



**SOCIEDADE BRASILEIRA DE INSTRUÇÃO
UNIVERSIDADE CANDIDO MENDES
INSTITUTO UNIVERSITÁRIO CANDIDO MENDES-CAMPOS**

Credenciada por Decreto de 24/11/1997

Rua Anita Peçanha, 100 - Parque São Caetano / Cep: 28040-320 / Campos dos Goytacazes - RJ

Telefone / Fax : (0xx22) 2733-4100

<http://www.ucam-campos.br>

CHRYSSTIANO BARBOSA DE SOUZA ARAÚJO
LEANDRO MORAES VALE CRUZ
LUCAS CARVALHO TEIXEIRA
THIAGO RIBEIRO NUNES

**ASPECTOS DE VISÃO COMPUTACIONAL NO
DESENVOLVIMENTO DO *TOOLKIT HORUS***

Campos dos Goytacazes - RJ
Agosto - 2009

CHRYSTIANO BARBOSA DE SOUZA ARAÚJO
LEANDRO MORAES VALE CRUZ
LUCAS CARVALHO TEIXEIRA
THIAGO RIBEIRO NUNES

ASPECTOS DE VISÃO COMPUTACIONAL NO DESENVOLVIMENTO DO *TOOLKIT HORUS*

Monografia apresentada à Universidade
Cândido Mendes como requisito obrigatório
para a obtenção do grau de Bacharel em
Ciências da Computação.

ORIENTADOR: Prof. D.Sc. Ítalo de Oliveira Matias

CO-ORIENTADOR: Prof. D.Sc. Dalessandro Soares Vianna

Campos dos Goytacazes-RJ
2009

ASPECTOS DE VISÃO COMPUTACIONAL NO DESENVOLVIMENTO DO *TOOLKIT HORUS*

Monografia apresentada à Universidade Cândido Mendes como requisito obrigatório para a obtenção do grau de Bacharel em Ciências da Computação.

Aprovada em ____ de _____ de 2009.

BANCA EXAMINADORA

Prof. Ítalo de Oliveira Matias - Orientador
D.Sc.em Sistemas Computacionais pela UFRJ

Prof. Dalessandro Soares Vianna
D.Sc. em Informática pela PUC-Rio

Prof. Fermín Alfredo Tang Montané
D.Sc. em Engenharia de Produção pela UFRJ

Dedico este trabalho a meus pais, avós, irmãos, namorada e amigos por todo apoio e compreensão prestado durante essa importante fase da minha vida.

Chrystiano

Agradeço a meus familiares e amigos pela paciência e incentivo, possibilitando calma e forças para o desenvolvimento desse projeto. Aproveito para agradecer, o apoio durante todos os momentos difíceis ao longo da graduação, assim como em todos os momentos da minha vida.

Leandro

Dedico este trabalho ao meu tio Marcelo Soares Carvalho.

Lucas

Dedico este trabalho a meus pais e a minha família por todo apoio prestado durante a sua realização.

Thiago

Agradecimentos

Agradecemos as nossas famílias e amigos, pela paciência e apoio, que possibilitou calma e forças para o desenvolvimento desse projeto, assim como durante todos os momentos difíceis ao longo de nossa graduação. Agradecemos também a todos nossos professores que estiveram conosco durante essa caminhada, em especial aos nossos orientadores Ítalo de Oliveira Matias e Dalessandro Soares Vianna pelo apoio e orientação para o desenvolvimento desse trabalho, e ao professor Fabricio Barros que, além de fornecer forte incentivo e motivação, foi o responsável pela formação e pela união deste grupo. Por fim, agradecemos a Universidade Cândido Mendes por propiciar um ambiente frutífero a pesquisa e ao aprendizado.

Resumo

Neste trabalho, serão apresentados os conceitos e algoritmos envolvidos na construção de um *toolkit Open Source*, de nome Horus, utilizado para o desenvolvimento e controle de aplicações que envolvam agentes inteligentes, com foco em dois problemas centrais. O primeiro problema refere-se à movimentação autônoma de um agente inteligente em ambientes desconhecidos. O segundo problema refere-se à visão computacional, onde o agente deve ser capaz de extrair informações do ambiente através da utilização de câmeras virtuais ou reais. Além de algoritmos para movimentação autônoma e visão computacional, o Horus também fornece abstrações para criação e configuração de agentes inteligentes com seus principais componentes e dispositivos. Embora o Horus tenha sido também desenvolvido com foco na movimentação autônoma de agentes inteligentes, nesse trabalho será dado maior enfoque à parte de visão computacional e às abstrações para criação e configuração de agentes inteligentes. Como forma de validação e prova de conceito das funcionalidades fornecidas pelo toolkit Horus, também foram desenvolvidas três aplicações: PyANPR, Teseu e Ariadnes. PyANPR é um sistema *web* de reconhecimento automático de placas de automóveis. Teseu é um sistema que simula o mapeamento automático de um ambiente desconhecido por um agente inteligente. Ariadnes é um sistema que simula a movimentação autônoma de um agente inteligente em um ambiente desconhecido através de técnicas de visão computacional e mapeamento de ambientes. Neste trabalho, além da descrição do Horus, também serão detalhadas a aplicação PyANPR e os aspectos de visão computacional do simulador Ariadnes.

Palavras-chave: Visão Computacional, Mapeamento de Ambientes, Horus

Abstract

This work presents the concepts and algorithms involved in the construction of an Open Source toolkit, named Horus, used in development and control of applications involving intelligent agents, focusing on two central problems. The first problem relates to the handling of autonomous intelligent agent in unknown environments. The second problem relates to computer vision, where the agent should be able to extract information from the environment through the use of virtual or real cameras. In addition to algorithms for computer vision and autonomous movement, the Horus also provides abstractions for the development and configuration of intelligent agents with their main components and devices. While Horus was also developed with a focus on movement of autonomous intelligent agents, this work is focused on the computer vision algorithms and abstractions for creation and configuration of intelligent agents. As a proof of concept and validation of the functionality provided by the toolkit Horus, were also developed three distinct applications: PyANPR, Teseu and Ariadnes. PyANPR is an automatic number plate recognition web system. Teseu is a system that simulates the automatic mapping of an unknown environment by an intelligent agent. Ariadnes is a system that simulates the movement of an autonomous intelligent agent in an unfamiliar environment using techniques of computer vision and unknown environments mapping. This work will also detail the PyANPR application and the computer vision aspects of the Ariadnes simulator.

Keywords: Computer Vision, Environment Mapping, Horus

Sumário

1	Introdução	12
2	Conceitos Básicos	15
2.1	Agentes Inteligentes	15
2.2	Computação Gráfica	18
2.2.1	Processamento de Imagem	20
2.2.2	Visão Computacional	23
2.2.3	Reconhecimento de Padrões	24
2.2.4	Reconhecimento Óptico de Caracteres	26
2.2.5	Extração de características	28
2.3	Inteligência Computacional	29
2.4	Redes Neurais Artificiais	30
2.4.1	Neurônio Biológico	31
2.4.2	O Neurônio Artificial MCP	33
2.4.3	Funções de Ativação	34
2.4.4	Arquiteturas de Redes Neurais	36
2.4.5	Processo de aprendizado	38
3	Horus	41
3.1	<i>Core</i>	42
3.2	Processamento de Imagem	45
3.2.1	<i>Thresholding</i>	47
3.2.2	Skeletonization	51
3.3	Visão	55
3.3.1	Extração de características	56
3.3.2	Reconhecimento de Objetos	60
3.4	Mapeamento e Navegação	61
3.5	Aplicações desenvolvidas com o Horus	62
4	PyANPR	64
4.1	Localização da Placa	65
4.2	Pré-processamento da placa	70
4.3	OCR	71

4.4 Aplicação Web	72
5 Ariadnes	73
6 Conclusões e Trabalhos Futuros	79
Apêndices	84
A Instalação	84

Listas de Figuras

2.1	Subáreas da computação gráfica.	19
2.2	Etapas do processo de OCR.	27
2.3	Neurônio biológico.	31
2.4	Modelo do neurônio de McCulloch e Pits.	34
2.5	Gráfico da função Limiar.	35
2.6	Gráfico da função sigmóide.	35
2.7	Gráfico da função Signum.	36
2.8	Gráfico da Tangente Hiperbólica.	36
2.9	Rede de uma única camada.	37
2.10	Rede de múltiplas camadas.	37
2.11	Exemplo de rede feedback.	38
2.12	Esquema de aprendizado supervisionado.	40
2.13	Esquema de aprendizado não supervisionado.	40
3.1	Arquitetura do módulo <i>core</i> .	42
3.2	Extensão da classe <i>Brain</i> do Horus por uma aplicação.	43
3.3	Comportamento <i>MyBehavior</i> que estende tanto de <i>NeuralNetworkBehavior</i> quanto de <i>MappingBehavior</i> .	45
3.4	(a) 4-vizinhança (b) d-vizinhança (c) 8-vizinhança.	46
3.5	Exemplo de imagem binária.	47
3.6	Utilização do <i>threshold</i> global implementado no <i>toolkit</i> Horus.	48
3.7	(a) Imagem original (b) Resultado do processamento	48
3.8	Imagen original (à direita) e imagem binarizada com thresholding global (à esquerda).	49
3.9	Localização do limiar por Chow e Kaneko.	50
3.10	Imagen original (à direita) e resultado da limiarização com <i>thresholding</i> local (à esquerda).	51
3.11	Imagen de um "T" e seu respectivo esqueleto.	51
3.12	Imagen de um "B" (preto) e seu respectivo esqueleto (branco).	52
3.13	8-vizinhança do pixel p_1 .	52
3.14	Exemplos das funções: (a) $B(p_1) = 2$, $A(p_1) = 1$ b) $B(p_1) = 2$, $A(p_1) = 2$.	53
3.15	(a) $B(p_1) = 7$ (b) $B(p_1) = 0$ (c) $B(p_1) = 1$.	53
3.16	Exemplos onde $A(p_1)$ é maior que 1.	54

3.17	Exemplo de situação em que linhas verticais com largura de dois pixels não serão inteiramente removidas.	54
3.18	Exemplo de situação em que linhas horizontais com largura de dois pixels não serão inteiramente removidas.	55
3.19	(a) padrão de entrada do algoritmo (b) deleção iterativa dos pixels das bordas (c) resultado após a execução do algoritmo.	55
3.20	Padrões completamente erodidos pelo algoritmo de Hilditch.	55
3.21	Matriz de pixel de um <i>bitmap</i>	56
3.22	Layout com seis regiões em três linhas e duas colunas.	58
3.23	Quatorze diferentes tipos de arestas.	58
3.24	Matrizes referentes aos tipos de arestas.	59
3.25	Vetor de Características.	59
3.26	(a) Exemplo de junção com quatro vizinhos (b) Exemplo de junção com três vizinhos (c) Exemplo de terminação de linha.	60
3.27	(a) Imagem original (b) Imagem negativa.	60
4.1	(a) Fotografia de um carro (b) <i>Band</i> (c) <i>Plate</i>	66
4.2	(a) Fotografia de um carro (b) imagem convertida para escala de cinza (c) imagem filtrada por Sobel-Vertical.	67
4.3	(a) Imagem filtrada por Sobel-Vertical (b) Projeção vertical.	67
4.4	(a) Imagem do band pré-processada (b) projeção horizontal	68
4.5	(a) Imagem original (b) Projeção vertical suavizada (c) Aplicação do <i>Threshold</i>	69
5.1	Arquitetura conceitual do simulador Ariadnes.	73
5.2	Placa informativa presente no ambiente.	74
5.3	Agente configurado com dispositivos de lasers.	75
5.4	Ambiente utilizado no Ariadnes.	76
5.5	Imagens capturadas no ambiente pelas câmeras do robô (a) Câmera para baixo. (b) Câmera frontal.	77
5.6	Projeção horizontal de uma placa informativa.	78
5.7	Projeção vertical das colunas da placa.	78

Capítulo 1

Introdução

Existem alguns tipos de ambiente que são inóspitos ao homem. Nesses casos é comum utilizar robôs para explorar e atuar em tais locais. A movimentação desses robôs pode ser automática (agentes autônomos), semi-automática (agentes semi-autônomos) ou manual. Neste trabalho, serão apresentados os passos para a construção de um *toolkit Open Source*, de nome Horus, utilizado para o desenvolvimento e controle de aplicações que envolvam agentes inteligentes, com foco em dois problemas centrais. O primeiro problema refere-se à movimentação autônoma de um agente inteligente em ambientes desconhecidos. O segundo problema refere-se à visão computacional, onde o agente deve ser capaz de extrair informações do ambiente através da utilização de câmeras virtuais ou reais.

Um Agente, por definição, é todo elemento ou entidade autônoma que pode perceber seu ambiente, por algum meio cognitivo ou sensorial, e de agir sobre esse ambiente por intermédio de atuadores. Podem-se citar como exemplos de agentes inteligentes, além de robôs autônomos ou semi-autônomos, personagens de um jogo, agentes de busca e recuperação de informação, entre outros.

Para que um agente autônomo seja capaz de atuar em um ambiente desconhecido é necessário anteriormente explorar esse local. Essa exploração pode ser feita através de um mapeamento desse ambiente. A forma como se mapeia o ambiente internamente no sistema é determinante na sua precisão e performance. As diferentes abordagens para controle de

agentes móveis autônomos interagem fortemente com a representação do ambiente. Uma proposta para o mapeamento do ambiente, ainda não implementada no Horus, consiste na construção de um ambiente virtual 3D associado a um ambiente real no qual um robô real está explorando. Essa abordagem exige que o agente reconheça padrões no ambiente explorado e represente-os no ambiente virtual.

Durante a exploração do ambiente nos simuladores construídos, o agente deverá ser capaz de estimar sua posição local para localizar-se globalmente e se recuperar de possíveis erros de localização. Um correto mapeamento do ambiente junto a aplicação correta das leis da cinemática pode resolver tal problema. Uma proposta para a localização de um agente no ambiente é a utilização do método Monte Carlo [FOX et al. 1999] ou do método SLAM (*Simultaneous Location and Mapping*) [Davison, Cid e Kita 2004], [Dellaert 2005]. O método selecionado para ser implementado no *toolkit* Horus foi o SLAM.

Um agente explora um ambiente através de sensores. O sensoriamento provê ao robô as informações necessárias para a construção de uma representação do ambiente onde está inserido e para uma interação com os elementos contidos nesse. Sistemas com uma variedade de sensores tendem a obter resultados mais precisos. A fusão de dados de sensores, ou como é mais conhecida, fusão de sensores, é o processo de combinação de dados de múltiplos sensores para estimar ou predizer estados dos elementos da cena. Neste trabalho, foram utilizados lasers, câmeras e odômetro como sensores.

Para simular a visão de um agente inteligente, são utilizadas câmeras virtuais. Na abordagem desse trabalho, a visão é a principal forma de percepção do ambiente. A visão possibilita reconhecer padrões e classificar obstáculos. Existem diferentes tipos de obstáculos. Estes podem ser classificados como transponível (aquele que não interrompe a trajetória), intransponível (aquele que o exigirá recalcular a trajetória por outro caminho) e redutor (aquele que permite o robô seguir pela trajetória, porém a uma velocidade mais lenta). Mesmo mediante a obstáculos transponíveis e redutores, pode ser conveniente recalcular o caminho devido ao aumento do custo do percurso. Uma proposta para o desvio de trajetória é o modelo baseado em Campos Potenciais proposto por [8]. A classificação de um obstáculo ocorre mediante a algum método de reconhecimento de padrões, baseado

em visão computacional.

O *toolkit* Horus, criado neste trabalho, propõe uma coleção de classes e algoritmos voltados a resolução de problemas pertencentes as áreas de visão computacional e mapeamento automático de ambientes. Nessa monografia, será dado foco à parte de visão computacional do Horus. De forma a validar a implementação desse *toolkit* e demonstrar a sua utilidade, foram desenvolvidas três aplicações distintas: Os simuladores Teseu e Ariadnes, e o PyANPR. Neste trabalho, além do *toolkit* Horus, serão, também, apresentados a parte de visão computacional utilizada no simulador Ariadnes e a aplicação para reconhecimento automático de placas de automóveis PyANPR.

Capítulo 2

Conceitos Básicos

Neste capítulo serão apresentados os principais conceitos das áreas de computação gráfica e inteligência computacional que foram utilizados na implementação do *toolkit* Horus. Esses conceitos são de fundamental importância para o entendimento dos algoritmos de visão computacional, processamento de imagens e reconhecimento de padrões implementados nos principais módulos do Horus.

2.1 Agentes Inteligentes

A Inteligência Computacional (IC) é uma área de estudo da ciência da computação que procura desenvolver sistemas computacionais capazes de ter ações e reações similares as capacidades humanas, tais como: pensar, criar, solucionar problemas, entre outros. Um Agente, por definição, é todo elemento ou entidade autônoma que pode perceber seu ambiente por algum meio cognitivo ou sensorial e de agir sobre esse ambiente por intermédio de atuadores [Russel e Norvig 2003]. Podem-se citar como exemplos de agentes inteligentes, além de um robô autônomo ou semi-autônomo, personagens de um jogo, agentes de busca e recuperação de informação e agentes de chats.

Existem diferentes definições para a arquitetura de um robô presentes na literatura. Este trabalho considera a definição abordada por Arkin [Arkin 1998] a qual considera que

uma arquitetura de um robô está mais relacionada com os aspectos de software que os de hardware. Apesar de algumas diferenças, os modelos de arquitetura para robôs móveis descrevem um mecanismo para construção de um sistema de desenvolvimento e controle de agentes inteligentes, apresentando, principalmente, quais os módulos presentes e como estes interagem.

De uma forma geral, os módulos de uma arquitetura de um agente inteligente se preocupam com aspectos como percepção, planejamento e atuação/execução. A percepção refere-se à compreensão do ambiente e dos elementos nele contido. Denomina-se *sequência de percepções*, a história completa de tudo que o agente já percebeu, ou seja, o conjunto de todas as percepções do agente até um dado momento; o planejamento à inteligência do robô; e a atuação, ou execução, é o modo como o robô procede no ambiente, ou seja, movimentos, captura de informações, etc.

Existem diversos modelos de arquitetura para robótica entre os quais ressaltam-se os modelos de três camadas como: SSS [CONNEL 1992], Atlantis [GAT 1991], 3T [BONASSO 1991]. Todas as arquiteturas se dividem semelhantemente da seguinte forma:

- camada reativa: orienta os sensores e atuadores, além de tomar decisões de baixo nível, como visão e movimento;
- camada deliberativa: responsável pela inteligência do robô, aspectos mais globais, que não são alterados a cada iteração;
- camada de execução: intermedia essas duas outras camadas.

As camadas reativa e deliberativa provêm processos denominados de comportamentos. Em termos matemáticos, o comportamento do agente é a função que mapeia qualquer percepção ou sequência de percepções para uma ação específica [Russel e Norvig 2003], essa função é conhecida como *função de agente*.

Com base nesses conceitos, definiu-se neste trabalho um agente como uma entidade composta de comportamentos, dispositivos e um programa de agente. Os dispositivos do agente, utilizados no sistema em questão, são de sensoriamento (lasers e câmeras) e de

movimentação (rodas). Os comportamentos do agente são: mapeamento, navegação e reconhecimento de objetos. O programa de agente é responsável pelo controle da execução de todos os comportamentos supracitados.

Os comportamentos são procedimentos implementados para representar ações e reações dos agentes. As ações são comportamentos ativos, ou seja, procedimentos que visam realizar um objetivo previamente definido. Por outro lado, reações são procedimentos realizados mediante a estímulos externos. Exemplos de ação e reação ocorrem no deslocamento de um agente de uma posição a outra. Para realizar o deslocamento é necessário traçar uma rota. Tal comportamento é definido como uma ação. Ao se deparar com algum obstáculo durante o percurso, esse agente deve gerar um comportamento de re-planejamento da rota para alcançar o objetivo inicial sem colidir com o obstáculo. A esse re-planejamento, denomina-se reação.

Comportamentos podem ser divididos em duas categorias. Os Comportamentos Primários: parar, reduzir velocidade, acelerar, desviar de obstáculos, virar, inverter direção, dirigir-se a meta, fotografar, disparar lasers e etc; e Comportamentos Inteligentes: mapear, reconhecer objetos, navegar e executar uma tarefa específica. Um Comportamento Inteligente executa um conjunto de comportamentos Primários para atingir seu objetivo. Os comportamentos primários ocorrem na camada reativa, enquanto que, os inteligentes ocorrem na camada deliberativa.

Entende-se por *Programa de Agente* o programa que recebe as percepções do ambiente como entrada e as mapeia para uma determinada ação através da função de agente. Além da estrutura citada, o programa de agente pode ser estruturado de outras maneiras. Um exemplo de estrutura é construir o programa de agente como um conjunto de sub-rotinas que serão executadas de forma assíncrona em relação ao ambiente. O programa permanece em *loop*, recebendo todas as percepções geradas pelo ambiente, e repassa cada percepção para uma sub-rotina que irá tratá-las.

Os programas de agente podem ser classificados em quatro categorias principais:

- Agentes reativos simples: esse tipo de programa de agente se baseia apenas na per-

cepção atual para executar as suas ações, onde a sequência de percepções até o momento é ignorada.

- Agentes reativos baseados em modelo: esse programa de agente armazena uma sequência de percepções e toma suas decisões levando em consideração as percepções dessa sequência.
- Agentes baseados em objetivos: esse programa de agente possui informações sobre o objetivo que deve alcançar. Logo, suas decisões são tomadas com base na combinação das percepções do ambiente e nas informações do objetivo.
- Agentes baseados na utilidade: esse tipo de programa de agente tem a preocupação, não só de alcançar o seu objetivo final, como também em determinar a melhor forma possível de alcançá-lo.

Em duas das aplicações desenvolvidas neste trabalho, os programas de agente utilizados se enquadram na categoria de agentes baseados na utilidade.

2.2 Computação Gráfica

A computação gráfica é a ciência que basicamente estuda um conjunto de técnicas que transforma dados em imagem apresentada através de um dispositivo gráfico. Estes dados podem ser organizados através de um modelo matemático cuja geometria se assemelhe a do objeto do mundo real a qual se deseja modelar. Segundo J. Gomes e L. Velho em [Gomes e Velho 2003] a computação gráfica pode ser dividida em quatro subáreas que relacionam dados e imagens. Essas subáreas são tão próximas que estão muitas vezes interligadas, em determinados momentos são até confundidas. Estas subáreas são: Modelagem Geométrica, Síntese de Imagem, Processamento de Imagem e Visão Computacional. A Figura 2.1 mostra estas quatro subáreas da computação gráfica e como se relacionam com os dados e a imagem.

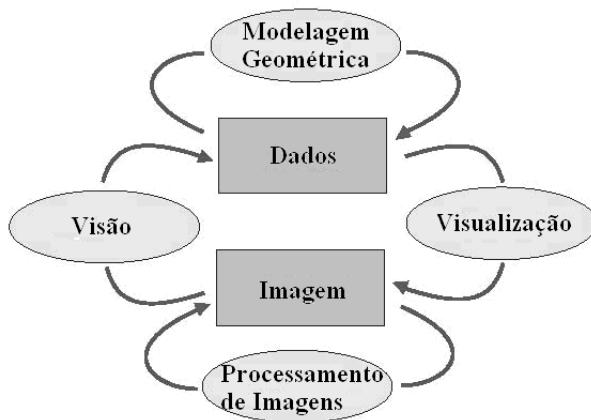


Figura 2.1: Subáreas da computação gráfica.

A Modelagem Geométrica trata do problema de descrever e estruturar dados geométricos no computador, ou seja, encontrar um modelo matemático capaz de representar o objeto desejado. Um exemplo de Modelagem geométrica é a obtenção de uma função que se aproxime de outra desconhecida através de um método de interpolação de alguns pontos de amostragem já conhecidos.

A Síntese de imagens, ou Visualização, processa os dados gerados pelo sistema na modelagem geométrica e produz uma imagem que pode ser vista em um dispositivo de saída gráfica. A partir do mapeamento de alguns pontos, distribuídos em torno de um determinado objeto, obtém-se um modelo matemático, que o representa, através da Modelagem Geométrica. De posse deste modelo é possível apresentar uma imagem que represente a forma do objeto. Este último processo é um exemplo de Síntese de Imagem.

O Processamento de Imagens assume como dados de entrada uma imagem, processa-a produzindo outra imagem como dado de saída. As técnicas desta área podem ser utilizadas, por exemplo, para melhorar a qualidade de uma imagem antiga ou danificada, produzindo outra, imagem com menos ruídos.

Por último, o processo de Visão Computacional, ou simplesmente Visão, tem por finalidade obter, a partir de uma imagem de entrada, informações físicas, geométricas e/ou topológicas sobre os dados do objeto apresentado na imagem. Um bom exemplo desta área

aparece nas transmissões de futebol, quando é necessário observar com segurança e certeza se um jogador está em posição de impedimento, ou seja, uma situação onde é necessário determinar a sua posição no campo em relação à posição dos jogadores adversários.

Para determinados pesquisadores a Computação Gráfica é apenas a área denominada de síntese de imagem. Porém para [Gomes e Velho 2003], além de outros pesquisadores desta área, esta ciência é composta pelas quatro linhas apresentadas acima. Este fato se justifica graças ao fato de cada vez surgir mais métodos e técnicas que solucionem problemas nestas quatro vertentes.

De um modo geral, visão computacional é a área da computação gráfica que extrai informações de uma imagem ou de um conjunto de imagens. Apesar da definição de visão computacional como uma subárea da computação gráfica, é válido ressaltar que isso não é aplicado em muitos grupos de pesquisa. A área chamada de visão pode ser renomeada para Análise de Imagem, e essa é, inquestionavelmente, uma subárea da Computação Gráfica.

O problema abordado nesse trabalho é o de Reconhecimento de Padrões. Esse é um problema clássico de Visão Computacional. Mas a maioria das técnicas de visão computacional é combinada com técnicas de processamento de imagem (normalmente o processamento de imagem entra como um pré-processamento da imagem a ser tratada pelo método de visão). Dessa forma, nesse trabalho serão apresentados esses dois temas da computação gráfica: Processamento de Imagem, na subseção 2 deste capítulo, e Visão Computacional, na subseção 3.

2.2.1 Processamento de Imagem

Os estudos referentes ao Processamento de Imagem originaram-se na Teoria dos Sinais (ou Processamento de Sinais). Sinais, assim como uma imagem, é um suporte carregado pontualmente de informações. Os sinais apresentam-se de diversas formas, entre as quais podemos citar: sinais sonoros, sinais eletromagnéticos, sinais cognitivos e sinais luminosos (que nos torna capaz de enxergar um ambiente, ou distinguir elementos de uma imagem).

O estudo dos diversos sinais existentes origina-se na obtenção de uma descrição matemática

deste. Após isso é necessário determinar meios efetivos de construir representações discretas e buscar de algoritmos que possam implementar as diversas técnicas de síntese, análise e processamento destes sinais em um computador.

Segundo o modelo proposto por [Gomes e Velho 2002] é possível obter três níveis de abstração no processo de representação dos sinais: sinais contínuos, sinais discretos e sinais codificados. Cada nível corresponde a uma descrição do sinal de modo conveniente a um determinado assunto da Teoria dos Sinais. Relacionando esses três níveis de abstração é possível definir quatro operações de transformações: a discretização (mudança de um sinal contínuo para discreto), a codificação (mudança de um sinal discreto para um codificado), a decodificação (mudança de um sinal codificado para um discreto) e a reconstrução (mudança de um sinal discreto para um sinal contínuo).

Basicamente um sinal deve ser representado por um elemento matemático que estabeleça a variação de uma determinada grandeza. Por exemplo, o som é a variação da densidade do ar ao longo do tempo; uma imagem é a variação da cor. Desta forma som e imagem podem ser considerados sinais, como mencionado anteriormente. A variação do sinal modelada matematicamente pode ser determinística, onde se utilizam uma função como modelo do sinal, ou não determinística, onde o sinal é descrito através de um processo estocástico.

No caso que o sinal é modelado por uma função $f : U \subset \mathbb{R}^m \rightarrow \mathbb{R}^n$ a grandeza física do sinal é representada por vetor n -dimensional variando em um suporte m -dimensional. Denomina-se espaço dos sinais a um subespaço de funções de $\{f : U \subset \mathbb{R}^m \rightarrow \mathbb{R}^n\}$, com U , m e n fixos. Desta forma um espaço de funções é um espaço vetorial cujas operações de soma de funções e multiplicação por escalar são as usuais.

$$(f + g)(t) = f(t) + g(t)$$

$$(\lambda f)(t) = \lambda f(t)$$

De posse desta informação pode-se definir um sinal contínuo, em um espaço de sinais, como a função $f : U \subset \mathbb{R}^m \rightarrow \mathbb{R}^n$. Porém vale ressaltar que o termo contínuo não quer dizer que necessariamente a função f é topologicamente contínua. Na verdade significa

que domínio e contradomínio são um ”continuum” de números, ou seja, conjuntos vetoriais cujas coordenadas dos elementos são números reais. Essa ressalva deve-se principalmente pelo fato que computacionalmente falando, não se trabalham com o conjunto dos números reais, mas na verdade utiliza-se um conjunto finito de elementos representado através de ponto flutuante. Para simplificar essa discussão vale citar o termo mais popular para este sinal contínuo: analógico.

Dado um sinal contínuo definido pela função $f : U \subset \mathbb{R}^m \longrightarrow \mathbb{R}^n$, entre as operações definidas entre os níveis de abstração dos sinais, a operação de discretização (ou representação) consiste em discretizar o domínio da citada função (processo conhecido como amostragem), enquanto a codificação é a discretização do contradomínio (processo conhecido como quantização).

Não é pretensão deste trabalho esgotar este assunto. Uma abordagem um pouco mais elaborada pode ser encontrada em [Gomes e Velho 2002].

O processamento de imagem, como definido anteriormente, consiste em, a partir de uma imagem, obter outra imagem com determinados aspectos realçados. Ou seja, processar uma imagem consiste em realizar sucessivas transformações para obter determinadas informações contidas nesta com mais facilidade.

Um desejo desta área é o de processar uma imagem analogamente como é realizado pelo sistema visual humano. Porém este sistema é extremamente complexo. Para realizar as mesmas operações utilizando máquinas é necessário fazer com que esta tenha uma real compreensão do ambiente analisado. Desta forma o processamento de imagem está extremamente inter-relacionado com sua área de aplicação, ao invés de um sistema amplo capaz de identificar qualquer objeto, a partir de uma imagem, em boas condições, dada.

Ao se estudar o processamento de Imagem deve-se anteriormente ter um domínio claro de o que é uma Imagem. Do ponto de vista da ótica, uma imagem é um conjunto de pontos, com uma identidade fotométrica, que converge formando um todo. Matematicamente falando, uma Imagem Contínua é uma aplicação $f : U \subset \mathbb{R}^2 \longrightarrow C$, onde C é o espaço das cores, f é chamada de Função Imagem, U é o suporte da Imagem e $f(U)$, um subconjunto de C , é o gamute de cores da imagem. Em geral $C = \mathbb{R}^n$, porém os casos mais comuns

são: para $n = 3$ (como RGB), ou $n = 1$ (como imagens em escala de cinza).

Inicialmente o termo imagem estava relacionado à compreensão do domínio da luz visível. Atualmente pode-se pensar em imagens como uma grande quantidade de dados representados bidimensionalmente, entre os quais se podem citar as imagens acústicas, sísmicas, de satélites, infravermelhas, magnéticas, entre outras.

A linha de pesquisa denominada Processamento de Imagem consiste em realizar operações sobre uma imagem. Essas operações normalmente são determinadas por funções, cujo domínio é a imagem a ser processada e o contradomínio é o espaço das funções.

2.2.2 Visão Computacional

Outra área da Computação Gráfica é a Visão computacional. Visão computacional ainda pode ser considerada como uma área imatura de pesquisa. Embora existam trabalhos reconhecidos, somente após o final da década de 1970 os estudos foram aprofundados com o advento do computador possibilitando processar grandes conjuntos de dados como imagens. Esse início teve grande impulso com David Marr, considerado o "pai" da Visão Computacional. David Marr morreu em 17 de novembro de 1980, porém seu livro [Marr 1982] foi lançado postumamente em 1982 e causou um forte impacto na comunidade de visão computacional. Entretanto, esses estudos foram geralmente originados de outros campos de pesquisa, e, consequentemente, não existe uma formulação padrão para o problema de visão computacional, assim como não existe uma formulação padrão de como os problemas de visão computacional devem ser resolvidos. Os que existem atualmente são diversos métodos para resolver várias tarefas bem definidas, no qual os métodos são bastante especializados e raramente podem ser generalizados para várias aplicações. Na maioria das aplicações de visão computacional, os computadores são pré-programados para resolver uma tarefa particular, mas métodos baseados em aprendizagem estão se tornando cada vez mais comuns.

Visão computacional é uma linha de pesquisa que procura definir procedimentos automatizados que imitem processos da visão humana (ou de outros animais). Normalmente

essa área baseia-se em técnicas estatísticas, ou probabilísticas, para estimar informações baseadas em imagens. Para desenvolver técnicas nessa linha de pesquisa é essencial uma boa compreensão sobre o funcionamento das câmeras (com suas falhas na aquisição de sinais que compõem uma imagem, inserção de ruídos, distorção de cores, etc), assim como seu posicionamento no espaço ambiente. Esse segundo problema é conhecido como calibração de câmera e é um tema largamente estudado, pois normalmente, para estimar informações em uma cena vista em duas dimensões é necessário ter noções sobre profundidade, que é estimada através da especificação da posição da câmera e de objetos na cena.

De um modo geral, visão computacional é a área da computação gráfica que extrai informações de uma imagem ou de um conjunto de imagens. Apesar da definição de visão computacional como uma subárea da computação gráfica, é válido ressaltar que isso não é aplicado em muitos grupos de pesquisa. A área chamada de visão pode ser renomeada para Análise de Imagem, e essa é, inquestionavelmente, uma subárea da Computação Gráfica.

Existem diversas aplicações de visão computacional em diversas áreas como detecção automática de tumores em imagens médicas; definição de posição de jogadores de futebol no campo, em um determinado instante; determinação de volume de um fluido em mares ou rios, etc.

Nesse trabalho foi estudado um tema particular de visão computacional, que é o de reconhecimento de padrões, seção 2.2.3. Para reconhecer padrões, assim como na maioria dos problemas de visão computacional, é necessário extrair características da imagem.

2.2.3 Reconhecimento de Padrões

Reconhecimento de padrões é uma atividade que os seres humanos fazem a todo tempo e, normalmente, sem um esforço consciente. Seres humanos recebem informações através de vários sensores orgânicos, as quais são processadas instantaneamente pelo cérebro. Essa habilidade é ainda mais impressionante no que diz respeito à assertividade do processo de reconhecimento mesmo quando as informações não se encontram em condições ideais, como por exemplo, em situações onde as informações são vagas, imprecisas, ou até mesmo,

incompletas. A área de Reconhecimento de Padrões é responsável por projetar algoritmos e abordagens que procuram aproximar as tarefas realizadas computacionalmente das habilidades humanas. Esse processo consiste em classificar e descrever objetos através de um conjunto de características ou propriedades. Um dos principais conceitos dentro de reconhecimento de padrões é o discriminante. Tal conceito consiste em medir uma distância de um determinado padrão para cada outro previamente conhecido. Logo, a classe de um determinado padrão será a mesma do seu vizinho mais próximo [Simpson 1992], [Simpson 1993], de menor distância ou o protótipo mais parecido [Eltoft 1998]. Espera-se de um sistema de reconhecimento de padrões que este seja capaz de aprender de uma forma adaptativa e dinâmica. Em sistemas de reconhecimento de padrões automáticos, as etapas de aprendizagem e reconhecimento são combinados a fim de atingir um objetivo desejado [Cagnoni et al. 1993]. Portanto, redes neurais artificiais é uma das principais técnicas utilizadas nesse sentido [Nigrin 1993] e será apresentada na seção 2.4. Um típico sistema de reconhecimento de padrões consiste em três partes: aquisição de dados, seleção/extracção de características e classificação/clustering [Pal e Mitra 2004].

- Aquisição de dados: é o processo de seleção dos dados que serão usados como entrada no processo de reconhecimento. Tais dados podem ser qualitativos, quantitativos ou ambos. Podendo ser numéricos, linguísticos, entre outros.
- Seleção/Extração de características: o objetivo principal desta etapa consiste em gerar o melhor conjunto de características necessárias para o processo de reconhecimento, de modo a maximizar a eficácia do sistema. A grande dificuldade dessa etapa está na determinação de um critério adequado para a escolha de um bom conjunto de características. Um bom critério é aquele que é imutável para qualquer variação possível dentro de uma classe, todavia, deve ser capaz de destacar as diferenças importantes a fim de discriminar entre diferentes tipos de padrões.
- Classificação/clustering: classificação é o processo de definição de qual classe um determinado padrão de entrada pertence. Essa classificação pode ser feita utilizando

técnicas determinísticas e probabilísticas. As classes são definidas a partir de um conjunto de amostras apresentadas na etapa de aprendizagem.

A área de reconhecimento de padrões possui grande variedade de aplicações, como reconhecimento de faces, leitura biométrica, identificação de circuitos impressos defeituosos, reconhecimento óptico de caracteres, reconhecimento de fala e de escrita cursiva, entre outros.

Um padrão em particular que podemos citar são caracteres. O campo de pesquisa, dentro da visão computacional, que se preocupa com o reconhecimento de caracteres é conhecido pelo acrônimo OCR, do inglês, Optical Recognition Character. Esse tema será abordado na seção 4.3.

2.2.4 Reconhecimento Óptico de Caracteres

Reconhecimento Óptico de Caracteres ou OCR (*Optical character recognition*) é um campo de pesquisa nas áreas de reconhecimento de padrões, inteligência artificial e visão computacional. Em computação, OCR é o processo de tradução eletrônica de imagens de textos para textos que possam ser editados computacionalmente, permitindo assim, a realização de operações que seriam inviáveis de serem realizadas sobre o texto em formato de imagem. Sistemas OCR, ou de reconhecimento de caracteres, datam do final dos anos 50 e têm sido amplamente utilizados em computadores desktop desde os anos 90.

Esses sistemas disponibilizam textos contidos em imagens, capturadas por dispositivos ou geradas computacionalmente, em textos editáveis por computador. Os textos gerados por sistemas OCR são, normalmente, utilizados por outras ferramentas que permitem operações, como a busca de determinado conteúdo de interesse.

Apesar de mais de 40 anos de pesquisa, sistemas OCR ainda estão muito longe de alcançar a eficácia de um ser humano. A eficácia desses sistemas está fortemente ligada à qualidade das imagens. Imagens limpas e de alta qualidade levam esses sistemas a atingirem uma taxa de aproximadamente 99% de eficácia [Rice, Nagy e Nartker 1999]. Porém, imagens com baixa resolução, com ruído ou com diferenças de luminosidade, por exemplo,

podem levar esses sistemas a cometerem erros grosseiros e confundirem diversos tipos de caracteres. Caracteres como "6" e "9", "B" e "8" e "o" e "0", são facilmente confundidos em imagens imperfeitas.

Nesse sentido, atualmente, boa parte das pesquisas em OCR está focada na melhoria da sua eficácia no que diz respeito à extração de textos de imagens que não se encontram em condições ideais. Além disso, o reconhecimento de textos escritos a mão em linguagem cursiva ainda são uma área de pesquisa muito ativa. O processo de OCR é constituído de várias etapas, com responsabilidades bem definidas, que ao final apresentam o texto editável. A Figura 2.2 apresenta um esquema básico do processo de OCR.

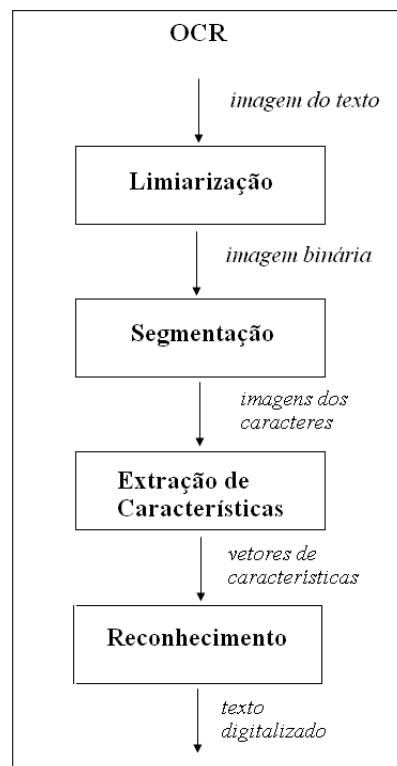


Figura 2.2: Etapas do processo de OCR.

Na Figura 2.2, têm-se as várias etapas de um processo clássico de OCR. Inicialmente, é necessário tornar a imagem binária, isto é, transformar a imagem, que se encontra em escala de cinza, em uma imagem com apenas duas cores: preto e branco, cores essas repre-

sentadas respectivamente pelos inteiros 0 e 255. Na etapa de segmentação, cada caractere presente na imagem é recortado da imagem binária para ser tratado individualmente. Após a segmentação, cada caractere é passado individualmente para a etapa de extração de características, onde um vetor numérico é extraído a partir das características desse caractere. Por último, é realizada a etapa de reconhecimento do vetor de características, que finalmente deverá apontar o caractere correto. Cada etapa desse processo pode ser implementada de diversas maneiras e por vários algoritmos diferentes. No capítulo 3 serão apresentados alguns algoritmos que podem ser utilizados no processo de OCR.

2.2.5 Extração de características

No campo de reconhecimento de padrões, extrair características significa extrair medidas associadas ao objeto que se deseja reconhecer, de forma que essas medidas sejam semelhantes para objetos semelhantes e diferentes para objetos distintos [Santos 2007]. Definir vetores de características é o método para representação de dados mais comum e conveniente para problemas de classificação e reconhecimento. Cada característica resulta de uma medição qualitativa ou quantitativa, que é uma variável ou um atributo do objeto [Guyon et al. 2006].

Para reconhecer um caractere de uma representação bitmap, há a necessidade de extrair características do mesmo para descrevê-lo de uma forma mais adequada para o seu processamento computacional e reconhecimento. Como o método de extração de características afeta significantemente a qualidade de todo o processo de reconhecimento de padrões, é muito importante extrair características de modo que elas sejam invariantes no que diz respeito às várias condições de iluminação, tipo de fonte e possíveis deformações nos caracteres causadas, por exemplo, pela inclinação da imagem.

Geralmente, a descrição de uma região de uma imagem é baseada em suas representações interna e externa. A representação interna de uma imagem é baseada em suas propriedades regionais, como cor ou textura. A representação externa é selecionada quando se deseja dar ênfase nas características da forma do objeto. Logo, o vetor de características

de uma representação externa inclui características como o número de linhas, a quantidade de arestas horizontais, verticais e diagonais, etc.

O conjunto de vetores de características forma um espaço vetorial. Cada caractere representa uma determinada classe, e todas as formas de representação desse caractere definem as instâncias dessa classe. Todas as instâncias do mesmo caractere devem ter uma descrição similar através de vetores numéricos chamados de "descritores", ou "padrões". Logo, vetores suficientemente próximos representam o mesmo caractere. Essa é a premissa básica para que o processo de reconhecimento de padrões seja bem sucedido.

No capítulo que trata do *toolkit* Horus, serão explicados alguns métodos de extração de características implementados no módulo de visão computacional.

2.3 Inteligência Computacional

A Inteligência Computacional (IC) é uma linha de pesquisa dentro de Ciência da Computação que tem o objetivo de desenvolver sistemas que imitem algumas capacidades específicas dos seres humanos [Russel e Norvig 2003]. Esses sistemas são denominados *Sistemas Inteligentes*. Como exemplos de capacidades humanas que inspiram o desenvolvimento desse tipo de sistema têm-se: aprendizado, percepção, raciocínio, evolução e adaptação. Nesse sentido, cada técnica de Inteligência Computacional foi fortemente influenciada por aspectos naturais dos seres humanos. Como exemplo de técnicas de IC pode-se citar:

- Sistemas Especialistas: são programas desenvolvidos para solucionar problemas de uma área específica do conhecimento humano. Esses sistemas armazenam o conhecimento de um especialista, de uma determinada área, em forma de código e são capazes de tomar decisões com base nesse conhecimento.
- Redes Neurais Artificiais: são modelos computacionais não lineares inspirados na forma de operação da rede neural do cérebro humano. Essa técnica visa reproduzir as capacidades de aprendizado, associação, generalização e abstração dos seres humanos, sendo bastante eficiente no aprendizado de padrões.

- Algoritmos Genéticos: são algoritmos que fornecem um mecanismo de busca adaptativa que se baseia no mecanismo de adaptação e evolução natural e recombinação genética de *Darwin*. Essa técnica possui uma população de indivíduos (soluções), representados por cromossomos. Cada indivíduo é então submetido a um processo de avaliação, onde é medida a sua aptidão para o problema. Os melhores indivíduos são selecionados e seus cromossomos são combinados. Esse processo é realizado de forma iterativa até obter um resultado satisfatório.
- Lógica *Fuzzy*: lógica *Fuzzy* tem o objetivo de modelar a capacidade de aproximação do raciocínio humano. Essa técnica tem o objetivo de desenvolver sistemas capazes de fornecer soluções aceitáveis em ambientes onde as informações de entrada são incertas ou imprecisas.

Dentre as várias técnicas de inteligência computacional, a técnica que foi utilizada nesse trabalho foi a de Redes Neurais Artificiais devido ao seu grande poder de aprendizado de padrões e generalização. Essas características são muito importantes na resolução do problema de reconhecimento de padrões abordado por este trabalho. A técnica de redes neurais artificiais será apresentada na próxima sessão.

2.4 Redes Neurais Artificiais

Redes Neurais Artificiais (RNAs) são estruturas computacionais que visam imitar a forma com que o cérebro humano processa as informações. O cérebro possui a capacidade de organizar e encadear seus elementos estruturais, conhecidos como neurônios, para realizar o processamento das informações de forma não linear e paralela. Dessa forma, as principais características das redes neurais são a habilidade de aprender relações complexas e não lineares entre os padrões de entrada e as saídas, utilizarem procedimentos de treinamento seqüencial e se auto-adaptar aos dados [Jain, Duin e Mao 2000].

Na sua forma geral, uma rede neural é uma máquina projetada para imitar a forma com que o cérebro realiza uma tarefa particular ou uma função de interesse, podendo

ser implementada em hardware ou simulada através de software. Para atingir uma performance razoável, as redes neurais empregam uma massiva interconexão de unidades de processamento simples, denominadas neurônios [Haykin 1994].

As redes neurais artificiais possuem a capacidade de aprendizado e, como consequência, de generalização. Generalização refere-se à produção de saídas racionais para entradas que não foram apresentadas a rede durante a fase de treinamento ou aprendizado. As capacidades de aprendizado e de generalização tornam as redes neurais capazes de resolver problemas complexos e não-lineares. A seguir, serão explicadas algumas características do neurônio biológico, fundamentais para o entendimento das RNAs.

2.4.1 Neurônio Biológico

O sistema nervoso é formado por um conjunto extremamente complexo de neurônios. Os neurônios estão conectados uns aos outros através de sinapses. Nos neurônios a comunicação é realizada através de impulsos elétricos, quando um impulso é recebido, o neurônio o processa, e passado um limite de ação, dispara um segundo impulso o qual flui do corpo celular para o axônio que, por sua vez, pode ou não estar conectado a um dendrito de outra célula. A Figura 2.3 apresenta uma representação de um neurônio.

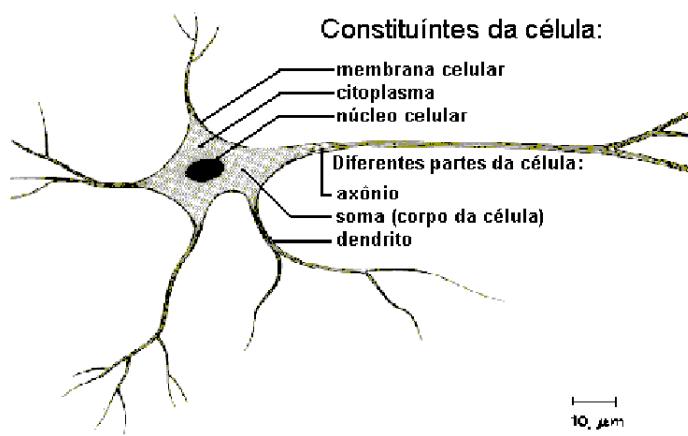


Figura 2.3: Neurônio biológico.

Os principais componentes de um neurônio são:

- Os dendritos: membrana que recebe os estímulos gerados por outras células. Os dendritos são as entradas do neurônio.
- Soma: é o corpo do neurônio, que é responsável por coletar e combinar informações vindas de outros neurônios;
- O axônio: membrana constituída de uma fibra tubular que é responsável por transmitir os estímulos para outras células. O axônio representa a saída do neurônio.

O neurônio biológico é constituído de um corpo celular denominado soma. Nesse local ocorre o processamento metabólico da célula nervosa ou neurônio. A partir da soma, projetam-se extensões filamentares denominadas dendritos, e o axônio. Este modelo anatômico foi identificado por Ramon Cajal em 1894. Com base nas pesquisas de Erlanger e Gasser, em 1920, e outras posteriores, passou-se a entender o comportamento do neurônio biológico como sendo o dispositivo computacional do sistema nervoso, o qual possui muitas entradas e uma única saída [Haykin 1994].

As entradas ocorrem através das conexões sinápticas, que conectam a árvore dendrital aos axônios de outras células nervosas. Os sinais que chegam pelos dendritos são pulsos elétricos conhecidos como impulsos nervosos ou potenciais de ação, e constituem a informação que o neurônio processará de alguma forma para produzir como saída um impulso nervoso no seu axônio.

Sinapse é o nome dado ao ponto de contato entre a terminação axônica de um neurônio e o dendrito de outro. É pelas sinapses que os nodos se unem funcionalmente, formando a rede neural. As sinapses funcionam como válvulas, e são capazes de controlar a transmissão de impulsos entre os nodos na rede [Braga, Ludermir e Carvalho 2000] e estão compreendidas entre duas membranas celulares: a membrana pré-sináptica, que recebe o estímulo vindo de uma célula, e a membrana pós-sináptica, que é a do dendrito. Na região pré-sináptica, se o estímulo nervoso recebido atinge um determinado limiar em um espaço curto de tempo, a célula "dispara", produzindo um impulso que é transferido para outras

células através de neurotransmissores presentes na membrana dendrital. Dependendo do neurotransmissor, a conexão sináptica é excitatória ou inibitória. A conexão excitatória provoca uma alteração no potencial da membrana que contribui para formação do impulso nervoso no axônio de saída, enquanto que a conexão inibitória age no sentido contrário.

O mecanismo como é criado o potencial de ação ou impulso nervoso é o seguinte: quando o potencial da membrana está menos eletronegativo do que o potencial de repouso, diz-se que a membrana está despolarizada e quando está mais negativo, diz-se que ela está hiperpolarizada. O impulso nervoso ou potencial de ação é uma onda de despolarização de certa duração de tempo, que se propaga ao longo da membrana. A formação de um potencial de ação na membrana axonal ocorre quando essa membrana sofre despolarização suficientemente acentuada para cruzar um determinado valor conhecido como limiar de disparo. Quando esse limiar é superado, os estímulos são passados para outras células através das ligações sinápticas.

2.4.2 O Neurônio Artificial MCP

McCulloch e Pitts [Mendel e McLaren 1970] propuseram um modelo matemático do neurônio artificial MCP. Esse modelo foi proposto com base nos principais conceitos do neurônio biológico. O modelo matemático do neurônio proposto por McCulloch e Pitts apresenta n terminais de entrada x_1, x_2, \dots, x_n (representando os dendritos) e apenas um terminal de saída y (representando o axônio). Os terminais de entrada do neurônio têm pesos acoplados w_1, w_2, \dots, w_n cujos valores podem ser positivos ou negativos caso as sinapses correspondentes serem inibitórias ou excitatórias. O efeito de uma sinapse particular i no neurônio pós-sináptico é dado por $x_i w_i$. Os pesos determinam o nível em que o neurônio deve considerar sinais de disparo que ocorrem naquela conexão. O modelo está ilustrado na Figura 2.4.

O corpo do neurônio realiza um simples somatório dos valores $x_i w_i$ que chegam a ele. Se o somatório dos valores ultrapassa o limiar de ativação ou *threshold*, o neurônio dispara (valor 1 na saída), caso contrário o neurônio permanece inativo (valor 0 na saída). A

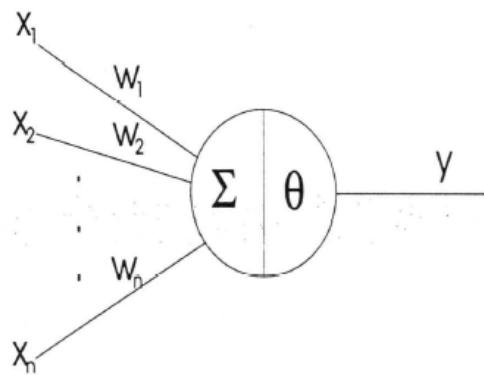


Figura 2.4: Modelo do neurônio de McCulloch e Pits.

ativação do neurônio é realizada através de uma função de ativação, que dispara ou não o neurônio dependendo do valor da soma ponderada das suas entradas. No modelo MCP original, a função de ativação ativará a saída quando:

$$\sum_{i=1}^n x_i w_i \geq \theta$$

Na equação acima, n é a quantidade de entradas do neurônio, w_i é o peso associado à entrada x_i e θ é o limiar do neurônio.

2.4.3 Funções de Ativação

A partir do modelo apresentado por McCulloch e Pits, surgiram vários outros modelos que permitem a produção de qualquer saída, não apenas zero ou um, e com várias funções de ativação. As funções de ativação mais comuns são:

- Função Limiar: função utilizada no modelo de McCulloch e Pits, caracterizada por "tudo ou nada", representada da seguinte forma:

$$f(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v \leq 0 \end{cases}$$

Onde v é igual ao valor produzido pelo somatório das entradas do neurônio.

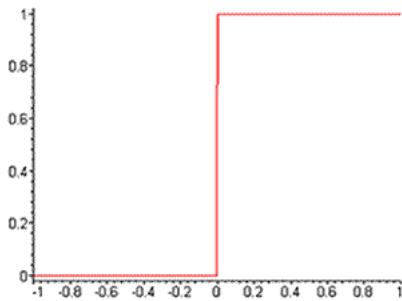


Figura 2.5: Gráfico da função Limiar.

- Função Sigmóide: essa é uma função semilinear, limitada e monotônica que pode assumir valores entre 0 e 1. Existem várias funções sigmodais, porém, a mais usada é a função logística definida pela equação:

$$f(v) = \frac{1}{1+e^{-av}}$$

Onde a é o parâmetro de inclinação da função sigmoidal e v é o valor de ativação do neurônio.

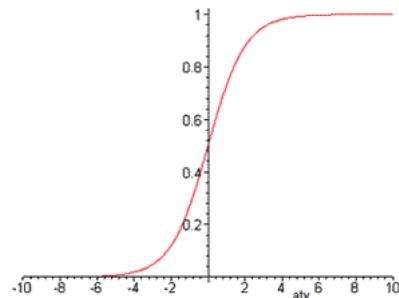


Figura 2.6: Gráfico da função sigmoidal.

- Função Signum: essa função apresenta as mesmas características da função limiar, porém, se limita ao intervalo entre 1 e -1. Essa função é representada por:

$$f(v) = b \frac{v}{|v|} \text{ para } v \neq 0$$

Onde b são os limites inferiores e superiores ($b = |1|$ no gráfico) e v é o valor de ativação.

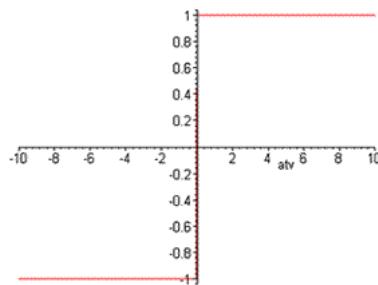


Figura 2.7: Gráfico da função Signum.

- Tangente Hiperbólica: seu gráfico é parecido com o da Função Sigmóide, assumindo valores entre 1 e -1 , sendo representada por:

$$f(v) = a \frac{e^{(bv)} - e^{(-bv)}}{e^{(bv)} + e^{(-bv)}}$$

Onde a é o parâmetro de inclinação da curva, b são os limites inferiores e superiores ($b = |1|$ no gráfico) e v o valor de ativação.

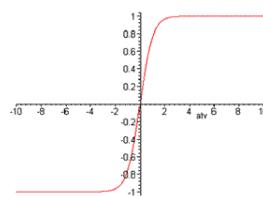


Figura 2.8: Gráfico da Tangente Hiperbólica.

2.4.4 Arquiteturas de Redes Neurais

A definição da arquitetura de uma RNA define a maneira com que os neurônios são estruturados na rede [Haykin 1994]. A arquitetura restringe o tipo de problema que

pode ser tratado pela rede. Redes com uma camada única de nodos MCP, por exemplo, só conseguem resolver problemas linearmente separáveis. Redes recorrentes, por sua vez, são mais apropriadas para resolver problemas que envolvem processamento temporal [Braga, Ludermir e Carvalho 2000]. A arquitetura de uma rede é definida pelos seguintes parâmetros: número de camadas da rede, número de neurônios em cada camada, tipo de conexão entre os neurônios e topologia da rede.

Em geral, as redes neurais podem ser classificadas quanto ao número de camadas, quanto ao tipo de conexão e quanto à conectividade entre os neurônios. Quanto ao número de camadas têm-se:

- Redes de uma única camada: existe apenas um nó entre uma entrada e uma saída da rede. A Figura 2.9 apresenta um exemplo de redes de única camada.

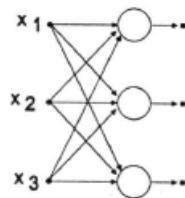


Figura 2.9: Rede de uma única camada.

- Redes de múltiplas camadas: existe mais de um neurônio entre alguma entrada e alguma saída (Figura 2.10).

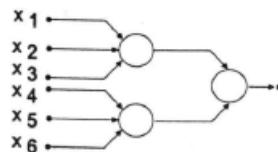


Figura 2.10: Rede de múltiplas camadas.

Quanto ao tipo de conexão têm-se:

- Feedforward, ou acíclica: a saída de um neurônio na i -ésima camada da rede não pode ser usada como entrada de nodos em camadas de índice menos ou igual a i (Figura 2.9).
- Feedback, ou cíclica: a saída de algum neurônio na i -ésima camada da rede é utilizada como entrada em neurônios da camada de índice menor ou igual a i (Figura 2.11).

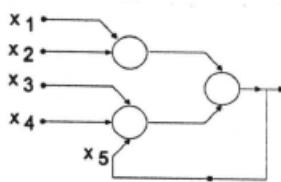


Figura 2.11: Exemplo de rede feedback.

Quanto à conectividade têm-se:

- Redes parcialmente conectadas (Figura 2.9).
- Redes completamente conectadas (Figura 2.10).

2.4.5 Processo de aprendizado

A principal propriedade de uma rede neural é a sua habilidade de aprender sobre o ambiente no qual está inserida de forma a melhorar a sua performance na resolução de problemas complexos. Essa melhora na performance é adquirida a cada instante do processo de aprendizado de acordo com uma forma de medição pré-estabelecida. Com isso, uma rede neural aprende sobre seu ambiente através de um processo iterativo de ajuste dos pesos sinápticos adquirindo mais conhecimento sobre o problema após cada iteração do processo de aprendizado. Segundo Mendel e McLaren [Mendel e McLaren 1970], no contexto de redes neurais, aprendizado é o processo pelo qual os parâmetros de uma rede neural são ajustados através de estímulos produzidos pelo ambiente no qual a rede está inserida. O tipo de aprendizado é determinado pela

maneira particular com que os parâmetros são modificados. O Horus disponibiliza diversos métodos de treinamento de redes, podendo ser agrupados em dois paradigmas principais: aprendizado supervisionado e aprendizado não-supervisionado. No entanto, outros dois paradigmas bastante conhecidos são os de aprendizado por reforço (que é um caso particular de aprendizado supervisionado) e aprendizado por competição (que é um caso particular de aprendizado não supervisionado).

– Aprendizado Supervisionado:

Esse tipo de aprendizado é o mais utilizado em RNAs. Esse aprendizado é dito supervisionado porque as entradas e as respectivas saídas desejadas são fornecidas por um supervisor externo, normalmente chamado de "professor". Seu objetivo é ajustar os pesos da rede, de forma a encontrar uma ligação entre os pares de entrada e saída fornecidos pelo professor. O professor é responsável por direcionar o processo de aprendizado fornecendo os padrões de entrada, as saídas desejadas e a taxa de erro desejada. A cada padrão de entrada submetido à rede pelo professor, compara-se a resposta desejada (que representa uma solução ótima para aquele padrão de entrada) com a resposta calculada, ajustando-se os pesos das conexões para minimizar o erro. A minimização da diferença é incremental, já que pequenos ajustes são feitos nos pesos a cada iteração do aprendizado. A soma dos erros quadráticos de todas as saídas é normalmente utilizada como medida de desempenho da rede e também como função de custo a ser minimizada pelo algoritmo de treinamento. A Figura 2.12 apresenta um esquema básico de aprendizado supervisionado.

Nesse trabalho, o algoritmo de aprendizado utilizado foi o *backpropagation*.

– Aprendizado Não Supervisionado:

No aprendizado não supervisionado não há um professor que oriente o processo de aprendizado. Ao contrário do aprendizado supervisionado, que possui pares de entrada e saída, para esses algoritmos somente são disponibilizados os padrões de entrada. De acordo com as regularidades estatísticas com que as

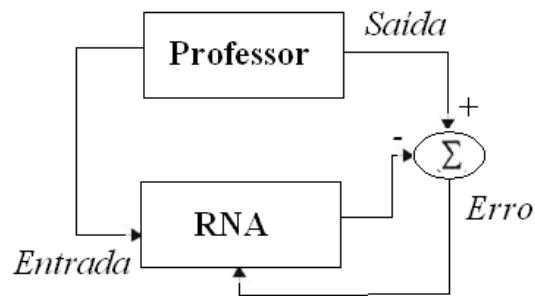


Figura 2.12: Esquema de aprendizado supervisionado.

entradas ocorrem, a rede cria classes de acordo com as características extraídas de cada padrão de entrada. Dessa forma, para cada entrada fornecida a rede, a saída será a classe a qual a entrada pertence. Caso a entrada não pertença a nenhuma classe pré-existente, a rede cria uma nova classe para essa entrada. A base para a utilização desse tipo de aprendizado é a redundância nos dados, sem redundância, seria praticamente impossível a utilização bem sucedida do aprendizado não supervisionado. A Figura 2.13 apresenta um esquema de um sistema de aprendizado não supervisionado.

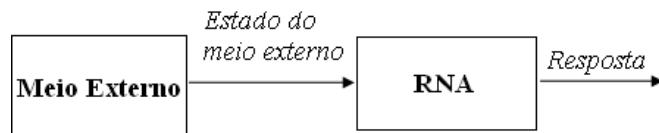


Figura 2.13: Esquema de aprendizado não supervisionado.

Capítulo 3

Horus

Horus é um *toolkit*, criado no presente trabalho, para desenvolvimento e controle de agentes inteligentes desenvolvido na linguagem de programação Python. Esse *toolkit* foi construído com o objetivo de fornecer classes e algoritmos voltados a resolução de problemas de mapeamento automático de ambientes e visão computacional. O Horus é um projeto *Open Source* que se encontra sob a licença de uso LGPL(*Lesser General Public License*). O projeto está disponível para uso no repositório de projetos *Open Source* da Google na url <http://code.google.com/p/finishstrike/>.

O *toolkit* Horus fornece os módulos *Core*, Mapeamento, Visão e Util. O módulo *Core* apresenta as abstrações que devem ser implementadas pelas aplicações para construir um agente inteligente. O módulo Mapeamento fornece algoritmos de localização, mapeamento e navegação para um agente. O módulo Visão fornece os algoritmos de visão computacional necessários na etapa de reconhecimento de padrões. Por último, o módulo Util fornece um conjunto de funções utilitárias que podem ser usadas tanto no *toolkit* Horus quanto em qualquer outra aplicação. Cada um desses módulos será explicado nas subseções seguintes.

3.1 Core

O Horus pode ser utilizado tanto como uma biblioteca de algoritmos para processamento de imagens, visão computacional e mapeamento de ambientes, como através de extensões das classes fornecidas pelo módulo *core*. Essas classes podem ser estendidas pelas aplicações, são chamadas de abstrações. A Figura 3.1 mostra a arquitetura deste módulo.

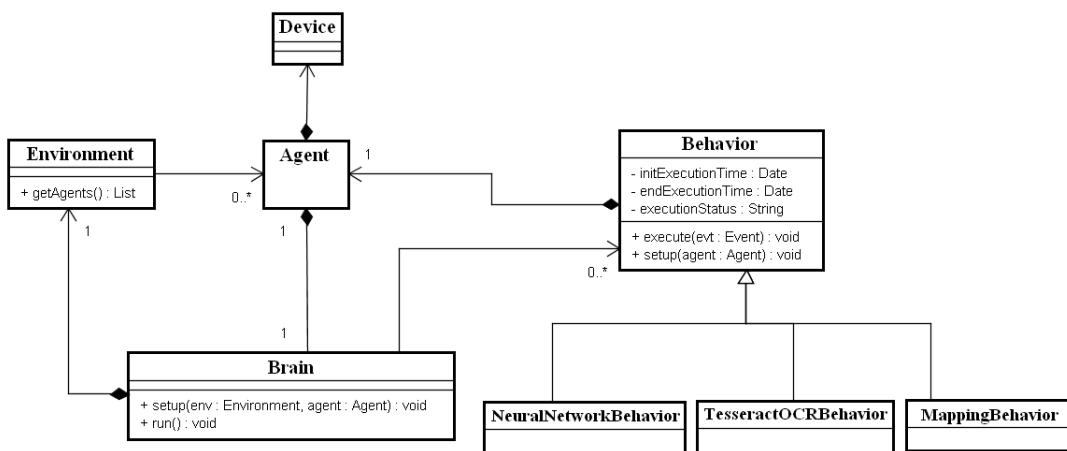


Figura 3.1: Arquitetura do módulo *core*.

A arquitetura acima apresenta classes que representam o ambiente (*Environment*), o agente inteligente (*Agent*), os dispositivos (*Device*), o programa de agente (*Brain*), eventos (*Event*) e a hierarquia de comportamentos (*Behavior*). Cada instância da classe *Agent* representa um agente inteligente na aplicação. Para que um agente inteligente possa ser utilizado por uma aplicação, ela deve configurá-lo com instâncias de *Device* e uma única instância da classe *Brain*, responsável pela inteligência do agente.

O programa de agente, responsável pela inteligência do agente, deve ser implementado em extensões da classe *Brain* (cérebro). Uma aplicação que deseja implementar um programa de agente para um agente em particular deve estender a classe *Brain* e

implementar o método *run()* dessa classe, como apresentado na Figura 3.2. Esse método é o *loop* principal da execução do agente. No diagrama acima, nota-se que a classe *Brain* pode estar relacionada a nenhum ou a muitos comportamentos. Essa é mais uma facilidade fornecida pelo módulo *core* do Horus que tem o objetivo de organizar a implementação dos comportamentos separadamente da implementação da classe *Brain*. Dessa forma, o programa de agente fica mais claro e simples de ser compreendido e mantido.

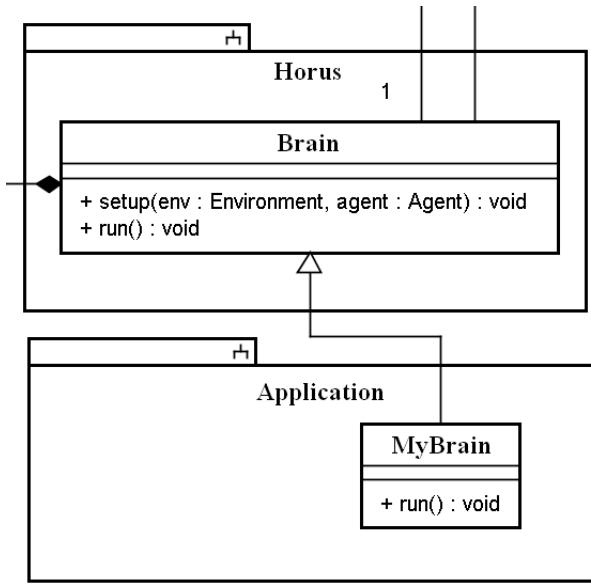


Figura 3.2: Extensão da classe *Brain* do Horus por uma aplicação.

Os comportamentos (*Behavior*) recebem eventos gerados pela aplicação e fazem com que o agente execute uma determinada ação com base no tipo de evento recebido. Como exemplo de eventos, pode-se citar a captura de uma cena, um obstáculo detectado, a leitura de um determinado dispositivo, etc. Um comportamento pode ser implementado diretamente como uma extensão da classe *Behavior* ou como uma extensão de uma ou várias de suas subclasses. As subclasses da classe *Behavior* (*NeuralNetworkBehavior*, *TesseractBehavior* e *MappingBehavior*) fornecem facilidades para a utilização de funcionalidades fornecidas pelo Horus. Logo, *NeuralNetworkBehavior*

fornecer métodos para a construção e treinamento de redes neurais na implementação de um comportamento. A classe *TesseractBehavior* disponibiliza a funcionalidade de OCR da *engine* Tesseract, presente no Horus. A classe *MappingBehavior* disponibiliza métodos para implementação de comportamentos de mapeamento de ambientes através da técnica SLAM. Dessa forma, uma aplicação que necessite de um comportamento que envolva mapeamento de ambientes e redes neurais, por exemplo, deve criar uma classe que estenda tanto da classe *MappingBehavior* como da classe *NeuralNetworkBehavior*. A Figura 3.3 mostra como ficaria o esquema desse comportamento, sendo representado pela classe *MyBehavior*.

A classe *Behavior* também possui atributos para armazenar informações sobre estado de execução de um comportamento. Essas informações são os horários de início e término da execução e o status de execução do comportamento. Essas informações são utilizadas para emissão de relatórios de atuação do agente.

Em certas aplicações, um agente inteligente não necessariamente se encontra sozinho no ambiente. Ele pode interagir com outros agentes, como no caso de aplicações que envolvam enxames de agentes. Para que um agente possa obter informações sobre outros agentes que se encontram no ambiente, ou sobre o próprio ambiente, foi criada a classe *Environment*, que representa o ambiente no qual o agente está inserido. Dessa forma, o cérebro do agente deve ser configurado tanto com uma instância da classe *Agent*, como com uma instância da classe *Environment* para operar.

Quando o cérebro do agente é configurado com diversos comportamentos, é necessário definir a ordem de execução desses comportamentos. Em alguns casos, os comportamentos podem possuir condições de execução. Logo, o cérebro é responsável por identificar as condições que cada comportamento necessita para ser executado e colocá-lo como ativo quando a sua condição de execução for satisfeita. Contudo, há casos em que dois ou mais comportamentos podem ter a sua condição de execução satisfeita. Nesses casos, é necessário definir prioridades de execução sobre os comportamentos ou executá-los em paralelo. A ordem de execução dos comportamentos define a

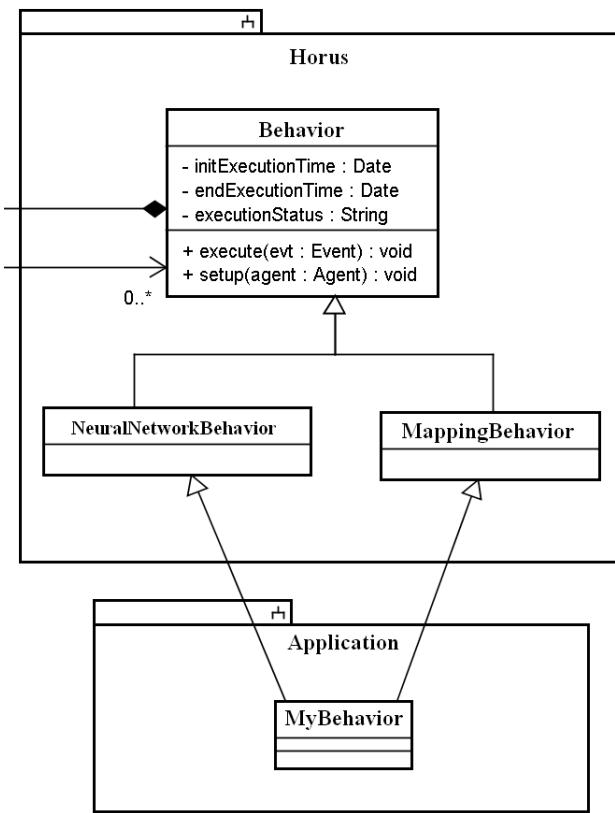


Figura 3.3: Comportamento *MyBehavior* que estende tanto de *NeuralNetworkBehavior* quanto de *MappingBehavior*.

máquina de estados de execução do agente inteligente. Sendo assim, a implementação do cérebro como uma máquina de estados se enquadra perfeitamente em aplicações que exijam a interação entre diversos comportamentos.

3.2 Processamento de Imagem

O termo processamento de imagens refere-se ao processamento de imagens de duas dimensões por um computador digital [Neto 1999], normalmente é utilizado como um estágio para novos processamentos de dados, tais como reconhecimento de padrões e aprendizagem de máquina. Esse tipo de processamento é utilizado em diversos tipos

de aplicações, entre elas, processamento de imagens médicas e de satélite, robótica, sensoriamento remoto, entre outras.

Para uma melhor compreensão dos conceitos e algoritmos utilizados durante esse trabalho, é necessária uma breve introdução sobre algumas propriedades de uma imagem digital. Essas propriedades são:

- Conectividade: esse conceito determina se dois pixels estão conectados entre si. Para isso, é necessário determinar se esses pixels são adjacentes, segundo algum critério, e se os seus níveis de cinza são, de alguma forma, similares. Definindo uma image binária onde os pixels somente assumem valores 0 e 1, dois pixels vizinhos só serão considerados conectados se possuírem o mesmo valor.
 - Adjacência: dois pixels p e q são adjacentes somente se estiverem conectados segundo algum critério. Dados os conjuntos de pixels C_1 e C_2 , esses conjuntos serão adjacentes se algum pixel de C_1 é adjacente a algum pixel de C_2 .
 - Vizinhança: dado um pixel p de coordenadas (x, y) , sua 4-vizinhança é definida como $(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)$, chamada de $N_4(p)$. Os quatro vizinhos diagonais do pixel p são definidos como $(x - 1, y - 1), (x - 1, y + 1), (x + 1, y - 1), (x + 1, y + 1)$, chamados de $N_d(p)$. Dessa forma, a união dos conjuntos $N_4(p)$ e $N_d(p)$ forma o conjunto da 8-vizinhança do pixel p , chamado de $N_8(p)$.
- A Figura 3.4 ilustra as possíveis vizinhanças de um pixel.

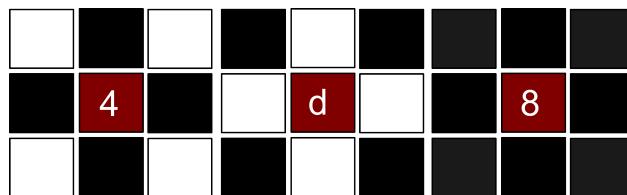


Figura 3.4: (a) 4-vizinhança (b) d-vizinhança (c) 8-vizinhaça.

Nas próximas subseções serão explicados os principais algoritmos de processamento de imagens implementados no *toolkit* Horus.

3.2.1 *Thresholding*

O processo de limiarização da imagem, também chamado de *thresholding*, consiste em transformar a imagem que inicialmente se encontra em escala de cinza, com 256 tons diferentes, para uma imagem com apenas dois tons: preto e branco, representados por 0 ou 255 (Figura 3.5). No processo de OCR, Essa etapa é utilizada para definir claramente as fronteiras dos caracteres na imagem, a fim de prepará-los para serem analisados pela etapa de extração de características. Para transformar uma imagem monocromática de 256 tons de cinza em binária podem ser utilizadas técnicas de *thresholding* global e adaptativo.



Figura 3.5: Exemplo de imagem binária.

3.2.1.1 *Thresholding* Global

Thresholding global é um algoritmo simples que depende de um único parâmetro denominado limiar. O limiar é um valor de intensidade de pixel utilizado como base de comparação. Sendo assim, em uma imagem monocromática de 256 tons de cinza, todo valor de pixel da imagem de entrada que se encontra abaixo do limiar é colocado como 0 e todo valor acima do limiar é colocado como 255 na imagem

resultante. Tendo v como o valor do pixel na imagem original e t como limiar, o novo valor de pixel computado é:

$$f(v) = \begin{cases} 255, & v \geq t \\ 0, & v \leq t \end{cases}$$

Na Figura 3.6 é apresentado um exemplo de utilização da função *globalThreshold* que pertecem ao módulo de processamento de imagens do Horus.

```
1. from horus.core.processingimage import image, processingimage
2.
3. img = image.Image(path="/tmp/horus.png")
4. img = processingimage.globalThreshold(img)
5. img.save("/tmp/output.png")
```

Figura 3.6: Utilização do *threshold* global implementado no *toolkit* Horus.

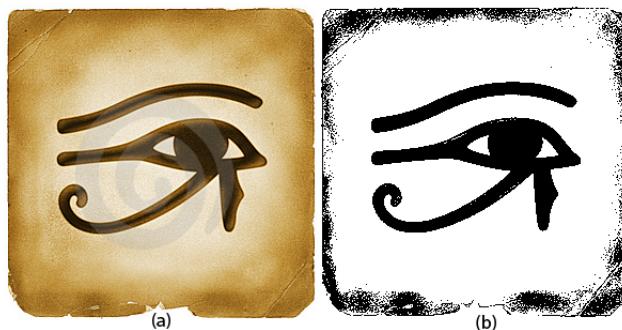


Figura 3.7: (a) Imagem original (b) Resultado do processamento

A Figura 3.7 exibe o resultado do processamento do algoritmo *threshold* global apresentado na Figura 3.6.

Uma das formas de se obter o valor do limiar t é através de uma inspeção visual do histograma da imagem. Uma vez que o mesmo limiar é utilizado para toda a imagem, o *thresholding* global pode falhar algumas vezes. No caso de imagens com diferenças de iluminação ou parcialmente sombreadas, informações que não pertencem ao plano

de fundo podem ser apagadas. A Figura 3.8 mostra um exemplo de aplicação de *thresholding* global com imagens sombreadas.

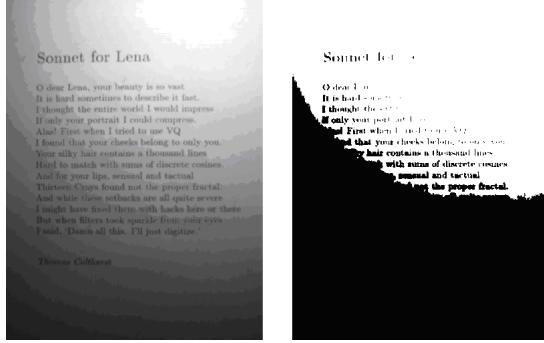


Figura 3.8: Imagem original (à direita) e imagem binarizada com *thresholding* global (à esquerda).

Em casos onde a imagem se encontra parcialmente sombreada, a melhor opção é utilizar a técnica de *thresholding* adaptativo.

3.2.1.2 *Thresholding* Adaptativo

No caso de imagens parcialmente sombreadas ou não uniformemente iluminadas, o algoritmo de *thresholding* global falha devido à utilização de um único limiar para a imagem inteira. Isso pode causar o efeito apresentado na Figura 3.8. O algoritmo de *thresholding* adaptativo resolve esse problema calculando o limiar para cada pixel da imagem com base na sua vizinhança. Há duas abordagens principais de *thresholding* adaptativo: Chow e Kaneko [Chow e Kaneko 1972] e *thresholding* local [Gonzales e Woods 1992].

- Abordagem Chow e Kaneko: esse método, assim como o de *thresholding* local, se baseia na teoria de que regiões menores da imagem têm maior probabilidade de possuírem uma iluminação aproximadamente uniforme. Com isso, o método de Chow e Kaneko divide a imagem em um vetor de sub-imagens sobrepostas e

então encontra o limiar ótimo para cada sub-imagem através da análise de seus histogramas. O limiar para cada pixel é encontrado interpolando os resultados extraídos das sub-imagens. Apesar de gerar resultados satisfatórios, o método de Chow e Kaneko possui um alto custo computacional. Na Figura 3.9 é mostrado como o limiar de uma sub-imagem é localizado e a posterior interpolação dos valores para encontrar o limiar de cada pixel na sub-imagem.

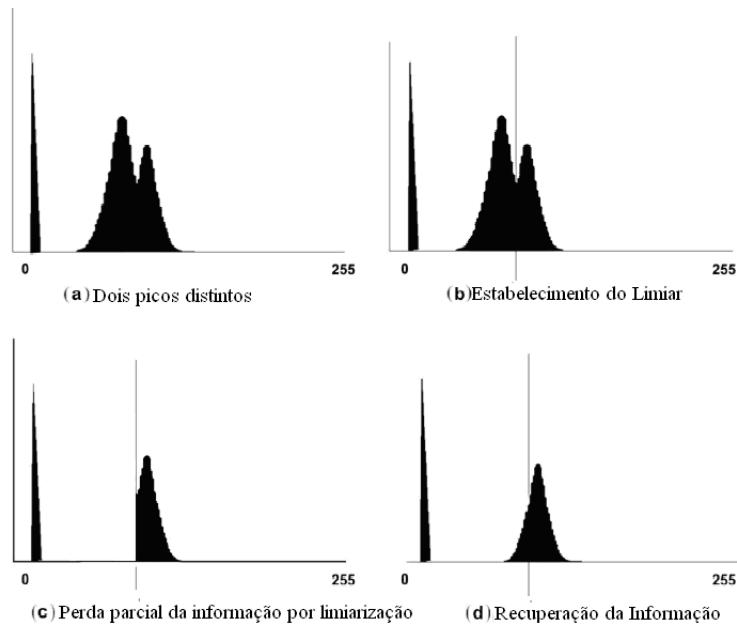


Figura 3.9: Localização do limiar por Chow e Kaneko.

- *Thresholding Local*: a segunda forma de encontrar o limiar de cada pixel é uma inspeção estatística da sua vizinhança, levando em consideração que os vizinhos de um pixel tendem a possuir maior chance de ter uma iluminação mais uniforme. No método de *thresholding* local, o limiar de um pixel é definido com base nos valores de um número predeterminado de vizinhos do mesmo. Dessa forma, aplica-se uma operação estatística nos valores da vizinhança de um pixel a fim de obter o seu limiar t . Essas operações podem ser a média, a mediana ou a média dos valores mínimo e máximo da vizinhança. Na Figura 3.10 é apresentado o resultado do algoritmo de *threshold* local.

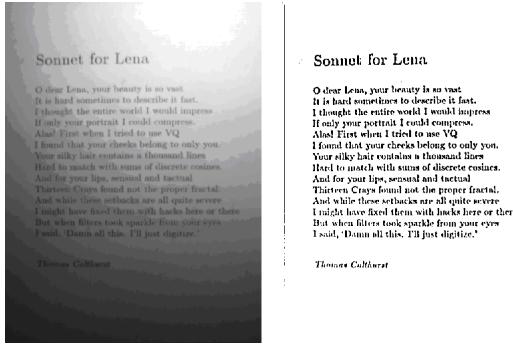


Figura 3.10: Imagem original (à direita) e resultado da limiarização com *thresholding local* (à esquerda).

3.2.2 Skeletonization

Skeletonization (Esqueletonização) é o processo de remoção dos pixels de uma imagem, o máximo quanto possível, de forma a preservar a estrutura básica ou esqueleto da imagem. O esqueleto extraído deve ser o mais fino quanto possível (largura de um pixel), conectado e centralizado. Quando estas propriedades são satisfeitas, o algoritmo deve parar. As Figuras 3.11 e 3.12 mostram exemplos de imagens e seus respectivos esqueletos.

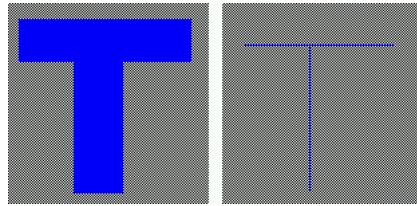


Figura 3.11: Imagem de um "T" e seu respectivo esqueleto.

Normalmente, o esqueleto de uma imagem enfatiza as propriedades geométricas e topológicas dos padrões e é extraído quando se deseja preservar as características estruturais da imagem, como por exemplo, junções, *loops* e terminações de linha. Essas características podem ser extraídas do esqueleto para serem utilizadas, posteriormente, em um processo de reconhecimento e classificação de formas através de



Figura 3.12: Imagem de um ”B” (preto) e seu respectivo esqueleto (branco).

técnicas de inteligência computacional.

O algoritmo de *skeletonization* implementado no Horus utiliza o conceito de ”*fire front*”. Esse conceito realiza a remoção iterativa dos pixels da borda dos padrões até que as condições de conectividade, centralização e espessura do esqueleto sejam satisfeitas. Esse algoritmo, denominado algoritmo de Hilditch, é um processo iterativo em que se aplicam sucessivamente dois passos aos pixels pertencentes à borda de um padrão. O primeiro passo concentra-se em selecionar os pixels das bordas que serão removidos e marcá-los para deleção. O segundo passo é remover todos os pixels marcados para deleção no passo anterior. A Figura 3.13 ilustra os oito vizinhos do pixel p_1 .

P9	P2	P3
P8	P1	P4
P7	P6	P5

Figura 3.13: 8-vizinhança do pixel p_1 .

A fim de estabelecer as condições para que um pixel da borda seja marcado para deleção, serão definidas duas funções:

- $B(p_1)$: número de vizinhos pretos do pixel p_1 .
- $A(p_1)$: número de transições de preto para branco (0 para 255) na seqüência $p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$.

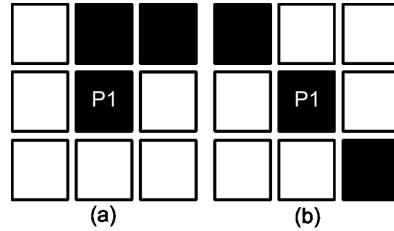


Figura 3.14: Exemplos das funções: (a) $B(p_1) = 2$, $A(p_1) = 1$ b) $B(p_1) = 2$, $A(p_1) = 2$.

A Figura 3.14 mostra exemplos dessas duas funções.

Há duas versões do algoritmo de Hilditch, uma usando uma janela 4×4 e outra usando uma janela 3×3 , nesse trabalho foi utilizada uma janela 3×3 . Utilizando as funções apresentadas acima, o algoritmo de Hilditch verifica os pixels pretos e marca para deleção aqueles que satisfazem as quatro seguintes condições:

- $2 \leq B(p_1) \leq 6$: essa condição assegura que o número de vizinhos pretos de um pixel seja maior ou igual a 2 e menor ou igual a 6. Isso garante que nenhuma terminação de linha ou pixel isolado, seja deletada e que o pixel em questão seja um pixel de fronteira.

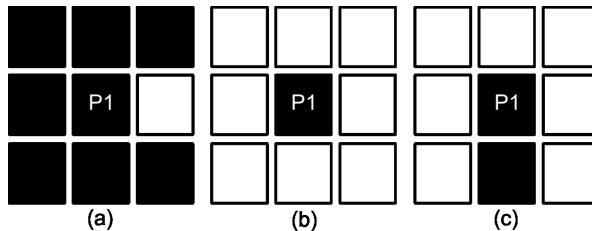


Figura 3.15: (a) $B(p_1) = 7$ (b) $B(p_1) = 0$ (c) $B(p_1) = 1$.

A Figura 3.15 apresenta três condições em que um determinado pixel p_1 não deve ser deletado. Quando $B(p_1)$ é igual a 7, o pixel não é um bom candidato,

pois, sua deleção pode quebrar a conectividade do padrão. Quando $B(p_1)$ é igual a 1, significa que o pixel p_1 é uma terminação de linha e já faz parte do esqueleto, portanto, não deve ser removido. Quando $B(p_1)$ é igual a 0 significa que o pixel p_1 é um pixel isolado e também não deve ser removido.

- $A(p_1) = 1$: essa condição representa efetivamente um teste de conexão. Os casos em que $A(p_1)$ é maior que 1, a deleção do pixel p_1 causa uma quebra na conectividade do padrão, como mostra a Figura 3.16.

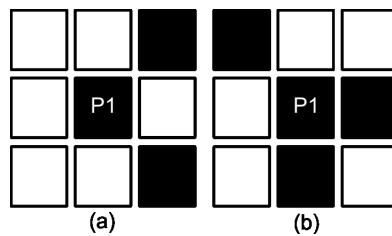


Figura 3.16: Exemplos onde $A(p_1)$ é maior que 1.

- $p_2 + p_3 + p_8 \geq 255$ ou $A(p_2) \neq 1$: essa condição assegura que linhas verticais com largura de dois pixels não serão inteiramente removidas pelo algoritmo. A Figura 3.17 apresenta uma situação em que a condição acima é satisfeita.

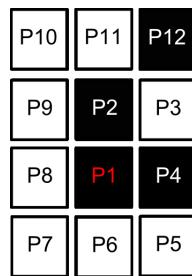


Figura 3.17: Exemplo de situação em que linhas verticais com largura de dois pixels não serão inteiramente removidas.

- $p_2 + p_4 + p_6 \geq 255$ ou $A(p_4) \neq 1$: essa condição assegura que linhas horizontais com largura de dois pixels não serão inteiramente removidas pelo algoritmo. A Figura 3.18 apresenta uma situação em que a condição acima é satisfeita.

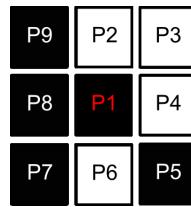


Figura 3.18: Exemplo de situação em que linhas horizontais com largura de dois pixels não serão inteiramente removidas.

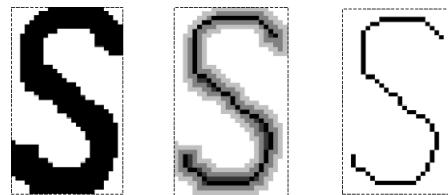


Figura 3.19: (a) padrão de entrada do algoritmo (b) deleção iterativa dos pixels das bordas (c) resultado após a execução do algoritmo.

A cada iteração do algoritmo, os pixels das bordas são analisados, alguns deles são marcados para deleção e então deletados. A Figura 3.19 ilustra o processo iterativo do algoritmo, onde, os pixels deletados em cada iteração são representados pelas diferenças nos tons de cinza da imagem.

O algoritmo de Hilditch é menos custoso do que o algoritmo de transformação de eixo mediano. Porém, esse algoritmo não funciona perfeitamente para todos os padrões. A Figura 3.20 apresenta dois tipos de padrões que são completamente erodidos pelo algoritmo.

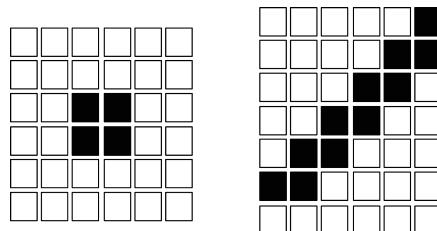


Figura 3.20: Padrões completamente erodidos pelo algoritmo de Hilditch.

3.3 Visão

O Sistema de Visão é um dos mais complexos e completos do ser humano, pois fornece um conjunto de informações necessárias à interação do homem com o ambiente. Tal processo se inicia com a captação dos estímulos luminosos do ambiente formando uma imagem, que juntamente aos outros estímulos captados por demais sensores do corpo (som, temperatura, pressão, umidade, cheiro, etc) e as informações contidas na memória, compõem uma cena compreendida pelo cérebro.

Esse módulo tem como principal objetivo o reconhecimento de padrões. Na aplicação Ariadnes, desenvolvida neste trabalho, o padrão a ser reconhecido é uma placa com o nome dos locais do ambiente e setas que indicam as direções dos mesmos. Para reconhecimento de uma placa é necessário identificar algumas características de uma imagem, que servirão de padrões de entrada para uma rede neural.

3.3.1 Extração de características

Para realizar o reconhecimento de objetos em uma cena, é necessário extrair características das imagens desse objeto, de forma a identificá-lo, independentemente das variações com que ele possa ocorrer na imagem. O *toolkit* Horus apresenta alguns algoritmos para extração de características, os principais deles serão explicados nos itens abaixo.

- Matriz de Pixel: a maneira mais simples de extrair características de um *bitmap* é associar a luminância de cada pixel com um valor numérico correspondente no vetor de características.

Esse método, apesar de simples, possui alguns problemas que podem torná-lo inadequado para o reconhecimento de caracteres. O tamanho do vetor é igual à altura do *bitmap* multiplicado pela sua largura, portanto, *bitmaps* grandes produzem vetores de características muito longos, o que não é muito adequado

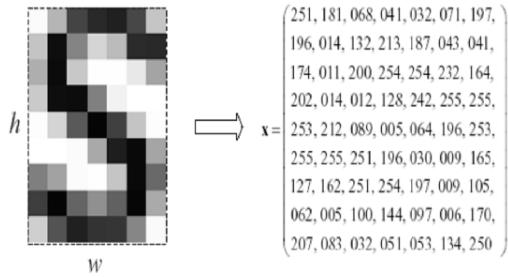


Figura 3.21: Matriz de pixel de um *bitmap*.

para o reconhecimento. Logo, o tamanho do *bitmap* é uma restrição para esse método. Além disso, este método não considera a proximidade geométrica dos pixels, bem como suas relações com a sua vizinhança. No entanto, este método pode ser adequado em situações onde o *bitmap* do caractere se encontra muito opaco ou muito pequeno para a detecção de arestas.

- Intensidade de pixel por região: o objetivo deste método de extração de características consiste em extrair a intensidade de pixels pretos em cada região da imagem [Juntanasub e Sureerattanan 2005]. Para isso, divide-se a imagem em 5×5 blocos. Para cada bloco, calcula-se a intensidade de pixels pretos. Isso pode ser calculado pela seguinte equação.

$$\sum_{i=0}^L \sum_{j=0}^M f(x_i, y_j)$$

Onde L M representam a altura e largura de cada bloco, respectivamente.

- Histograma de Arestas por Regiões: esse método extrai o número de ocorrências de determinados tipos de arestas em uma região específica do *bitmap*. Isso torna o vetor de características desse método invariante com relação à disposição das arestas em uma região e a pequenas deformações do caractere. Sendo o *bitmap* representado pela função discreta $f(x, y)$, largura w e altura h , onde $0 \leq x < w$ e $0 \leq y < h$. Primeiramente é realizada a divisão do *bitmap* em seis regiões (r_0, r_1, \dots, r_5) organizadas em três linhas e duas colunas. Outros quatro layouts podem ser utilizados para a divisão do *bitmap* em regiões.

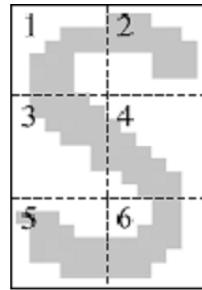


Figura 3.22: Layout com seis regiões em três linhas e duas colunas.

Definindo a aresta de um caractere como uma matriz 2×2 de transições de branco para preto nos valores dos pixels, têm-se quatorze diferentes tipos de arestas, como ilustrado na Figura 3.23.

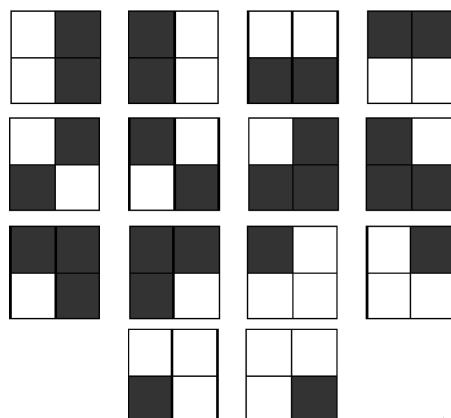


Figura 3.23: Quatorze diferentes tipos de arestas.

O vetor de ocorrências de cada tipo de aresta em cada sub-região da imagem é normalmente muito longo o que não é uma boa prática em reconhecimento de padrões, onde o vetor de características deve ser tão menor quanto possível. Com isso, pode-se agrupar tipos de arestas semelhantes para reduzir o tamanho do vetor de características. Por questões de simplicidade, o agrupamento dos tipos de aresta será desconsiderado no algoritmo de extração de características. Sendo n igual ao número de tipos de arestas diferentes, onde h_i é uma matriz 2×2 que corresponde ao tipo específico de aresta, e p igual ao número de regiões

retangulares em um caractere têm-se:

$$\begin{array}{ll}
 h_0 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} & h_1 = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \\
 h_4 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & h_5 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\
 h_8 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} & h_9 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \\
 h_{11} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} & h_{12} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}
 \end{array}
 \quad
 \begin{array}{ll}
 h_2 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} & h_3 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \\
 h_6 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} & h_7 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \\
 & h_{10} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \\
 & h_{13} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}
 \end{array}$$

Figura 3.24: Matrizes referentes aos tipos de arestas.

O vetor de características de saída é ilustrado pelo padrão abaixo. A notação $h_j @ r_i$ significa "número de ocorrências de um tipo de aresta representado pela matriz h_j na região r_i ", 3.25

Outra forma de extração de características é a análise estrutural do padrão. Através desse tipo de extração é possível diferenciar padrões por suas características mais substanciais. No caso de reconhecimento de caracteres, a análise estrutural leva em consideração estruturas mais complexas, como junções, terminação de linha e *loops*.

- Terminação de Linha: é representada por um ponto que possui exatamente um vizinho de pixel preto na 8-vizinhança (Figura 3.26 (c)).
- Junções: consiste em um ponto que possui pelo menos três pixels pretos na 8-vizinhança. No presente trabalho, consideraram-se apenas dois tipos de junções: com três e quatro vizinhos (Figura 3.26 (a),(b)).

$$\mathbf{x} = \underbrace{\left(h_0 @ r_0, h_1 @ r_0, \dots, h_{\eta-1} @ r_0 \right)}_{\text{Região } r_0}, \underbrace{\left(h_0 @ r_{\rho-1}, h_1 @ r_{\rho-1}, \dots, h_{\eta-1} @ r_{\rho-1} \right)}_{\text{Região } r_{\rho-1}}$$

Figura 3.25: Vetor de Características.

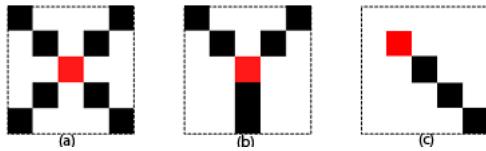


Figura 3.26: (a) Exemplo de junção com quatro vizinhos (b) Exemplo de junção com três vizinhos (c) Exemplo de terminação de linha.

- *Loops*: esta é a característica estrutural mais complexa de ser extraída em um caractere. Neste trabalho, o processo de contagem de *loops* trabalha com a imagem negativa do caractere (Figura 3.27), ou seja, o fundo da imagem é representado pela cor preta, enquanto que o caractere é representado pela cor branca. O número de loops pode ser calculado como o número de grupos de pixels pretos na imagem negativa, representado na Figura 3.27 pelos números 1, 2, 3, subtraído de um. Essa subtração é feita para desconsiderar o fundo da imagem como um *loop*.

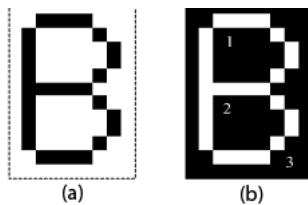


Figura 3.27: (a) Imagem original (b) Imagem negativa.

O *toolkit* Horus também fornece algumas implementações de algoritmos para análise estrutural de caracteres.

3.3.2 Reconhecimento de Objetos

Reconhecimento de objetos é o processo de identificação de um determinado objeto através de suas características. Normalmente, esse processo se inicia com a captura de informações sobre o objeto através de câmeras ou outros tipos de sensores, como

sonares por exemplo. Em seguida, essas informações passam pelo processo de extração de características com a finalidade de se extrair um vetor de informações que identifique unicamente o objeto independente das variações que ele possa apresentar. Por fim, esse vetor de características é passado para o processo de reconhecimento, o qual identifica o objeto através de suas características.

Para tarefas de reconhecimento, o Horus disponibiliza funções para construção e treinamento de redes neurais através da utilização de uma biblioteca denominada FANN (*Fast Artificial Neural Network*). O FANN é uma biblioteca de código aberto implementada em linguagem C que fornece conectores para diversas linguagens de alto nível, dentre elas pode-se citar: Java, C++, Python e Ruby.

Outra funcionalidade disponibilizada pelo Horus para reconhecimento de objetos é o módulo de OCR. Esse módulo utiliza uma engine OCR *Open Source* chamada de Tesseract. Essa engine está sob a licença Apache e é escrita nas linguagens de programação C e C++. O módulo OCR do Horus pode ser utilizado em aplicações onde haja a necessidade de se reconhecer textos existentes em imagens.

O módulo OCR é utilizado nas aplicações ANPR e Ariadnes. No ANPR, esse módulo é utilizado para reconhecer o texto que se encontra nas placas dos automóveis. Já na aplicação Ariadnes, o agente inteligente utiliza esse módulo para reconhecer os textos que se encontram nas placas informativas presentes no ambiente.

3.4 Mapeamento e Navegação

Chamamos de mapeamento o processo de identificar locais no ambiente do simulador e representá-los em um grafo. O mapeamento no Horus utiliza uma técnica genérica denominada SLAM. Nessa técnica, um agente consegue realizar o mapeamento e a localização no ambiente de forma simultânea. Os dispositivos utilizados pela implementação da técnica SLAM são lasers, para identificar obstáculos, e um odômetro, para medir distâncias percorridas.

O SLAM é composto por vários procedimentos interligados. Cada um desses procedimentos pode ser implementado de diversas formas. Dentre os procedimentos implementados no Horus, podemos citar:

1. *Landmark Extraction*: procedimento responsável pela extração de marcos no ambiente.
2. *Data Association*: procedimento que associa os dados extraídos de um mesmo marco por diferentes leituras de lasers.
3. *State Estimation*: procedimento responsável por estimar a posição atual do robô com base em seu odômetro e nas extrações de marcos no ambiente.
4. *State Update*: procedimento que atualiza o estado atual do agente.
5. *Landmark Update*: procedimento que atualiza as posições dos marcos no ambiente em relação ao agente.

Neste trabalho, a proposta utilizada é mapear o ambiente através de um grafo conexo, cujos nós referem-se a: entradas/saídas do ambiente, acessos aos cômodos, obstáculos fixos e esquinas. O peso das arestas é calculado de acordo com a distância euclidiana entre os vértices.

O problema de navegação consiste na localização e definição do caminho que o agente deve seguir. Após a construção de uma representação do ambiente em forma de um grafo, o agente é capaz de se localizar e se movimentar pelo ambiente através dos vértices e arestas, previamente mapeados no grafo. Para a utilização de grafos, o Horus fornece classes para sua construção e algoritmos para cálculo de caminho mínimo.

3.5 Aplicações desenvolvidas com o Horus

As aplicações desenvolvidas nesse projeto têm o objetivo de validar e demonstrar a utilidade do *toolkit* Horus no desenvolvimento de aplicações envolvendo agentes in-

teligentes, visão computacional, mapeamento automático de ambientes e matemática. São três as aplicações desenvolvidas no projeto como um todo, porém, neste trabalho serão detalhadas somente as aplicações que possuem aspectos de visão computacional.

As aplicações desenvolvidas foram: um sistema de reconhecimento automático de placas de automóveis denominado *PyANPR*, descrito no Capítulo 4, um simulador para mapeamento automático de ambientes desconhecidos através da técnica SLAM, chamado *Teseu* e um simulador de movimentação autônoma em ambientes desconhecidos, utilizando visão computacional, denominado *Ariadnes*.

Teseu é um sistema que simula a movimentação autônoma de um agente inteligente em um ambiente desconhecido. Nele é possível simular o comportamento de um robô real no que tange mapeamento e navegação. O agente tem o objetivo de explorar todo o ambiente utilizando técnicas de localização, mapeamento e exploração implementadas no Horus. A construção do ambiente virtual foi realizada utilizando um software de modelagem 3D *Open Source* chamado Blender 3D. Esse é um produto criado pela *Blender Foundation*, feito na linguagem de programação Python e está sob a licença GNU(*General Public License*). Outra ferramenta utilizada na construção dos simuladores é o Panda3D, criada pela equipe *Walt Disney* para renderização de jogos e ambientes em terceira dimensão. Esse produto também foi criado na linguagem de programação Python e encontra-se sob a licença de uso BSD(*Berkeley Software Distribution*). O uso do Panda3D no Horus restringe-se apenas na manipulação e renderização do ambiente e atores criados previamente no Blender3D.

O simulador Ariadnes foi construído utilizando as mesmas ferramentas utilizadas na construção do Teseu, porém, com objetivo e técnicas diferentes. O Ariadnes será mais bem descrito no Capítulo 5.

Capítulo 4

PyANPR

Nesse capítulo serão abordados os principais conceitos sobre um sistema de Reconhecimento Automático de Números de Placas (*Automatic Number Plate Recognition - ANPR*). Nesse trabalho será adotado o acrônimo do termo em inglês ANPR. Atualmente, ANPR se tornou a principal técnica para muitos sistemas de transporte automatizado, como monitoramento de tráfego em estradas, pagamento automático de pedágios e controle de acesso a pontes e estacionamentos. Esse tipo de sistema consiste em, a partir de uma imagem de um carro, localizar a placa e identificar os caracteres presentes nesta. Com base no reconhecimento dos caracteres presentes na placa, é possível identificar o carro e levantar informações sobre o mesmo automaticamente.

Sistemas ANPR também são conhecidos como: *Automatic licence plate recognition* (ALPR), *Automatic vehicle identification* (AVI), *Car plate recognition* (CPR), *Licence plate recognition* (LPR) e *Lecture Automatique de Plaques d'Immatriculation* (LAPI). A idéia de um sistema ANPR foi criado em 1976, pelo Departamento de Desenvolvimento Científico da Polícia no Reino Unido, com o objetivo de identificar carros roubados. O primeiro protótipo desenvolvido ficou pronto em 1979 e foi desenvolvido pela empresa Britânica EMI *Electronics*. Esse sistema começou a ser testado na rodovia A! e no Túnel Dartford, ambos na Inglaterra. Somente em 1981 ocorreu

a identificação de um carro roubado, seguido da prisão do delinquente, através da utilização deste sistema.

Há algum tempo atrás, a identificação de veículos era um trabalho completamente manual, porém, com o crescimento do tráfego de veículos nas rodovias, a tarefa de identificação manual de veículos se tornou inviável. Com a evolução dos componentes eletrônicos, foi criado um sistema AVI, onde, o código do veículo era armazenado em um *transponder* que era instalado em cada veículo. Quando o veículo passava por alguns pontos de controle da rodovia, unidades de leitura identificavam a sua presença e realizavam a leitura do seu código, presente no *transponder*. Muitos sistemas AVI eram equipados com um sistema de captura de vídeo, aumentando o seu custo. Com o aumento considerável de veículos, a produção e manutenção em larga escala desses sistemas passaram a se tornar extremamente custosa. Com a evolução das técnicas de visão computacional, tornou-se possível a identificação de veículos pela leitura de suas placas através de câmeras de monitoramento, diminuindo o custo para implantação desse tipo de sistema.

Na aplicação PyANPR, desenvolvida nesse trabalho, existem três etapas entre a inclusão da imagem de um carro até a extração do caractere. A primeira consiste na localização da placa. A segunda consiste num pré-processamento da placa, para binarizar a imagem e remover ruídos. E a terceira etapa consiste em aplicar uma função de OCR a fim de identificar os caracteres da placa. Cada uma das etapas serão descritas nas seguintes seções.

4.1 Localização da Placa

O processo de localização é a etapa mais importante e vulnerável de todo o ANPR. Existem duas suposições básicas em sistemas ANPR [SHAPIRO, GLUHDEV e DIMOV 2006]

- As placas são orientadas horizontalmente;

- A placa é caracterizada por uma frequência de alterações entre o caractere (*Foreground*) e a placa (*Background*).

Essa etapa consiste em definir uma região (normalmente um retângulo com a orientação da imagem, mas também pode ser um retângulo rotacionado de modo a se ajustar otimamente a placa) que contenha a imagem. Quanto melhor for o ajuste entre o retângulo e a placa, melhor será o processo de reconhecimento dessa. O processo de geração desse retângulo, utilizado nesse trabalho, foi dividido em duas etapas. Na primeira aplica-se um corte vertical definindo o intervalo de linhas que inclua a placa. Essa região foi denominada de *Band*. A segunda etapa é semelhante, realiza-se um corte horizontal no *band* definindo a região que contém a placa, denominada de *Plate*. A Figura 4.1 mostra essas duas etapas.



Figura 4.1: (a) Fotografia de um carro (b) *Band* (c) *Plate*.

O método de localização do *Band* inicia-se com um pré-processamento na imagem do carro. Esse pré-processamento consiste em aplicar um filtro de detecção de arestas verticais (filtro de sobel - vertical) em uma imagem em escala de cinza. Esse processo é mostrado na Figura 4.2.

Após o pré-processamento aplica-se uma projeção vertical. Essa projeção consiste em somar a intensidade de cor dos pixels da imagem filtrada linha a linha. Matematicamente, pode-se definir a projeção vertical como uma função $pv : 0, \dots, h - 1 \rightarrow \mathbb{N}$ onde $pv(i) = \sum_{j=0}^{w-1} I(i, j)$ sendo $I : [0, w] \times [0, h] \rightarrow 0, 1, \dots, 255]$ a função de intensidade de luz da imagem pré-processada cuja largura é w e altura é h . A Figura 4.3

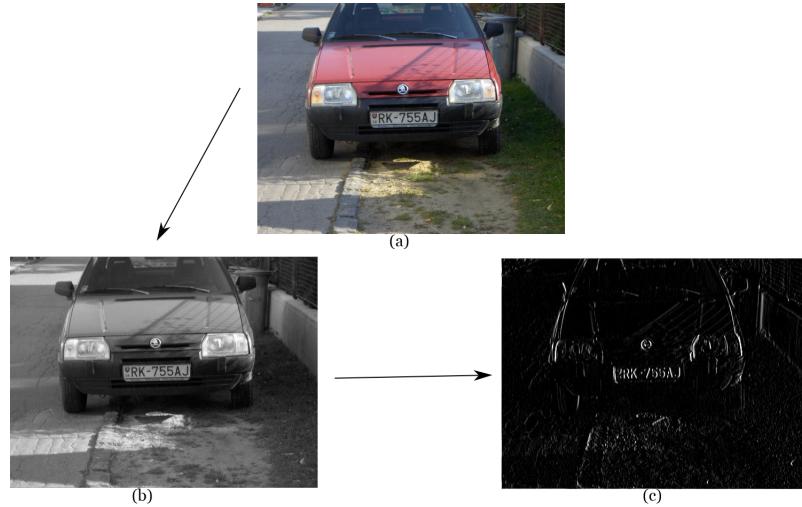


Figura 4.2: (a) Fotografia de um carro (b) imagem convertida para escala de cinza (c) imagem filtrada por Sobel-Vertical.

mostra o exemplo de uma projeção vertical de uma imagem de um carro.

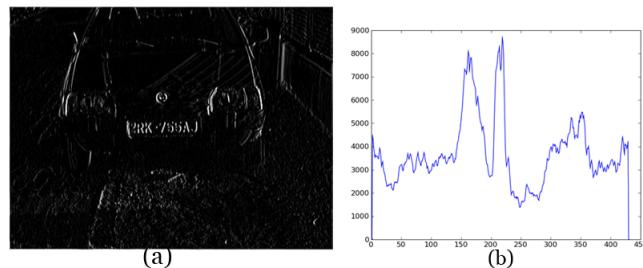


Figura 4.3: (a) Imagem filtrada por Sobel-Vertical (b) Projeção vertical.

Analogamente, no processo de detecção do *plate* aplica-se um pré-processamento, que difere-se do pré-processamento do *band* somente no fato que o filtro de detecção de arestas usado é o filtro de sobel - horizontal. No resultado do pré-processamento aplica-se uma projeção horizontal. Essa projeção é análoga a projeção vertical. Somam-se, coluna a coluna, as intensidades de cor de cada pixel. Matematicamente a projeção horizontal é dada pela função $ph : 0, \dots, w - 1 \longrightarrow \mathbb{N}$ onde $ph(j) = \sum_{i=0}^{h-1} I(i, j)$. A Figura 4.4 mostra o exemplo de uma projeção horizontal de uma

imagem de um *band*.

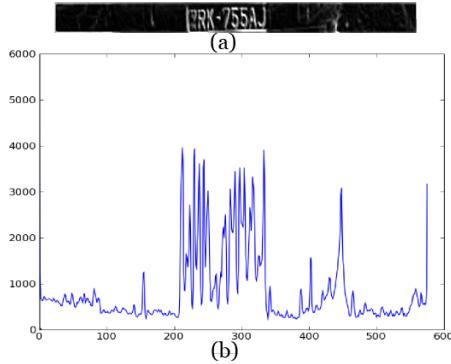


Figura 4.4: (a) Imagem do band pré-processada (b) projeção horizontal

Normalmente o máximo global da projeção vertical está na região que contém a placa. Isso se deve ao fato que em uma imagem de carro filtrada por Sobel-Vertical, normalmente, a região da placa é a região que contém mais arestas e, portanto, essa é a região que contém o máximo global da função. Sendo essa uma função discreta, o processo de detecção de máximo global pode ser determinado por inspeção, sem grandes custos computacionais. A esse valor de máximo global, denominaremos de *pico*.

Para definir o *band* é necessário encontrar a linha inicial e a linha final da imagem do carro que o determinarão. Para tal é necessário definir uma região próximo ao pico. A abordagem utilizada nesse projeto para definir essa região divide-se em duas etapas. Na primeira, aplica-se uma suavização na projeção vertical. Para isso aplica-se um processo de convolução na imagem da função projeção. Essa convolução calcula a média de valores em uma determinada vizinhança. Após isso, na segunda etapa aplica-se uma função *threshold*, que consiste em, dado um valor x que nesse caso é a média dos valores da imagem da projeção, para cada elemento da imagem da projeção verifica se este é menor que x : caso seja, seu valor é zerado. Após

essa etapa, tem-se uma função cuja imagem está dividida em intervalos não nulos. Normalmente a placa está no intervalo que contém o pico. Logo, basta extrair o índice inicial e o final que definem esse intervalo e considerá-los como as linhas inicial e final, respectivamente, usadas para o corte do *band*. A Figura 4.5 mostra o processo de projeção para determinar o *band*.

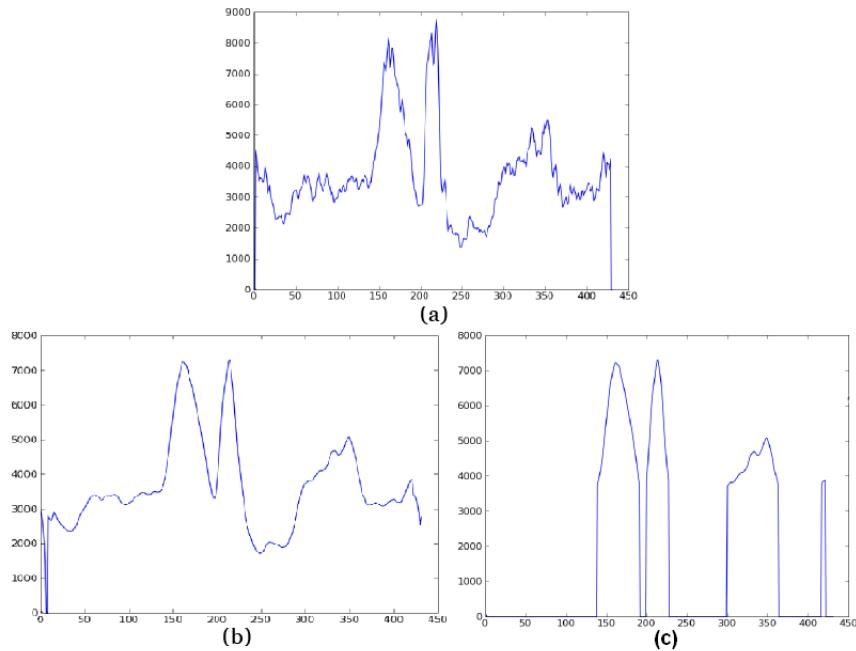


Figura 4.5: (a) Imagem original (b) Projeção vertical suavizada (c) Aplicação do *Threshold*.

O processo de detecção do *plate* é semelhante. Aplica-se a projeção horizontal na imagem do *band* pré-processada, como dito anteriormente. Após essa etapa, aplicam-se as duas etapas de processamento na projeção, assim como feito na projeção vertical, o que resultará em uma função cuja imagem estará dividida em intervalos não nulos. Normalmente a placa está no intervalo de maior comprimento.

Como dito ao longo do texto, esse processo não é determinístico. Ele baseia-se na alta probabilidade de acerto verificada empiricamente. Porém, pode-se adotar algumas técnicas caso o *band* adotado não seja o que contenha o pico. Isso pode acontecer

quando alguma outra região da imagem que contenha o carro contenha algum objeto com riqueza de detalhes, ou haja excesso de ruídos concentrado em uma região. Nessas possibilidades haverá região de alta frequência, além da região da placa. No caso do *band*, pode-se ordenar os intervalos obtidos na etapa final após a projeção. Essa ordenação baseia-se em picos locais (picos nos intervalos). Então os intervalos são organizados decrescentemente baseados nos picos locais. Vale destacar que o conjunto dos picos locais não é o conjunto de máximos locais, pois em um intervalo podem existir mais de um máximo local. Com essa ordenação, continua-se o processo a partir do primeiro intervalo. Caso no fim do processo não encontre nenhuma placa, o processo com o *band* é feito na segunda posição. Analogamente o *plate* pode não estar no intervalo de maior comprimento. Dessa forma realiza-se uma ordenação decrescente baseada no tamanho dos intervalos. Outra alternativa é utilizar algum método para estimar a probabilidade de haver uma placa no *band* e no *plate*. Um método que pode ser utilizado é a Transformada de Hough e verificar se existe um quadrilátero na região determinada pelo *band* ou *plate*. Mais uma vez, esse é um método probabilístico. Porém, vale destacar que a taxa de eficiência do PyANPR é satisfatória.

4.2 Pré-processamento da placa

Normalmente a imagem da placa contém ruídos ou está em uma baixa resolução. Esses fatores costumam baixar a taxa de acerto do reconhecimento dos caracteres (OCR). Para melhorar a qualidade da imagem, aumentando a taxa de acerto do reconhecimento de caracteres no PyANPR, adotou-se um processo dividido em três etapas. Na primeira etapa, converte-se a imagem para escala de cinza. Na segunda extrai-se o ruído através de um filtro de convolução cujo núcleo baseia-se na mediana. Por fim aplica-se um *threshold* (seção 3.2). Um possível valor do limiar é 128, mas este pode ser alterado caso insira-se alguma outra etapa intermediária. Uma etapa

que pode ser utilizada é aplicar um filtro que realce arestas.

Dessa forma, após o pré-processamento, obtém-se uma imagem binária da placa que servirá de entrada para a próxima etapa, a de reconhecimento de caractere.

4.3 OCR

O *toolkit* Horus disponibiliza diversos algoritmos para implementação de cada uma das etapas que constituem um sistema OCR (seção 2.2.4). Os algoritmos necessários para a construção das etapas de pré-processamento e segmentação dos caracteres estão disponíveis no módulo de processamento de imagens do Horus, alguns desses algoritmos são: detecção de arestas, binarização, suavização, remoção de ruídos, projeções, entre diversos outros algoritmos.

Após a segmentação dos caracteres, extrai-se um vetor numérico (vetor de características) de cada um desses. Tal vetor é utilizado como padrão de entrada para uma rede neural no processo de reconhecimento. O processo de extração de características é uma das etapas mais importantes de um sistema OCR, a qualidade dessa etapa influencia no resultado de todo o sistema. Portanto, a escolha de quais características serão utilizadas como padrão de entrada no processo de reconhecimento é extremamente importante. A construção do vetor de características pode ser feito utilizando os diversos algoritmos implementados no módulo de extração de características do Horus (seção 3.3.1) podem ser: histograma de arestas por regiões, terminação de linha, *loops*, entre outros. Por último, é realizada a etapa de reconhecimento do vetor de características, que finalmente deverá apontar o caractere correto. Uma abordagem usual para a construção da etapa de reconhecimento consiste na utilização de redes neurais artificiais (seção 2.4). O Horus disponibiliza algoritmos para criação e utilização de redes neurais através de uma API *Open Source* chamada *Fast Artificial Neural Network*(FANN).

Atualmente, o *toolkit* Horus não possui um sistema OCR próprio, essa funcionali-

dade será implementada em um trabalho futuro. Porém, esse *toolkit* disponibiliza funcionalidades de um sistema OCR através da utilização de um OCR *Open Source* chamado Tesseract.

4.4 Aplicação Web

Durante o desenvolvimento deste trabalho, mais especificamente do sistema PyANPR, foi desenvolvido um sistema web cujo objetivo é: através do *upload* de uma imagem de um carro, o sistema realizará o ANPR nessa imagem e retornará todas as informações da placa contidas em seu banco de dados. Esse sistema foi construído utilizando um *framework web* de altíssimo nível chamado Django. Esse *framework* é escrito na linguagem de programação Python e é baseado no padrão arquitetural *Model View and Controller*(MVC) [Fowler 2003].

Capítulo 5

Ariadnes

Ariadnes é um sistema que simula a navegação e movimentação autônoma de um agente inteligente em um ambiente completamente desconhecido. Inicialmente, o agente tem o objetivo de chegar a uma determinada sala no ambiente utilizando técnicas de localização e mapeamento de ambientes e de visão computacional. Da mesma forma que o Teseu (seção 3.5), o ambiente e atores do Ariadnes foram modelados utilizando Blender e são manipulados utilizando o Panda3D.

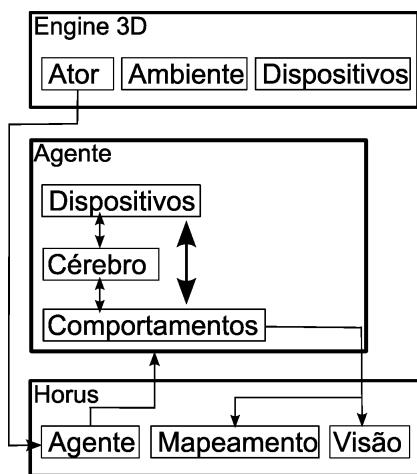


Figura 5.1: Arquitetura conceitual do simulador Ariadnes.

Esse simulador é composto por quatro partes principais baseadas no *toolkit* Horus. A Figura 5.1 apresenta a arquitetura conceitual do simulador Ariadnes com seus principais componentes.



Figura 5.2: Placa informativa presente no ambiente.

As principais partes do simulador Ariadnes são: ambiente, engine 3D, agentes e dispositivos. O ambiente, nesse simulador, é composto basicamente de diversos cômodos interligados por corredores, em alguns pontos desse ambiente existem placas informativas cujo objetivo é orientar o agente durante o mapeamento, como mostra a Figura 5.2.

Por último, agentes e dispositivos são extensões de abstrações fornecidas no módulo *Core* do Horus para implementação de tais partes. O agente configurado com dispositivos emissores de lasers é mostrado na Figura 5.3.

Inicialmente, o agente é configurado com os comportamentos de Navegação, Mapeamento, Localização e Leitura de placas. Após a configuração, o agente é inserido em um ambiente totalmente desconhecido com a missão de chegar a uma sala específica. Na Figura 5.4 é apresentado a vista superior do ambiente utilizado no Ariadnes.

Após o estabelecimento da missão, o agente inicia o mapeamento do ambiente utilizando os algoritmos do módulo Mapeamento do Horus para construir uma representação do ambiente em forma de grafo.

A máquina de estados utilizada no Ariadnes é composta por: mapeamento, navegação,

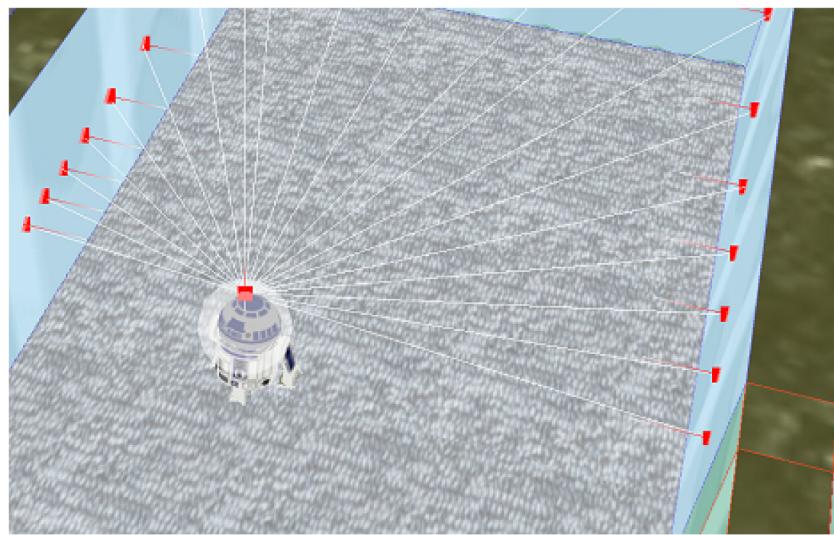


Figura 5.3: Agente configurado com dispositivos de lasers.

reconhecimento de objetos e execução. O agente tem por objetivo partir de um ponto inicial para um ponto final, para isso, ele tenta definir uma rota. Nesse momento, o agente se encontra no estado de navegação. Caso ele não consiga definir uma rota, o estado desse agente muda para mapeamento. Nesse estado, ele explorará o ambiente até que encontre uma placa ou algum ponto já mapeado. Caso encontre a placa, seu estado mudará para reconhecimento de objetos. Nesse estado, caso a placa seja a identificação do cômodo destino, seu estado muda para execução. Caso contrário, o agente muda para o estado de navegação ou retorna para o estado de reconhecimento de objetos, caso a placa identificada seja informativa. No estado de execução, uma tarefa específica, previamente determinada, é realizada. No estado de mapeamento, ao encontrar um ponto já mapeado, ele verifica se agora é possível traçar uma rota até seu objetivo. Caso não seja possível, o agente continua no estado de mapeamento.

O agente é capaz de estimar sua posição local para se localizar globalmente e recuperar-se de possíveis erros de localização. A proposta utilizada para realização das etapas de mapeamento e localização é a técnica SLAM. Durante o mapeamento, uma câmera é utilizada pelo agente para localizar as placas informativas presentes no ambiente.

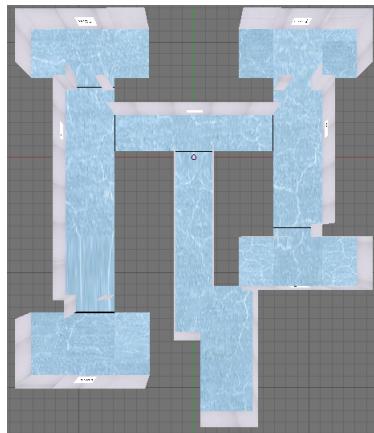


Figura 5.4: Ambiente utilizado no Ariadnes.

Quando este se depara com uma dessas placas, interrompe o processo de mapeamento e utiliza algoritmos de visão para interpretar o conteúdo existente na placa, a fim de estabelecer a direção para a qual ele deve seguir no ambiente. A captura da cena por câmeras permite a utilização de procedimentos de visão computacional, como extração de características, OCR, localização de placas e reconhecimento de informações de direção presentes no ambiente. No ambiente utilizado no Ariadnes existem marcos, localizados no chão, com os objetivos de: auxiliar a localização das placas identificadoras de cômodos e direções e diminuir o tempo necessário no processo de localização das placas. Para isso, o agente é configurado com duas câmeras: uma delas apontando para o chão, utilizada para localizar os marcos no chão do ambiente; e a segunda câmera é apontada para frente do agente, simulando sua visão, essa câmera é utilizada no processo de localização e reconhecimento das placas. A Figura 5.5 mostra uma foto capturada por cada uma das câmeras citas.

Ao encontrar um marco, o agente fotografa a placa. Com essa imagem, inicia-se o processo de reconhecimento da placa. Esse processo consiste nas etapas de: local-

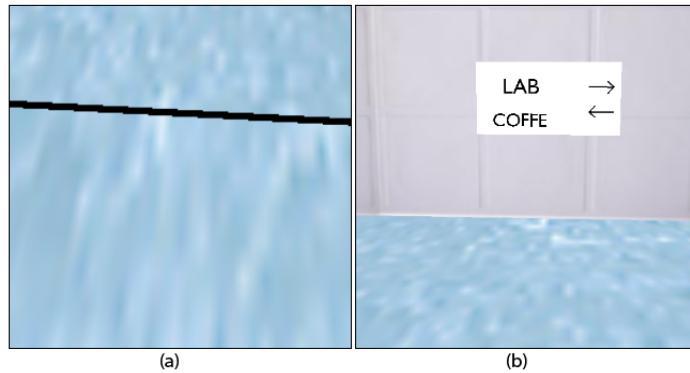


Figura 5.5: Imagens capturadas no ambiente pelas câmeras do robô (a) Câmera para baixo. (b) Câmera frontal.

ização e segmentação da placa, distinção de setas e textos, reconhecimento do texto, identificação da direção da seta. O processo de localização da placa pode ser resolvido em duas etapas. A primeira etapa consiste em localizar a região da placa na imagem. Inicialmente, aplica-se o filtro de Sobel, a fim de detectar as arestas horizontais e verticais. Após a detecção das arestas, pretende-se encontrar a região com maior densidade de arestas, pois, geralmente a placa se encontra nessa região. Esse processo é análogo a localização de placas descrito na seção 4.

A placa é constituída de várias linhas, cada linha contém duas colunas. A primeira coluna de cada linha fornece o nome de um determinado local no ambiente, enquanto que, a segunda coluna fornece a direção desse local em forma de setas. Como apresentado na Figura 5.2.

Para realizar o reconhecimento dos locais e direções apresentados na placa, é necessário uma etapa de segmentação dessa placa em duas regiões, textos e direções. Para isso, aplica-se a projeção horizontal para identificar cada região, a Figura 5.6 mostra o resultado da projeção horizontal na imagem da placa.

Após a etapa de segmentação da imagem em duas regiões, separa-se cada coluna em linhas. Isso é feito através da projeção vertical, como mostra a Figura 5.7

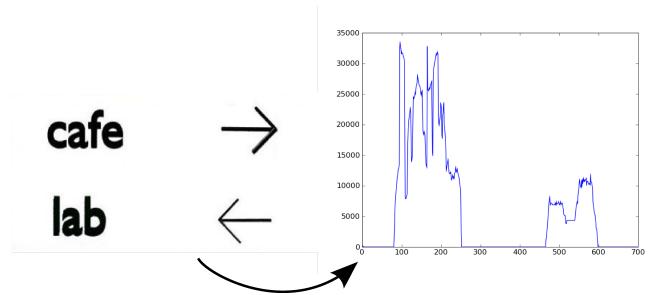


Figura 5.6: Projeção horizontal de uma placa informativa.

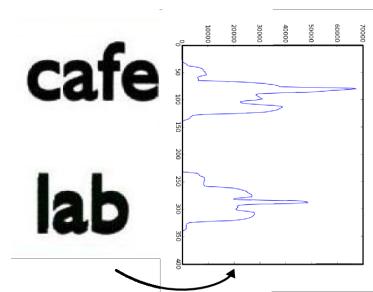


Figura 5.7: Projeção vertical das colunas da placa.

Para a identificação do texto e da seta de cada linha, utilizam-se os métodos de extração de características do Horus, utilizando-os para criação dos vetores de características que servirão como padrões de entrada para uma rede neural artificial, também implementada no Horus. Por fim, o agente identifica os locais e suas respectivas direções, presentes na placa, e define a direção para qual ele deve seguir baseado em seu objetivo pré-estabelecido.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado o *toolkit* Horus, uma ferramenta utilizada para desenvolvimento e controle de aplicações que envolvem agentes inteligentes. Esse *toolkit* apresenta algoritmos que são comumente utilizados em aplicações que envolvem visão computacional e mapeamento automático de ambientes. Além disso, o Horus fornece abstrações que permitem a criação e configuração de agentes inteligentes com suas principais partes conceituais como, programas de agentes, comportamentos e atuadores. O foco deste trabalho se concentra nos algoritmos utilizados para visão computacional e nas abstrações para a construção de agentes inteligentes fornecidas pelo Horus. É válido destacar que o Horus se encontra disponível para utilização, esse *toolkit* é de código-fonte aberto.

A fim de validar os algoritmos e abstrações fornecidas pelo Horus, foram desenvolvidas, também, as aplicações ANPR, Teseu e Ariadnes. ANPR é uma aplicação para reconhecimento automático de placas de automóveis, onde, a partir de uma imagem de um carro, ela é capaz de extrair a placa de identificação do mesmo utilizando técnicas de visão computacional. Teseu é uma aplicação de mapeamento automático de ambientes por um agente inteligente utilizando a técnica SLAM. Ariadnes é uma aplicação que integra técnicas de visão computacional e mapeamento de ambientes. No Ariadnes, um agente utiliza algoritmos de mapeamento e visão computacional

para explorar um ambiente desconhecido e se orientar através de placas informativas presentes no mesmo. Com o desenvolvimento dessas três aplicações, constatou-se que, com a utilização de seus algoritmos e abstrações, o *toolkit* Horus é capaz de agilizar o desenvolvimento de aplicações que envolvem agentes inteligentes, principalmente no que diz respeito aos aspectos de visão computacional, mapeamento automático de ambientes e matemática.

Como trabalhos futuros, espera-se a eliminação da dependência do OCR Tesseract com a implementação de um OCR próprio do *toolkit* Horus, uma vez que, a maioria dos algoritmos utilizados nessa funcionalidade já se encontram implementados no Horus. Outro trabalho futuro, importante para a continuação do *toolkit* Horus, seria a integração com a biblioteca para visão computacional e processamento de imagens *OpenCV*. Isso tornaria o módulo de visão computacional do Horus mais robusto e adicionaria funcionalidades de mapeamento de ambientes às funcionalidades fornecidas pelo *OpenCV*. Atualmente, o Horus apenas dá suporte a construção de aplicações que envolvem simulação em ambientes virtuais, como trabalhos futuros, seriam desenvolvidos *drivers* para que as abstrações presentes no módulo *Core* do Horus possam ser utilizadas tanto em robôs virtuais como em robôs reais. Esses *drivers* funcionariam como *middlewares*, gerenciando a comunicação entre o *toolkit* Horus e os diversos dispositivos reais, de diversos fabricantes, que são disponibilizados no mercado atualmente. Além disso, outro trabalho futuro é disponibilizar no Horus uma funcionalidade que torne possível o mapeamento do ambiente em 3D. Essa funcionalidade pode ser aplicada em situações onde exista a necessidade de se conhecer ambientes inóspitos ao homem. Nesses casos, robôs reais utilizando o Horus seriam capazes de representar um ambiente real desconhecido em 3D, possibilitando o conhecimento de qualquer ambiente em que seria impossível a exploração por seres humanos.

Referências Bibliográficas

- Arkin 1998 ARKIN, R. *Behavior-based robotics*. 1998. MIT Press.
- BONASSO 1991 BONASSO, R. P. Integrating reaction plans and layered competences through synchronous control. *Twelfth International Joint Conference on Artificial Intelligence*, p. 1225–1231, 1991.
- Braga, Ludermir e Carvalho 2000 BRAGA, A.; LUDERMIR, T.; CARVALHO, A. *Redes Neurais Artificiais, Teoria e Aplicações*. [S.l.]: Livros Técnicos e Científicos, 2000.
- Cagnoni et al. 1993 CAGNONI et al. Neural network segmentation of magnetic resonance spin echo images of the brain. *Jounal of Biomedical Engineering*, v. 15, p. 335–362, 1993.
- Chow e Kaneko 1972 CHOW, C.; KANEKO, T. Automatic boundary detection of the left ventricle from cineangiograms. *Comp. Biomed. Res.*(5),, p. 388–410, 1972.
- CONNEL 1992 CONNEL, J. A hybrid arquitecture applied to robot navigation. *IEEE Int Conf. on Robotics and Automation*, p. 2719–2724, 1992.
- Davison, Cid e Kita 2004 DAVISON, A.; CID, Y.; KITA, N. Real-time 3d slam with wide-angle vision. *IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.
- Dellaert 2005 DELLAERT, F. Square root slam: Simultaneous location and mapping via square root information smoothing. *Robotics: Science and Systems*, 2005.
- Eltoft 1998 ELTOFT, T. A new neural network for cluster-detection-and-labeling. *IEEE Transactions on Neural Networks*, v. 9, p. 1021–1035, 1998.

Fowler 2003 FOWLER, M. *Patterns for Enterprise Application Architeture.* [S.l.]: Bostom, 2003.

FOX et al. 1999 FOX, D. et al. Monte carlo localization: Efficient position estimation for mobile robots. *The National Conference on Artificial Intelligence (AAAI)*, 1999.

GAT 1991 GAT, E. *Reliable Goal-Directed Reactive Control for Real-World Autonomus Mobile Robots.* Tese (Doutorado) — Dept. of Computer Science, Virginia potytechnic Institute and State University,, 1991.

Gomes e Velho 2002 GOMES, J.; VELHO, L. *Computação Gráfica: Imagem.* [S.l.]: IMPA, 2002.

Gomes e Velho 2003 GOMES, J.; VELHO, L. *Fundamentos da Computação Gráfica.* [S.l.]: IMPA, 2003.

Gonzales e Woods 1992 GONZALES, R.; WOODS, R. *Digital Image Processing.* [S.l.]: Addison-Wesley Publishing Company, 1992. 443-452 p.

Guyon et al. 2006 GUYON, I. et al. *A practical guide to splines.* 2006. Springer.

Haykin 1994 HAYKIN, S. *Neural Networks: A comprehensive foundation.* [S.l.]: Macmillan College Publishing., 1994.

Jain, Duin e Mao 2000 JAIN, A. K.; DUIN, R. P.; MAO, J. Statistical pattern recognition: A review. *IEEE Trans*, v. 22, p. 4 – 47, 2000.

Juntanasub e Sureerattanan 2005 JUNTANASUB, R.; SUREERATTANAN, N. Car license plate recognition through haudorff distance technique. *Conference on Tools with Artificial Intelligence*, v. 5, 2005.

Marr 1982 MARR, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* [S.l.]: W.H.Freeman & Co Ltd, 1982.

Mendel e McLaren 1970 MENDEL, J. M.; MCLAREN, R. W. Adaptive learning and pattern recognition systems: theory and applications. *Academic Press.*, 1970.

- Neto 1999 NETO, O. M. F. e H. V. *Processamento digital de imagens.* [S.l.]: Brasport, 1999.
- Nigrin 1993 NIGRIN, A. Neural networks for pattern recognition. *MIT Press*, 1993.
- Pal e Mitra 2004 PAL, S. K.; MITRA, P. *Pattern Recognition Algorithms for Data Mining.* [S.l.]: Chapman & Hall/CRC, 2004.
- Rice, Nagy e Nartker 1999 RICE, S. V.; NAGY, G.; NARTKER, T. A. Optical character recognition: An illustrated guide to the frontier. *Kluwer Academic Publishers, Norwell, Massachusetts*, 1999.
- Russel e Norvig 2003 RUSSEL, S.; NORVIG, P. *Inteligência Artificial.* [S.l.]: Editora Campus, 2003.
- Santos 2007 SANTOS, J. C. *Extração de atributos de forma e seleção de atributos usando algoritmos genéticos para classificação de regiões.* Dissertação (Mestrado) — Instituto Nacional de Pesquisas Espaciais (INPE)., 2007.
- SHAPIRO, GLUHDEV e DIMOV 2006 SHAPIRO, V.; GLUHDEV, G.; DIMOV, D. Towards a multinational car license plate recognition system. *Machine Vision and Applications*, v. 17, p. 173–183, 2006.
- Simpson 1992 SIMPSON, P. Fuzzy min-max neural networks - part1 : Classification. *IEEE Transactions on Neural Networks*, v. 3, p. 776–786, 1992.
- Simpson 1993 SIMPSON, P. Fuzzy min-max neural networks - part2 : Classification. *IEEE Transactions on Neural Networks*, v. 1, p. 32–45, 1993.

Apêndice A

Instalação

O guia de instalação do Horus, assim como a sua lista de dependências, encontra-se disponível no site do projeto. Podendo ser acessado na url <http://code.google.com/p/finishstrike/>.