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

JÔNATAS OLIVEIRA LOPES SOARES
MAYCON BARRETO LOPES
WALLACE GOMES DE SOUZA

MÓDULO DE MAPEAMENTO DO TOOLKIT HÓRUS

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

ORIENTADOR: Prof. D.Sc. Ítalo Matias

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

Campos dos Goytacazes-RJ
2009

JÔNATAS OLIVEIRA LOPES SOARES
MAYCON BARRETO LOPES
WALLACE GOMES DE SOUZA

MÓDULO DE MAPEAMENTO 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: MAPEAMENTO, LOCALIZAÇÃO, SIMULTANEOUS LOCALIZATION AND MAPPING, SLAM, TOOLKIT, AGENTE INTELIGENTE.

Abstract

Keywords: MAPPING, LOCALIZATION, SIMULTANEOUS LOCALIZATION AND MAPPING, SLAM, TOOLKIT, HÓRUS, INTELIGENT AGENT.

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 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 classifcar 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 classifcação de um obstáculo ocorre mediante a algum método de reconhecimento de padrões, baseado

em visão computacional.

Com isso, 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, serão apresentados a parte de visão computacional do simulador Ariadnes e a aplicação para reconhecimento automático de placas de automóveis ANPR Django.

Capítulo 2

Horus

Horus é um toolkit de 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.

2.1 Core

O horus é um toolkit que 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.

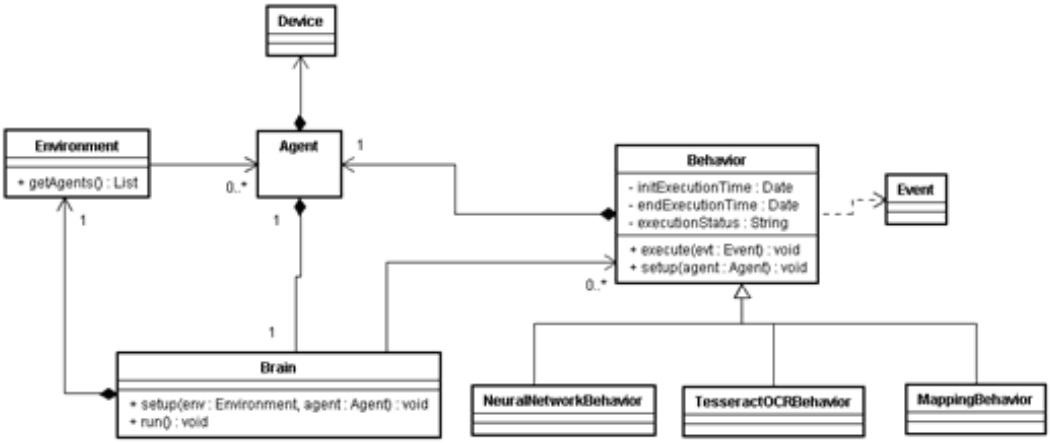

Figura 2.1: Arquitetura do módulo core

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 por toda a 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 pelo 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.

Os comportamentos (Behavior) recebem eventos gerados pela aplicação e fazem com

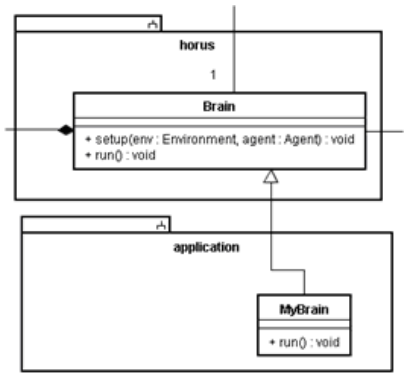

Figura 2.2: extensão da classe Brain do horus por uma aplicação.

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.

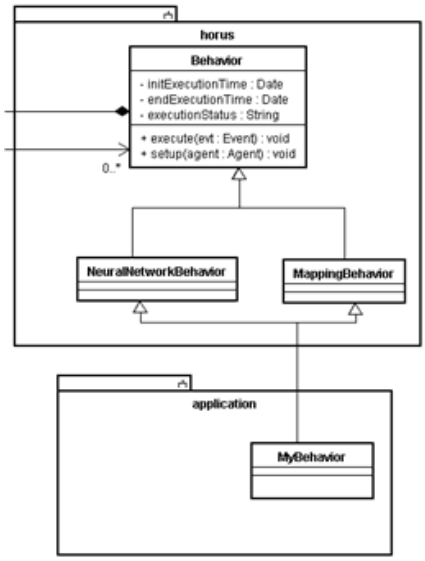

Figura 2.3: Comportamento MyBehavior que estende tanto de NeuralNetworkBehavior quanto de MappingBehavior.

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 forma 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 par-

alelo. 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.

2.2 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 aprendeizagem 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.

- 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.

- 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.

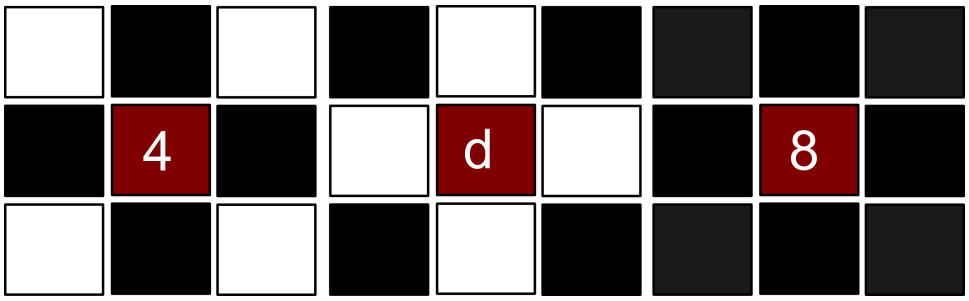

Figura 2.4: (a) 4-vizinhança, (b) d-vizinhança e (c) 8-vizinhança.

- 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.

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

2.2.1 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.

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*".

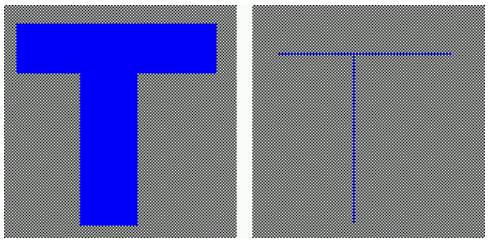

Figura 2.5: Imagem de um "T"e seu respectivo esqueleto

Figura 2.6: Imagem de um "B"(preto) e seu respectivo esqueleto (branco).

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.

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

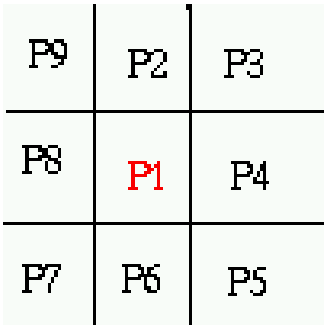

Figura 2.7: 8-vizinhança do pixel p_1

- $B(p_1)$: número de vizinhos que não são brancos do pixel p_1.
- $A(p_1)$: números 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.

A figura abaixo mostra exemplos dessas duas funções em uma imagem.

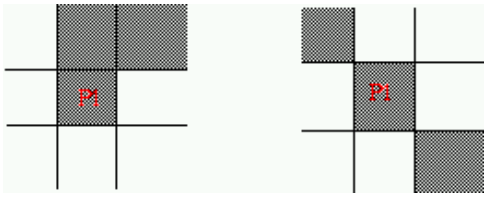

Figura 2.8: (a) $B(p_1) = 2$, $A(p_1) = 1$ b) $B(p_1) = 2$, $A(p_1) = 2$

Há duas versões do algoritmo de Hilditch, uma usando uma janela $4x4$ e outra usando uma janela $3x3$, nesse trabalho foi utilizada uma janela $3x3$. 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 não 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 ($B(p_1) = 6$).

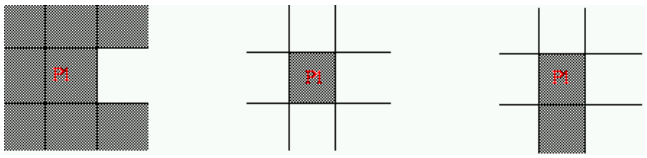

Figura 2.9: a) $B(p_1) = 7$ b) $B(p_1) = 0$ c) $B(p_1) = 1$

A figura acima 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.

- $p_2 + p_3 + p_8 \geq 255 ou A(p_2) \neq 1$: essa condição assegura que linhas verticais com largura de dois pixels não serão inteiramente removidas pelo algoritmo. A figura 6 abaixo apresenta uma situação em que a condição acima é satisfeita.

- $p_2 + p_4 + p_6 \geq 255$ 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 7 abaixo apresenta uma situação em que a condição acima é satisfeita.

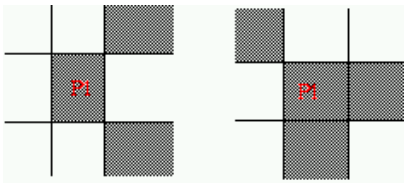

Figura 2.10: Exemplos onde $A(p_1)$ é maior que 1.

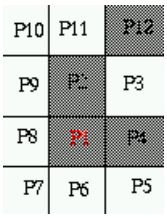

Figura 2.11: $A(p_2) \neq 1$ e $p_2 + p_3 + p_8 \geq 255$

A cada iteração do algoritmo, os pixels das bordas são analisados, alguns deles são marcados para deleção e então deletados. A figura 8 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 2.12: $p_2 + p_4 + p_6 \geq 255$

Figura 2.13: a) padrão de entrada do algoritmo b) deleção iterativa dos pixels das bordas c) resultado após a execução do algoritmo

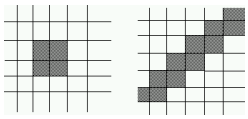

Figura 2.14: Padrões completamente erodidos pelo algoritmo de Hilditch.

2.3 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. Inicia-se 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. No Ariadnes 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.

2.3.1 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.

- 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 $(r0, r1, ..., r5)$ 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 $2x2$ de transições de branco para preto

nos valores dos pixels, tem-se quatorze diferentes tipos de arestas, como ilustrado na figura 4.

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 matriz h_j na região r_i":

- Intensidade de blocos falta falar

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 substancais. No caso de reconhecimento de caracteres, a análise estrutural leva em consideração estruturas mais complexas como junções, terminação de linhas e loops.

- Terminação de Linhas: 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 junções: com três e quatro 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 lagos, grupo de pixels preto na imagem negativa representado na Figura 00000 pelos números 1, 2, 3 subtraido de um, grupo de pixels preto que representa o fundo da imagem.

O toolkit Horus fornece algumas implementações de algoritmos para análise estrutal de caracteres.

2.3.2 Reconhecimento de Objetos

Reconhecimento de objetos é o processo de identificar 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 engina está sobre a licensa Apache e é escrita nas linguagens de programação c e c++. O modulo 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, agente inteligente utiliza esse módulo para reconhecer os textos que se encontram nas placas informativas presentes no ambiente.

2.4 Mapeamento e Navegação

Chamamos de mapeamento ao 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 será 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 su

Capítulo 3

Aplicações

3.1 Anpr

3.2 Teseu

3.3 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 **??** apresenta a arquitetura conceitual do simulador Ariadnes com seus principais componentes. 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.

As principais partes do simulador Ariadnes são: ambiente, engine 3D, agentes e dispositivos. O ambiente, nesse simulador, foi construído utilizando o modelador Blender3D.

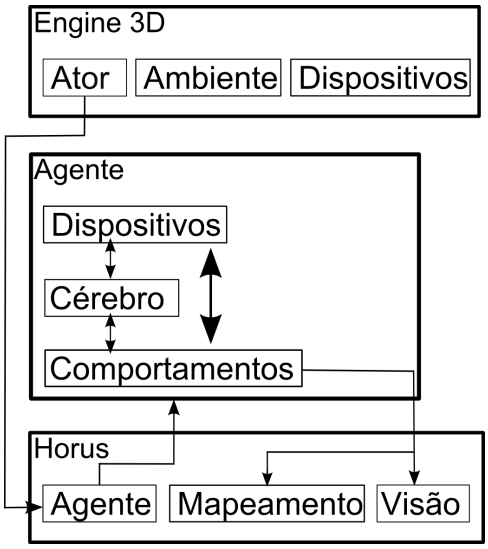

Figura 3.1: Arquitetura conceitual do simulador Ariadnes.

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 8. 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.

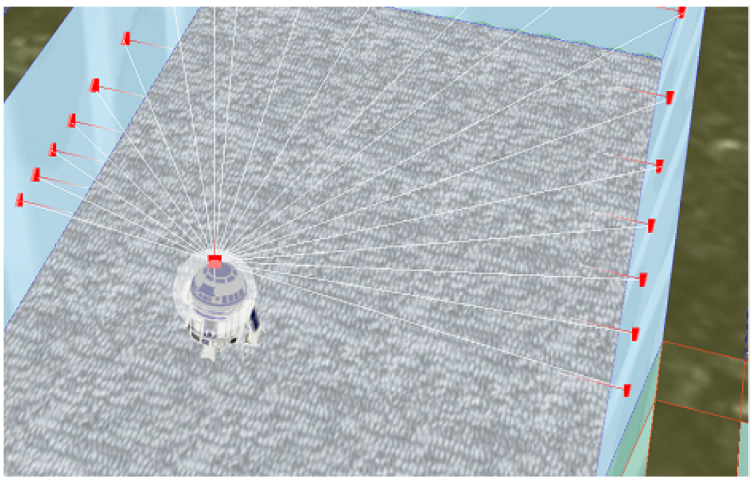

Figura 3.2: Agente configurado com dispositivos de lasers.

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.

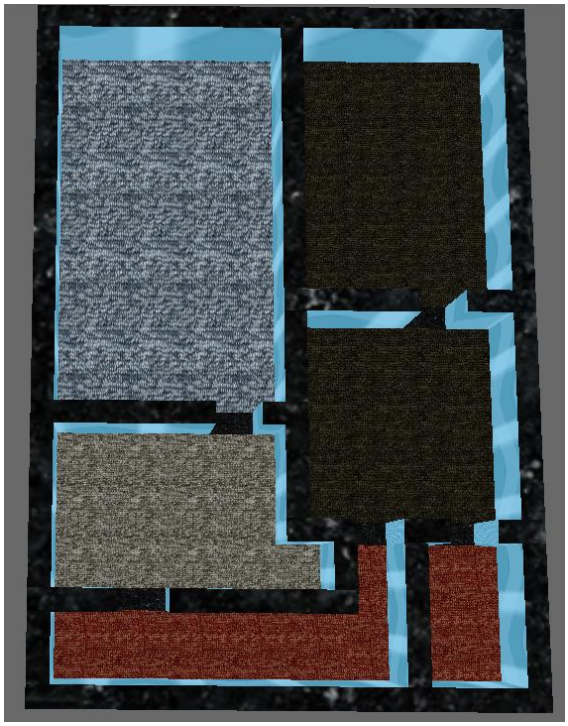

Figura 3.3: Ambiente utilizado no Ariadnes

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, 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. No estado de reconhecimento

de objetos, 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. A projeção horizontal (vertical) consiste em, para cada linha (coluna), aplicar o somatório da intensidade de luminância. 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.

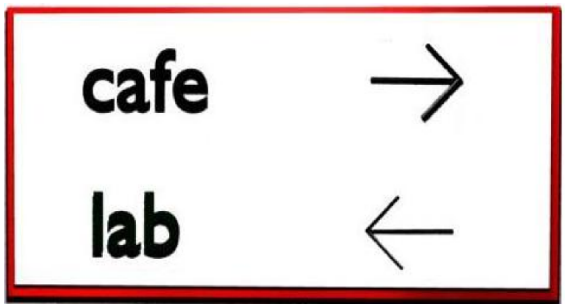

Figura 3.4: Placa informativa.

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 a 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 toolkit criado.

Capítulo 4

Conclusões e Trabalhos Futuros

Referências Bibliográficas

[1] RUSSEL, S.; NORVIG, P. *Inteligência Artificial*. [S.l.]: Editora Campus, 2003.

Apêndice A

Dependências do Tool kit

Apêndice B

Instalações