UNIVERSIDADE CÂNDIDO MENDES

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

DEFINIR O TÍTULO

Campos dos Goytacazes - RJ
Junho - 2009

MÓDULO DE VISÃO DO TOOLKIT HÓRUS

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

Aprovada em ____ de _____ de 2009.

BANCA EXAMINADORA

Prof. D.Sc. Ítalo Matias - Orientador
Doutor pela UFRJ

Prof. D.Sc. Dalessandro Soares
Doutor pela PUC-Rio

Prof. BLABLABLA
Univeridade de Londres

Agradecimentos

Resumo

Palavras-chave:

Abstract

Keywords:

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Existem alguns tipos de ambiente que são inóspitos ao homem. Nesses casos é comum utilizar um robô para explorar e atuar em tais locais. A movimentação desses robôs pode ser automática (agentes autônomos), semi-automática (agentes semi-autônomos) ou manuais. Neste trabalho, serão apresentados os passos para a construção de um *toolkit*, 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 a movimentação autônoma de um agente inteligente em ambientes desconhecidos. O segundo problema refere-se á 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. Pode-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, 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 mapeia-se 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 mapeamento do ambiente, ainda não implementada no Horus, é construir 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, 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 podem resolver tal problema. Uma proposta para a localização de um agente no ambiente é a utilização do método Monte Carlo [5] ou do método SLAM (*Simultaneous Location and Mapping*) [6], [7]. 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 se classifcados como transponível (aquele que não interrompe a trajetória), intransponível (aquele que o exigirá recalcula a trajetória por outro caminho) e redutor (aquele que permite ao 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 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 do *toolkit* e demonstrar a sua utilidade, foram desenvolvidas três aplicações distintas: Os simuladores Teseu e Ariadnes, e o ANPR Django. 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 ANPR Django.

Capítulo 2

Inteligência Computacional

2.1 Agentes Inteligentes

A Inteligência Computacional (IA) 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 [Citar Referencia]. Pode-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 cite(Ark98) 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 pre-

ocupam 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 ressaltamos os modelos de três camadas como: SSS cite(CON92), Atlantis cite(Gat91), 3T citeBon91. 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 provêem processos denominados de comportamentos. Em termos matemáticos, o comportamento do agente á a função que mapeia qualquer percecpção ou sequência de percepções para uma ação específica[LivroWallace] 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 re-

alizados 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 replanejamento da rota para alcançar o objetivo inical sem colidir com o obstáculo. A esse replanejamento, 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 objeto, 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 acima, 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-la.

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

- Agentes reativos simples: esse tipo de programa de agente se basea apenas na percepção 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 Reconhecimento de Padrões

Reconhecimento de padrões é um atividade que os 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 a 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 [Simp 92], [Simp 93], de menor distância ou o protótipo mais parecido [Torb 98]. 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 [Cagn 93] [Valli 98]. Portanto, redes neurais artificiais é uma das principais técnicas utilizadas nesse sentido

[Nigr 93] e será apresentada na seção seguinte. Um típico sistema de reconhecimento de padrão consiste em três partes: aquisição de dados, seleção/extração de características e classificação/clustering [Pattern recognition book].

- 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 amostas apresentadas na etapa de aprendizagem.

A técnica de reconhecimento de padrões possui uma grande variedade de aplicações. Dentre elas, pode-se citar: reconhecimento de faces, leitura biométrica, identificação de circuitos impressos defeituosos, reconhecimento óptico de caracteres, reconhecimento de fala e de escrita cursiva, entre outros.

2.3 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 [2].

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 [1].

As redes neurais artificais possuem a capacidade de aprendizado e, como conseqüência, 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.

2.3.1 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 2.1 apresenta uma representação de um neurônio.

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;

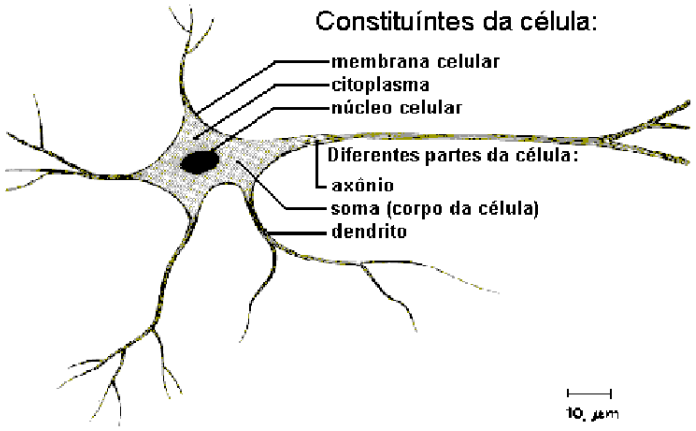

Figura 2.1: Neurônio biológico

- 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[Referencia Livro].

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 trans-

missão de impulsos entre os nodos na rede [3] 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.

2.3.2 O Neurônio Artificial MCP

McCulloch e Pits [4] 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 Pits apresenta n terminais de entrada $x_1, x_2, ..., x_n$ (representando os dendritos) e apenas um terminal de saída y (representando o axônio). Os terminais de entrada do neurônio têm pesos acoplados $w_1, w_2, ..., w_n$ cujos valores podendo ser positivos ou negativos dependendo de 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. Uma descrição do modelo está ilustrada na Figura 2.2.

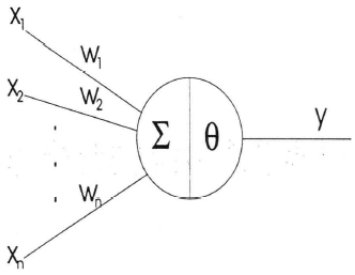

Figura 2.2: Modelo do neurônio de McCulloch e Pits

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 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.

2.3.3 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:

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

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

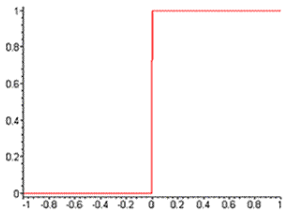

Figura 2.3: Gráfico da função Limiar

- 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 sigmóide e v é o valor de ativação do neurônio.

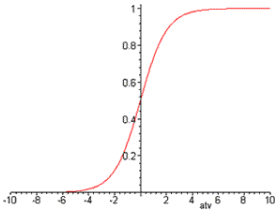

Figura 2.4: Gráfico da função sigmóide

- 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.

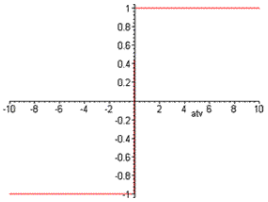

Figura 2.5: Gráfico da função Signum

- 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.

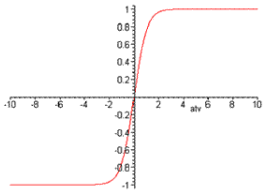

Figura 2.6: Gráfico da Tangente Hiperbólica

2.3.4 Arquiteturas de Redes Neurais

A definição da arquitetura de uma RNA define a maneira com que os neurônios são estruturados na rede [1]. 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 [3]. 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 2.7 apresenta dois exemplos de redes de única camada.

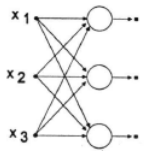

Figura 2.7: Redes de uma única camada

- Redes de múltiplas camadas: existe mais de um neurônio entre alguma entrada e alguma saída (Figura 2.8).

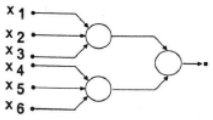

Figura 2.8: Rede de múltiplas camadas

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 2.7.

- 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 2.9).

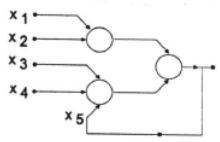

Figura 2.9: Exemplo de rede feedback.

Quanto à conectividade têm-se:

  - Redes parcialmente conectadas (Figura 2.7).
  - Redes completamente conectadas (Figura 2.8).

2.3.5 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 [4], 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. Foram desenvolvidos 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).

– Arendizado Supervisionado:

Esse tipo de aprendizado é o mais utilizado em RNAs. Esse aprendizado é dito supervisionado porque as entrada 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 2.10 apresenta um esquema básico de aprendizado supervisionado.

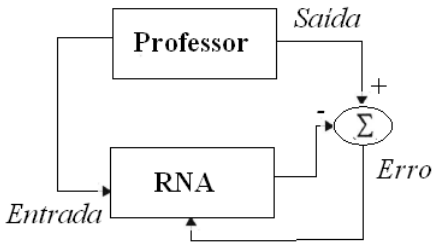

Figura 2.10: Esquema de aprendizado supervisionado.

Nesse trabalho, o algoritmo de aprendizado utilizado nesse trabalho foi o back-propagation.

– 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 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 2.11 apresenta um esquema de um sistema de aprendizado não supervisionado.

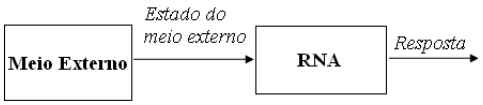

Figura 2.11: Esquema de aprendizado não supervisionado.

2.4 Visão Computacional

Adicionar Introdrução

2.4.1 Extração de características

No campo de reconhecimento de padrões, extrair caracerísticas 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 apropriada 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 dos 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.

2.4.2 Reconhecimento Óptico de Caracteres

Reconhecimento Óptico de Caracteres ou OCR (*Optical character recoginition*) 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 [1]. 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. Há vários sistemas OCR, livres e proprietários, presentes no mercado hoje. O processo de OCR é constituído de várias etapas, com responsabilidades bem definidas, que ao final apresentam o texto editável. A figura 1 apresenta um esquema básico do processo de OCR.

Figura 1: Etapas do processo de OCR

Na figura 1, tem-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 representadas 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.

Capítulo 3

Horus

Horus é um *toolkit* 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 *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, que podem ser estendidas pelas aplicações, são chamadas de abstrações. A Figura 000000 mostra a arquitetura deste módulo.

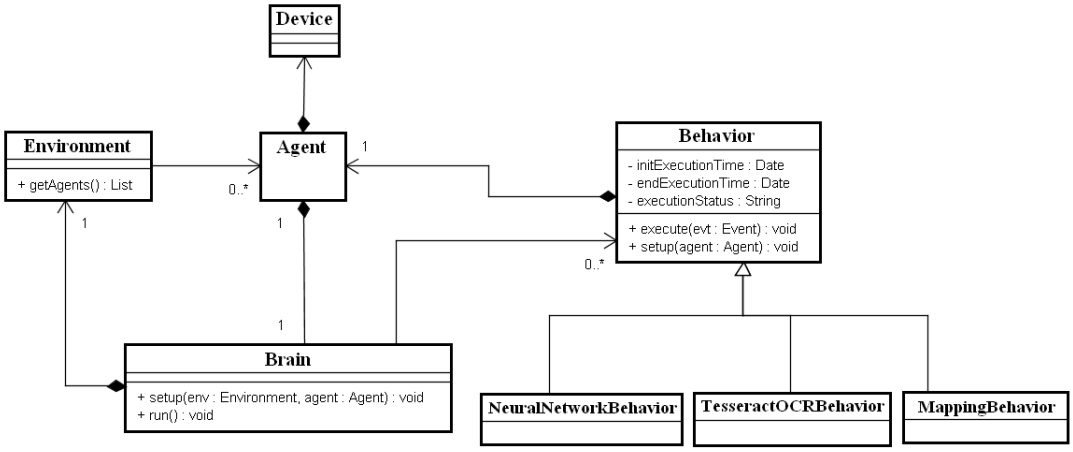

Figura 3.1: Arquitetura do módulo core

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 (hierarquia *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 2. Esse método é o *loop* principal da execução do agente, sendo executado dez vezes por segundo. 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.

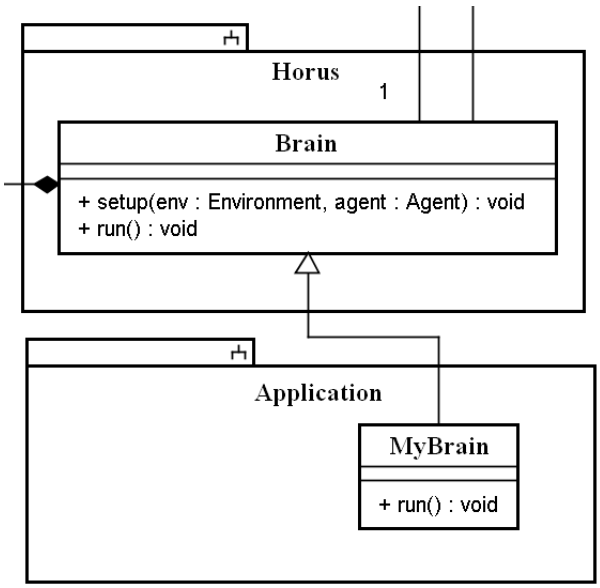

Figura 3.2: extensão da classe *Brain* do horus por uma aplicação.

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* fornece 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 2 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.

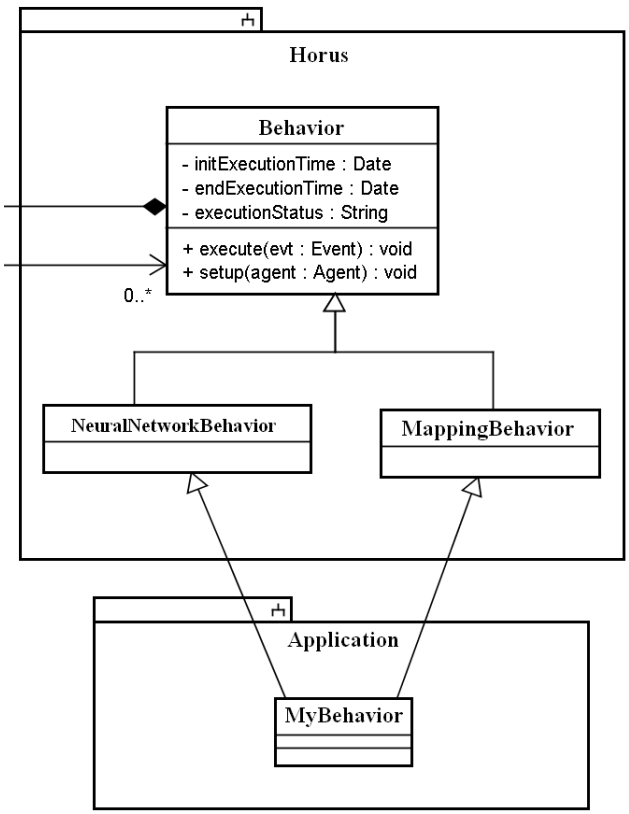

Figura 3.3: Comportamento *MyBehavior* que estende tanto de *NeuralNetworkBehavior* quanto de *MappingBehavior*.

Em certas aplicações, um agente inteligente não necessariamente se encontra sozinho

no ambiente. Ele pode interagir com outros agentes inteligentes, como no caso de aplicações que envolvam enxames de agentes. Para que um agente possa obter informações sobre outros agentes que se encontrem 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 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 [LIVRO FUNDAMENTALS OF DIGITAL IMAGE PROCESSING]. Processamento de imagens 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 necessário 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 $(x-1, y-1), (x-1, y+1), (x+1, y-1), (x+1, y+1)$, chamada 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 000000 ilustra as possíveis vizinhanças de um pixel.

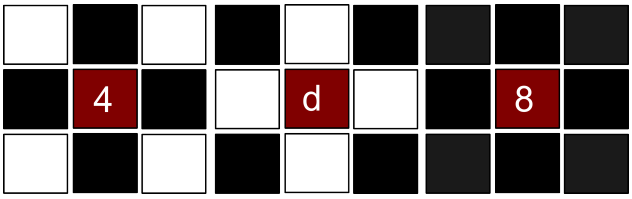

Figura 3.4: (a) 4-vizinhança, (b) d-vizinhança e (c) 8-vizinhança.

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

3.2.1 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 1 e 2 mostram exemplos de imagens e seus respectivos esqueletos.

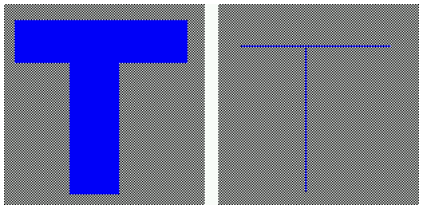

Figura 3.5: Imagem de um "T"e seu respectivo esqueleto

Figura 3.6: Imagem de um "B"(preto) e seu respectivo esqueleto (branco).

Normalmente, o esqueleto de uma imagem enfatiza as propriedades geométricas e topológicas dos padrões e é extraído quando se desejam 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 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 00000000 ilustra os oito vizinhos do pixel p_1.

Figura 3.7: 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 3.8: (a) $B(p_1) = 2$, $A(p_1) = 1$ b) $B(p_1) = 2$, $A(p_1) = 2$

A Figura 3.8 mostra exemplos dessas duas funções em uma imagem.

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.

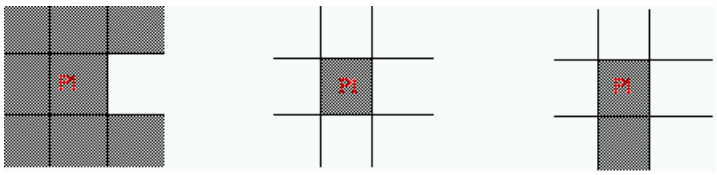

Figura 3.9: a) $B(p_1) = 7$ b) $B(p_1) = 0$ c) $B(p_1) = 1$

A Figura 3.9 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 5.

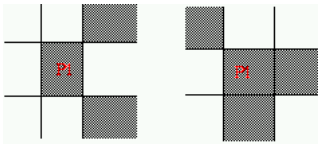

Figura 3.10: Exemplos onde $A(p_1)$ é maior que 1.

– $p_2 + p_3 + p_8 \geq 255$ 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 abaixo apresenta uma situação em que a condição acima é satisfeita.

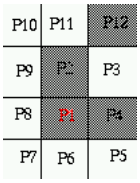

Figura 3.11: $A(p_2) \neq 1$ e $p_2 + p_3 + p_8 \geq 255$

– $p_2 + p_4 + p_6 \geq 255$ 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 abaixo apresenta uma situação em que a condição acima é satisfeita.

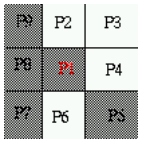

Figura 3.12: $p_2 + p_4 + p_6 \geq 255$

A cada iteração do algoritmo, os pixels das bordas são analisados, alguns deles são marcados para deleção e então deletados. A figura abaixo 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 8 apresenta dois tipos de padrões que são completamente erodidos pelo algoritmo.

Figura 3.13: a) padrão de entrada do algoritmo b) deleção iterativa dos pixels das bordas c) resultado após a execução do algoritmo

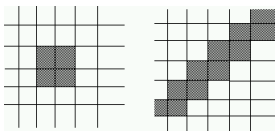

Figura 3.14: Padrões completamente erodidos pelo algoritmo de Hilditch.

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 três algoritmos para extração de características. Cada um deles será explicado 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 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.

Figura 3.15: Matriz de pixel de um bitmap.

– 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 $(r_0, r_1, ..., r_5)$ organizadas em três linhas e duas colunas. Quatro layouts podem ser utilizados para a divisão do bitmap em regiões.

Definindo a aresta de um caractere como uma matriz 2×2 de transições de branco para preto nos valores dos pixels, tem-se quatorze diferentes tipos de

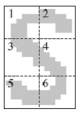

Figura 3.16: Layout com seis regiões em três linhas e duas colunas.

arestas, como ilustrado na figura 4.

Figura 3.17: 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:

O vetor de características de saída é ilustrado pelo padrão abaixo. A notação $h_j@r_i$ siginifica "número de ocorrências de um tipo de aresta representado pela

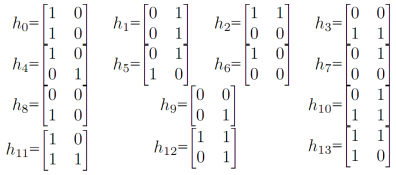

Figura 3.18: Matrizes referentes aos tipos de arestas

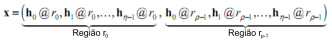

Figura 3.19: Vetor de Características

matriz h_j na região r_i":

Uma outra forma de extração de características é a análise estrutural do padrão. Através desse tipo de extração é possível diferenciar padrões por suas caracteristias 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.
- Junções: consiste em um ponto que possui pelo menos três pixels pretos na 8-viznhança. No presente trabalho, considerou-se apenas dois tipos de junções: com três e quatro vizinhos. A Figura 000000 mostra um exemplo de cada caso.
- *Loops*: este 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 00000), ou seja, o fundo da imagem é reprensetado 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 00000 pelos números 1, 2, 3, subtraido de um. Essa subtação é feita para não considerar o fundo da imagem como um *loop*.

O *toolkit* Horus fornece algumas implementações de algoritmos para análise estrututal 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 identifiquem unicamente o objeto independente das variações com que ele se apresente. 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 linguagems 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 licensa Apache e é escrita nas linguagens de programação C e C++. O módulo OCR do Horus pode ser utilizado em aplicações em que 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 representa-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 o custo de processamento no deslocamento entre a posição de um nó ao outro.

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.

Capítulo 4

Aplicacoes

As aplicações desenvolvidas nesse projeto tem o objetivo de validar e demonstrar a utilidade do *toolkit* Horus no desenvolvimento de aplicações que envolvem agentes inteligentes, 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 *ANPR*, 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*. As aplicação ANPR e o simulador Ariadnes serão detalhados nas próximas seções.

4.1 Anpr

4.2 Ariadnes

Ariadnes é um sistema que simula a movimentação autônoma de um agente inteligente em um ambiente. Nele é possivel simular o comportamento de um robô real no que tange mapeamento e navegação. 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. O simulador Ariadnes é composto por quatro partes principais, onde duas delas utilizam o *toolkit* Horus. A Figura 4.1 apresenta a arquitetura conceitual do simulador Ariadnes com seus principais componentes.

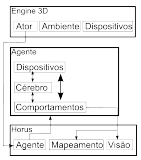

Figura 4.1: Arquitetura conceitual do simulador Ariadnes.

As principais partes do simulador Ariadnes são: ambiente, engine 3D, agentes e dispositivos. O ambiente, nesse simulador, foi construído utilizando o modelador

Blender3D. Esse ambiente 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 0000.

A engine 3D utilizada no controle do ambiente e do agente foi o Panda3D. 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 00001.

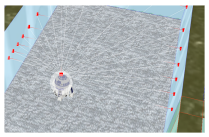

Figura 4.2: Agente configurado com dispositivos de lasers.

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 000002 é apresentado a vista de cima 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,

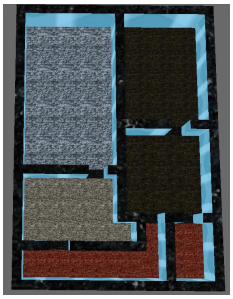

Figura 4.3: Ambiente utilizado no Ariadnes

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 especifica, 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, descrita na Seção 0000. Durante o mapeamento, uma câmera é utilizada pelo agente para localizar as placas informativas presentes no ambiente. Quando este depara-se com uma dessas placas, interrompe o processo de mapeamento e utiliza algoritmos de visão para interpretar o

conteúdo existente na placa, a fim de estabalecer 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 a seguir mostra uma foto capturada por cada uma das câmeras citas.

Ao encontrar um marco, (FALAR do ALINHAMENTODO ROBO ANTES DE FOTOGRAFAR) o agente fotografa a placa. Com essa imagem, inicia-se o processo de reconhecimento da placa. Esse processo consiste nas etapas de: localização 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, descrito na Secao 000000, 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. Para isso, a técnica utilizada consiste na utilização da projeção vertical e da horizontal, ambas descritas na Seção 000. Após o cálculo das projeções, calcula-se a média e aplica-se um *threshold* global. Isso define intervalos de valores, na projeção, diferentes de zero. A placa, normalmente, encontra-se no intervalo que contém o ponto de máximo da função projeção, pode-se observar na FIGURA 000000.

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 V.

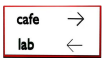

Figura 4.4: Placa informativa.

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 a seguir 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 a seguir.

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

Capítulo 5

Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado o *toolkit* para desenvolvimento e controle de aplicações que envolvem agentes inteligentes, Horus. 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.

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 ténicas 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.

Referências Bibliográficas

Apêndice A

Dependências do Tool kit

Apêndice B

Instalações