Ariadnes: um sistema de mapeamento e navegação autônoma em ambientes virtuais

Thiago R. Nunes*, Chrystiano S. Araújo†, Leandro M. Cruz‡, Italo O. Matias§
*Universidade Candido Mendes
Campos dos Goytacazes - RJ - Brasil
thiagorinu@gmail.com
† chrystianosaraujo@gmail.com
‡ lcruz@impa.br
§ italo@ucam-campos.br

***Resumo*—Este artigo apresenta arquitetura e fundamentos para o desenvolvimento de simuladores para controle de agentes inteligentes móveis. Entende-se por movimentação os processos de mapeamento e navegação. Embora esses processos envolvam diversas características do agente e do ambiente, priorizou-se nesse trabalho, conceitos de percepção do ambiente e dos elementos nele contidos, através de visão computacional. Durante o desenvolvimento desse projeto, foram desenvolvidos um toolkit de desenvolvimento e controle de agentes inteligentes, chamado Horus; e um simulador de movimentação autônoma de um agente inteligente, chamado Ariadnes. Essa aplicação simula computacionalmente o processo de exploração de um robô real em um ambiente desconhecido. Durante a exploração, o agente localiza marcos no ambiente através de uma imagem obtida via uma câmera virtual, e identifica-os através de algoritmos para extração de características presentes no Horus.**

***Keywords*-Agentes Inteligentes; Visão Computacional; Simulação**

I. Introdução

Neste trabalho serão apresentados arquitetura e fundamentos para a construção de um simulador para mapeamento e navegação autônoma de um agente inteligente em um ambiente virtual. Esse agente é capaz de movimentar-se em um ambiente desconhecido realizando inicialmente o seu mapeamento e a construção de um mapa através do qual ele navegará a posteriori. Dependendo do problema abordado em cada aplicação, o agente pode utilizar técnicas de visão computacional a fim de otimizar a sua percepção com relação ao ambiente no qual está inserido. Como prova dos conceitos apresentados no presente trabalho, foi contruído um simulador, denominado Ariadnes, e um toolkit para desenvolvimento e controle de agentes inteligentes, chamado Horus.

O objetivo do simulador, apresentado neste trabalho, é representar as questões relacionadas com a movimentação de um robô real presente em um ambiente desconhecido. Um dos principais aspectos na robótica móvel é a ênfase nas questões referentes à locomoção em ambientes complexos e que se modificam dinamicamente. Para cumprir seus objetivos, um robô deve ser capaz de adquirir e utilizar informações extraídas do ambiente, estimar sua posição e direção e reconhecer padrões de elementos no ambiente. Essas habilidades tornam-o capaz de reagir, em tempo real, às diferentes ocorrências ao seu redor. Tais habilidades podem ser generalizadas para outros agentes inteligentes.

Reconhecimento de padrões é um problema clássico de visão computacional. Os procedimentos, já conhecidos na literatura, referem-se a padrões específicos, ou seja, padrões distintos são reconhecidos de formas distintas. Dessa forma, no módulo de visão do toolkit Horus existe uma abstração com essa finalidade e um conjunto de algoritmos para extração de características que podem ser utilizados em diferentes procedimentos. Além disso, também foram implementados algoritmos de processamento de imagens que normalmente são utilizados, como pré-processamento, em sistemas de reconhecimento de padrões.

O toolkit Horus provê diversas funcionalidades com o obetivo de auxiliar o desenvolvimento de agentes inteligentes. Essas funcionalidades estão agrupadas em áreas como: visão computacional, mapeamento e navegação. Utilizando esse toolkit, foi desenvolvido o simulador Ariadnes. Nessa aplicação, o agente inteligente simula um robô real movimentando-se em um ambiente. Para isso, esse agente inicialmente mapeia o ambiente para posteriormente conseguir se localizar e locomover. Durante a movimentação, em certos momentos, o agente necessita reconhecer alguns padrões, através de visão computacional, como obstáculos e placas informativas.

Este artigo está organizado da seguinte forma: na seção II são apresentados os principais conceitos de agentes inteligentes. Na seção III são apresentados alguns conceitos referentes à movimentação autônoma através de reconhecimento de padrões. Na seção IV é apresentado o toolkit Horus e os principais algoritmos utilizados na implementação de aplicações como o simulador Ariadnes, apresentado na seção V.

II. Agentes inteligentes

A Inteligência Computacional (IA) é uma área de estudo da ciência da computação que procura desenvolver sistemas computacionais capazes de realizar 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. 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.

Existem diferentes definições para a arquitetura de um robô presentes na literatura. Este trabalho considera a definição abordada por Arkin [1] a qual considera que uma arquitetura de um robô está mais relacionada com os aspectos de software que os de hardware. Apesar de algumas diferenças, os modelos de arquitetura para robôs móveis descrevem um mecanismo para construção de um sistema de desenvolvimento e controle de agentes inteligentes, apresentando, principalmente, quais os módulos presentes e como estes interagem.

De uma forma geral, os módulos de uma arquitetura de um agente inteligente se preocupam com aspectos como percepção, planejamento e atuação/execução. A percepção refere-se à compreensão do ambiente e dos elementos nele contido; o planejamento à inteligência do robô; e a atuação, ou execução, ao 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 [2], Atlantis [3], 3T [4]. 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. Tais 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.

Com base nesses conceitos, definiu-se neste trabalho um agente como uma entidade composta de comportamentos, dispositivos e um cérebro. 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. Já o cérebro, é o responsável pelo controle de execução de todos os comportamentos supracitados.

III. Movimentação Autônoma através de Reconhecimento de Padrões

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 alguns conceitos de agentes autônomos, ou seja, agentes capazes de explorar o ambiente de forma independente.

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. No Ariadnes, o mapa é feito na forma de um grafo conexo. 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.

O sistema 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 [6], [7].

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.

As câmeras simulam a visão em um agente inteligente. Na abordagem desse trabalho, a visão é a principal forma de percepção do ambiente. 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.

A visão possibilita reconhecer e classificar obstáculos. Existem diferentes tipos de obstáculos. Estes podem ser classificados como transponível (aquele que não interrompe a trajetória), intransponível (aquele que o exigirá recalcular a trajetória por outro caminho) e redutor (aquele que permite 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.

IV. HORUS

O Horus é um toolkit de desenvolvimento e controle de agentes inteligentes, desenvolvido na linguagem de programação Python. Além do Ariadnes, existem outros exemplos de aplicações desenvolvidas com o Horus: um sistema de Reconhecimento Automático de Placas de Automóveis, um sistema de Processamento de Imagens e Extração de Características e um sistema de mapeamento autônomo de ambientes chamado Teseu.

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.

A. Core

O Horus pode ser utilizado de dois modos. A primeira forma é como uma coleção de algoritmos, que podem ser utilizados de forma independente. A segunda forma consiste na extensão das abstrações fornecidas pelo módulo core do Horus. Nesse módulo existem abstrações para implementação de agentes, dispositivos e comportamentos.

Os comportamentos são procedimentos implementados para representar ações e reações dos agentes. As ações são comportamentos ativos, ou seja, procedimentos que visam realizar um objetivo previamente definido. Por outro lado, reações são procedimentos realizados mediante a estimulos 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. Levando em consideração que durante o percurso, esse agente se deparou com algum obstáculo. Isso deve gerar um comportamento de replanejamento da rota para alcançar o objetivo inical sem colidir com o obstáculo. A esse replanejamento, denomina-se reação.

A ordem de execução dos comportamentos define a máquina de estados de cada agente de acordo com cada aplicação. O toolkit Horus, desenvolvido nesse projeto, permite a configuração dessa máquina de estados.

B. Visão

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.

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 [9]. 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 [10]. 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 significativamente 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.

Nas sessões seguintes, serão explicados alguns métodos de extração de características implementados no toolkit Horus.

- 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.

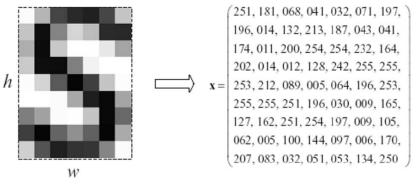

Figura 1. Matriz de pixel de um bitmap.

- 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 $f(x, y)$, onde w é a largura, h a altura, $0 \leq x < w$ 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, têm-se quatorze diferentes tipos de arestas, como ilustrado na Figura 3.
  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

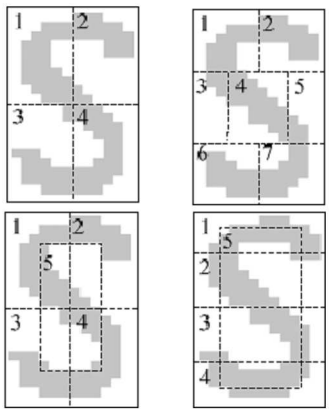

Figura 2. Layout com seis regiões em três linhas e duas colunas.

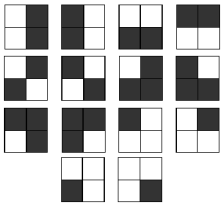

Figura 3. Quatorze diferentes tipos de arestas

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:

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

Figura 4. Matrizes referentes aos tipos de arestas

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":

$$\mathbf{x} = \underbrace{\left(\mathbf{h}_0 @ r_0, \mathbf{h}_1 @ r_0, \ldots, \mathbf{h}_{\eta-1} @ r_0\right)}_{\text{Região } r_0}, \underbrace{\mathbf{h}_0 @ r_{\rho-1}, \mathbf{h}_1 @ r_{\rho-1}, \ldots, \mathbf{h}_{\eta-1} @ r_{\rho-1}\right)}_{\text{Região } r_{\rho-1}}$$

Figura 5. Quatorze diferentes tipos de arestas

C. 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 sua construção e algoritmos para cálculo de caminho mínimo, como Dijkstra e A*.

V. 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 6 apresenta a arquitetura conceitual do simulador Ariadnes com seus principais componentes.

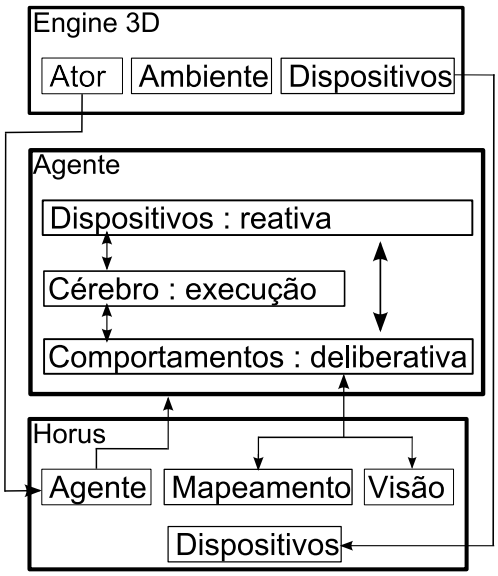

Figura 6. Arquitetura conceitual do simulador Ariadnes.

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 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 7

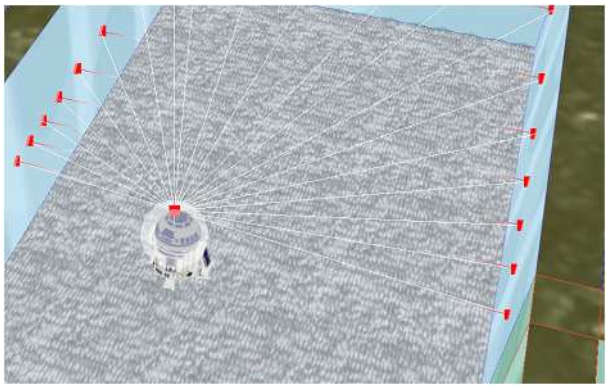

Figura 7. Agente configurado com lasers.

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 8 é apresentado a vista de cima do ambiente utilizado no Ariadnes.

Figura 8. Visão superior do ambiente do 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 (Simultaneous Localization and Mapping). Essa técnica é utilizada em agentes autônomos para mapeamento de ambientes desconhecidos. Uma implementação do SLAM encontra-se no módulo de mapeamento do toolkit Horus.

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 o objetivo de auxiliar a localização das placas identificadoras de cômodos e direções. Ao encontrar um marco, o agente fotografa a placa. Com essa imagem, inicia-se o processo de reconhecimento da placa. Esse processo consiste nas etapas de: 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, um filtro para detecção de 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. 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.

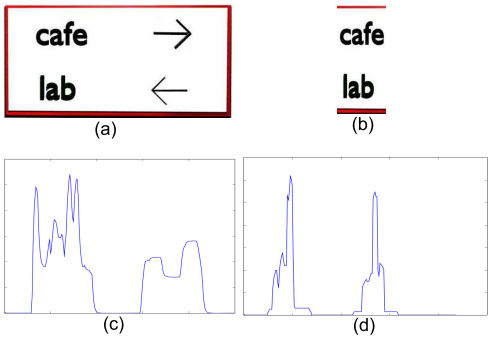

Figura 9. (a) Imagem da placa. (b) Recorte da coluna de textos. (c) Projeção horizontal da placa. (d) Projeção vertical da coluna.

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. 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. Logo, essas regiões são segmentadas através da projeção vertical, identificando cada linha da placa, conforma mostra a Figura V. 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.

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.

VI. Conclusão

Este trabalho apresentou alguns aspectos relacionados a construção de um sistema de controle e desenvolvimento de agentes inteligentes chamados Ariadnes. Nesse sistema, o agente, independentemente, mapeia e se locomove através de um ambiente desconhecido com o objetivo de construir um grafo desse e atingir um determinado lugar, orientando-se através de placas informativas interpretadas através de técnicas de visão computacional. Dessa forma, o grafo construído durante o processo de mapeamento é utilizado posteriormente no processo de navegação.

Esse sistema foi desenvolvido como parte do projeto do toolkit para desenvolvimento de agentes inteligentes Horus. Neste toolkit estão implementados todos os algoritmos de mapeamento, localização, navegação e visão computacional que foram utilizados no sistema.

Além da utilização do toolkit Horus, proposto no presente trabalho, na contrução de ferramentas para exploração e atuação de agentes em ambientes virtuais, esse toolkit também pode ser utilizado em aplicações reais. Um exemplo desse tipo de aplicação é o uso de robôs reais na exploração de ambientes inacessíveis para seres humanos.

Agradecimentos

Agradecemos a Universidade Candido Mendes, por ter propiciado um ambiente frutífero para pesquisa e aos demais colaborados para o desenvolvimento do toolkit Horus: Dalessandro Vianna, Jonathas Lopes, Lucas Carvalho, Maycon Lopes e Wallace Gomes.

Referências

[1] R. ARKIN, *Behavior-based robotics*. MIT Press, 1998.

[2] J. CONNEL, "A hybrid arquitecture applied to robot navigation." *IEEE Int Conf. on Robotics and Automation*, pp. 2719–2724, 1992.

[3] E. GAT, Dept. of Computer Science, Virginia potytechnic Institute and State University, 1991, reliable Goal-directed Reactive Control for Real-World Autonomus Mobile Robots.

[4] R. P. BONASSO, "Integrating reaction plans and layered competences through synchronous control." *Twelfth International Joint Conference on Artificial Intelligence*, pp. 1225–1231, 1991.

[5] D. FOX, W. BURGARD, F. DELLAERT, and S. THRUN, "Monte carlo localization: Efficient position estimation for mobile robots." *The National Conference on Artificial Intelligence (AAAI).*, 1999.

[6] A. Davison, Y. Cid, and N. Kita, "Real-time 3d slam with wide-angle vision," *IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.

[7] F. Dellaert., "Square root slam: Simultaneous location and mapping via square root information smoothing." *Robotics: Science and Systems*, 2005.

[8] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[9] J. C. Santos, Instituto Nacional de Pesquisas Espaciais - INPE, 2007, extração de atributos de forma e seleção de atributos usando algoritmos genéticos para classificação de regîões.

[10] I. Guyon, S. Gum, M. Nikravesh, and I. Zulch, *A practical guide to splines*. Springer, 2006.