastrado no banco de conhecimento.

Lembrando que a saída do algoritmo de treinamento deve gerar os pesos ajustados das sinapses e servir como entrada para o algoritmo operacional. O algoritmo operacional deve ser utilizado quando o robô estiver em funcionamento em um dos ambientes modelados neste trabalho e encontrar um obstáculo.

A saída do algoritmo operacional deve informar qual o obstáculo que foi encontrado, para o exemplo da figura 9.2: empilhadeira, humano ou caixa. Caso contrário deve retornar a informação de obstáculo desconhecido e, se for o caso, a solução será encontrada pelo AG. Estas informações são apresentadas nas 4 saídas da RNA, onde a saída do tipo degrau bipolar, ou sinal, como é mais conhecida, deve informar com um valor positivo qual obstáculo foi reconhecido ou informar que o obstáculo encontrado é desconhecido, assim, ativando o AG. Como exemplo, o vetor de saída [-1 1 -1 -1] informa que o obstáculo humano foi encontrando, pois, apenas a segunda posição do vetor é positiva e, esta, representa o humano. O funcionamento de reconhecimento da RNA em relação às suas entradas foi explicado no capítulo 7.

No ambiente dinâmico 1, o ambiente industrial didático, foram cadastrados no banco de conhecimento da RNA três obstáculos de maior frequência, sendo estes: Os alunos (humanos); as empilhadeiras e os carros de transporte manual.

No ambiente dinâmico 2, o ambiente industrial real, foram cadastrados na RNA três obstáculos de maior frequência, sendo estes: Os humanos; as empilhadeiras e as caixas utilizadas na linha de produção. O terceiro obstáculo aqui utilizado não segue a lógica de níveis de problemática para que uma diversificação na modelagem do ambiente seja feita se comparada ao ambiente 1.

Estas escolhas foram baseadas nos dados coletados nos ambientes dinâmicos reais. O critério de escolha foi baseado no grau de movimentação (GM) e a taxa de ocupação (TO) destes obstáculos na rota do robô. Uma melhor descrição destes obstáculos pode ser vista no capítulo 5.

O tempo máximo gerado para o treinamento da RNA usando os três obstáculos em ambos os ambientes se manteve em 1.8 segundos.

Este custo de tempo não deve interferir no rendimento do algoritmo neurogenético em funcionamento, pois deve ser feito o aprendizado apenas em casos de mudança de características do ambiente e com a suspensão das funções do robô. Em outras palavras o robô deve estar parado e consequentemente livre do seu trabalho.

O tempo máximo gerado para a operação da RNA em tempo real usando três obstáculos também se manteve em 10 milissegundos.

O custo máximo de tempo da rotina de solução se manteve em 4 milissegundos para todos os obstáculos cadastrados, pois as alterações na rotina são apenas feitas nas variáveis de deslocamento.

O custo máximo de tempo do algoritmo operacional é consideravelmente menor que o exigido pelo AG. Porém o algoritmo operacional da RNA não gera efetivamente uma solução de desvio, apenas direciona suas saídas a um banco de rotinas para desvio de obstáculos. Então o custo de tempo do algoritmo operacional e o custo de tempo da rotina de solução foram somados gerando o tempo máximo de: 10 ms (algoritmo operacional) + 4 ms (rotina de solução) = 14 milissegundos.

Na figura 9.3 pode ser observado o gráfico gerado a partir da análise do número de iterações e o tempo da RNA e suas rotinas de soluções para obstáculos de maior frequência. Este algoritmo mostra o mesmo nível de desempenho para diferentes tipos de obstáculos.

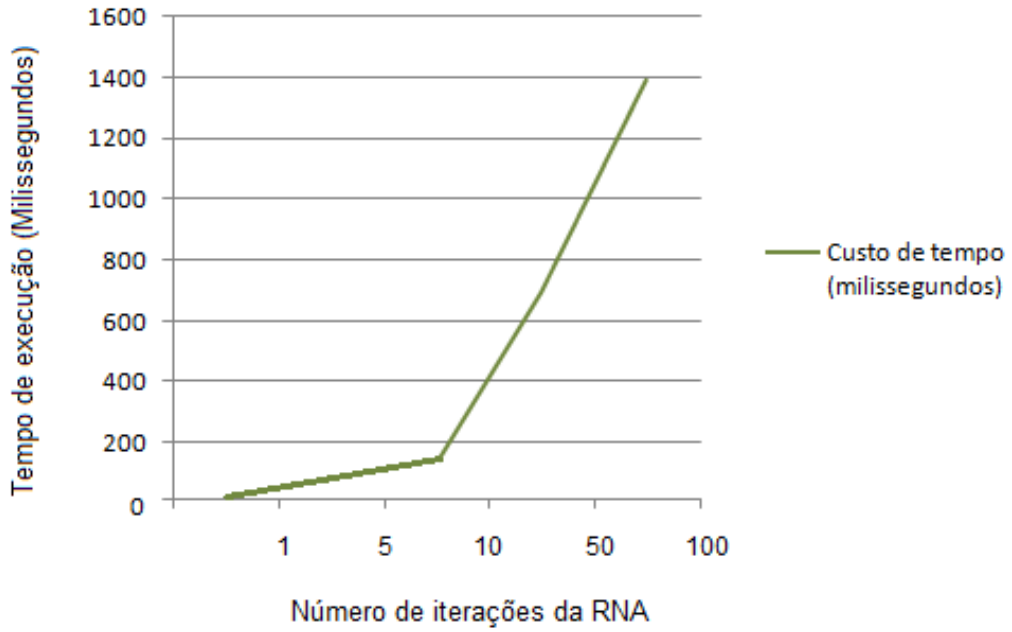


Figura 9.3: Gráfico de custo de tempo para *n* iterações da RNA

O motivo para o custo da RNA combinada com suas rotinas de solução de obstáculos frequentes permanecer constante é que a RNA tem sempre as mesmas variáveis de entrada (cor e tamanho) e as rotinas de solução sempre devem alterar apenas os deslocamentos D1 e D2. Sendo, assim, independentemente do tipo de obstáculo conhecido a solução terá o mesmo custo. Lembrando que o custo apresentado no gráfico da figura 9.3 é referente apenas ao tempo de execução da RNA operacional e suas rotinas de soluções.

Os ajustes que podem ser necessários na aplicação da RNA estão ligados às seguintes alterações: (a) número de saídas caso necessite cadastrar mais obstáculos no banco de conhecimento e (b) número de entradas, caso seja necessário extrair novas características (além de cor e tamanho) dos obstáculos por meio do sensoriamento e

enviá-las à rede. Os ajustes são altamente dependentes do ambiente em que o robô deve realizar sua função. Estes ajustes foram melhores explicados no capítulo 3.

No conjunto de tabelas 9.2 são apresentados os dados de treinamento da RNA. Estes dados são os padrões que foram apresentados a rede *perceptron* para o seu treinamento. Estes dados tiveram como base as informações coletadas nos ambientes dinâmicos reais apresentados no capítulo 4.

Tabela 9.2: Conjunto de tabelas para treinamento da RNA (a) entradas para cor; (b) entradas para tamanho (lateral 1); (c) entradas para tamanho (lateral 2) e (d) saídas esperadas

Entradas RNA - [COR RGB]				
		CÓDIGO RGB		
OBSTÁCULO	COR	Vermelho	Verde	Azul
HUMANO	AZUL	0	0	255
CAIXA	MARROM	102	51	0
EMPILHADEIRA	AMARELO	255	255	0

(a)

Entradas RNA - [TAMANHO] - LATERAL 1	
OBSTÁCULO	TAMANHO
HUMANO	40cm
CAIXA	100cm
EMPILHADEIRA	350cm

(b)

Entradas RNA - [TAMANHO] - LATERAL 2	
OBSTÁCULO	TAMANHO
HUMANO	30cm
CAIXA	120cm
EMPILHADEIRA	200cm

(c)

SAÍDAS - [RECONHECIMENTO] - COR/TAMANHO	
ENTRADAS	SAÍDAS
NÃO RECONHECIDO	[1 -1 -1 -1]
HUMANO	[-1 1 -1 -1]
CAIXA	[-1 -1 1 -1]
EMPILHADEIRA	[-1 -1 -1 1]

(d)

Para a variável cor foi utilizado o código RGB da imagem detectada pela câmera. Na figura 9.4 é apresentada a codificação de cores RGB para a cor azul. Já para o tamanho foi utilizado a unidade de centímetros, que é estimada pela contagem de *pixels* na horizontal e vertical. A relação de contagem de *pixels* e tamanho em centímetros é dependente da resolução e distância de captura da câmera. Esta problemática foi melhor explicada no capítulo 4.

A saída da RNA gera um vetor contendo quatro posições. O único valor positivo do vetor representa o tipo de obstáculo reconhecido ou retorna a resposta de obstáculo desconhecido. O vetor com o valor saída = [-1 -1 -1 1] representa o reconhecimento do obstáculo empilhadeira.

Para o treinamento da RNA foram apresentadas quatro variações da tonalidade de cor de cada tipo de obstáculo para que mudanças na cor do obstáculo possam ser reconhecidas pela RNA operacional.

Variações de tamanho também foram feitas mantendo o mesmo nível de variação feito para as cores.

Para os ambientes utilizados neste trabalho, as variáveis de cor e tamanho foram eficientes sem a necessidade de outras variáveis.

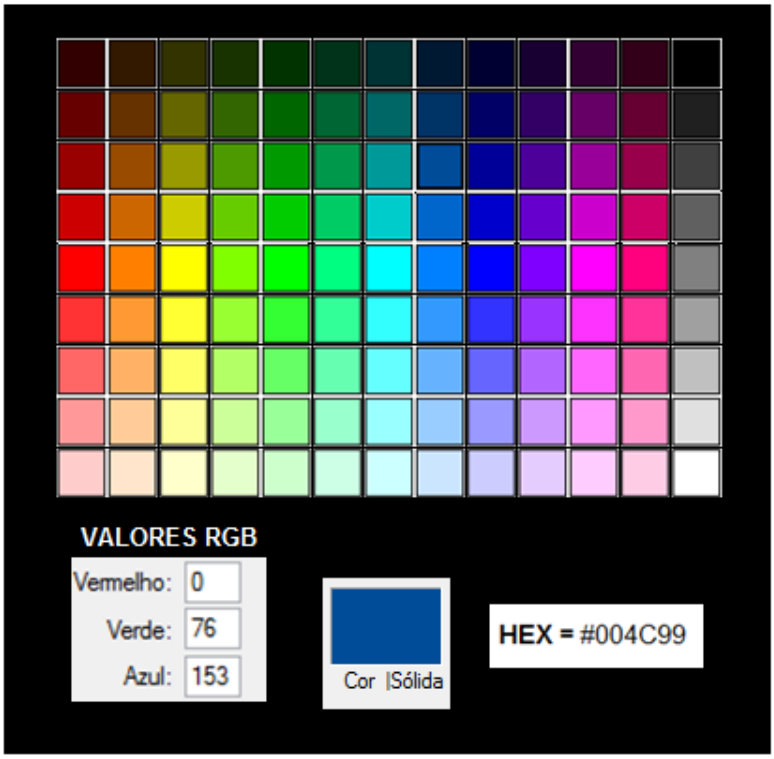


Figura 9.4: Tabela de cores para o padrão RGB

O uso de três obstáculos de maior frequência foi decidido a partir da análise dos ambientes reais, e não exigiu o cadastro de novos obstáculos no banco de conhecimento, pois o AG solucionou estes obstáculos de menor frequência sem grandes problemas.

9.3. Síntese dos resultados apresentados

Nesta seção é apresentada uma síntese dos resultados referentes a cada um dos experimentos apresentados no capítulo 8. Estes resultados foram analisados e discutidos nas seções anteriores deste capítulo. O objetivo desta síntese, é apresentar para o leitor os resultados obtidos neste trabalho de maneira sistêmica e simplificada, assim, facilitando o entendimento das conclusões apresentadas nas próximas seções.

Os resultados apresentados nesta seção foram obtidos por meio das simulações realizadas nos 3 ambientes dinâmicos modelados para testes do controlador robótico, assim, apresentando a média dos resultados obtidos para os 3 ambientes em questão.

Na tabela 9.3 é apresentada a síntese dos resultados obtidos por meio de simulações do controlador neurogenético e seu sistema de sensoriamento. Primeiramente é feita a apresentação dos resultados obtidos para o custo de tempo de cada um dos algoritmos pertencentes ao controlador neurogenético.

Para a apresentação destes resultados, cada algoritmo passou por algumas variações na quantidade de iterações, sendo estas, 1 iteração, 100 iterações e 1000 iterações. Cada iteração do algoritmo neurogenético representa a tentativa do controlador robótico em solucionar um obstáculo encontrado na rota do robô.

Tabela 9.3: Síntese dos resultados obtidos

Síntese dos resultados obtidos por meio de simulações				
Custo de tempo para cada algoritmo do controlador híbrido				
Sistema		Custo de tempo (ms)		
		1 iteração	100 iterações	1000 iterações
RNA		10	1000	10000
Rotinas de desvio		4	400	4000
AG		30	3000	30000
Custo de tempo para o controlador híbrido				
	Sistema	Tipo de análise	Número de iterações	Resultado (ms)
1	RNA + Rotina	tempo	100	1400
2	AG	tempo	100	3000
Resultados do sistema de sensoriamento robótico				
				Resultado (%)
3	Sensoriamento por visão e RNA	Obstáculos falso - positivos	100	4
4	Sensoriamento por visão e RNA	Obstáculos falso - negativos	100	21
5	Sensoriamento por visão e RNA	Obstáculos reconhecidos	100	75
6	Sensoriamento ultrassônico	Detecção de obstáculos	100	100

A segunda parte dos testes desenvolvidos, está ligada ao sensoriamento do robô, sendo assim, foram testados os dois principais sensores utilizados para que o controlador consiga gerar soluções de desvio, assim, evitando colisões com obstáculos que surgirem na rota de navegação do robô.

Os dois sensores que passaram por uma análise experimental, por meio de simulações, foram o sensoriamento por visão em conjunto com a RNA e o sensoriamento ultrassônico.

O sensoriamento por visão em conjunto com o algoritmo neural apresentou bons resultados, tendo um reconhecimento de obstáculos igual a 75% de uma amostra de testes igual a 100 obstáculos (figura 9.5). Os outros 25% estão ligados aos obstáculos falso - positivos e falso negativos.

O sensoriamento por visão em conjunto com o algoritmo neural apresentou 4% de obstáculos falso - positivos, ou seja, obstáculos que não são cadastrados no banco de conhecimento do sistema neural, porém, são declarados reconhecidos pela RNA.

O sensoriamento por visão em conjunto com o algoritmo neural apresentou 21% de obstáculos falso - negativos, ou seja, obstáculos que foram cadastrados no banco de conhecimento do sistema neural, porém, são declarados desconhecidos pela RNA.

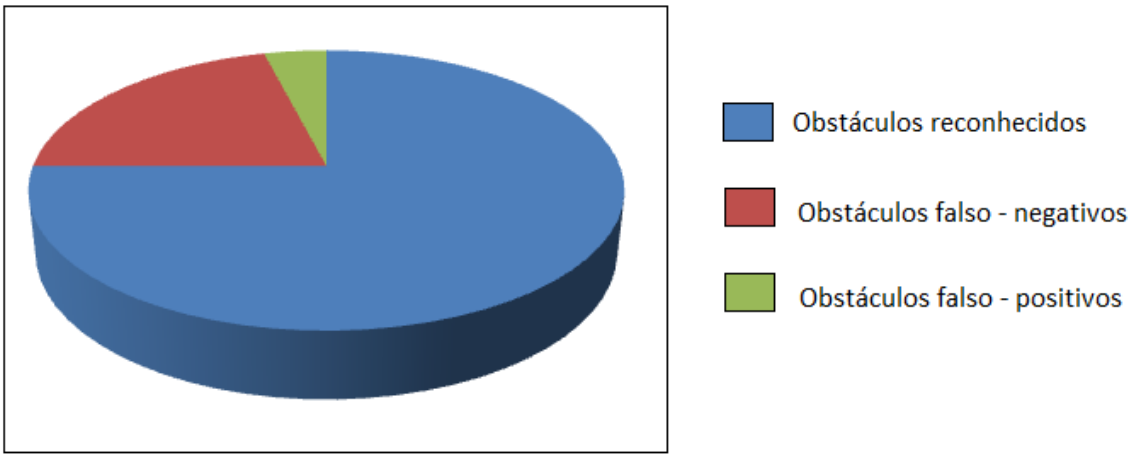


Figura 9.5: Gráfico referente ao reconhecimento de obstáculos

Os resultados obtidos pelo sistema de visão em conjunto com o algoritmo neural foram satisfatórios para este trabalho, já que, para um valor mínimo de 39% (tabela 9.5) de obstáculos solucionados por meio da RNA e suas rotinas de solução o controlador neurogenético apresenta resultados melhores que o AG "puro".

O sistema de sensoriamento ultrassônico apresentou ótimos resultados, assim, detectando todos os obstáculos que impediram a rota de navegação do robô.

9.4. Conclusões e resultados sobre as análises comportamentais e de custo computacional do algoritmo neurogenético

Nesta seção, são apresentadas as conclusões sobre as análises feitas por meio do levantamento de dados nas simulações do algoritmo neurogenético.

O objetivo maior foi validado, sendo este a verificação a partir de simulações que o controlador híbrido pode desenvolver suas funções para garantir que o robô atinja seus objetivos sem realizar colisões e otimizando as soluções para reduzir o custo computacional, de tempo e de energia para cada solução.

O controlador apresentou alguns problemas que foram relevantes para a navegação do robô, porém as problemáticas não afetaram o sistema de controle neurogenético a ponto de não atingir o funcionamento esperado.

O robô foi capaz de navegar em seus ambientes modelados neste trabalho apresentando resultados satisfatórios e consequentemente o controlador híbrido foi validado com base nas simulações feitas.

Uma ideia de bastante relevância surgiu durante as simulações. Esta ideia está ligada ao chaveamento entre os algoritmos de RNA e AG. A contribuição teve como objetivo reduzir o custo de tempo do controlador desativando a RNA quando o robô é colocado em um ambiente desconhecido/ e ou o ambiente atual recebe novos objetos que se tornaram obstáculos de grande frequência e a RNA não foi treinada novamente para este novo caso do ambiente. Na tabela 9.4 é apresentada a comparação de custo de tempo entre o algoritmo neurogenético e do AG "puro".

Na tabela 9.4 é apresentado o comportamento do algoritmo neurogenético e do AG "puro" para a solução de 100 obstáculos encontrados durante a navegação do robô nos ambientes modelados neste trabalho. Lembrando que, para um obstáculo desconhecido, deve-se utilizar a RNA junto com o AG para se obter uma solução de desvio, já para solucionar um obstáculo reconhecido é utilizado a RNA mais a rotina de solução correspondente ao obstáculo.

Tabela 9.4: Comparação de custo do algoritmo neurogenético com o AG

Comparação de custo do algoritmo neurogenético com o AG para 100 obstáculos			
Obstáculos não conhecidos (%)	**Obstáculos conhecidos (%)**	**Custo total (ms)**	**Custo total (ms)**
[RNA + AG]	**[RNA + ROTINA]**	**[Neurogenético]**	**[AG]**
0	100	1400	3000
10	90	1660	3000
20	80	1920	3000
30	70	2180	3000
40	60	2440	3000
50	50	2700	3000
60	**40**	**2960**	**3000**
70	30	3220	3000
80	20	3480	3000
90	10	3740	3000
100	0	4000	3000

Na tabela 9.4 pode ser observado que para ser vantajoso o uso do algoritmo neurogenético o controlador deve solucionar no mínimo 40% de obstáculos conhecidos de um total de 100%. Os outros 60% deverão ser os obstáculos tratados pelo AG. O tempo do AG trabalhando de maneira independente permanece constante em 3000 ms, pois ele deve solucionar todos os 100 obstáculos gerando o mesmo custo de tempo. Na figura 9.6 são apresentadas estas informações com maior simplicidade em um gráfico.

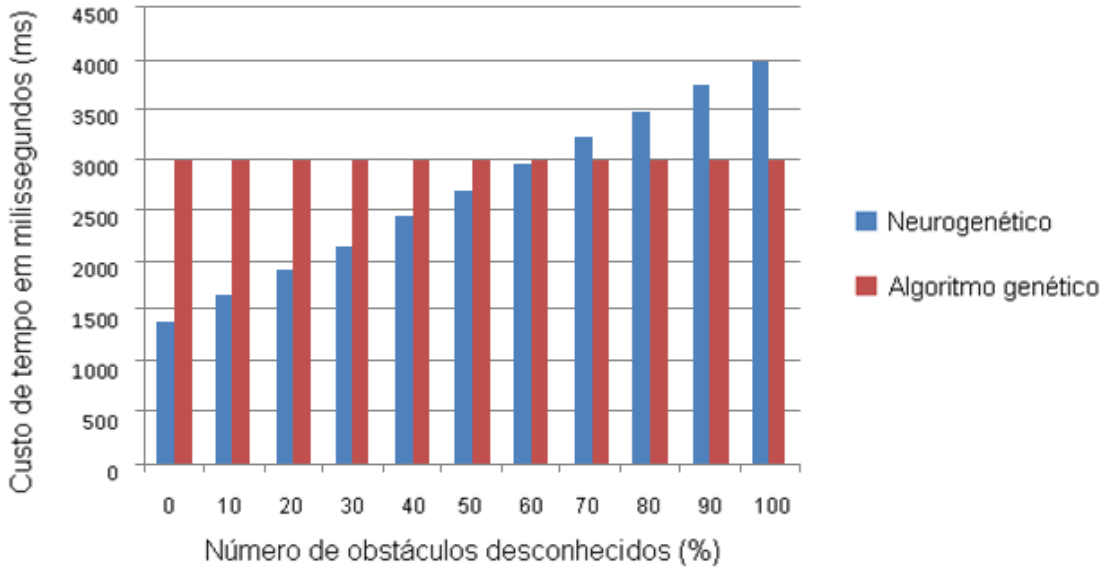


Figura 9.6: Gráfico comparativo de tempo entre os algoritmos. De 0 a 100% de obstáculos desconhecidos

O problema é que todo obstáculo que não é reconhecido pelo algoritmo neurogenético e deve ser solucionado pelo AG teve que ser antes analisado pela RNA, ou seja, foi um custo de tempo adicionado e sem vantagens para este caso. O inverso acontece para quando o algoritmo neurogenético reconhece o obstáculo e tem sua solução pré-definida.

Na tabela 9.5 pode ser observado com maior precisão o ponto em que o algoritmo neurogenético não apresenta mais vantagens sobre o AG trabalhando de maneira independente. Quando o número de obstáculos conhecidos for inferior a 39% não é mais vantajoso o uso da RNA para tratar os obstáculos.

Como consequência neste caso deve-se desativar a RNA e todas as soluções serem geradas pelo AG. Na figura 9.7 são apresentadas estas informações com maior simplicidade em um gráfico.

A solução encontrada para que o controlador continuasse seu funcionamento com o melhor desempenho possível foi desativar a RNA do algoritmo neurogenético quando o seu algoritmo de treinamento não tem condições de ser executado novamente e os obstáculos desconhecidos são iguais ou superiores a 62%.

Tabela 9.5: Comparação de custo do algoritmo neurogenético com o AG (comparação feita para a margem de 40% a 30% de obstáculos conhecidos pela RNA)

Comparação de custo do algoritmo neurogenético para 100 obstáculos			
(comparação feita para a margem de 30% a 40% de obstáculos conhecidos pela RNA)			
Obstáculos não conhecidos (%)	**Obstáculos conhecidos (%)**	**Custo total (ms)**	**Custo total (ms)**
[RNA + AG]	**[RNA + ROTINA]**	**[Neurogenético]**	**[AG]**
60	40	2960	3000
61	**39**	**2986**	**3000**
62	38	3012	3000
63	37	3038	3000
64	36	3064	3000
65	35	3090	3000
66	34	3116	3000
67	33	3142	3000
68	32	3168	3000
69	31	3194	3000
70	30	3220	3000

Os motivos podem estar ligados à reconfiguração do ambiente dinâmico atual, ou mudança de ambiente utilizado. Com isso conseguiu-se que o controlador híbrido apenas funcionasse de maneira vantajosa em todos os casos possíveis.

Para isso foi colocado um contador no número de chamadas às rotinas pré-definidas, pois elas são o resultado de que a RNA reconheceu os obstáculos. Quando estas chamadas são inferiores a 39% (tabela 9.5), a RNA é desativada. Consequentemente só o AG é chamado para solucionar os obstáculos.

O AG deve funcionar de maneira independente até que a RNA seja treinada novamente ou o robô volte a trabalhar no ambiente que estava aplicado a realizar suas funções anteriormente. Em casos que o robô deve trabalhar por um curto prazo no novo ambiente e depois voltar ao seu ambiente treinado, deve-se estudar este caso e avaliar se é compensador todo o processo de treinamento da RNA.

O custo para um novo treinamento da RNA está altamente ligado ao seu tempo. Porém não é só o custo de tempo que está envolvido, uma vez que é preciso também que as rotinas de solução para cada novo tipo de obstáculo de maior frequência sejam alteradas e um estudo seja feito sobre o novo ambiente em que o robô deve atuar, assim o caracterizando para um novo treinamento da rede. Este processo de caracterização deve ser similar ao apresentado no capítulo 5.

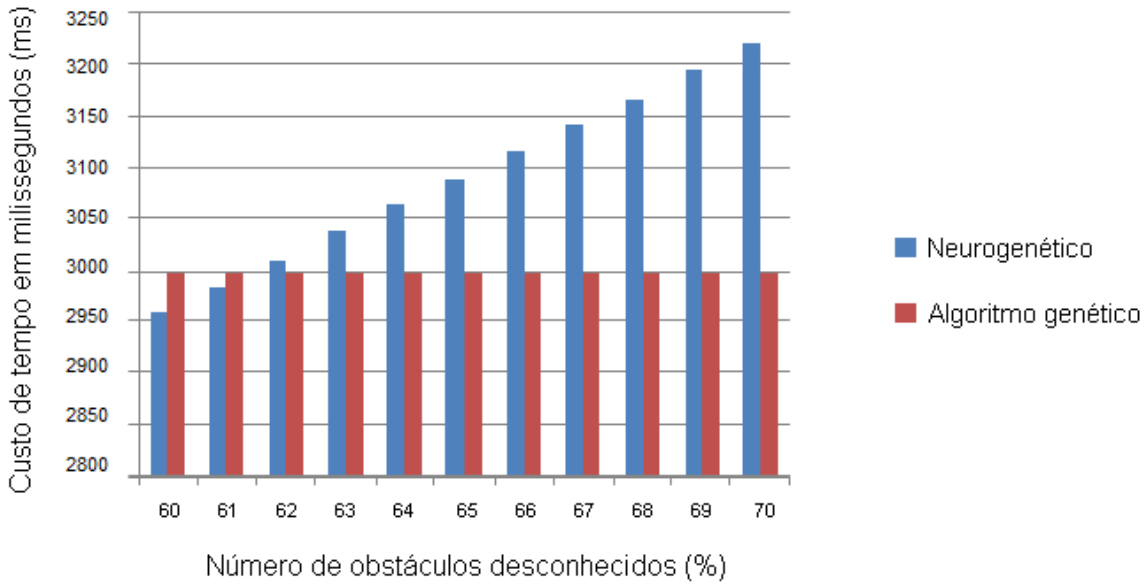


Figura 9.7: Gráfico comparativo de tempo entre os algoritmos. De 60 a 70% de obstáculos desconhecidos

Os ambientes reais descritos no capítulo 5 serviram de base para todas as simulações feitas neste capítulo. Todas as situações coletadas em ambiente real foram abordadas nestas análises. O objetivo para a aplicação desta sistemática foi deixar os ambientes modelados no simulador V-REP o mais próximo da realidade. Foram modeladas três variações de ambientes dinâmicos, todos baseados nos ambientes industriais analisados.

Capítulo 10

Conclusão

O trabalho apresentado nesta dissertação de mestrado está ligado a uma área de pesquisas que, atualmente, é de grande potencial e de grande interesse científico. O potencial explorado nesta área do conhecimento tem contribuição nas diversas aplicações envolvendo robôs móveis autônomos para auxiliar a humanidade em tarefas de risco e nas quais se exige precisão nas funções realizadas.

Nesta linha de pesquisa existem várias técnicas utilizadas para auxiliar o robô em sua navegação, porém na maioria das vezes, são utilizadas de maneira independente. Neste trabalho foi desenvolvido um controlador neurogenético unindo o potencial dos algoritmos bioinspirados em genética e redes neurais.

Alguns trabalhos utilizando as mesmas técnicas bioinspiradas foram estudados e apresentados na revisão bibliográfica dessa dissertação.

Porém, as mesmas técnicas são aplicadas na realização de outras funções no sistema híbrido se comparado ao controlador neurogenético desenvolvido neste trabalho. Consequentemente o controlador apresentado neste trabalho mostrou um comportamento diferenciado se comparado aos controladores neurogenéticos existentes atualmente.

Para as aplicações desenvolvidas na plataforma *Pioneer,* comprovou-se que o controlador neurogenético é eficiente para auxiliar o robô na sua navegação autônoma.

10.1. Resultados apresentados

Os resultados apresentados nesta seção tiveram como base os trabalhos de alguns pesquisadores da área de ciência da computação e engenharias que desenvolveram controladores para sistemas robóticos com vistas para o controle de rotas inteligentes, e mostraram eficiência do Algoritmo Genético (AG) para este tipo de problema a ser tratado, assim como comparações com outros métodos inteligentes (SHI; CUI, 2010);(TUNCER; YILDIRIM, 2012) ; (PURIAN, FAROKHI, 2013) ; (SAMADI, OTHMAN, 2013) ; (SHILTAGH, JALAL, 2013) ; (ABINAYA ; KUMAR, 2014) ; (KHELCHANDRA ; HUANG, 2014) ; (SANTHOSH, 2014) ; (ABBASPOUR; ALPOUR, 2015) ; (FETANAT; HAGHZAD, 2015) ; (PANDA ; CHOUDHURY, 2015).

O principal trabalho utilizado para esta análise de resultados foi o trabalho de Ferreira (2015), que teve como objetivo maior a escolha dos melhores operadores genéticos para o planejamento de rota de robôs móveis autônomos. O Algoritmo Genético (AG) de Ferreira (2015) foi utilizado como base para as comparações de tempo apresentadas

neste trabalho, já que este algoritmo foi otimizado com vistas para a redução de custo de tempo e foi aplicado no controlador neurogenético desenvolvido neste trabalho.

O algoritmo neural contribuiu potencialmente para o controlador híbrido, deixando-o mais preciso e robusto, assim, desenvolvendo sua função de reconhecimento de obstáculos de maior frequência de maneira positiva. Outros pesquisadores também utilizaram algoritmos neurais para auxiliar a navegação de robôs móveis autônomos (BESSA ; BARROSO, 2015) ; (LUO, YANG, 2015) ; (LYRIO ; SANTOS, 2015).

Por outro lado, o Algoritmo Genético (AG) apresentou eficiência para a solução de obstáculos que não são reconhecidos pelo algoritmo neural. Foram apresentados, também, os casos em que o algoritmo neurogenético apresenta vantagens sobre o algoritmo genético "puro".

As soluções geradas pelo algoritmo neural e suas rotinas de soluções pré-definidas mostraram melhores resultados se comparadas as soluções geradas pelo AG. As rotinas de soluções pré-definidas têm o potencial de fazer um desvio preciso, assim, evitando uma rota de desvio com uma distância maior que a necessária, pois o banco de conhecimento que representa cada obstáculo reconhecido fornece *a priori* as distâncias exatas para um bom desvio.

O AG deve ser vantajoso apenas quando a RNA não estiver bem treinada por motivos de troca de ambiente e/ou inserção de novos obstáculos no ambiente atual. Pois, o AG tem um maior custo de tempo e suas soluções não são tão precisas quanto as soluções geradas pelas rotinas pré-definidas no banco de conhecimento da RNA.

Um outro ponto que merece atenção está relacionado ao treinamento da RNA que necessita de um novo estudo a cada troca de ambiente dinâmico para que as características dos novos obstáculos sejam aplicadas ao banco de conhecimento do controlador de maneira empírica.

Em resumo, o controlador neurogenético apresenta vantagens potenciais se comparado ao AG "puro", principalmente em termos de custo de tempo e custo de energia, já que as soluções geradas pelo algoritmo neural são previamente conhecidas e seus desvios geram os menores caminhos possíveis. Porém para o aproveitamento destas potenciais características, o treinamento da RNA deve ser feito de maneira rigorosa.

As aplicações com os operadores de Visão Computacional foram realizadas no ambiente computacional *Scilab* com vistas para a modelagem e simulação do sistema de visão do robô. O sistema de visão foi implementado efetivamente em linguagem LUA internamente ao simulador V-REP.

As simulações no ambiente *Scilab* apresentaram resultados parciais do sistema de Visão Computacional com o objetivo de servir como ferramenta de análise para o funcionamento de cada algoritmo utilizado para a extração de características dos obstáculos.

O objetivo para a implementação de um sistema de visão simples e de baixo custo computacional neste trabalho está ligado ao processamento em tempo real que é exigido para extrair as características de um obstáculo e enviá-las para o controlador neurogenético encontrar uma solução de desvio para o obstáculo.

Em outras palavras, uma boa solução de desvio é altamente dependente do custo de tempo do sensoriamento, tanto ultrassônico como de visão computacional.

Então, o tempo que o sistema de percepção do robô deve demorar para extrair as características necessárias de cada obstáculo e enviá-las para o controlador de movimento do robô a fim de encontrar uma solução para desviar do obstáculo é de grande importância para quantificar o sistema de soluções do controlador neurogenético.

O reconhecimento de obstáculos cadastrados no banco de conhecimento da RNA se mostrou eficiente na maioria dos casos. Problemas foram apresentados, mas nenhum deles foi relevante ao ponto de neutralizar as vantagens da RNA utilizada no controlador neurogenético.

10.2. Problemas encontrados

A extração de características dos obstáculos foi um processo que gerou problemas de grande potencial para este trabalho.

Primeiramente foi utilizado um sensoriamento óptico para analisar o tamanho dos obstáculos encontrados pelo sensoriamento ultrassônico. Porém, este método apresentou grandes problemas relacionados a medição. Estes problemas foram apresentados no capítulo 3. Por este motivo foi descartado este tipo de sensoriamento para a análise de obstáculos.

Como alternativa para a resolução do problema de medição, foi utilizado um sistema de Visão Computacional para fazer a extração de características dos obstáculos encontrados na rota de navegação do robô.

Outro problema encontrado neste trabalho está relacionado à união de todos estes algoritmos pertencentes ao controlador neurogenético para que trabalhassem de maneira híbrida. O controlador foi implementado externamente ao simulador de robôs em linguagem *C,* utilizando uma aplicação cliente/servidor para a comunicação de dados entre o robô e o seu controlador de movimentação.

A problemática neste sistema de algoritmos trabalhando em conjunto teve ligação com o alto custo de tempo consequente da grande taxa de transmissão de dados dos sensores pela comunicação cliente/servidor ao controlador do robô.

A solução para o problema do alto custo de tempo para o sensoriamento e sua alta taxa de transmissão de dados foi a implementação de um sistema de percepção do robô em

linguagem LUA internamente ao simulador. Consequentemente conseguiu-se uma redução no custo de tempo, pois todo o tratamento dos sensores do robô foi feito internamente ao simulador.

10.3. Sugestões para trabalhos futuros

Uma sugestão para um futuro trabalho seria a implementação do controlador neurogenético em *hardware,* utilizando a linguagem VHDL *(VHSIC* (*Very High Speed Integrated Circuits) Hardware Description Language*) para a programação de um controlador FPGA *(Field Programmable Gate Array)* embarcado na plataforma *Kihon.*

Outro trabalho sugerido seria a tentativa de melhoramento do sistema de Visão Computacional utilizado para extração de características, já que o objetivo principal deste trabalho não foi voltado ao sistema de visão do robô. O sistema de Visão Computacional utilizado neste trabalho foi o mais simples possível e capaz de extrair as características necessárias dos obstáculos encontrados, assim, auxiliando a RNA no reconhecimento de obstáculos de maior frequência.

O sistema de visão aplicado neste trabalho foi eficiente para desenvolver suas funções em um ambiente de simulações com iluminação totalmente controlada, ou seja, o ambiente modelado possui uma iluminação uniforme que facilita para o sistema de visão realizar suas funções de extração de características. Para uma implementação deste sistema de visão para desenvolver suas funções em um ambiente real será necessário tratar este problema relacionado à iluminação, já que em ambientes reais é de extrema dificuldade conseguir uma iluminação controlada.

Uma alternativa para resolver o problema de variação de luminosidade, seria utilizar outro modelo de cores, que não, o RGB, já que neste modelo não é possível controlar a intensidade de luminosidade encontrada em ambientes reais que possuem iluminação não uniforme. Uma outra sugestão seria a implementação do sistema de visão em *hardware*, assim, diminuindo o custo computacional e de tempo em prol do sistema de reconhecimento de obstáculos do robô.

Outra modificação sugerida para este sistema de controle robótico está ligada à troca do sensoriamento de detecção de obstáculos ultrassônico por um sensoriamento óptico, assim, sendo possível detectar obstáculos com uma distância superior, consequentemente, aumentando o campo de visão da câmera utilizada em prol à análise de obstáculos, já que a câmera só deve ser acionada quando um impedimento é detectado na rota atual do robô.

Outra indicação de trabalhos futuros seria a adaptação de um braço robótico no robô *Kihon* para o carregamento de peças em sua plataforma. Este trabalho aumentaria potencialmente o grau de autonomia deste robô.

10.4. Contribuição científica

A principal contribuição científica deste trabalho está ligada ao uso de um algoritmo neurogenético com vistas para o planejamento de rotas de robôs móveis autônomos em ambientes dinâmicos.

Outra contribuição científica deste trabalho está ligada ao uso de um método simplificado e, de baixo custo computacional, para a identificação de objetos, baseado no algoritmo de "binarização".

Referências

ABBASPOUR, A. ; ALPOUR, K.; JAFARI, H, Z.; MOOSAVIAN, S, A. Optimal formation and control of cooperative wheeled mobile robots. Comptes Rendus Mecanique, p. 307-321, (2015).

ABINAYA, S. ; KUMAR, V, H. ; TAMILSELVI, D. ; Hybrid Genetic Algorithm Approach for Mobile Robot Path Planning. Advances in Natural and Applied Sciences, 8(17) Special, p. 41-47, (2014).

ALLAIRE F.C.J., M.; LABONTÉ, G.; FUSINA, G. FPGA implementation of genetic algorithm for uav real-time path planning. Journal of Intelligent and Robotic Systems, v. 54, n. 54, p. 495–510,(2009). ISSN 0921-0296. Disponível em: <http://dx.doi.org/ 10.1007/s10846-008-9276-8>. Acesso em: 20 Outubro 2015.

ALSINA, P. CINEMÁTICA: Representação de Posição e Orientação. 2008. Disponível em: <http://www.dca.ufrn.br/~pablo/FTP/robotica/cap2.pdf> - Acesso em: 1 nov. 2015.

ASIMOV, I. Eu, "Robô. 1.ed. São Paulo: Aleph, 2014.

AYROSA, P. P.; BRUNETTO, Maria Angélica de Oliveira Camargo; Attrot, Wesley; MANTOVANI, R. G. ; PERIOTTO, Alvaro Jose. Utilização de redes neurais de spikes para tarefas de navegação de agentes robóticos autônomos. 2011. Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina.

BÄCK, T. ; HOFFMEISTER, F. Extended selection mechanisms in genetic algorithms. In: R. K. Belew and L. B. Booker, eds., Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kaufmann. 1991.

BARRETO, J. M. Introdução às redes neurais artificiais, In: ESCOLA REGIONAL DE INFORMÁTICA DA SBC REGIONAL SUL, 5, 1997, Florianópolis. Anais. Florianópolis, Maringá, 1997. Disponível em: <http://www.gsigma.ufsc.br/~popov/aulas/ia/modulo9.pdf>. Acesso em: 10 Outubro 2015.

BECKMANN, M. Algoritmos Genéticos como estratégia de pré-processamento em conjuntos de dados desbalanceados. 2010. 1 v. Dissertação (Mestrado) - Curso de Ciência da Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2010.

BERRI, R.A.; Grassi Jr.,V.; Osorio, F. S. "Simulação de Robôs Móveis e Articulados: Aplicações e Prática". (Capítulo 6). 34a. JAI - Jornada de Atualização de Informática, XXXV Congresso da SBC, Recife - PE, pp.274-322. Julho 2015. Disponível em: <https://www.sites.google.com/site/vrepjai/>. Acesso em: 24 Agosto 2015.

BESSA, A, J. ; BARROSO, NETO, A, R. ; ALEXANDRIA, A, R. Global Location of Mobile Robots Using Artificial Neural Networks in Omnidirectional Images. IEEE LATIN AMERICA TRANSACTIONS, VOL. 13, NO. 10, p. 3405-3414, (2015).

BOTASSOLI, G, T .; ALBERTI, R, A.; FURTADO J, C. COMPUTER SIMULATION FOR THE OPTIMIZATION OF PROCESS QUEUES. Revista GEINTEC, São Cristóvão/SE – 2015 Brasil.p. 2121–2135. Disponível em: <https://www.researchgate.net/publication/281182669_SIMULACAO_COMPUTACIONAL_PARA_OTIMIZACAO_DE_FILAS_EM_PROCESSOS> Acesso em: 08 Agosto 2015.

BUENO, F. Métodos Heurísticos – Teoria e Implementações - Tutorial para os alunos dos cursos de ciências exatas - IFSC/Araranguá, 2009. Disponível em: <https://wiki.ifsc.edu.br/mediawiki/images/b/b7/Tutorial_m%C3%A9todos_heur%C3%ADsticos.pdf>. Acesso em: 5 Novembro 2015.

CABREIRA, T.; DIMURO, G.; AGUIAR, M. de. An evolutionary learning approach for robot path planning with fuzzy obstacle detection and avoidance in a multi-agent environment. In: Social Simulation (BWSS), 2012 Third Brazilian Workshop on. [S.l.: s.n.], (2012). p. 60–67.

CAMPBELL, S.; Chancelier J.-P., Nikoukhah R.. *Modeling and Simulation in Scilab/Scicos*. New York: Springer, 2006. ISBN 978-0-387-27802-5. Disponível em: < http://www.sze.hu/~molnarka/SCILAB/book_SCIALB.pdf> Acesso em: 02 Novembro 2015.

CARVALHO, A. Redes Neurais Artificiais. ICMC - USP- São Carlos (2015). Disponível em: <http://www.icmc.usp.br/~andre/research/neural/>. Acesso em: 01 Dezembro 2015.

CHAVES, J. F. Síntese de estruturas bio-inspiradas baseada em redes de neurônios pulsantes. 2007. 1 v. Dissertação (Mestrado) – Curso de Ciência da Computação, CEFET-MG, Belo Horizonte, 2007.

CMAKE - Software para geração automatizada - Disponível em : https://cmake.org/. (2015).

COPELIA ROBOTICS; V- REP – (Virtual Robot Experimentation Platform) – COPELIA ROBOTICS; Zurich, Switzerland (2015) e seu *download* pode ser feito no seguinte elo: <http://www.coppeliarobotics.com/downloads.html>. Acesso em: 10 Agosto 2015.

CRAIG, J. Introduction to Robotics: Mechanics and Control. Pearson Education, Incorporated, (2005). (Addison-Wesley series in electrical and computer engineering: control engineering). ISBN 9780201543612. Disponível em: <http://books.google.com.br/books?id=MqMeAQAAIAAJ>.

DARPA, (*Defense Advanced Research Projects Agency)* Grand Challengewww.darpa.mil (2015).

DE LA MAZA, M. ; Tidor, B. Boltzmann Weighted Selection Improves Performance of Genetic Algorithms. A.I. Memo 1345, MIT Artificial Intelligence Laboratory. 1991.

FERREIRA, W. Implementação de um sistema de navegação para robôs autônomos baseado em algoritmos genéticos. 2015. 1 v. TCC (Graduação) - Curso de Ciência da Computação, Instituto de Biociências, Letras e Ciências Exatas, São José do Rio Preto, 2015.

FETANAT, S. ; HAGHZAD, S. ; SHOURAKI, S, B. Optimization of Dynamic Mobile Robot Path Planning based on Evolutionary Methods. AI & Robotics (IRANOPEN), p. 1-7, (2015).

GAT, E. Three-layer architectures in: Artificial intelligence and mobile robots: case studies of successful robot systems, Cambridge, MA, USA: MIT Press, 1998, p, 195-210.

GILBERT, E., JOHNSON, D., KEERTHI, S., “A fast procedure for computing the distance between complex objects in three-dimensional space”, Robotics and Automation, IEEE Journal of , v. 4, n. 2, pp. 193 –203, apr 1988.

GOLDBERG, D. Genetic algorithms in search, optimization, and machine learning. [S.l.]: Addison-wesley Reading Menlo Park, (1989).

GOLDBERG, D. E., and Deb, K. 1991. A comparative analysis of selection schemes used in genetic algorithms.In G. Rawlins, Foundations of Genetic Algorithms. Morgan Kaufmann.

GONZALEZ, R. C. and Woods,R.E. – "*Digital image processing*", Reading, Mass - Addison- Wesley 3ª Ed. - 2011.

HATA, Y, A. Mapeamento de ambientes externos utilizando robôs móveis. 2010. Dissertação (Mestrado) - Curso de Pós-Graduação em Ciências Matemáticas e de Computação, ICMC - USP, São Paulo, 2010.

HEINEN. R, OSÓRIO, F. Controle Inteligente do Caminhar de Robôs Móveis Utilizando Algoritmos Genéticos e Redes Neurais Artificiais. 2006. 1 v. Dissertação (Mestrado), Curso de Ciência da Computação, UFRGS, Porto Alegre, 2006.

HOLANDA, Aurélio Buarque de. Dicionário Aurélio da língua portuguesa. Curitiba: Editora Positivo, 2010.

JONES, G. A.; LAYER, D. H.; OSENKOWSKY, T. G. National Association of Broadcasters Engineering Handbook: NAB Engineering Handbook. [S.l.]: Taylor & Francis, 2007.

KDevelop: Compilador para distribuições Unix: software livre. 2015. Disponível em:< https://www.kdevelop.org/> Acesso em: 11 Setembro 2015.

KHELCHANDRA, T. ; HUANG, J.; DEBNATH, S. Path planning of mobile robot with neuro-genetic-fuzzy technique in static environment. International Journal of Hybrid Intelligent Systems 11, p. 71–80, (2014).

KHOURY, G. M.; Saad, M.; Kanaan, H. Y.; Asmar, C. (2004) Fuzzy PID Control of a Five DOF Robot Arm, Journal of Intelligent and Robotic Systems, V. 40, pp. 299-320.

LUA - A linguagem de programação - (PUC - Rio) Pontifícia Universidade Católica do Rio de Janeiro) - 2015. Disponível em: < http://www.lua.org/portugues.html>. Acesso em : 05 Novembro 2015.

LUDWIG JR., O.; MONTGOMERY, D. C. Redes Neurais: fundamentos e aplicações com programas em C. Rio de Janeiro: Ciência Moderna, 2007.

LUO, C. ; YANG, S. ; MO, H. ; LI, X. Safety Aware Robot Coverage Motion Planning with Virtual-obstacle-based Navigation. Proceeding of the 2015 IEEE International Conference on Information and Automation, Lijiang, China,p. 2110-2115, (2015).

LYRIO, L, J. ; SANTOS, T, O. ; BADUE, C. ; SOUZA, A, F. Image-Based Mapping, Global Localization and Position Tracking Using VG-RAM Weightless Neural Networks. 2015 IEEE International Conference on Robotics and Automation (ICRA), Washington State Convention Center, p. 3603-3610, (2015).

MATSUMURA. T. Desenvolvimento de uma plataforma aberta de robô móvel para propósitos gerais. 2014. 1 v. Dissertação (Mestrado), Curso de Ciência da Computação, UNESP, São José do Rio Preto, 2014.

McCULLLOCH, W. S; PITTS, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, New York, n. 9, p. 127-147, 1943.

MENDONÇA, M. ; FINOCCHIO, M. A. ; VALLIM, M. B. R. ; WATANABE, A. Navegação Robótica Autônoma através de sistema Neuro-Fuzzy. In: Simpósio Brasileiro de Automação Inteligente (SBAI), 2013, Fortaleza. Simpósio Brasileiro de Automação Inteligente (SBAI), 2013.

MITCHELL, M. An Introduction to Genetic Algorithms - Massachusetts Institute of Technology - First MIT Press paperback edition, 1998.

MOHANTY, P. K.; PARHI, D. R. Controlling the motion of autonomous mobile robot using various techniques: a review. Journal of Advance Mechanical Engineering, v. 1, n. 1, p. 24–39, (2013).

MORAVEC, H. Robot: Mere machine to transcendent mind. [. [S.l.]: Oxford University Press, (1999).

NASA, - (National Aeronautics and Space Administration) - Curiosity Rover – (2015) < http://www.nasa.gov/mission_pages/msl/index.html>. Acesso em: 22 Junho 2015.

OLIVI. L. R. Controle de Locomoção do Robô Khepera Utilizando Redes Neurais Artificiais. 2005. Monografia (Trabalho Final de Curso em Engenharia de Controle e Automação). Escola de Minas, Universidade Federal de Ouro Preto, Minas Gerais, 2005.

PANDA, R, K. ; CHOUDHURY, B, B.; An Effective Path Planning of Mobile Robot Using Genetic Algorithm. IEEE International Conference on Computational Intelligence & Communication Technology. p. 287-291, (2015).

PESSIN, G ; Osório, F ; MUSSE, S - "Utilizando Redes Neurais Artificiais no Controle Robusto de Navegação de Robôs Móveis" (2007) , (UNICEN). Disponível em: <http://www.sucesumt.org.br/mtdigital/anais/files/UtilizandoRedesNeuraisArtificiaisnoControle Robusto.pdf> Acesso em : 05 Agosto 2015.

(PIONEER, 2011) Datasheet. Disponível em: <http://www.mobilerobots.com/Libraries/Downloads/Pioneer3AT-P3AT RevA.sflb.ashx>.

PURIAN, K. F,FAROKHI FARDAD. Comparing the Performance of Genetic Algorithm and Ant Colony Optimization Algorithm for Mobile Robot Path Planning in the Dynamic Environments with Different Complexities - Journal of Academic and Applied Studies - Vol. 3(2) February (2013), pp. 29-44 (2013).

QU, H.; XING, K.; ALEXANDER, T. An improved genetic algorithm with coevolutionary strategy for global path planning of multiple mobile robots. Neurocomputing,v. 120, n. 0, p. 509 – 517, 2013. ISSN 0925-2312. Image Feature Detection and Description. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0925231213005195>.

RAJA, P.; PUGAZHENTHI, S. Optimal path planning of mobile robots: A review. International Journal of Physical Sciences, v. 7, n. 9, p. 1314–1320, (2012).

SAMADI.M, OTHMAN.M.F., —Global Path Planning for Autonomous Mobile Robot using Genetic Algorithm‖ , International Conference on Signal-Image Technology & Internet-Based Systems (2013).

SANTHOSH, S. Innovative Methodology and Designing of Intelligent Mobile Robots Computer Vision with Genetic Algorithm SLE Mechanism and Advanced Sensors in Managing Disaster. International Conference on Circuit, Power and Computing Technologies [ICCPCT], p. 1692-1697, (2014).

SAUVOLA, J.;Pietaksinen, M., "*Adaptive document image binarization*", *Pattern Recogn,* 2000. Vol. 33, p. 225–236, 2000.

SECCHI, H. Uma Introdução aos Robôs Móveis. 2008. 1 v . Tese (Doutorado), Ingeniería de Sistemas de Control, Universidade Nacional de San Juan, San Juan (Argentina), 2008.

SHI, P.; CUI, Y. Dynamic path planning for mobile robot based on genetic algorithm in unknown environment. In: Control and Decision Conference (CCDC), 2010 Chinese. [S.l.: s.n.], 2010. p. 4325–4329.

SHILTAGH, N. A.; JALAL, L. D. Path planning of intelligent mobile robot using modified genetic algorithm. International Journal of Soft Computing and Engineering (IJSCE), v. 3, p. 2231 – 2307, (2013). ISSN 2231-2307.

SIEGWART, R.; NOURBAKHSH, I. Introduction to Autonomous Mobile Robots. Bradford Book, (2004). (A Bradford book). ISBN 9780262195027. Disponível em: <http://books.google.com.br/books?id=gUbQ9_weg88C>.

SIERAKOWSKI, C. A. Inteligência Coletiva Aplicada a Problemas de Robótica Móvel. Dissertação (Mestrado), Pontifícia Universidade Católica do Paraná – PUCPR, Curitiba, 2006.

SILVA, I. N. da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. REDES NEURAIS ARTIFICIAIS: PARA ENGENHARIA E CIÊNCIAS APLICADAS. São Paulo: Artliber, 2010.

SLOWIK, A. ; BIALKO M. Training of Artificial Neural Networks Using Differential Evolution Algorithm, IEEE, HIS, pp. 60-65, Krakow, Poland, May. 2008.

SPACE, lost. Direção: Irwin Allen, 1965. Produção: Irwin Allen e Gerard Ervin, Irwin Allen Productions (1965).

TRIPATHI and V.Rihani - MOTION PLANNING OF AN AUTONOMOUS MOBILE ROBOT USING ARTIFICIAL NEURAL NETWORK - Natarajan Meghanathan, et al. (Eds): SIPM, FCST, ITCA, WSE, ACSIT, CS & IT 06, pp. 367–373, 2012.

TUNCER, A.; YILDIRIM, M. Dynamic path planning of mobile robots with improved genetic algorithm. Computers & Electrical Engineering, v. 38, n. 6, p. 1564 – 1572, (2012). ISSN 0045-7906. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0045790612001255>. Acesso em: 10 Julho 2015.

VON NEUMANN, J. The Computer and The Brain. New Haven, CT – Yale University Press, 1958.

WARS, Star. Direção: George Lucas. Produção: Gary Kurtz. LucasFilm, 1977, son., cor.

WATKINS, A. Ga-based path planning for mobile robots: An empirical evaluation of seven techniques. Journal of Computers, v. 8, n. 8, p. 1912 – 1922, 2013. ISSN 1796203X. Disponível em: <http://search-ebscohost-com.ez87.periodicos.capes.gov.br/login.aspx?direct=true&db=iih&AN=89903290&lang=pt-br&site=ehost-live>. Acesso em: 12 Junho 2015.

ZIVKOVIC, F. van der Heijden, Efficient Adaptive Density Estimapion per Image Pixel for the Task of Background Subtraction, Pattern Recognition Letters, vol. 27, no. 7, pages 773-780, 2006.