UNIVERSIDADE CÂNDIDO MENDES

JÔNATAS OLIVEIRA LOPES SOARES
MAYCON BARRETO LOPES
WALLACE GOMES DE SOUZA

MÓDULO DE MAPEAMENTO DO TOOLKIT HÓRUS

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
Doutorado em Sistemas Computacionais pela UFRJ.

Prof. D.Sc. Dalessandro Soares - Co-orientador
Pós-Doutorado em Engenharias, Ciências Exatas e da Terra pela UFF.

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

Dedico este trabalho a minha mãe, irmã e namorada.

Jônatas

Dedico este trabalho à minha família, amigos e a minha noiva.

Maycon

Dedico este trabalho a meus pais, amigos e noiva.

Wallace

Agradecimentos

Agradecemos a Deus, pois sem Ele nada do que se fez poderia ter sido feito; a Ele que nos deu forças pra superar as dificuldades e vencer as barreiras.

Agradecemos aos nossos pais por incetivarem e auxiliarem em nossos estudos desde o ínicio, quando ainda estávamos brincando de aprender até o período atual onde a brincadeira de aprender se tornou uma fome de conhecimento.

Agradecimentos ao orientador e co-orientador que tornaram possível esse momento com seu esforço e dedicação.

Agradecemos aos demais integrantes da Banca Examinadora, os quais, pelo menos em algum momento, desde a origem até a conclusão do trabalho, deram a sua contribuição.

Agradecemos a todos os professores da Universidade Cândido Mendes do curso de ciências da computação, que nos acompanharam e nos ensinaram nessa fase única e marcante de nossas vidas, que foi nossa formação acadêmica.

Eu, Jônatas, agradeço em especial à minha mãe que me deu o suporte muito além do que lhe era cabível e motivou-me a continuar mesmo em momentos de fraqueza, a minha irmã que com sua doçura me manteve sempre otimista e por último, mas não menos importante,

a minha namorada que esteve ao meu lado me incentivando e apoiando em grande parte do processo de graduação .

Eu, Maycon, agradeço em especial à minha família, que me deu a oportunidade de iniciar o meu curso de graduação. Agradeço à força e auxílio de muitos amigos inclusive os que fazem parte deste trabalho. Agradeço também à minha noiva, pelo seu companheirismo e por suas palavras de incentivo dando-me a certeza de que este projeto seria possível.

Eu, Wallace, agradeço em especial aos meus pais, por sempre estarem dispostos a ajudar, pelo incentivo e apoio. Agradeço a minha noiva, pelo seu carinho e paciência por passar semanas longe por ter que estudar ou trabalhar. Agradeço aos meus velhos e novos amigos pelos momentos de alegria que ajudaram a chegar ate aqui

Agradecemos também à Chrystiano, Leandro, Lucas e Thiago que participaram diretamente da nossa formação e, juntamente conosco, proporcionaram a conclusão deste trabalho.

E os nossos sinceros agradecimentos a todas as pessoas que, direta ou indiretamente contribuíram para que este trabalho fosse concluído.

Resumo

O presente trabalho demonstra a utilização do *Toolkit* Horus, com foco no módulo de mapeamento, para a criação de agentes inteligentes para ambientes desconhecidos onde agiram reconhecendo e explorando o ambiente. Foi usada a filosofia de mapeamento SLAM (*Simultaneos Location and Mapping*), além de outras técnicas desenvolvidas para resolver os problemas dos ambientes-tarefa propostos.

Dentre as outras técnicas está a apresentação de um método de mapeamento baseado em marcos (*markpoints*) que o agente cria durante a sua navegação a fim de identificar posteriormente lugares nos quais já esteve.

Também apresentaremos um método de mapeamento próprio baseado na idéia de punição e recompensa, no qual o agente ganha ao encontrar novos marcos e perde ao reencontrá-los, criando assim uma condição de parada desejada quando a sua perda atender a um requisito previamente criado.

Neste trabalho, o agente utiliza apenas dois tipos de sensores para comprir sua tarefa de navegação, localização e mapeamento simultâneo: laser e odômetro.

Além disso, foi produzida uma aplicação com a finalidade de integração entre os módulos já produzidos do *Toolkit* Horus, nesta aplicação o agente tem como objetivo navegar e mapear o ambiente ao mesmo tempo em que busca por placas informativas disponibilizadas por todo o ambiente utilizando de duas câmeras virtuais para concretizar esta busca.

Finalmente, o trabalho pretende criar uma prova de conceito para o módulo desenvolvido comprovando assim o seu funcionamento. O módulo SLAM desenvolvido é responsável apenas pelas tarefas de localização e mapeamento simultâneo, sendo a aplicação

que o utiliza a responsável por conceder o agente (ator), sua navegação e os dispositivos usados.

Palavras-chave: MAPEAMENTO, LOCALIZAÇÃO, SIMULTANEOUS LOCALIZATION AND MAPPING, SLAM, TOOLKIT, AGENTE INTELIGENTE.

Abstract

This work demonstrates the use of the Horus Toolkit, with focus on mapping module, for creating intelligent agents for unknown environments where act recognizing and exploring the environment. The philosophy of mapping SLAM (Simultaneous Location and Mapping) was used and other techniques developed to solve the problems of the proposed task-environments.

Among the other techniques is the presentation of a mapping method based on marks (markpoints) that he creates during its navigation in order to identify further sites where already been.

Also present a method of mapping based on the idea of punishment and reward, in which the agent gets credit for finding new markers and lose them if find twice the same marks, thus creating a condition to stop your loss when the desired answer to a previously established requirement .

In this work, the agent uses only two types of sensors to accomplish its task of navigation, simultaneous localization and mapping: laser and odometer.

Furthermore, an application was implemented for the purpose of integration between the modules already produced the Toolkit Horus, in this application the agent has the objective navigate and map the environment while they search for information signs available in the entire environment using two virtual cameras to through this search.

Finally, the work aims to create a proof of concept for the module developed thus showing its operation. The SLAM module developed and responsible only for the tasks of simultaneous localization and mapping, the application uses the agent responsible for the

grant, and its navigation devices used.

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

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A robótica era um sonho até pouco tempo atrás, hoje em dia a sua existência é tão comum que muitas vezes nos passam despercebidos os grandes avanços da área. Cada vez mais robos, ou melhor, agentes inteligentes estão presentes em nosso dia-a-dia a fim de facilitar nossos afazeres e com isso nos tornar mais produtivos.

Com esse intuito de otimizar nosso tempo, a robótica voltou-se ainda mais para inteligência artificial de seus agentes tornando-os mais independentes e mais capazes de solucionar problemas. Assim os agentes poderiam solucionar problemas previamente definidos ou não, este último através de técnicas avançadas de inteligência artificial.

A robótica móvel é um campo da robótica que estuda as vantagens da mobilidade dos agentes. Essa mobilidade os tornam capazes de avançar ainda mais em ambientes de difícil acesso ao ser humano, por exemplo, pesquisas subaquáticas ou, até mesmo, em outros planetas. Alguns fatores que serão apresentados nessa monografia levam em consideração conceitos como: inteligência artificial, agentes inteligentes, mapeamento, localização e outros conceitos da robótica móvel.

No capítulo 2 foram introduzidos conceitos importantes de inteligencia artificial, como estruturas de agentes, já no capítulo 3 abordou-se sobre o Ambiente tarefa e como ele influencia no projeto de um agente, o capítulo 4 foram apresentados tópicos sobre arquiteturas de robôs relevantes para o desenvolvimento da aplicação, no capítulo 5 o foco maior foi no

SLAM e como ele foi utilizado na aplicacao, enquanto no capítulo 6 foram demonstrados, do *Toolkit* Horus, os modulos de visao, mapeamento, core e utils e o capítulo 7 refere-se a aplicação com um agente autônomo em um ambiente desconhecido.

Capítulo 2

Inteligência Artificial

É uma área de estudo da ciência da computação que se preocupa em fazer com que dispositivos computacionais tenham ações e reações similares as capacidades humanas, tais como: pensar, criar, solucionar problemas entre outros.[1]

Tal inteligência pode ser aplicada de várias formas para tornar um agente, um agente inteligente, hábil e capaz.

2.1 Agentes Inteligentes

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.[2]

Algumas definições do termo agente na lingua portuguesa tais como "O que opera ou é capaz de operar", "O que promove negócios alheios"e "Autor". Existem definições de agentes em várias áreas do conhecimento humano.[3]

- Sociologia: Dentro dos estudos sociologicos, a definição de agentes inteligentes está relacionada aos seres humanos.
- Economia: Os agentes inteligentes são aqueles que operam de forma mais astuta dentro de um ambiente econômico.

- Robótica: Na robótica, que é nosso foco, o agente inteligente é visto como um agente que possui uma inteligência (artificial) e se utiliza dela para ter autonimia, pró-atividade e até mesmo tomada de decisões.

Nesse projeto será abordada a utilização de agentes inteligentes na área de tecnologia, que engloba robótica e também a área de software com agentes em ambientes virtuais. Dentro desse propósito, o agente é todo aquele que é capaz de perceber o ambiente através de sensores e efetuar transformações nesse ambiente através de efetuadores.

2.2 Estruturas de agentes

Uma estrutura básica consiste no programa do agente e arquitetura do agente. O programa do agente é aquele que contêm a inteligência dele que analisará os dados obtidos dos sensores e passará para os efetuadores quais as ações necessitam ser tomadas, já a arquitetura é todo o aparato de dispositivos que o agente tem a sua disposição para perceber o ambiente.

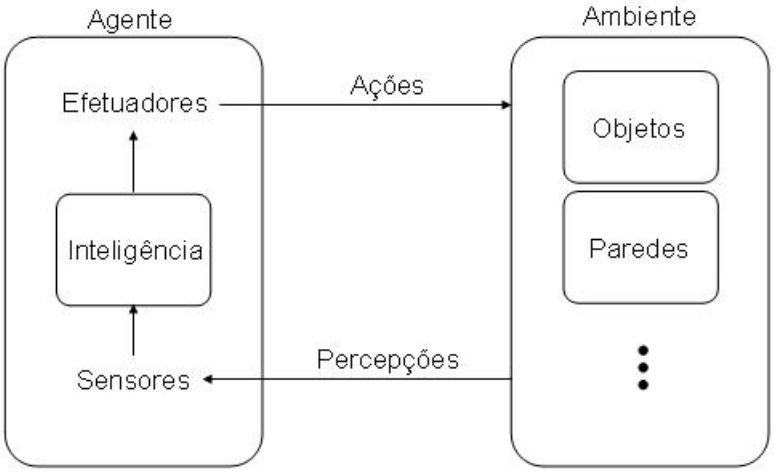

Figura 2.1: Ações entre ambiente e agentes inteligentes.

Um arquitetura de agente deve estar preparada para receber as requisições que forem enviadas à ela pelo programa de agente, não seria correto pedir que a arquitetura seguisse recomendações como Ative o Laser se a mesma não contiver o dispositivo. É necessário que as percepções e ações estejam adequadas com a inteligência que as controlará.

Os tipos de programas de agentes podem ser definidos em quatro categorias gerais. Que são:

- Agentes reativos simples.
- Agentes reativos baseados em modelo.
- Agentes baseados em objetivos.
- Agentes baseados em utilidade.

2.2.1 Agentes reativos simples

O tipo mais simples de agente, ele se utiliza apenas da condição atual para definir suas ações, deixando de lado todo o histórico de percepções que poderiam influenciar diretamente na ação atual. Ele utiliza-se apenas do que é chamado de regra da condição-ação, que define que toda vez que uma percepção atual do agente passa por uma condição, ele deve agir conforme o descrito naquela condição sem se quer analisar as anterioes. Essa é mesma inteligência que há no corpo humano no que diz respeito a reflexos inatos, aqueles que ocorrem de forma involuntária.

Tal agente tem regras muito específicas de como agir em determinadas sitações e como agir no geral (uma espécie de *default*) e normalmente capturam através dos sensores e passam para um mecanismo de inferência para simplesmente determinar em qual condição a leitura do ambiete se adequa e qual ação será tomada.

2.2.2 Agentes reativos baseados em modelos

Modelos são as informações de como é o funcionamento de um determinado conhecimento que é migrado para um conhecimento computacional e torna-se assim um modelo do mundo (o conhecimento do ambiente em questão). O agente que se utiliza do conhecimento prévio de "como as coisas funcionam"para poder tomar decisões de alteração de estado ou alteração do ambiente é o agente baseado em modelos.

Com isso temos que o agente tem que armazenar o seu estado interno com o histórico de percepções previamente inseridas para assim ter com o que comparar no momento em que se colocar em uma situação com as mesmas características.

2.2.3 Agentes reativos em objetivos

Quando levamos em conta um destino ou um objetivo que o agente deve atingir, assim como um robô que deve localizar uma determinada sala, esse agente está agindo com foco no objetivo. Além da necessidade de armazenar o estado, quando se leva em consideração um objetivo, o agente deve conhecer algo que diga a ele quando parar (algum marco do objetivo). Assim ele poderá analisar se ele está mais próximo do objetivo e se para onde ele der ir.

2.2.4 Agentes baseados em utilidade

No momento em que um agente questiona se está seguindo por um caminho que terá maior êxito, ele está questionando a utilidade daquela ação. Os agente baseados em utilidade são aqueles que sabendo de seus objetivos e seu estado atual definem a melhor possibilidade para alcançar o objetivo(s) e poderando sempre se a importância do objetivo e seu estado atual.

2.3 Teste de *Turing*

O Teste de *Turing* é utilizado para analisar se um programa, no nosso caso um agente artificial, é capaz de interagir com um ser humano como se fosse um outro ser humano.

O teste inicialmente foi feito da seguinte forma, dois humanos e um agente artificial eram colocados isolados e por meio de uma comunicação (um computador) um interregogador, que estaria se comunicando com os três simultâneamente, deverá conseguir identificar quem é o agente. Caso o interrogador não consiga identificar ele através desse "diálogo"o agente passaria no teste, caso contrário falharia.[4]

Mesmo sendo uma representação simples, muitas empresas ainda utilizam o Teste de *Turing* (de forma mais adequada aos propósitos delas) para testar alguns softwares.

2.4 Cognição

A capacidade de processar informações que qualquer sistema tem, isso é o que define a cognição. Seja através de percepções, pensamentos, mémoria ou mesmo o raciocínio sobre algo. A palavra foi definida pela primeira vez na época de Platão e citada por várias vezes por seu discípulo Aristóteles.

O campo da ciência cognitiva reuni conhecimento de inteligência artificial e conhecimentos de psicologia no que diz respeito a construção de modelos precisos e verificáveis dos processos de funcionamento da mente humana. A ciência cognitiva precisa necessáriamente de vários seres humanos ou animais. Já para fazer os testes de inteligência artificial leva-se em consideração, em nosso caso, apenas um computador.

No início da IA, era idetificado que um bom algoritmo para uma determinada tarefa era necessariamente um bom modelo de desempenho para o humano e vice-versa. Na atualidade são vistos como campos distintos, esse desacoplamento fez com que ambos os campos desenvolvem separamente, o que auxiliou ambas, pois as evoluções em uma área poderiam ser usadas pela outra de forma "não-obrigatória".

Capítulo 3

Ambiente

De uma forma geral, um ambiente é um conjunto das situações e/ou condições onde existe determinado objeto ou ocorre uma determinada ação. No caso desse projeto o ambiente significa o local onde está inserido um agente que deve navegar, mapear e explorar

Antes de imaginar um agente inteligente, deve-se pensar em um ambiente de tarefas, que será utilizado para ser o "problema"em si que os agentes devem utilizar para gerar suas "soluções". Esse problema é chamado de ambiente de tarefa.[5]

3.1 Especificação de um ambiente

O ambiente de tarefa, onde o agente irá realizar suas tarefas, tem que ser o mais completo possível. Lembrado que o agente deve ser inserido em um ambiente que seja apropriado ao seu propósito. Alguns fatores serão levados em consideração ao projetar um agente [3]:

1. Medida de desempenho: Qual será a o objetivo do projeto? minimizar custos, minimizar consumo de energia, otimizar rotas, melhorar mapeamente etc.

2. Ambiente: Que tipo de ambiente? Estático, determinístico, previsível etc.

3. Agente: Que tipo de agente? Humanóide, próprio para ambiente aquático, voador etc.

4. Sensores: Que tipo de sensores devem ser inseridos no agente? Lasers, sonares, camêras etc.

Esses fatores serão de enorme importância ao se determinar um ambiente.

3.2 Propriedades de ambientes de tarefas

A quantidade de ambientes de tarefas que podem ser criados é enorme, no entanto, podemos subdividir em categorias. Essas categorias determinam um projeto apropriado para o agente e as principais técnicas que devem ser implementadas no agente. As categorias são:

3.2.1 Completamente observável e parcialmente observável

Se os sensores de um agente permitem acesso ao estado completo do ambiente em cada instante, dize-se que o ambiente é completamente observável. Um ambiente tarefa é completamente observável se os sensores detectam todos os aspectos que são relevantes para escolha de uma ação, relevância tal que depende da medida de desempenho. Um ambiente é parcialmente observável quando devido a ruído, sensores imprecisos, sensores mal configurados ou parte do ambiente estão ausentes nos dados do sensor.

3.2.2 Determinístico e Estocástico

Se o próximo estado do ambiente é completamente determinado pelo estado atual e pela ação executada pelo ambiente, dizemos que o ambiente é determinístico; caso contrário, ele é estocástico. A princípio, um agente não precisa se preocupar com a incerteza em um ambiente completamente observável e determinístico. Porém, se o ambiente for parcialmente observável, ele poderá parecer estocástico. Isso é verdadeiro se o ambiente é complexo, tornando-se difícil de controlar todos os aspectos não-observados.

3.2.3 Episódico e Sequencial

Em um ambiente de tarefa episódico, a experiência do agente é dividida em episódios atómicos. Cada episódio consiste na percepção do agente, e depois na execução de uma única ação. É crucial que o episódio seguinte não dependa das ações anteriores, executadas em episódios anteriores. Por outro lado, em ambientes sequenciais, a decisão atual poderá, na maioria das vezes certamente irá, afetar todas as decisões futuras, como por exemplo o jogo de xadrez onde cada mudança afeta todas as posteriores.

3.2.4 Estático e Dinâmico

Caso o ambiente mude enquanto o agente está se movimentando (seja para mapear, navegar etc), dizemos que o ambiente é dinâmico para esse agente; caso contrário, ele é estático. Ambientes estáticos são muito mais fáceis de manipular e observar, por que o agente não precisa continuar a observar o mundo enquanto está decidindo sobre a realização da ação, nem precisa se preocupar com o tempo, mas o nível de desempenho do agente se alterar, podemos dizer que o ambiente é semi-dinâmico.

3.2.5 Discreto e Contínuo

A distinção entre discreto e contínuo pode se aplicar ao estado do ambiente, do modo como o tempo é tratado, e ainda às percepções e ações do ambiente. Por exemplo, um ambiente de estados discretos como um jogo de xadrez tem um número finito de estados distintos. A entrada proveniente de câmeras digitais é discreto, mas em geral é tradada como representação de intensidades e posições que variam continuamente, logo, é contínuo.

3.2.6 Agente único e Multiagente

A distinção entre ambiente do agente único e o ambiente dos multiagentes pode parecer bastante simples. Por exemplo, um agente que resolve um jogo de palavras cruzadas sozinho está claramente em um ambiente de agente único, enquanto um agente que joga

Poker está claramente em um ambiente multiagente.

3.3 Ambiente tarefa para agente virtual

Os ambientes tarefa que são utilizados com agentes virtuais, normalmente, tem seu foco em simular ambientes reais sejam eles quais forem. Dessa forma os ambientes tarefa funcionam como simuladores para os agentes virtuais agirem a fim de "solucionar"o problema que aquele ambiente propõe.

Em sua grande maioria os ambientes tarefas virtuais são completamente observáveis para facilitar a implementação do agente, mesmo sabendo que isso pode variar e muito no ambiente real, bem como o ambiente também é determinístico para que o agente saiba exatamente qual será o próximo estado para o qual ele irá.

Um ambiente tarefa virtual pode ser dinâmico ou estático de acordo com o propósito, assim como discreto ou estático. Em quase todos os casos também se observa que são episódicos, também para facilitar a implementação do agente.

Os agentes únicos ou multiagentes (vários agentes) são amplamente utilizados. Mesmo com propósitos diferentes, os agentes únicos são bons para testar a capacidade de um ser naquele ambiente, sendo ele único e com uma inteligência mais robusta, já ao utilizar-se de multiagentes, o uso de uma grande massa permite que a inteligência seja limitada porém a área de cobertura, comunicação entre outros atributos tornam eles tão capazes (se não mais capazes) que os agentes únicos.

Capítulo 4

Robótica

A ciência robótica é reponsável pela parte da tecnologia que tem por intuito otimizar tarefas feitas por humanos e, em alguns casos, substituí-los por motivos que vão desde a preservação da integridade do ser humano até mesmo a ocupação de seu cargo de trabalho. Alheios a um mundo de filmes e preconceitos, os robôs tornam os resultados dos serviços melhores e sua precisão é muito maior que a de um funcionário humano.[3]

Figura 4.1: À esquerda, robô que trabalha como policial, no Japão. À direita, robô criado para ser maestro de uma orquestra.

Os robôs são agentes que executam tarefas no mundo físico. Para isso utilizam dispositivos para realizar sua ações: os sensores e os efetuadores. A grande maioria dos robôs podem ser incluídos em três categorias principais:

1. Robôs manipuladores
2. Robôs móveis
3. Robôs híbridos

4.1 Sensores

A robótica se utiliza de vários dispositivos para emular os sentidos e as reações humanas em determinadas situações. Esses dispositivos tratam diferentemente cada um dos sentidos humanos. Os sensores passivos são aqueles que fazem a observação, propriamente dita, do ambiente, como por exemplo câmeras, que observam os sinais gerados por outras fontes no ambiente. Em contra partida, os sensores ativos são aqueles que a partir dele emite-se uma energia para obter medições do ambiente, assim como um sonar, que é um transdutor ultra-sônico.

Muitos agentes utilizam telêmetro, sensores capazes de medir distâncias, para definir o quão próximo ele está do objeto. Um desses sensores é o telêmetro a laser que através de uma varredura obtem-se a distância dos pontos onde a emissão do laser colidiu com um objeto. Alguns sensores medem distâncias mais curtas e são chamados de táteis. Estes são sensores de pequenas distâncias, painéis de choque e pele sensível. Eles são usados para tarefas de maior precisão como segurar um vidro, por exemplo.

Outro tipo de sensores são os de tratamentos de imagem, tais como câmeras que analisam o ambiente para extrair suas característicase. Entre vários tipos de visão computacional, uma deve ser levada em consideração quando falamos em robótica, é a estereoscópica que analisa a imagem quanto a sua profundidade. Os odômetros também são importantes e se alocam na categoria sensores proprioceptivos que são aqueles que fornecem informações ao agente sobre ele próprio.

4.2 Efetuadores

Os sensores são os dispositivos para obter informações do ambiente, já os efetuadores são aqueles que agem sobre o ambiente a fim de mover o agente por ele ou efetuar uma alteração de forma no ambiente, ou em sí. O grau de liberdade (GLD) é um conceito usado para saber a quantidade de direções independentes que um agente pode se mover. Todas essas direções em um agente são chamados de estado cinemático. Já um estado dinâmico é aquele que inclui uma dimensão adicional para a taxa de mudança de cada dimensão cinemática.

Enquanto agentes não-rígidos podem ter por exemplo um grau de liberdade adicional, como um cotovelo que garante mais um grau de liberdade e é inerente ao agente, os agentes rígidos tem um grau de liberdade a menos com a falta de articulações como a anterior.

Para os agentes móveis a capacidade de locomoção pode ser através de vários tipos de tração. Tração diferencial possuem duas rodas (ou esteiras) acionadas de forma independente uma de cada lado. Quando ambas as rodas se movem na mesma velocidade e sentido movem o agente para frente ou para trás. Caso girem em sentidos opostos ele gira no mesmo ponto. Já na tração sincronizada cada roda é independente e pode mover-se e girar em seu próprio eixo. Tal movimentação é pouquíssimo usada pois se não for gerenciada corretamente pode causar problemas na movimentação incorreta.

Outro tipo movimentação é a utilização de pernas na movimentação do agente. Elas tem a capacidade de percorrer terrenos acidentados mais facilmente porém são extremamente lentas e sua contrução é difícil. Existem duas categorias de movimentação através dos pés:

- Dinamicamente estável: Onde o agente pode permanecer de pé desde que não pare de fazer movimentações, ou seja, ele necessita de movimentar-se para poder ficar de pé.

- Estaticamente estável: O agente tem capacidade de ficar de pé mesmo que esteja parado (ou seja, estático).

Dentre os mecanismos de movimentação são observadas várias formas de energia que mantêm a movimentação dos agentes. A grande maioria utiliza motores elétricos por ser

mais barato e ter mais dispositivos que utilizam esse tipo de energia no mercado. Existem também os tipos de atuação pneumática, que utiliza gás comprimido e a hidráulica que usa fluídos pressurisados.

4.3 Categorias da robótica

Os robôs manipuladores são aqueles que tem sua base fixa de forma a não se movimentar além do alcance de suas partes móveis, assim como um braço. Esse tipo de robô é muito utilizado nas linhas de produção em indústrias automotivas, sideúrgicas, metalúrgicas etc. Esses robôs normalmente possuem efetuadores que alteram a forma do objeto afetado, como soldas, cortadores, efetuadores de pressão, entre outros. Também existem manipuladores em tarefas mais precisas como em hospitais no auxílio em cirurgias delicadas.

Se os robôs manipuladores tem a característica de ficarem estáticos, os móveis são o oposto, eles podem se movimentar através de efetuadores de movimento. Robôs móveis são muito utilizados para alcançar locais que são insalubres aos seres humanos, tais como fundo do mar, superfície de vulcões, dutos de ar condicionado e inclusive superfícies de outros planetas, assim como foi feito em Marte com o Sojourner, nome do robô dado pela NASA, que peregrinou pelo solo do planeta vermelho.

Os robôs híbridos são equipados como os manipuladores e também são móveis, normalmente são vistos na sua forma humanóide. Muitos cientistas tentam ensinar esses robôs a efetuar tarefas domésticas, em alguns casos até mesmo interagir com humanos.

4.4 Arquiteturas em Robótica

Uma forma de estruturar os algoritmos usados para gerenciar o robô, assim como as ferramentas e linguagens utilizadas para produzir a inteligência do robô em questão é chamada de arquitetura de software de robótica. Ela tabém engloba todos os programas que participaram do software.

Existem várias arquiteturas, dentro da literatura as mais citadas são a arquitetura de subsunção e a arquitetura em camadas.

4.4.1 Arquitetura de subsunção

Essa arquitetura é uma estrutura formada por controladores a partir de máquinas de estados finitos. Cada nó é representado por uma dessas máquinas que contém testes de variáveis ou para rastreio da execução da máquina de estados relativo a um teste. Já os arcos são usados para envio de mensagens para outras máquinas ou para dispositivos do robô. As máquinas resultantes contém um relógio interno que controla a duração do percurso da mensagem no arco, por esse motivo, elas são chamadas de MEFAs (máquinas de estados finitos ampliadas).

O principal problema da utilização de MEFAs é que normalmente elas são implementadas com os valores brutos dos sensores, ou seja, sem tratamento quando se tem um mecanismo de inferência que analisará de forma precisa as informações para tomar uma decisão não se pode ter entradas imprecisas. Isso causa problemas no momento de um MEFA aferir uma entrada. Por esse motivo a arquitetura de subsunção é raramente usada em larga escala, apenas para fins didáticos

4.4.2 Arquitetura de três camadas

A arquitetura em três camadas consiste em uma camada reativa, outra executiva e a última deliberativa.

A camada reativa é aquele que se encontra no mais baixo nível, que controla o robô. Ela oferece um controle de análise repetitiva de sensor-ação. Tal ciclo capturará as leituras atuais dos sensores e atuará caso seja ordenado por uma camada superior.

A camada executiva serve como união entre a camada reativa e a deliberativa. Recebe as reações obtidas da camada reativa e as propaga para a deliberativa assim como recebe um conjunto de pontos gerados pela camada deliberativa e passa para a camada reativa atuar.

A camada deliberativa gera as soluções de planejamento baseadas em modelos previamente inseridos ou em modelos desenvolvidos através de aprendizado. Em geral todas as informações usadas por esta camada são retiradas da camada executiva.

Existem algumas variações da arquitetura de três camadas em vários softwares de robôs, porém devido sua rigidez, são adicionadas outras camadas para torná-la mais adaptável aos outros tipos de problemas.

Capítulo 5

SLAM

O mapeamento é uma funcionalidade que é tratada de muitas formas dentro da literatura. O uso de algoritmos de mapeamento permite que um agente móvel possa identificar sua posição em um ambiente desconhecido e identificar o local em que está inserido. Com o ambiente devidamente mapeado é possível otimizar a rota uma vez que o agente já o conhece.[6]

Algumas técnicas de mapeamento que foram estudadas:

- Técnica utilizando o algoritmo *Djkistra* e Subida de Montanha.[7]
- Método incremental convencional.
- Técnica baseada em grafos de visibilidade.
- SLAM (Simultaneous Localization and Mapping).

O método de mapeamento que será incluído no Toolkit Hórus será o SLAM (Simultaneous Localization and Mapping), tendo em vista que ele soluciona dois problemas clássicos da teoria das posições, que define a dificuldade de se localizar em um ambiente desconhecido e a dificuldade de mapear um ambiente onde não se sabe onde está.[8]

5.1 SLAM

O *Simultaneous Localization and Mapping* é uma técnica utilizada em agentes autônomos para o mapeamento de ambientes desconhecidos levando em consideração a sua posição atual como a posição inicial para início do mapeamento. Os sensores que podem ser utilizados para a implementação do mapeamento são diversos. Para a prova de conceito foi utilizado o odômetro, dispositivo que mensura distâncias percorridas, e o laser, dispositivo para detectar a presença de objetos na cena.[4]

O SLAM é composto por vários segmentos que são independentes e tem suas comunicações muito bem estabelecidas o que os torna mais flexíveis quanto aos algoritmos utilizados em cada um dos segmentos. Cada um dos segmentos tem uma enorme gama de algoritmos que o compõe. Foram incorporadas ao *Toolkit* apenas as mais otimizadas e relevantes para melhor utilização no processo.[9]

Esses segmentos são:

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

5.1.1 Landmark Extraction

A forma de gestão dos marcos (objetos) e dos pontos de movimentação (áreas de movimentação do agente) foi feita através de um grafo. A escolha dessa estrutura foi baseada na sua credibilidade e largo uso na literatura.[10]

Existem dois algoritmos que foram analisados para ser incorporados nesse segmento: o *RANSAC* e o *SPIKE*.

5.1.1.1 RANSAC

O RANSAC (*Random Sampling Consensus*) método interativo para estimativa dos parâmetros de um modelo matemático a partir de um conjunto de dados observados, que contém linhas. É um algoritmo não-determinístico, sentido em que produz um resultado razoável apenas com uma certa amostra dos dados de entrada e testa as outras partes dos dados, não escolhida inicialmente, para verificar se o modelo obtido e satisfatório. Em conjunto com o SLAM o RANSAC identifica uma linha de acordo com os pontos passados pelo laser, pode-se concluir que ali existe uma parede que impossibilita a transposição do agente.

while

- Houverem leituras de laser não associadas;
- **E** o número de leituras for maior que o limiar;
- **E** o número de iterações não for maior que o limite.

do
- Selecionar uma leitura de laser na lista.
- Seleciona uma quantidade S de exemplos de leitura do laser que estão associadas a uma quantidade D de graus daquele laser.
- Usando esses exemplos S e a leitura original para calcular o menor quadrado que se ajuste a linha.
- Determinar quantas leituras do laser estão dentro de X unidades de medida que melhor se ajustam a linha.

if O número de leituras do laser que ficam sobre a linha é maior que o Consensus **then**
- Calcular o novo mínimo quadrado que melhor se ajuste a linha, com base em todas as leituras e a linha formada anteriormente.
- Adicionar o melhor ajuste baseado no calculo anterior.
- Remover o número de leituras que cruzam a linha do total de leituras não associadas.

end if
end while

Tabela 5.1: Algoritmo RANSAC

Uma vantagem de RANSAC é a sua capacidade para fazer estimativa robusta nos parâmetros do modelo, ou seja, é possível estimar os parâmetros com um elevado grau de precisão, mesmo quando significativas quantidades de linhas estão presentes no conjunto de dados e por causa dessa robustez que o RANSAC foi escolhido para ser implementado.

5.1.1.2 SPIKE

O algoritmo de extração de marcos *SPIKE* faz a extração através da análise de um determinado montante de valores do laser, estimada uma diferença muito grande de um marco para outro, por exemplo, 0.6 metros, defini-se um *SPIKE* no marco que diferiu dos demais, isso serve para identificar grandes mudanças no ambiente, como por exemplo um laser lançado através das pernas de uma mesa que geraria grandes diferenças entre os feixes de laser que tocarem a(s) perna(s) e os laser que passarem por entre ela(s). Nesse momento o marco que foi detectado na perna da cadeira se torna um *SPIKE*.

O algoritmo leva em consideração um ambiente onde exitem muitas diferenças entre dois marcos, em ambientes onde não existem tantas diferenças, esse algoritmo não tem eficiência.

5.1.2 Data Association

O segmento *Data Association* (tradução livre: Associação de Dados) é responsável pela filtragem e associação dos dados obtidos através dos dispositivos do agente.

Uma vez que um marco seja visualizado em um passo do agente e esse mesmo marco é visto novamente em um novo passo, a sua posição mudou, em relação ao agente. O Data Association faz a analíse da posição autal do marco com a sua posição imediatamente aterior, com esse paralelo certifica-se que o marco existe ou se ele foi removido da cena.

Este segmento tem como saída para o State Estimation um lista indexada pela posição do agente contendo os marcos da cena. Dessa forma torna-se mais fácil a localização do marco e em que posição ele foi identificado.

5.1.3 State Estimation

O segmento *State Estimation* (tradução livre: Estimação do Estado) tem com objetivo analisar as informações passadas pelo Data Association e estimar as posições dos marcos e do agente, com essas informações ele prepara as posições de cada elemento na cena e analisa o estado anterior já gravado.

As posições dos marcos devem coicidir com as mesmas posições deles em posições anteriores do agente (caso os marcos sejam reobservados), essa garantia de que a posição irá coincidir é tida pelo cálculo da posição do marco num plano cartesiano. O cálculo é:

$$Landmark(x, y) = (AgentActualPositionX + \cos\theta, AgentActualPositionY + \sin\theta)$$

a posição do marco leva em consideração posição do robô para identificar em que posição o marco está a partir dele, isso torna o agente capaz de mapear baseado na posição dele e o ângulo é em relação ao laser do agente que colide com marco.

5.1.4 State Update

O *State Update* (tradução livre: Atualização do Estado) faz a gravação do estado atual do agente, sua posição, em relação a posição inicial, valores relativos ao odômetro. Esse passo do SLAM apesar de ter uma resposabilidade aparentemente pequena, leva-se muito tempo para definir o tipo de estrutura que será usada para gravação dos estados. Algumas, com o foco em performance, armazenam em listas devido alta velocidade de gravação, outros preferem uma lista indexada (também conhecido como *map*) para facilitar na busca pelas informações. A mais recomendada é uma lista indexada, utilizando a posição atual como chave.

5.1.5 Landmark Update

Com o mesmo objetivo e com focos diferentes, o Landmark Update (tradução livre: Atualização de marcos) faz a gravação do estado atual dos marcos, sua posição, em relação

a posição do agente. Assim como é no *State Update*, no *Landmark Update* é difícil definir a melhor estrutura de dados para armazenamento. A estrutura mais indicada nesse caso também é a lista indexada, que também deve ser indexada pela posição atual do agente.

Capítulo 6

O Toolkit Horus

O Horus é um *Toolkit*, ou seja, uma coleção de ferramentas (nesse caso módulos) que servem para gerenciar agentes inteligentes, escrito em *Python*. Duas partes estão sendo desenvovdidas a princípio: Módulo de Visão e Módulo de Mapeamento.

No módulo de visão estão os mais variados algoritmos de visão computacional e no módulo de mapeamento trata-se do problema de mapear ambientes a partir de dispositivos de leitura do ambiente e de efetuadores.

6.1 Objetivo

O objetivo do Toollkit é prover ferramentas necessárias para produção de agentes inteligentes, focando nos processos de navegação e mapeamento no que diz respeito à movimentação dos agentes em um ambiente e também na identificação de placas e reconhecimento dos caracteres no que diz respeito à parte de visão computacional disponibilizada neste toolkit denominado Horus

6.2 Módulos do Horus

As principais funcionalidades de cada um dos módulos são:

- Core do Horus: Contêm módulos que serão utilizados como suporte para os módulos principais. Sendo assim não necessitam ser utilizados diretamente pelo usuário.
- Módulo de Visão: Tem por objetivo tratar as principais técnicas de visão computacional de um agente inteligente.
- Modulo de Mapeamento: O módulo de mapeamento é responsável por gerenciar os tipos de mapeamento bem como os dispositivos utilizados para mapear e navegar no ambiente.
- Utils: Módulo que contêm funções de suporte para auxiliar no funcionamento dos demais módulos.

6.2.1 *Core* do Horus

O Core do Horus é onde se encontram as partes relativas ao agente e seus comportamentos.

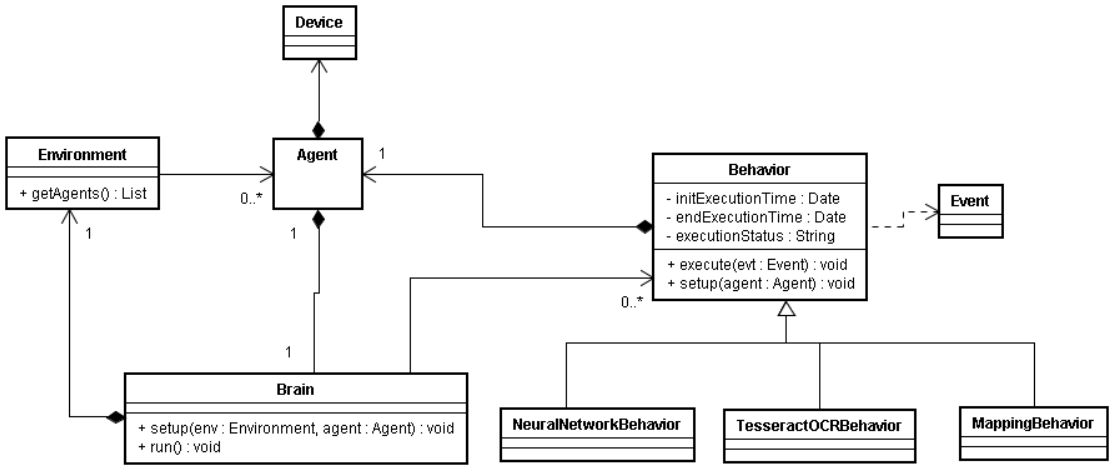

Figura 6.1: Diagrama que demonstra a arquitetura do Core do Horus.

A arquitetura representada acima demonstra de forma geral os integrantes do Toolkit Horus que são o agente inteligente (*Agent*), os dispositivos usados por esse agente (*Device*), o programa de agente (*Brain*), os eventos (*Event*) e a hierarquia de comportamentos

(*Behavior*).

Uma vez criado o agente ele precisa de dispositivos (classe *Device*), como lasers ou câmeras, que são necessários para cumprir uma determinada tarefa, ou seja, para atingir o seu objetivo. Além disto, é necessário atribuir a este agente a inteligência (classe *Brain*), isto é, os algoritmos que poderá usar para processar as informações obtidas com seus dispositivos e transformá-las em dados para a fase de pos-processamento.

Para que a aplicação possa usar um programa de agente ela precisa estender a classe *Brain* e implementar o método *run()* dessa classe.

Os comportamentos (classe *Behavior*) são os resposáveis por indicar ao agente o que é necessário ser feito durante o pos-processamento das informações capturadas, isto é, dos eventos que ocorrem e que são compreendidos pelo agente através de seus dispositivos e de seu cérebro.

Entende-se por eventos todos os dados compreendidos pelo agente e que influenciam nas suas decisões, como a detecção de colisões ou a captura e tratamento de imagens no ambiente.

O *Toolkit* Horus oferece algumas classes de comportamentos já definididas que podem ser usadas por qualquer aplicação. São elas:

- *NeuralNetworkBehavior*: Disponibiliza métodos para a construção e treinamento de redes neurais na implementação de um comportamento;
- *TesseractBehavior*: Disponibiliza a funcionalidade de *OCR* da *engine tesseract*, presente no Horus;
- *MappingBehavior*: Disponibiliza métodos responsáveis pelo processo de navegação, localização e mapeamento de ambientes desconhecidos com base na técnica SLAM.

A classe *Environment* representa o ambiente sobre o qual os eventos ocorrem e o agente trabalha. Para que o agente possa obter informações sobre o ambiente é preciso que o *Brain* seja configurado tanto com uma instância da classe *Agent* quanto uma instância da classe *Environment*.

É necessário definir a ordem de execucão quando o cérebro do agente é configurado com diversos comportamentos. Os comportamentos podem mudar de acordo com as condições identificadas, logo, o cérebro identifica as condições que cada comportamento necessita para ser executado e o ativa quando a sua condição de execução for satisfeita. Porém, 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 lembrando sempre que a maior prioridade deve ser a integridade do agente. A ordem de execução dos comportamentos define a máquina de estados de execução do agente inteligente, que se encaixa com a proposta e a utilização do cérebro no *Toolkit*.

6.2.2 Módulo de Visão

Módulo onde ficam as funções relativas a visão computacional para agentes inteligentes. Ele leva em consideração o uso de redes neurais para fazer a extração de características e também vários algoritmos de segmentação. Esse módulo tem como principal objetivo o reconhecimento de padrões. Para reconhecimento de uma placa é necessário identificar algumas características de uma imagem que servirão de padrões de entrada para uma rede neural.

Isso ocorre através de dois processos: Extração de características e reconhecimento de objetos.

- Extração de características - O objetivo deste processo é abstrair as características de um objeto para, a partir delas, reconhecê-lo usando padrões e usá-los posteriormente pelo método de reconhecimento de objetos.

- Reconhecimento de objetos - Esse processo utiliza as características que foram obtidas no processo anterior para a partir delas utilizar um classificador e com isso reconhecer o objeto em questão. No caso do Horus foi utilizada como classificador uma rede neural.

6.2.3 Módulo de Mapeamento

O módulo responsável pela inteligência do agente, no que diz respeito à localização, movimentação, mapeamento e navegação.

- Localização: Entende-se por localização a capacidade do agente localizar-se em um ambiente.
- Movimentação: É a capacidade de locomoção em um ambiente.
- Navegação: O agente se move pelo ambiente mapeado visando um objetivo, com tarefas como otimização de rotas.
- Mapeamento: É o modo como o agente localiza marcos para identificar a forma do ambiente.

6.2.4 Localização

A localização do agente é dada em coordenadas X, Y e Z. Inicialmente o agente conhece apenas a sua posição, que é dada pelas coordenadas (0, 0, 0). Independentemente da posição do agente, no que diz respeito ao ambiente, a inicial será sempre (0, 0, 0).

Conforme o agente mapeia o ambiente, ele seguirá alterando sua posição a partir da sua posição anterior com base em seu odômetro (dadas as métricas proporcionais).

No *Toolkit* isso funciona da seguinte forma:

- Obtém-se uma leitura da posição atual.
- Calcula-se o ângulo de rotação do agente para onde ele navegará.
- Armazena-se a distância percorrida baseando-se no odômetro.
- Calcula-se a nova posição com a distância percorrida e o ângulo de rotação.

6.2.5 Movimentação

A capacidade que o agente tem de ir de um ponto a outro usando recursos voluntários é o que chamamos do processo de movimentação. A movimentação não é orientada, não possui um objetivo, tratando-se apenas do processo físico de deslocamento, diferentemente da navegação que é planejada e executada obedecendo regras pré-determinadas ou geradas durante a execução do processo.

A movimentação é de total responsabilidade da aplicação.

6.2.6 Navegação

O proceso de navegação é um conjunto de processos de movimentação que seguem uma ou mais regras, por exemplo, ir sempre para o ponto mais distante possível. É fato que a navegacão tem influência direta no mapeamento, logo um método de navegação eficiente deve ser implementado na aplicação a fim de que o mapeamento seja também eficiente. Em nossa aplicação testamos duas variações de meios de navegação. Antes de tudo é interessante informar que apesar da navegação ser implementada na aplicação, a escolha do caminho a ser tomado pertence ao Horus e é feita através do método "seeNewWay"cujos passos serão mostrados abaixo. O primeiro método de navegação desenvolvido baseava-se em fazer com que o agente identificasse o ponto mais distante dentre todas as leituras do laser e então navegasse para aquela direção. Caso houvesse mais do que uma leitura a ser escolhida (por exemplo dois ou mais pontos infinitos identificados) o agente escolheria a primeira leitura entre elas. Essa forma de navegação era muito simples e o agente com frequência retornava para lugares já mapeados.

O segundo método de navegação desenvolvido usa o mesmo pricípio do primeiro, ou seja, a escolha do maior caminho, mas há algumas restrições para essa escolha. Essas restrições compõe uma série de passos (um algoritmo) que foi desenvolvida justamente para melhorar a navegação.

Os passoas abaixo fazem parte do método seeNewWay presente no módulo SLAM desenvolvido:

- Cada conjunto de leituras dos lasers deve ser armazenado temporariamente em uma estrutura de dados do tipo lista.
- A lista deve ser dividida em três partes: pontos que estão à esquerda do agente, pontos que estão à direita do agente e o ponto central, ou seja, o laser do meio.
- Dentre todos os pontos desta leitura, deve-se precisar a quantidade de pontos já mapeados encontrados em cada lado –direito e esquerdo.
- Escolher o lado que tenha menos pontos já mapeados, caso dê empate então o ponto central é escolhido.
- Tendo sido escolhido o lado, verificar qual a maior leitura deste lado e navegar para aquela direção.

Com este novo método, é possível garantir que o agente sempre preferirá ir para lugares onde hajam menos pontos mapeados.

6.2.7 Mapeamento

O mapeamento do agente baseia-se em um processo chamado tentativas de criação de *mark points*. O Horus disponibiliza esse processo através de um método criado em nosso trabalho chamado *tryAMarkPoint*.

Mark points (pontos de marcos, ou simplesmente marcos) são circunferências criadas durante a navegação do agente. O algoritmo que gera os *mark points* é uma nova proposta apresentada pelo corrente trabalho para resolver o problema de mapeamento.

Em contrapartida ao SPIKE e ao RANSAC que buscam extrair *landmarks* utilizando-se das colisões dos lasers e esperando que essas colisões atendam a um requisito, no caso do RANSAC espera-se a formação de uma linha, no caso do SPIKE espera-se uma diferença muito grande entre um marco e outro, o processo de criação de *mark points* leva em consideração que todas as colisões dos lasers são *landmarks*, mas nem todos são armazenados.

O landmark escolhido é denominado mark point e é aquele que não esteja em um ponto por onde o agente já passou.

Essa identificação é feita a partir do método de criação de *mark points* que nesse trabalho é chamado *tryAMarkPoint*.

O algoritmo do *tryAMarkPoint* é o seguinte:

tryAMarkPoint(*tupleposicao_robo*, *floatrotacao_robo*)
for all *mark_point in lista_mark_point* **do**
  RESULTADO ← *isPointInCircle*(*mark_point*, *posicao_robo*, *raio_circunferencia*)
  if RESULTADO is *FALSE* **then**
    ADIOCIONAR ← *VERDADEIRO*
  else
    TENTATIVAS ← TENTATIVAS - 1
  end if
end for
if ADICIONAR is **then**
  TENTATIVAS ← TENTATIVAS + 2
  Adiciona a posição do robô à lista de *mark points*
end if
return *TENTATIVAS*

Tabela 6.1: Algoritmo *tryAMarkPoint*

NOTA: isPointInCircle é um método criado no módulo matemático responsável por identificar se um determinado ponto está ou não presente em um círculo de raio N. Os *mark points* nada mais são do que círculos criados com raio N e centro na posição do agente no momento da descoberta do mark point.

O método retorna o número de tentativas para se criar *mark points*. A aplicação que usa o método deve verificar a condição de parada do agente utilizando esse número de tentativas.

Este é um conceito de punição e recompensa, ou seja, se o agente encontra um novo mark point –que significa um ponto ainda não explorado –então ele é recompensando com créditos, caso ele passe por um ponto já explorado ele é punido com a perda de créditos. No nosso trabalho, o agente mapeia o ambiente enquanto busca sempre encontrar novos *mark points*, ou seja, busca sempre aumentar os seus créditos e pára de mapear quando

esses créditos assumem um valor negativo ou zero, o que significa que não há mais pontos a serem descobertos (mapeamento completo)

6.2.8 Outros

Dentro deste módulo também se enquandram o módulo Graph e suas funcionalidades, que são:

- *ShortestPath* - Função que retorna o menor caminho de um grafo tendo em vista o algoritmo que for passado, na atual conjultura do projeto apenas um algoritmo foi implementado, o Dijkstra.
- *ShowAllPaths* - Retorna todos caminhos possíveis até um dedterminado nó independente de ser o mais distante ou o mais próximo.
- *BuildaGraph* - Como a construção de um grafo necessita da criação correta dee nós e arestas, foi criado este método para construir um grafo e retorná-lo ao final do processo.
- *Geters* e *Seters* de nós e de arestas do grafo.

6.2.9 Utils

Conforme o nome do módulo sugere, ele torna os códigos dos demais módulos livres de reimplementações de coisas comuns a todos os módulos. Tal módulo pode ser usado tanto em conjunto com o toolkit quanto a parte, devido sua usabilidade alta e seu nível de acoplamento baixo.

Algumas funções e métodos presentes no *Utils* do Horus estão dispostas como segue:

6.2.9.1 O módulo *"math_module"*

Este módulo possui funções matemáticas usadas pelo Horus. Ele está subdividido em algumas categorias dentre elas, funções trigonométricas e regressão linear.

As funções trigonométricas são:

- **getXCateto(self, hypotenuse, angle)** - responsável por encontrar a coordenada X em um plano cartesiano tendo como parâmetros a distância (hipotenusa) até o ponto e o ângulo de rotação atingido para observar aquele ponto.
- **getYCateto(self, hypotenuse, angle)** - responsável por encontrar a coordenada Y em um plano cartesiano tendo como parâmetros a distância (hipotenusa) até o ponto e o ângulo de rotação atingido para observar aquele ponto.
- **isPointInCircle(self, center_tuple, point_tuple, radius)** - responsável por definir se um ponto (X, Y) qualquer está ou não contido em um círculo, dado o centro deste e o raio.

Além das funções trigonométricas existem também as funções relativas a regressão linear que é responsável por fazer uma identificação de uma condicional de uma variável y, tendo dados de vários x.

Capítulo 7

Aplicação com ambiente virtual e agente autônomo

A fim de produzir uma prova de conceito, foi implementada uma aplicação onde foi possível demonstrar a eficiência do algoritmo de mapeamento. Foi produzido um simulador com gravidade, colisão, rederização de texturas, objetos e atores (o agente nesse caso) e, tendo esse simulador como base, foi criada a aplicação que tem um agente utilizando-se do Toolkit Horus para mapeamento e navegação dentro do ambiente simulado. Aguns conceitos foram tirados do [11]

7.1 Simulador

Um simulador é todo software (em nosso caso) que simula um comportamento de algum sistema. Deve ser capaz de reproduzir, de forma mais fiel possível, a realidade na qual ele tenta emular. No nosso caso foi desenvolvido um simulador de um ambiente, a fim de virtualizar um ambiente real.

O simulador desenvolvido foi tratado como uma aplicação a parte, pois não estava incluído no escopo do projeto, porém tendo em vista a necessidade de um simulador customizado para a realidade de um agente móvel (e na linguagem selecionada) o simulador

foi incluido ao projeto.

Alguns requisitos foram necessários para que o simulador fosse projetado. Alguns softwares que foram utilizados: *Blender* e *Panda3d*. Como uma das linguagens utilizadas foi *Python*, buscou-se ferramentas que fossem compatíveis (na verdade ambas são em *Python*) com as linguagens utilizadas.

7.1.1 Python

Python é um linguagem orientada a objetos de tipagem dinâmica, onde os tipos dos dados são atribuídos durante a interpretação do código dinamicamente. O Python tem foco na produtividade com uma sitaxe legível e simples, bem como módulos auxiliares (grande maioria OpenSource) que garatem um excelente desempenho.

7.1.2 Blender

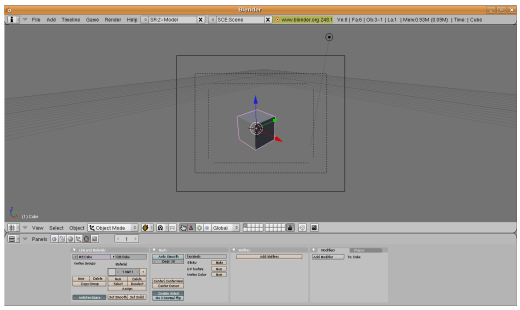

Figura 7.1: Tela inicial do Blender.

O Blender é um produto da Blender Fundation, conforme pode ser visto na imagem 7.1, que é open source e desenvolvido em Python para modelagem de objetos 3D que está disponível para vários sistemas operacionais sobre a licença GNU (*General Public License*). Por ser multiplataforma permite que as modelagens criadas possam ser executadas em outros sistemas operacionais que não o de origem. O Blender foi usado para criar os modelos usados nas aplicações desenvolvidas como prova de conceito do *Toolkit* Horus

7.1.3 Panda3D

O *Panda3D* é um produto da equipe de desenvolvimento da Walt Disney para renderização de jogos e ambientes virtuais em terceira dimensão. Estão sobre licença da *BSD License* (com algumas modificações para adequação a realidade do projeto). Foi desenvolvido em Python e é multiplataforma. Por isso, assim como o Blender, foi escolhido para uso em conjunto do *Toolkit* Horus. O uso do Panda3D no *Toolkit* Horus restringe-se apenas à manipulação dos modelos previamente criados no Blender para então criar a aplicação que foi a prova de conceito. Em outras palavras, o papel do Panda3D não está diretamente relacionado ao funcionamento do *Toolkit*, ele é usado apenas para criar a aplicação que usa o *Toolkit* Horus.

Abaixo, um exemplo desenvolvido com no Panda3D:

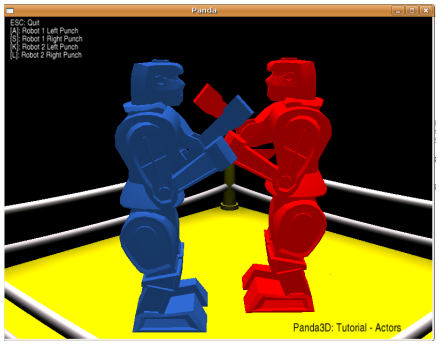

Figura 7.2: Um exemplo de aplicação utilizando Panda3D.

7.2 Ambiente

O ambiente em que o agente foi testado consiste em uma área que simula um galpão com 5 cômodos onde o agente inicia sua movimentação no cômodo 1 e pode alcançar qualquer ponto do ambiente a partir de sua posição inicial. O ambiente que foi desenvolvido tem como objetivo emular, em menor escala, um ambiente real.

O agente utilizado foi inspirado no personagem de um filme que também é um agente

inteligente, é possível notar a semelhança na figura 7.3.

Figura 7.3: O agente utilizado nos testes. Modelado em *Blender3D*.

A figura 7.4, que demonstra o ambiente que foi utilizado, pode-se ter uma visão em perspectiva do ambiente modelado. Ele foi modelado em Blender, tal ferramenta proporcionou os recursos de modelagem UV, texturização, determinação de medidas precisas, entre outros. O Blender foi selecionado, também, por conta da sua legibilidade de fácil assimilação tendo em vista que o *Toolkit* foi desenvolvido em *Python* e a ferramenta utiliza a mesma linguagem para produzir os modelos.

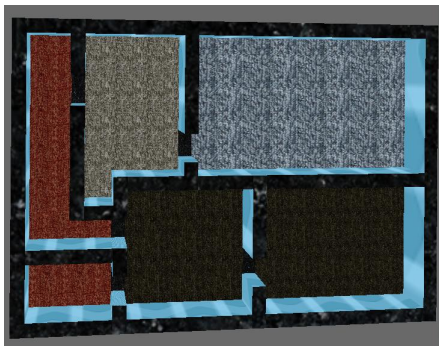

Figura 7.4: O ambiente utilizado para a prova de conceito. Modelada em Blender3D.

O Panda3d foi utilizado para fazer todos os tratamentos de colisão e virtualização de camêras e renderização do agente no ambiente. Os lasers, áreas de colisão são tratados conforme a imagem 7.5 Como o Blender3D agilizou o processo o fato do *Panda3D* ser em

Python.

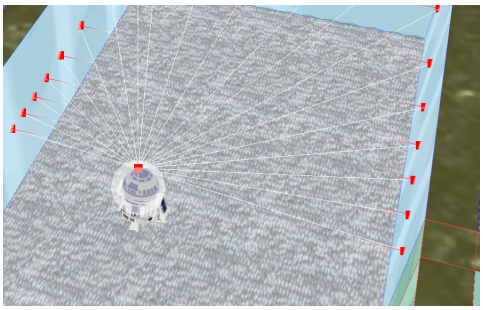

Figura 7.5: Os lasers projetados a partir do agente. Gerenciados pelo *Panda3D*.

Capítulo 8

Resultados obtidos

Durante o desenvolvimento muitos testes e ajustes foram executados na expectativa de atingir o objetivo de forma mais rápida e eficiente, ou seja, mapear todo ambiente com o menor tempo possível. Para isso alteramos o tamanho do raio dos *mark points*, o tamanho dos lasers e o tamanho do ambiente empiricamente, ou seja, testando e observando os resultados obtidos.

8.1 Quanto ao tamanho do ambiente

Ambientes muito grandes não prejudicaram o processo de mapeamento, mas o tornou muito lento. Enquanto que ambientes muito pequenos fizeram com que o número de colisões do agente nas paredes aumentasse consideravelmente, prejudicando a navegacão.

O ambiente ideal deve ter o triplo do tamanho do agente, sendo que o dobro também garante boa navegacão e mapeamento. Matematicamente: sendo *xa, ya e za,* as coordenadas ideais do ambiente, e *xr, yr, zr* as coordenadas do agente, $xa = 3xr$, $ya = 3yr$ e $za = 3zr$

8.2 Quanto à distribuicão dos lasers e sua quantidade

Nestes testes levamos sempre em consideracao o custo em processamento de informação (para o caso de uma quantidade muito grande de lasers) e a exatidão da interpretação dos dados a fim de proporcionar uma boa navegação pelo ambiente, pré-requisito para um bom mapeamento.

Em outras palavras, o agente deve ter a menor quantidade de lasers possível para não sobrecarregar o processamento das informações e garantir agilidade. Contudo, se por um lado muitos lasers causam um desperdício de processamento, poucos lasers resultam em um mapeamento ineficiente, pois o agente volta várias vezes em pontos muito próximos dos que já foram mapeados, ou seja, o número de *mark points* aumenta de forma considerável e desnecessária.

O número de lasers do agente deve ser em torno de 22, variando em até 3 unidades.

Um outro problema que causa um mapeamento ineficiente é se os lasers forem muito curtos, pois o agente tenderá a andar pelos cantos das salas e quando estes cantos estiverem mapeados ele irá para longe das paredes e ficará andando em círculos até que seus créditos terminem. O ideal é que o tamanho do laser seja a média aritmética das coordenasdas *(x, y, z)* do agente.

Matematicamente: sendo $\mathbf{l}$ o tamanho ideal de cada laser,

$$l = \frac{xr + yr + zr}{3}$$

A distribuição ideal dos lasers é a que proporciona um campo de visão de 120 graus, sendo 60 graus para a esquerda e 60 para direita, apesar de um campo de 180 graus também proporcionar uma navegação e mapeamento eficientes, contudo, neste último caso o cuidado com o tamanho dos lasers deve ser maior, pois tendo o agente uma visão muito ampla (de 90 graus para cada lado), ele poderá retornar para um cômodo já mapeado com mais freqüência. Um campo de visão acima de 180 graus é desencorajado, visto que o agente quase sempre retornará para um cômodo já mapeado.

Em contrapartida, um campo de visão estreito (menor do que 120 graus) faz com que o agente ignore portas e passagens freqüentemente a menos que estas estejam à sua frente.

8.3 Quanto ao raio dos mark points

O agente cria mark points conforme navega e descobre novos caminhos. Quanto menor o raio *mais mark* points são encontrados, mais créditos o agente ganha, mais vezes ele retorna nos cômodos e mais tempo demora o mapeamento. O contrário ocorre quanto menor for o raio.

O raio ideal, para um abiente ideal, é a média aritmética das coordenadas (x, y, z) desse ambiente.

Matematicamente: sendo **r** o raio dos *mark points*,

$$r = \frac{xa + ya + za}{3}$$

8.4 Quanto à aplicação integrada

Ao termino do SLAM e da aplicação que foi sua prova de conceito, uma outra aplicação que foi desenvolvida utilizando a parte de mapeamento em conjunto com a parte de visão do Tool Kit Horus, comprovando assim a interoperabilidade entre os módulos.

A aplicação integrada tem todas as características do módulo de mapeamento desenvolvido e utiliza a parte de reconhecimento de padrões do modulo de visão, com isso o agente pode reconhecer marcações distribuídas pelo ambiente para ajudar na navegação e atingir o seu objetivo de forma mais rápida.

Para isso, um novo ambiente virtual foi construído. Neste ambiente disponibilizamos placas com setas indicadoras de salas e suas direções (direita ou esquerda). O agente é capaz de não apenas identificar a placa no ambiente, mas também compreender o que está escrito nesta placa e decidir qual direção tomar posteriormente. Isto é feito através de marcos no chão que são disponibilizados imediatamente antes de cada placa.

Nesta aplicação o agente possui duas câmeras, uma com sua visão de primeira pessoa e outra com a visão voltada para o chão. O propósito desta última é identificar aqueles marcos, pois quando o agente encontra um marco é certo que por aquela região existe uma placa, então a câmera de primeira pessoa entra em ação em busca da placa.

É interessante informar que o uso de marcos no chão não é obrigatório, o agente poderia apenas procurar pelas placas e ter só uma câmera - a de primeira pessoa - mas isso tornaria a navegação extremamente lenta, pois o agente iria buscar a placa em vão com muita freqüência.

Uma vez que a placa tenha sido encontrada, o algoritmo de reconhecimento de padrão entra em ação para identificar os caracteres presentes na placa e também as setas e suas direções. A imagem abaixo mostra uma das placas que estão no ambiente:

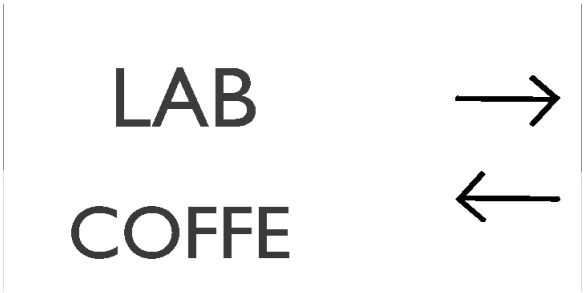

Figura 8.1: Placa utilizada na aplicação integrada.

Portanto, a aplicação integrada comprova o funcionamento em conjunto dos módulos desenvolvidos de mapeamento e de visão, atribuindo ao agente as características essenciais para alcançar o seu objetivo de forma mais eficaz, ainda que em um ambiente virtual.

Capítulo 9

Conclusões e Trabalhos Futuros

Há muito que melhorar para garantir que o processo de mapeamento seja mais produtivo.

Como trabalhos futuros iremos em primeiro momento concluir o ransac, método responsável por encontrar landmarks a partir de pontos extraídos com leitura de lasers. Com isso, o agente será capaz de identificar paredes e poderá usá-las para a sua navegação e mapeamento.

Apesar de o método ransac estar pronto e fazer parte do nosso módulo SLAM, ele ainda não está em uso, pois não foi necessário no ambiente-tarefa das aplicações. Contudo, testes individuais executados no método indicam um bom funcionamento com taxas de erros muito baixa. Entretanto na aplicação o numero de *landmarks* encontrados esta abaixo do esperado. Os ajustes no ransac serão o de desconsiderar as leituras dos lasers das pontas, por possuírem uma discrepância em relação a outras leitoras e gerarem erro no RANSAC, excluir os pontos com o valor considerado infinito na aplicação e fazer o calculo do intervalo de confiança da linha encontrada.

Com o RANSAC pronto iniciaremos o módulo que criará um grafo completo a partir dos marcos encontrados. Com os landmarks é possível fazer a poda do grafo e usar esse grafo para buscar o melhor caminho com algoritmo genético. No atual ambiente tarefa não foram considerados os erros dos sensores. No entanto, com a adição desses erros, módulos de tratamento de incerteza utilizando Lógica *Fuzzy* e Filtro de *Kalman* passarão a entregar

o Horus.

Referências Bibliográficas

1 BELLMAN, R. *An introduction to artificial intelligence: Can Computers think?* [S.l.]: Paperback, 1978.

2 BITTENCOURT, G. *Inteligência Artificial: Ferramentas e Teorias.* [S.l.]: Editora da UFSC, 2001.

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

4 ROSENSCHEIN, S. J. *Formal theories of knowledge in AI and robotics.* [S.l.]: New Generation Computing, 1985.

5 ESPERANçA, F. N. e C. Implementando células e portais no blender. *Seminário LCG*, Julho 2006.

6 DALFOVOS, C. R. C. de Abreu e D. O. Simulação de robôs aplicando o algoritmo de dijkstra e subida da montanha. *Seminário de Computação, 2005, Blumenau - SC. XIV SEMINCO. Blumenau - Sc : FURB*, Outubro 2005.

7 BIGGS, N. L. *Graph Teory.* [S.l.]: Oxford University Press, 1989.

8 AHO, A. V.; HOPCROFT, J. E.; ULLMAN, J. D. *The design and analysis of computer algorithms.* [S.l.]: Paperback, 1974.

9 MONTEMERLO, M. et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Pro-ceedings of the AAAI National Conference on Articial Intelligence*, 2002.

10 LOZANO-PéREZ, T.; WESLEY, M. A. *An algorithm for planning collision-free paths among polyhedral obstacles.* [S.l.]: Communications of the ACM, 1979.

11 SOUZA, J. P. R. P. de. *Mapeamento e localização de robôs móveis: Uma proposta baseada em marcos visuais e regras de produção.* Dissertação (Mestrado) — Universidade Federal da Bahia, 2005.

Apêndice A

Dependências do Tool kit

Módulos Python para o funcionamento completo do toolkit:

- FANN [http://leenissen.dk/]
- MATPLOTLIB [http://matplotlib.sourceforge.neT/]
- NUMERICS [http://numpy.scipy.org/]
- PIL [http://www.pythonware.com/products/pil/]
- SWIG [HTTP://www.swig.org/]
- TESSERACT [http://code.google.com/p/tesseract-ocr/]

Apêndice B

Siglas

- GNU - General Public License
- IA - Inteligência Artificial
- MEFA - Mmáquina de Estados Finitos Ampliadas
- NASA - National Aeronautics and Space Administration
- OCR - Optical Character Recognition
- RANSAC - Random Sampling Consensus
- SLAM - Simultaneos Location and Mapping
- UV - Refere-se a latitude e longitude.