THIAGO BECKER

NAVEGAÇÃO DE UM ROBÔ MÓVEL BASEADO EM UM MODELO DE CONSCIÊNCIA ARTIFICIAL

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Curitiba PR
2015

THIAGO BECKER

NAVEGAÇÃO DE UM ROBÔ MÓVEL BASEADO EM UM MODELO DE CONSCIÊNCIA ARTIFICIAL

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Área de concentração: *Engenharia de Sistemas Computacionais*

Orientador: João Alberto Fabro
Co-orientador: André Schneider de Oliveira

Curitiba PR
2015

Dados Internacionais de Catalogação na Publicação

B396n
2015

Becker, Thiago
Navegação de um robô móvel baseado em um modelo de consciência artificial / Thiago Becker.-- 2015.
91 f. : il. ; 30 cm

Texto em português, com resumo em inglês
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada, Curitiba, 2015
Bibliografia: f. 87-91

1. Inteligência artificial. 2. Robôs - Sistemas de controle. 3. Computação - Dissertações. I. Fabro, João Alberto, orient. II.Oliveira, André Schneider de, coorient. III. Universidade Tecnológica Federal do Paraná - Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD: Ed. 22 -- 621.39

Biblioteca Central da UTFPR, Câmpus Curitiba

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 39

Aos 14 dias do mês de agosto de 2015 realizou-se na sala B-204 a sessão pública de Defesa da Dissertação de Mestrado intitulada "Navegação de um Robô Móvel baseado em um modelo de consciência artificial", apresentada pelo aluno **Thiago Becker** como requisito parcial para a obtenção do título de Mestre em Computação Aplicada, na área de concentração “Engenharia de Sistemas Computacionais”, linha de pesquisa “Sistemas Embarcados”.

Constituição da Banca Examinadora:

Prof. Dr. João Alberto Fabro, UTFPR - CT (Presidente) _____

Prof. Dr. André Schneider de Oliveira, UTFPR - CT _____

Prof. Dr. Eduardo Todt, UFPR _____

Prof. Dr. Ricardo Dutra da Silva, UTFPR- CT_____

Em conformidade com os regulamentos do Programa de Pós-Graduação em Computação aplicada e da Universidade Tecnológica Federal do Paraná, o trabalho apresentado foi considerado _____ (aprovado/reprovado) pela banca examinadora. No caso de aprovação, a mesma está condicionada ao cumprimento integral das exigências da banca examinadora, registradas no verso desta ata, da entrega da versão final da dissertação em conformidade com as normas da UTFPR e da entrega da documentação necessária à elaboração do diploma, em até _____ dias desta data.

Ciente (assinatura do aluno): _____

(para uso da coordenação)

A Coordenação do PPGCA/UTFPR declara que foram cumpridos todos os requisitos exigidos pelo programa para a obtenção do título de Mestre.

Curitiba PR, _____/_____/_____ _____

"A Ata de Defesa original está arquivada na Secretaria do PPGCA".

À minha esposa Jéssica.

Resumo

O presente trabalho tem por objetivo realizar um estudo sobre a arquitetura Baars-Franklin, a qual define um modelo de consciência artificial, e aplicá-la a uma tarefa da navegação em um robô móvel autônomo. A inserção de robôs móveis em ambientes dinâmicos acarreta uma alta complexidade nas tarefas de navegação, visto que para lidar com a constante mudança no ambiente, é essencial que o robô seja capaz de se adaptar a este dinamismo. A abordagem utilizada neste trabalho é de realizar a execução destas tarefas de uma maneira mais próxima da utilizada por seres humanos quando reagem às mesmas condições, por meio de um modelo de consciência computacional. A arquitetura LIDA (*Learning Intelligent Distribution Agent*) é uma proposta de sistema cognitivo que procura modelar alguns dos aspectos cognitivos humanos, desde as percepções sensoriais (aspectos cognitivos de baixo nível) até o processo de tomada de decisão, os mecanismos de atenção e a memória episódica (considerados aspectos cognitivos de alto nível). Neste trabalho é utilizada uma implementação computacional da arquitetura LIDA, sendo apresentada a realização de um estudo de caso que visa avaliar a aplicabilidade e o desempenho de uma proposta de navegação cognitiva a robô móvel em ambientes dinâmicos e desconhecidos. São realizados experimento utilizando tanto um ambiente virtual (simulação) quanto um ambiente e robô reais. Este estudo concluiu que pode se obter benefícios na utilização de modelos cognitivos conscientes na tarefa de navegação de robôs móveis, apontando pontos positivos e negativos desta abordagem.

Palavras-chave: Robótica Móvel, Consciência Artificial, Modelo LIDA (Learning Intelligent Distribution Agent).

Abstract

This work presents a study about a the Baars-Franklin architecture, which defines a model of computational consciousness, and use it in a mobile robot navigation task. The insertion of mobile robots in dynamic environments carries a high complexity in navigation tasks, in order to deal with the constant environment changes, it is essential that the robot can adapt to this dynamism. The approach utilized in this work is to make the execution of these tasks closer to how human beings react to the same conditions by means of a model of computational consciousness. The LIDA architecture (Learning Intelligent Distribution Agent) is a cognitive system that seeks to model some of the human cognitive aspects, from low-level perceptions to decision making, as well as attention mechanism and episodic memory. In the present work, a computational implementation of the LIDA architecture was evaluated by means of a case study, aiming to evaluate the capabilities of a cognitive approach to navigation of a mobile robot in dynamic and unknown environments, using experiments both with virtual environments (simulation) and a real robot in a realistic environment. This study concluded that it is possible to obtain benefits by using conscious cognitive models in mobile robot navigation tasks, presenting the positive and negative aspects of this approach.

Keywords: Mobile Robotics, Artificial Consciousness, LIDA (Learning Intelligent Distribution Agent) model.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

GWT	Global Workspace Theory (Teoria do Workspace Global)
ABF	Arquitetura Baars-Franklin
PAM	Perceptual Associative Memory (Memória Associativa Perceptual)
SDM	Sparse Distributed Memory (Memória Esparsamente Distribuída)
LIDA	Learning Intelligent Distribution Agent
SLAM	Simultaneous Localization and Mapping(Mapeamento e Localização Simultâneos)
FIC	Fluent Interactive Codelets Framework
RTC	Real Time Controller
RTA	Robot Task Adviser
PPGCA	Programa de Pós-Graduação em Computação Aplicada
UTFPR	Universidade Tecnológica Federal do Paraná
ROS	Robot Operating System
IC	Iniciação Científica
IDA	Intelligent Distribution Agent
CAV	Veículo Autônomo Consciente (Conscious Autonomous Vehicle)
CSM	Current Situational Model (Modelo Situacional Corrente)

Capítulo 1

Introdução

Nas últimas décadas houve um avanço tecnológico tanto na robótica industrial quanto na robótica de serviços [Bennewitz, 2004]. No campo de serviços, foi desenvolvida uma grande gama de robôs projetados para operar em ambientes onde circulam seres humanos. Existem robôs que podem ser utilizados em escritórios, hospitais, museus entre diversos outros ambientes com circulação de pessoas [Burgard et al., 1998] [Takahashi et al., 2009]. Estes robôs realizam tarefas de entregas, limpeza, entretenimento e educação. Existe também o desenvolvimento da robótica voltada para acessibilidade, como cadeiras de rodas autônomas [Faria et al., 2014]. Outra área de aplicação que tem se desenvolvido nos últimos anos é o transporte de passageiros por carros autônomos [Belbachir et al., 2013].

Nestas condições estes robôs necessitam realizar suas tarefas em ambientes dinâmicos, com a presença de objetos, animais e pessoas em movimento. Em tais ambientes, o problema da navegação aumenta consideravelmente em comparação com robôs que efetuam suas tarefas em ambientes mais controlados, como é o caso da indústria. Por se tratar de um problema complexo, diversas técnicas são utilizadas na tentativa de solucionar este problema. Dentre elas estão a aquisição e o processamento de dados sensoriais, tomada de decisão, planejamento de trajetória e controle de movimento.

A navegação cognitiva é uma área que relaciona os problemas da navegação com a inteligência artificial, combinando a robótica móvel com o processamento cognitivo, realizado a partir de percepções de alto nível obtidas do ambiente. Nos modelos de navegação cognitiva, atualmente, utilizam-se controladores com duas camadas, onde a camada superior é responsável por identificar os aspectos de alto nível e enviar planejamentos para a camada inferior, que os executam, como proposto em [Arkin and Balch, 1997].

A camada de baixo nível pode ser denominada *navegação numérica*, e compreende todos os atributos numéricos e geométricos que o robô deve processar para realizar sua autolocalização e navegação básica. Uma estimativa precisa da localização do robô em um mapa consistente do ambiente é essencial para a realização desta navegação. Estes objetivos podem ser alcançados utilizando métodos de Mapeamento e Localização Simultâneos (SLAM-*Simultaneous Localization and Mapping*) [Durrant-Whyte and Bailey, 2006]. Estes métodos consistem em realizar estas duas tarefas simultaneamente, visto que, para se localizar em um ambiente o robô deve ter um mapa consistente, e para obter este mapa, é necessária uma estimativa apurada de sua localização. Os métodos atuais para a realização destas tarefas baseiam-se em distribuições probabilísticas da localização do robô dado um conjunto de medições realizadas pelos seus sensores.

Na camada de alto nível deve ocorrer a interpretação semântica do ambiente, responsável por identificar o modelo situacional corrente e realizar o planejamento das ações que devem ser executadas pelo robô, coordenando a camada de baixo nível na realização destas tarefas. Entre as tarefas realizadas pela camada de alto nível estão: o reconhecimento de obstáculos já vistos anteriormente, o reconhecimento de novos obstáculos, a identificação de ambientes perigosos e a definição de novas ações a serem realizadas. O modelo cognitivo baseado na Teoria do Workspace Global (GWS - *Global Workspace Theory*) [Baars, 1988] e na arquitetura LIDA (*Learning Intelligent Distribution Agent*) [Franklin et al., 2012], que são modelos de *consciência* artificial, permite a implementação da camada de alto nível da navegação cognitiva.

A *consciência* artificial consiste em uma área de estudo da inteligência artificial que busca aplicar os aspectos funcionais da consciência humana em sistemas computacionais. Conforme é discutido na seção 1.1.2, a consciência pode ser apresentada como um mecanismo que permite aos seres humanos lidar com situações complexas e diversificadas. Desta forma, aplicando os modelos de consciência em um ambiente computacional, pode-se obter um agente que possua maior flexibilidade e adaptabilidade ao ambiente. A Teoria do Workspace Global utiliza algumas características conhecidas da consciência, como atenção e memória episódica, descartando as características subjetivas e controversas do que é compreendido como consciência.

1.1 Contextualização

Nesta seção são apresentados brevemente os conceitos necessários para o entendimento deste trabalho. A subseção 1.1.1 apresenta os princípios, dificuldades e as principais técnicas para a realização da navegação autônoma de robôs móveis. A subseção 1.1.2 apresenta os conceitos de consciência e seus aspectos gerais. A subseção 1.1.3 apresenta propostas de modelos existentes para funcionamento da consciência. A subseção 1.1.4 apresenta propostas de aplicações de modelos de consciência artificial em ambientes computacionais. A seção 1.2 apresenta os objetivos gerais e específicos deste projeto de pesquisa.

1.1.1 Navegação

Para que seja possível que sistemas robóticos autônomos consigam operar em ambientes dinâmicos, seus sistemas de controle e sensoriamento devem enfrentar os desafios colocados pela incerteza de suas observações do ambiente e pelas diferentes condições de realização da tarefa. Um dos problemas encontrados nestas situações é que normalmente não é possível prever com precisão o resultado de cada ação realizada. Isto implica que não é possível idealizar *a priori* sequências de ações que irão resolver um problema sob todas as circunstâncias [Huber, 2000]. Uma forma de alcançar a realização das tarefas nestes ambientes é a utilização de um sistema de controle reativo que consiga reagir a eventos inesperados baseando-se exclusivamente em seu sensoriamento. Esta abordagem pode ser robusta ao ponto de que seja possível se recuperar de forma autônoma dos distúrbios ocorridos, sem a necessidade de replanejar toda a tarefa. Desta forma, esta abordagem de baixo nível pode ter uma resposta para situações inesperadas, o que permite que o robô realize sua tarefa apesar da dinamicidade do ambiente.

Para tratar as mudanças mais significativas no ambiente ou alterações nos requisitos necessários para completar a tarefa, é necessário que o robô seja capaz de alterar suas políticas de controle e planejamento para se adaptar às novas condições [Huber, 2000]. Em geral para alcançar esta flexibilidade, é necessária uma arquitetura de controle adaptativa que, dependendo

das condições de operação, possa utilizar mecanismos de aprendizado de máquina, como redes neurais e aprendizado por reforço [Burch et al.,], a fim de obter adaptabilidade ao ambiente.

Nas seções a seguir são apresentados detalhes sobre cada uma das tarefas necessárias para a navegação: localização, seleção de ação e planejamento de trajetória.

Localização

Dentre os problemas encontrados para a realização da navegação, um dos pontos chave é a localização. Sem uma estimativa precisa do posicionamento do robô dentro do ambiente, é impossível realizar a navegação. Para se localizar em um ambiente, uma forma é o robô ter um mapa consistente e, para obter um mapa consistente ele precisa de uma estimativa apurada de sua localização, logo um paradoxo se apresenta. A solução mais plausível para este problema está em realizar a localização e o mapeamento simultaneamente, visto que um é necessário para a realização do outro. Este é um problema clássico da robótica, denominado Mapeamento e Localização Simultâneos (SLAM). Para solucionar este problema, deve-se perguntar se é possível, para um robô móvel posicionado em uma localização desconhecida de um ambiente ainda não mapeado, construir um mapa consistente do ambiente, enquanto determina sua localização dentro deste mapa. O SLAM foi formulado e resolvido como um problema teórico de diferentes formas e implementado em diferentes domínios [Lee et al., 2013] [Kim et al., 2007], em robôs de ambiente interno, externo, sub-aquáticos e aéreos. Do ponto de vista teórico e conceitual, existem soluções para o problema SLAM que podem ser utilizadas diretamente na navegação de robôs [Durrant-Whyte and Bailey, 2006].

Seleção de Ação

A seleção de ação é o processo onde um agente determina em um certo instante, qual será a próxima ação a ser realizada [Bryson, 2007]. Esta é uma área que é extensamente pesquisada nas áreas como agentes de software e robótica [Jaafar et al., 2007]. Existem dois principais componentes para o problema: primeiro, decidir o que é uma ação válida (quais são as opções válidas para seleção), e segundo, determinar qual ação realizar em determinado momento.

No processo de decisão de qual ação deve ser tomada, diferentes objetivos devem ser considerados simultaneamente, sujeitos às restrições ditadas pelas limitações do agente. Um grande problema no projeto de sistemas para controlar um agente autônomo é a formulação de um mecanismo que combina múltiplos objetivos em estratégias para um comportamento racional e coerente [Jaafar et al., 2007].

Mecanismos de seleção de ação podem ser classificados em dois grupos: arbitragem e fusão de comandos. O primeiro permite que um comportamento ou um conjunto de comportamentos tomem o controle por um período de tempo, ou até que outro comportamento seja ativado. O segundo permite que múltiplos comportamentos contribuam para o controle do agente combinando recomendações de diferentes comportamentos para criar uma ação de controle que represente um consenso entre eles [Jaafar et al., 2007].

Planejamento de Trajetória

O planejamento de trajetória permite a identificação e a seleção do caminho por onde o robô possa atravessar, sem que haja uma colisão [Sariff et al., 2006]. Um planejamento de

trajetória preciso permite que robôs móveis possam seguir um caminho ótimo e livre de colisões a partir de uma posição original até a posição de destino, sem colidir com nenhum obstáculo no ambiente. O planejador de trajetória deve ser capaz de tratar as incertezas no modelo do ambiente, para minimizar o impacto de objetos com o robô e para encontrar o caminho ótimo no menor tempo possível, principalmente se a trajetória deve ser atualizada regularmente [Sariff et al., 2006].

Diversos algoritmos foram desenvolvidos para criar um sistema de planejamento de trajetória em tempo real para robôs autônomos. Estes algoritmos normalmente realizam buscas em estruturas de dados que contemplam uma representação do ambiente externo (como estruturas de representação métricas, topológicas ou mesmo híbridas [Thrun et al., 1998]), a partir dos dados fornecidos pelo mapeamento e a localização. Dentre estes algoritmos estão: busca em profundidade, A*, algoritmos genéticos e algoritmo de campos potenciais [Shwail et al., 2013] [Vaščák, 2007].

1.1.2 Consciência

Não existe uma definição globalmente aceita do que é o fenômeno da consciência humana. [Goldsmiths, 2009] afirma que esta incerteza sobre como definir o que é a consciência está ligada a forma em que a consciência é concebida nas diversas teorias sobre a natureza da própria existência. Na tradição clássica ocidental, o dualismo, como descrito por Platão e posteriormente por Descartes, é uma proposta que divide a natureza entre duas existências fundamentais, a matéria e a substância associada a consciência e ao espírito. Segundo [Goldsmiths, 2009] as teorias do século XX se tornaram puramente materialistas devido aos avanços das neurociências e outras pesquisas relacionadas, onde este materialismo apresenta diferentes formas de se definir o que é a consciência. O dualismo define que a consciência é uma propriedade não-física, mas que surge a partir dos sistemas físicos cerebrais, uma vez que estes atingem um certo nível de complexidade, como é o caso de [Sperry, 1969]. Em contrapartida, os reducionistas afirmam que a consciência não é nada mais que um estado ou função cerebral, como é o caso de [Crick, 1994] e [Dennet, 1991].

O termo "consciência" refere-se a experiência em si [Goldsmiths, 2009], isto é, procura-se identificar se uma entidade possui experiências sobre algo, caso sim ela está consciente, do contrário ela não está consciente. Visto que este trabalho tem como foco de seu estudo a aplicação de um modelo consciente para realização de tarefas de navegação de robôs móveis, serão apresentados alguns modelos teóricos sobre consciência para em seguida apontar um modelo adequado para a implementação da abordagem deste trabalho.

1.1.3 Modelos de Consciência

Teoria dos rascunhos múltiplos

Daniel Dennett, filósofo que escreveu vários livros a respeito da consciência, combate o dualismo cartesiano rejeitando a característica imaterial da mente [Dennet, 1991]. O modelo de Dennet propõe que o cérebro seja um hardware altamente paralelizado, onde a consciência

é formada por um complexo de memes[1], formando uma espécie de máquina virtual sobre o hardware, como um computador cria uma máquina virtual para cada aplicativo que irá executar.

Dennet sugere na teoria dos rascunhos múltiplos que o cérebro possui diversas linhas de processamento paralelo [Dennet, 1991], e em determinado momento existem vários fragmentos em diferentes estados que podem vir a se unir em um evento para determinar o comportamento do organismo para a situação. Dennet afirma que a sensação que há uma sequência de eventos conscientes ocorre quando essas linhas de processamento são exploradas.

Modelo de Crick e Koch

Crick e Koch (2003) criaram um modelo baseado no funcionamento do sistema cortical (esse sistema engloba o córtex cerebral e algumas outras regiões a ele intimamente associadas). O córtex é uma rede neural altamente interconectada que possui muitas ações interneurais excitatórias e inibitórias, formando grupos transientes de neurônios em forma de coalizões que fornecem suporte mútuo entre si e competição contra outras coalizões. O suporte que os membros de uma coalizão fornecem aos outros ocorre pelo aumento de seu nível de ativação em conjunto, dado que em um momento a coalizão vencedora define sobre o que o ente está consciente naquele momento.

Os eventos iniciais em uma coalizão correspondem ao processamento preliminar e inconsciente. Quando esta coalizão vence com um nível de ativação maior que as outras coalizões, ela vai permanecer no poder por algum tempo e tentar influenciar ou controlar eventos futuros [Crick and Koch, 2003]. Coalizões podem variar de tamanho e caráter. Os autores citam o exemplo de uma coalizão produzida por uma imaginação visual (com os olhos fechados) que pode ser menor que uma coalizão formada por um estímulo visual vívido e constante do ambiente.

O trabalho de Crick e Koch tem foco no sistema visual, deixando de lado alguns aspectos da consciência como a emoção e a autoconsciência, mas também pode ser aplicado para outras modalidades sensoriais [Crick and Koch, 2003].

Teoria do Workspace Global

A teoria do *Workspace Global* proposta em [Baars, 1988] apresenta a consciência como um acesso global que auxilia o recrutamento e integração de funções independentes do cérebro. A proposta de Baars sugere que o sistema nervoso contém vários processadores especializados que executam suas funções de forma autônoma e inconsciente. Estes processadores são altamente especializados e eficientes na execução de suas tarefas, trabalhando em paralelo e criando um sistema de grande capacidade, como é o sistema nervoso central. Os processadores especializados podem cooperar entre si formando coalizões a fim de desempenhar tarefas, como no modelo de [Crick and Koch, 2003]. As coalizões podem ser compostas de processadores especializados e também de outras coalizões destes processadores.

Um processador pode levar o seu conteúdo para a consciência quando o seu nível de ativação é maior do que todos os outros processadores. Um processador aumenta o seu nível de ativação quando este traz informações novas. Quando um processador traz informações de rotina seu nível de ativação se mantém baixo. O nível de ativação de um processador colabora

[1]Um meme é forma de transmissão cultural, como uma unidade de informação que se multiplica de cérebro para cérebro ou entre locais onde a informação é armazenada [Dawkins, 1976].

com o nível de ativação da coalizão que este participa, aumentando a chance de ter acesso à consciência.

Baars em sua teoria propõe um sistema de contextos, que são coalizões de processadores inconscientes. Os contextos representam conhecimento prévio ou pressuposições que, quando encontram uma situação fora do comum, tendem a levá-la ao nível da consciência. O autor propõe quatro tipos de contexto: contexto de objetivo, contexto de percepção, contexto conceitual e contexto cultural.

O *workspace global* é a estrutura central da teoria de Baars, sendo neste ponto onde se realiza a experiência consciente. Este *workspace* funciona como uma memória de trabalho que intermedia a troca de informações e interações entre os processadores. De forma antagônica aos processadores especializados, o *workspace global* é uma estrutura serial com uma pequena capacidade de processamento. Devido a esta característica, apenas uma coalizão pode estar ativa no *workspace global* em cada momento. Desta forma, surge uma competição, onde apenas a coalizão com o maior nível de ativação ganha acesso ao *workspace global*, transmitindo seu conteúdo de forma consciente aos outros processadores.

Conteúdos conscientes são sempre guiados por *contextos inconscientes*. Um contexto é uma coalizão de processadores inconscientes que afetam diretamente a experiência consciente, evocando ou modelando mensagens globais ou conteúdos conscientes. Para explicar o funcionamento de sua teoria, [Baars, 1997] utiliza uma metáfora chamada *metáfora do teatro interativo*. No teatro de Baars encontra-se o palco, um holofote, os atores, o público e os bastidores. O holofote conduz a atenção do público aos atores e estes competem pelo brilho do holofote. O público é composto por processadores especializados em determinadas tarefas, e nos bastidores ficam os processadores auxiliares que influenciam o que está sob o holofote. No decorrer da peça deste teatro, ocorrem diversas atividades no palco, mas apenas o que está iluminado pelo holofote é consciente. Os eventos continuam a ocorrer no palco e os auxiliares (processadores inconscientes) influenciam os eventos que estão sob a luz do holofote. Apenas os atores mais importantes são iluminados, e estes transmitem sua mensagem para a plateia e para os bastidores. No teatro interativo de Baars, quando os espectadores se identificam com a mensagem que está sendo transmitida eles podem subir ao palco e realizar sua performance junto com os atores sob o holofote, ou na parte escura do palco.

Esta metáfora apresenta a coalizão mais importante, que são os atores que estão sob a luz do holofote, transmitindo sua mensagem aos outros processadores especializados, exprimindo a limitação do conteúdo consciente. Estas coalizões que estão no *workspace global* (recebendo a atenção dos processadores) estão realizando tarefas relacionadas a situações novas ou conflitantes, e os processadores inconscientes, ao encontrar uma mensagem relevante, executam sua função específica.

1.1.4 Modelos cognitivos computacionais baseados em consciência

Existem trabalhos que propõem a aplicação das características da consciência em ambientes computacionais. Esta área de estudo é chamada de consciência artificial. Uma proposta de sistema cognitivo que segue essa linha é o sistema cognitivo neural de Haikonen [Haikonen, 2000], onde propõe-se que a consciência é criada através da percepção. Assim como o conhecimento do ambiente é gerado a partir da percepção do ambiente, a autoconsciência é criada pela percepção do corpo e de seus processos. Os conteúdos conscientes e inconscientes operam da mesma forma, pois esta proposta afirma que um conteúdo se torna consciente quando

vários circuitos operam tendo a mesma entidade como foco. Ao invés de utilizar sistemas especializados em blocos explicitamente programados, o sistema de Haikonen consiste em uma rede neural recorrente, utilizando blocos generalizados objetivando estabelecer uma máquina cognitiva.

Outra proposta de arquitetura cognitiva relacionada a consciência artificial é o CLARION (*Connectionist Learning with Adaptive Rule Induction On-line*). Esta proposta aborda o problema da cognição artificial utilizando uma representação dualista do conhecimento: implícito e explícito. Cada subsistema que compõe o CLARION possui dois níveis. O nível superior codifica o conhecimento explícito e o inferior o conhecimento implícito. A arquitetura é composta por quatro principais subsistemas: subsistema centrado em ações (ACS - *Action-Centered Subsystem*), susbsistema não centrado em ações (NACS - *Non-Action-Centered Subsystem*), subsistema metacognitivo (MCS - Meta-Cognitive System) e subsistema motivacional (MS - *Motivational Subsystem*) [Hélie et al., 2008]. Para a arquitetura CLARION, a consciência está ligada à união entre o aprendizado implícito (inconsciente) e o aprendizado explícito (consciente) [Sun, 1996].

Baars e Franklin (2007) apresentaram uma arquitetura cognitiva altamente distribuída, formada por componentes chamados "codelets" que são implementados como pequenos processadores de propósito específico. Esta arquitetura integra a *Global Workspace Theory* (GWT) proposta por Bernard Baars. Esta arquitetura estabelece que para que um agente seja capaz de realizar tarefas do mundo real, ele deve ser capaz de sentir o ambiente e agir de uma forma significativa no ambiente, além de ser capaz de agir de acordo com uma agenda de objetivos própria. As agendas de um agente biológico são criadas a partir da sua evolução e modificadas durante o seu desenvolvimento. Em um agente autônomo artificial, o autor determina que as agendas são criadas pelo desenvolvedor, e a partir de quando o agente é solto no ambiente, sua agenda pertence a si, e não é mais influenciada pelo desenvolvedor.

Atualmente o modelo cognitivo de consciência de um cérebro biológico não é completamente conhecido, logo não é proposto que o modelo possua consciência fenomenal[2], mas propõe o uso de um mecanismo de consciência não-fenomenal, que pode realizar as funções conhecidas da consciência humana. Com base na *Global Workspace Theory*, o modelo propõe que os conteúdos conscientes invocam uma ativação cerebral generalizada; que conteúdos conscientes recrutam funções inconscientes da memória de trabalho; e que todo o aprendizado significativo é invocado por conteúdos conscientes, mas o processo de aprendizagem e seus resultados podem ser inconscientes.

A arquitetura Baars-Franklin utiliza, além da teoria do *Workspace Global*, diversas outras teorias cognitivas como a Teoria do Pandemônio [Jackson, 1987] [Negatu, 2006], a Arquitetura *Copycat* [Hofstader and Mitchell, 1994], e Rede de Comportamentos [Maes, 1989], entre outras. Um *framework* desenvolvido na linguagem Java pela equipe de Franklin, disponibiliza a implementação genérica da arquitetura para ser utilizada na construção de agentes cognitivos baseados na arquitetura Baars-Franklin. Tal *framework* é denominado "LIDA Framework".

A partir das arquiteturas que implementam computacionalmente as características de consciência, foi escolhida para este trabalho a utilização da arquitetura Baars-Franklin, visto que para a aplicação em um robô móvel, uma arquitetura com enfoque nos processos de atenção

[2] A consciência fenomenal ou P-consciência está ligada a experiência qualitativa de um fenômeno, por exemplo quando vemos, ouvimos ou sentimos dor. As propriedades P-conscientes incluem as características da experiência das sensações, sentimentos e percepções [Block, 2002].

(como é a teoria do *Workspace Global*) se mostra promissora, já que o sistema estará inserido em um ambiente com situações adversas que poderão requerer mudanças no contexto de atenção. Outro fator que foi considerado para a escolha desta arquitetura foi o *framework* LIDA, que já possui uma implementação em software das teorias utilizadas na arquitetura integradas aos processos cognitivos.

Com base nos conceitos até agora descritos, a seguir são apresentados os objetivos deste trabalho.

1.2 Objetivos

Este trabalho tem como objetivo propor e desenvolver um controlador em duas camadas para realizar a navegação de um robô móvel autônomo, sendo que na camada superior serão utilizados mecanismos de percepção, atenção, aprendizado e seleção de ação. Esta camada tem como objetivo realizar planejamento e tomadas de decisão de alto nível. Juntamente com esta camada, uma camada reativa de baixo nível realiza o mapeamento e localização simultâneos (SLAM) e o controle de movimentação do robô. A camada superior é implementada utilizando o *framework* LIDA, que possui os requisitos necessários para a realização computacional da Arquitetura Baars-Franklin. A camada inferior é implementada utilizando o ambiente de controle de robôs *Robot Operating System* (ROS)[3], que provê ferramentas para o controle de diversos tipos de robôs, incluindo robôs móveis.

1.2.1 Objetivos Específicos

- Definição de uma arquitetura para a navegação cognitiva em duas camadas utilizando o *framework* LIDA e o ambiente *ROS*;
- Modelagem do problema de navegação sob a perspectiva do modelo cognitivo proposto, aplicando conceitos de memória, atenção e seleção de ação;
- Realização de experimentos em um ambiente simulado para avaliar o funcionamento da arquitetura;
- Comparação do desempenho da navegação do robô móvel utilizando o modelo proposto (cognitivo em dois níveis) com a abordagem tradicional de navegação reativa integrada ao SLAM (disponível no ROS - denominada *navigation stack*);
- Realização de experimentos com um robô móvel real como prova de conceito do modelo proposto.

1.3 Organização

Este trabalho está organizado como se segue. Além deste capítulo introdutório, o capítulo 2 aborda mais profundamente os detalhes da Arquitetura Baars-Franklin e sua implementação, o *Intelligent Distribution Agent* (IDA), apresentando os detalhes de seu funcionamento e

[3] `http://www.ros.org`

dos módulos que compõem a arquitetura. No capítulo 3 é apresentada a evolução do IDA, a arquitetura LIDA, apresentando os componentes de aprendizado que a diferenciam da arquitetura original, bem como os experimentos iniciais de navegação baseada em consciência realizados em ambientes de simulação. No capítulo 4 é apresentada a proposta detalhada desta dissertação, bem como a descrição dos experimentos com um robô real, modelo Pioneer P3-DX. No capítulo 5, são apresentadas as conclusões, discussões e trabalhos futuros.

Capítulo 2

Navegação Consciente e a Arquitetura Baars-Franklin de Consciência Computacional

2.1 Introdução

Neste capítulo são apresentadas inicialmente abordagens de navegação baseadas em modelos cognitivos, e posteriormente é detalhado o modelo de consciência utilizado no restante deste trabalho. A seção 2.2 discute os trabalhos desenvolvidos recentemente que visam a aplicar características de seres conscientes em aplicações para robôs reais e simulados. A seção 2.3 apresenta a arquitetura Baars-Franklin e o sistema *Intelligent Distribution Agent* (IDA), que consiste no modelo cognitivo utilizado como base para o desenvolvimento do controlador de alto nível proposto neste trabalho.

2.2 Aplicações de Modelos Cognitivos Conscientes à Robótica Móvel

Os dois principais paradigmas de controle para robôs móveis são o paradigma reativo e o paradigma deliberativo. Nas arquiteturas de controle reativas, o robô realiza suas ações a partir de informações sensoriais, utilizando pares pré-programados de percepção/ação, não utilizando-se de modelos internos do ambiente e planejamento com base em modelos. Um par de percepção/ação é chamado de comportamento reativo e é definido pela realização de um mapeamento de determinado estímulo sensorial à uma resposta. O desenvolvimento de diferentes comportamentos e a forma com que estes comportamentos são combinados fazem com que o robô realize as tarefas para quais ele foi desenvolvido. Dependendo da complexidade das tarefas e variedade das situações as quais o robô vai operar, o desenvolvimento de um sistema reativo pode ser algo trabalhoso [Junior, 2006]. O custo computacional para a execução dos comportamentos reativos é muito baixo, tornando o controlador reativo uma ótima solução para quando se necessita de respostas rápidas aos estímulos do ambiente. Rodney Brooks [Brooks, 1991], afirma que a inteligência se apresenta pelas relações de um sistema com o ambiente, de forma que não existe uma necessidade de núcleo de raciocínio em si, mas o resultado de todos os comportamentos de um sistema em relação ao ambiente é o que apresenta a inteligência. A

arquitetura de subsunção introduzida por Brooks [Brooks, 1985] foi um dos primeiros modelos de arquitetura reativa, e mais importantes para o desenvolvimento desta linha de pesquisa.

As arquiteturas deliberativas, em contrapartida, possuem dependência em um modelo interno do ambiente no qual o robô opera. A ações que o robô deve executar são determinadas com base neste modelo interno que realiza um planejamento de execução para a tarefa a ser realizada [Junior, 2006]. A capacidade de planejamento e de representação interna confere ao sistema maior flexibilidade, generalidade e adaptabilidade às diferentes situações que o robô possa encontrar. Um problema das arquiteturas deliberativas está no custo computacional necessário para sua execução. No caso de ambientes extremamente dinâmicos onde seriam necessários muitos replanejamentos em pequenos períodos de tempo, o custo computacional de uma arquitetura deliberativa pode tornar inviável sua utilização.

A utilização de arquiteturas híbridas, que combinam as duas abordagens, minimiza as restrições de cada abordagem usada de forma isolada. Nas arquiteturas híbridas, o planejamento das ações do robô é realizado a partir de suas representações internas de forma deliberativa, e a execução é realizada reativamente.

No restante deste capítulo, são apresentados os conceitos necessários para o entendimento da proposta deste trabalho, que consiste em desenvolver uma arquitetura híbrida em duas camadas, com um controlador deliberativo/cognitivo de alto nível, construído utilizando características relacionadas a consciência artificial, e um controlador reativo de baixo nível, construído utilizando o ambiente de robótica ROS.

O restante desta seção apresenta os trabalhos desenvolvidos recentemente que visam a aplicar características de seres conscientes em robôs reais e simulados, descrevendo como foram aplicados os conceitos de consciência artificial para a resolução de problemas relacionados a navegação.

2.2.1 Veículo Autônomo Consciente (CAV - *Conscious Autonomous Vehicle*)

Em [da Silva and Gudwin, 2009] é apresentada a aplicação da Teoria do Workspace Global em um Veículo Autônomo Consciente (CAV), para controlar uma criatura artificial em um ambiente virtual. A criatura virtual constrói um mapa do ambiente baseado em informação sensorial. Essa criatura possui duas motivações que são navegar de um ponto inicial a um ponto destino evitando colisões, e evitar que a carga de suas baterias se acabe, sendo possível carregá-las entrando em contato com objetos carregadores de bateria. O CAV implementa um ciclo cognitivo baseado no modelo *Intelligent Distribution Agent* (IDA) [Baars and Franklin, 2007], de forma simplificada, retirando os componentes que não são necessários para completar os objetivos do CAV.

O CAV possui uma arquitetura cognitiva com módulos de memória de trabalho, consciência e rede de comportamentos. Os módulos do CAV são implementados de forma distribuída no formato de codelets (processadores especializados da arquitetura de Baars-Franklin). Os codelets assim como em IDA, são simples e altamente especializados, possuindo características de autonomia, percepção, processo e ação, contando com seus próprios objetivos.

2.2.2 Neurobot

Em [Fountas et al., 2013] é apresentado um sistema baseado na arquitetura de Baars-Franklin para o controle de um avatar em um jogo digital, o qual busca simular o comportamento de um jogador humano. A arquitetura do Workspace Global foi implementada utilizando redes neurais pulsadas (*Spiking Neural Networks*), em uma abordagem inspirada biologicamente. O avatar acessa a informação sobre o jogo, não através da informação visual que um jogador real teria, mas sim de uma forma mais abstrata. Ele possui acesso direto a sua localização, direção, velocidade, local de itens visíveis e jogadores visíveis. Para utilizar estes dados que contém informações a respeito do jogo, é feita uma conversão em padrões de pulsos que alimentam a rede neural.

O avatar é controlado utilizando comandos para controle de movimento, rotação, tiro e salto. Para simplificar o controle, o avatar é tratado como um robô diferencial, onde as taxas de disparo de duas camadas de neurônios são utilizadas para definir a velocidade dos dois motores virtuais. O sistema possui outros conjuntos de camadas dedicados a responderem por cada tarefa de navegação e comportamento, como pode ser visto na Figura 2.1.

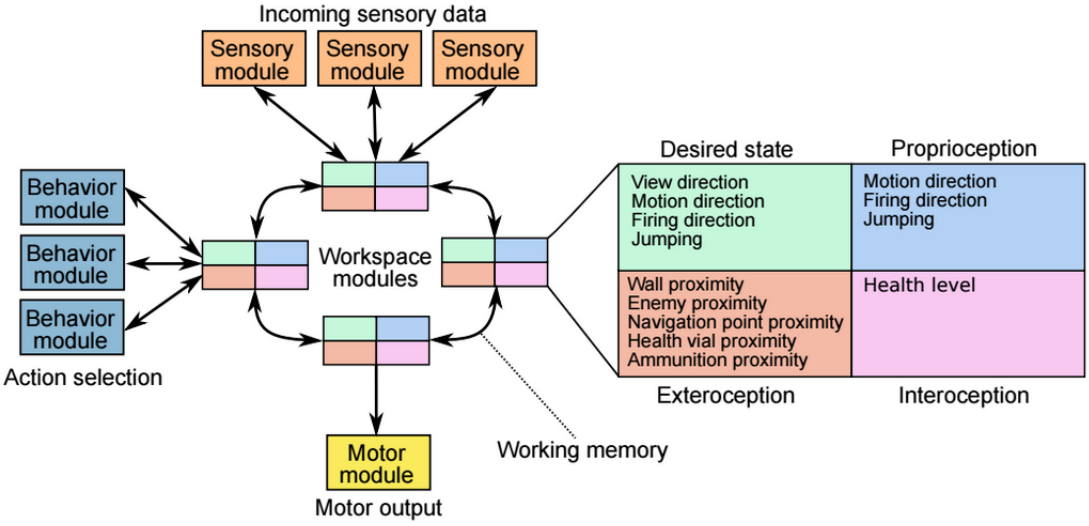


Figura 2.1: Arquitetura do Neurobot [Fountas et al., 2013]

O *workspace* global facilita a cooperação e competição entre os módulos de comportamento e garante que apenas um módulo controle a saída motora em qualquer momento. A arquitetura consiste em quatro camadas de neurônios conectados com conexões excitatórias e inibitórias. A informação em cada camada é idêntica e as quatro camadas são utilizadas como uma espécie de memória de trabalho que possibilita que a informação reverbere enquanto os módulos sensoriais e comportamentais competem entre si. As camadas do *workspace* são divididas em áreas representando os diferentes tipos de informações a serem difundidas.

2.2.3 *Fluent Interactive Codelets Framework*

Em [Daniela et al., 2011] é apresentado um protótipo de um robô móvel chamado FIC (*Fluent Interactive Codelets Framework*). FIC utiliza dois controladores para realizar o movimento da plataforma, sendo um deles chamado RTC (*Real Time Controller*) que é responsável por tomadas de decisões imediatas. O RTA (*Robot Task Adviser*) é o outro controlador, de alto-nível, responsável pelas estratégias de médio e longo prazo. O RTA utiliza uma arquitetura cognitiva chamada ALGOC que é um modelo baseado na arquitetura de Baars e Franklin.

Quando o robô é inicializado, os dados dos sensores são enviados ao RTC onde são processados e utilizados para atualizar o mapa inicial do ambiente. Parte desta informação também é enviada ao RTA para realizar a inicialização do sistema. Quando o robô envia um fluxo de dados ao RTC, ele responde com um comando após um pequeno espaço de tempo e envia dados ao controlador de alto nível. Quando o RTA detecta um obstáculo, ele determina que tipo de conceito corresponde a aquele obstáculo. Se não existe um conceito atual para o obstáculo, um novo conceito é inserido e retorna um meta-comando, denotando o obstáculo como um objeto desconhecido e associando-o com a ação previamente definida. Em caso de um conceito conhecido, a informação disponível e os conceitos relacionados são utilizados para selecionar o melhor comando a ser enviado. Utilizando meta-comandos e informação atualizada do ambiente, o RTC decide se deve alterar ou não o comportamento do robô.

2.3 Arquitetura Baars-Franklin

A Arquitetura Baars-Franklin (ABF) é um modelo cognitivo de consciência artificial baseado na Teoria do Workspace Global (GWT) de Bernard Baars [Baars, 1988]. A ABF realiza uma série de processos cognitivos, incluindo Percepção, Seleção de Ação, Aprendizado, Metacognição, entre outros. Para alcançar estes objetivos, a arquitetura utiliza diversas abordagens desenvolvidas anteriormente, como a arquitetura *copycat* de Hofstader [Hofstader and Mitchell, 1994], Rede de Comportamentos [Maes, 1989], Memória Esparsamente Distribuída [Kanerva, 1988] e a Teoria do Pandemônio [Jackson, 1987] [Negatu, 2006]. A arquitetura é composta por vários módulos, como pode ser visto na Fig. 2.2. A Arquitetura Baars-Franklin utiliza um ciclo cognitivo para realizar seus processos de forma que o sistema executa múltiplos ciclos de sentir o ambiente, decidir e agir. Assim o sistema itera continuamente sobre esse ciclo ativando seus módulos.

O projeto *IDA (Intelligent Distribution Agent)* é uma implementação baseada no modelo conceitual da Arquitetura Baars-Franklin, sendo o resultado de uma série de estudos realizados desde 1996 [Franklin et al., 1996] [Newman et al., 1997] [Franklin et al., 1998]. Este projeto foi desenvolvido com o objetivo de elaborar, de maneira completamente autônoma, a distribuição de trabalhos aos marinheiros da marinha americana. Ao final de cada jornada de trabalho de cada marinheiro, ele recebe um novo trabalho de forma que leve em consideração as preferências do marinheiro, as necessidades da marinha e uma série de regulamentações. A marinha emprega cerca de 300 pessoas em tempo integral para realizar estas atribuições. O IDA consegue automatizar completamente o papel de um humano para realização desta tarefa e com um desempenho comparável ao dos trabalhadores humanos. IDA é o sucessor do agente *CMattie* [Ramamurthy et al., 1998], que tinha o papel de coordenar as informações dos seminários do departamento de matemática da Universidade de Memphis. *LIDA - Learning Intelligent Distribution Agent* é o sucessor do IDA, sendo uma extensão de seu predecessor, adicionando

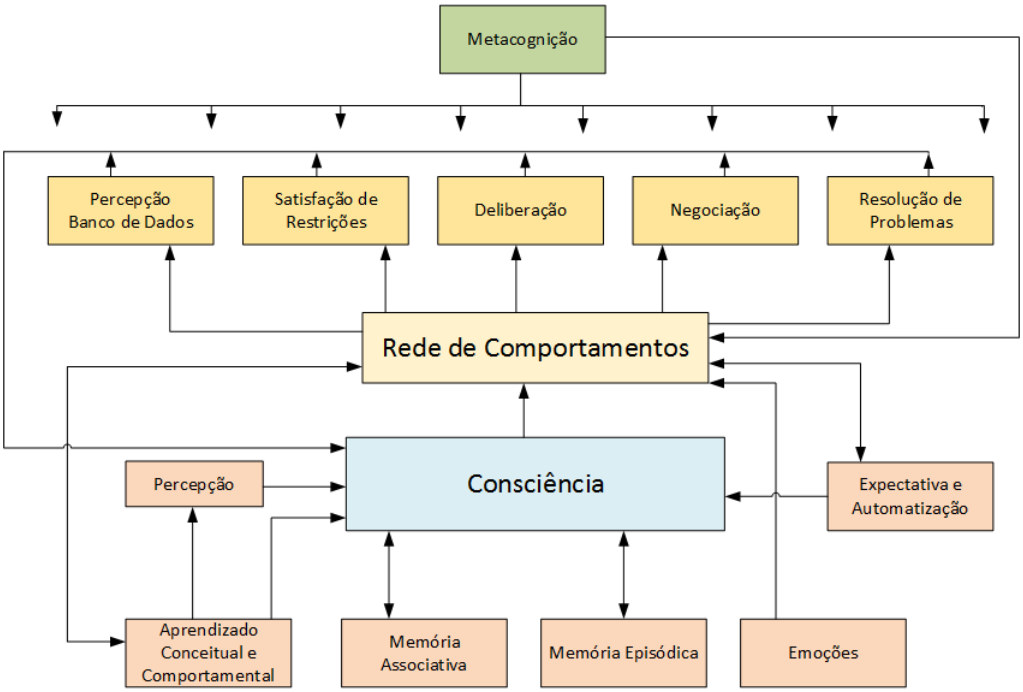


Figura 2.2: Arquitetura Baars-Franklin em IDA. (Adaptado de [Negatu, 2006])

três mecanismos fundamentais de aprendizado: aprendizado perceptual; aprendizado episódico; e aprendizado procedural [Ramamurthy et al., 2006]. As subseções seguintes apresentam uma visão geral da arquitetura Baars-Franklin, como definida em IDA, e seus componentes.

2.3.1 Codelets

O termo *codelet* é proveniente da nomenclatura utilizada por Hofstader na arquitetura *Copycat* [Hofstader and Mitchell, 1994]. Os *codelets* correspondem a processadores especializados em uma tarefa que é executada de forma independente. Na GWT os processadores especializados formam a base da cognição humana. Cada *codelet* é implementado como uma pequena porção de código sendo executada em uma *thread* independente [Negatu, 2006] Apesar de ser especializado na execução de uma tarefa simples, um *codelet* normalmente trabalha em conjunto com outros *codelets* formando coalizões com o propósito de construir comportamentos de alto nível [Capitanio, 2009]. Os *codelets* comumente atuam como *daemons* [Selfridge, 1959] aguardando uma condição apropriada para agir, como é apresentado em [Ramamurthy et al., 2006].

Os *codelets* possuem um nível de ativação, e ao notar uma situação que seja conveniente, um *codelet* irá aumentar seu nível de ativação em função da correspondência da situação e suas preferências. O nível de ativação de uma coalizão é aumentado quando o nível de ativação dos *codelets* pertencentes a esta aumentam [Ramamurthy et al., 2006]. Associações com alto nível de ativação também servem para criar uma ponte para interações de baixo nível entre os *codelets*, além de poderem ser vistas como uma rede de *codelets* simultaneamente ativos [Capitanio, 2009].

Os *codelets* podem ser divididos em categorias, como *codelets* de percepção, de atenção, de informação, de comportamento e de expectativa. Outros tipos de *codelets* podem ser criados dependendo das necessidades.

2.3.2 Percepção

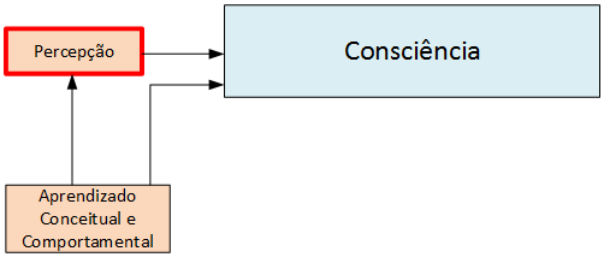


Figura 2.3: Módulo de Percepção

O módulo de percepção (Fig.2.3) da arquitetura Baars-Franklin, percebe exogenamente (percepções provenientes de eventos externos ao agente) e endogenamente (percepções relativas aos eventos internos do sistema cognitivo) utilizando como guia para tal o sistema de modelos perceptuais de Barsalou [Barsalou, 1999]. A base de conhecimento do agente utiliza um mecanismo chamado de Memória Associativa Perceptual (*PAM - Perceptual Associative Memory*), que combina mensagens recebidas com modelos pré-estabelecidos através de uma rede semântica chamada emphSlipnet (este mecanismo faz parte da arquitetura *Copycat* de Hofstader [Hofstader and Mitchell, 1994]).

A *Slipnet* é uma rede de conceitos correlacionados onde cada conceito é representado por um nó e cada relação entre dois conceitos é representada por um *link* contendo um valor numérico que representa a "distância conceitual" entre os nós envolvidos. Quanto menor a distância entre os conceitos dá-se um maior nível de ativação da relação [Hofstader and Mitchell, 1994]. Os conceitos representados em forma de nós atuam como os símbolos resultantes das percepções do agente, como indivíduos, categorias e até mesmo ideias de alto nível [Ramamurthy et al., 2006]. O nível de ativação de um nó reflete o grau de relevância para a situação corrente.

Estímulos sensoriais externos e internos são recebidos e interpretados pelo módulo de percepção de forma inconsciente onde um significado é associado. Na fase da percepção prévia, as entradas são captadas e capturadas por *codelets* de percepção especializados. Os *codelets* que encontram características relevantes a sua especialidade ativam os nós apropriados na *Slipnet*. Na fase seguinte, chamada *Chunk Perception*, a ativação passa de nó para nó na *Slipnet*, então a *Slipnet* se estabiliza, trazendo a convergência de vários fluxos de diferentes significados e agrupando partes de significados em partes maiores. Estas partes maiores na *Slipnet* formam um "Percepto" [Baars, 2003].

2.3.3 Memória Associativa

A memória associativa de longo prazo (Fig.2.4) é implementada na arquitetura Baars-Franklin utilizando uma memória esparsa distribuída (SDM - *Sparse Distributed Memory*). A SDM foi proposta por Pentti Kanerva como um modelo para a memória de longo prazo humana

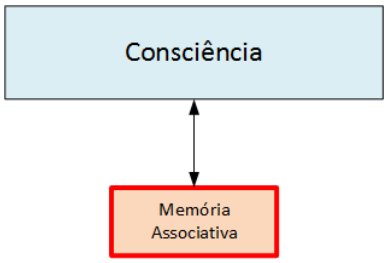


Figura 2.4: Memória Associativa

em [Kanerva, 1988]. A arquitetura de Kanerva permite o armazenamento de padrões e sua posterior recuperação, baseando-se em combinações parciais com entradas sensoriais correntes [Denning, 1989], ou seja, é uma memória de conteúdo endereçável de forma que parte do seu conteúdo é utilizada como entrada (ao invés de seu endereço de armazenamento) na recuperação de um item.

Devido ao fato da memória ser endereçada pelo próprio conteúdo, seu espaço de endereçamento deve ser grande o suficiente para acomodar todos os dados de uma entrada sensorial. No modelo de Kanerva, as entradas sensoriais são representadas por longos vetores de bits, contendo milhares ou dezenas de milhares de bits. Kanerva ilustra seu projeto com um exemplo de uma palavra de 1000 bits, o que leva a um espaço de endereçamento de $2^{1.000}$ possíveis padrões [Kanerva, 1988], este espaço de endereçamento requer uma quantidade de memória impraticável com a tecnologia atual, então é escolhido um número menor de endereços físicos que seja gerenciável na prática para serem realmente utilizados. Este conjunto de endereços constitui o subconjunto esparso do espaço de memória, sendo cada conjunto chamado de *hard location*. Uma *hard location* consiste em um número de contadores igual ao número de bits da palavra da memória, ou seja, para uma memória com palavra de 1.000 bits, existem 1.000 contadores de 8 bits. Além disto, existe um buffer de entrada e um buffer de saída, onde são armazenados o endereço de entrada e os dados provenientes da leitura, respectivamente.

Neste espaço, 1/1.000 dos padrões estão a uma distância de 451 bits de qualquer outro padrão e todos menos 1/1.000 dos padrões estão a uma distância de 549 bits [Kanerva, 1988]. Este grande número de padrões que estão tão perto (aproximadamente 49 bits) para uma distância média de 500 bits entre dois padrões aleatórios, são cruciais para a habilidade de fazer conexões entre padrões que aparentemente tem pouco a ver entre eles.

Cada *hard location* possui um decodificador de endereço que compara seu próprio endereço com o padrão de entrada para decidir se irá ser um participante da próxima operação de armazenamento ou recuperação, caso o endereço tiver uma distância (calculada pela distância de Hamming[1]) menor que d. Kanerva recomenda um d=*451* para uma alocação de memória de 1.000.000 de *hard locations* com padrões de 1.000 bits. Com estes parâmetros, aproximadamente 1/1.000 das *hard locations* serão selecionadas para qualquer endereço de entrada. Desta forma, vários locais são selecionados para a escrita ou leitura de um dado na memória, possuindo então um armazenamento distribuído. Devido a esta propriedade e o fato de possuir uma alocação esparsa de seu espaço de endereçamento, dá-se o nome de memória esparsa distribuída.

No processo de escrita de uma palavra na SDM, um padrão de entrada x é apresentado para todas as *hard locations*, que por sua vez verificam se participarão da operação (conforme

[1] O valor da distância de Hamming é a quantidade de bits que se diferem entre duas palavras.

descrito acima). O conjunto de endereços que participarão da operação é chamado de esfera selecionada por x. Em seguida o padrão de entrada é inserido em cada elemento da esfera de x. Como cada *hard location* na esfera selecionada possui um padrão diferente salvo em sua memória, não deve-se simplesmente sobrescrever o dado sobre a memória. O processo de escrita é realizado utilizando os contadores da *hard location*, quando um bit da palavra de entrada é 1 então o contador relativo à aquela posição tem seu valor incrementado, quando o bit é 0 então o valor do contador é decrementado.

Na leitura, uma entrada pode alcançar vários *hard locations* e o resultado da leitura é uma reconstrução estatística da palavra, determinada por todas as locações selecionadas. A esfera é selecionada da mesma forma que no processo de escrita e a palavra é construída somando-se os contadores de todas as *hard locations* selecionadas. Em seguida é realizado um processo de *threshold* onde os bits maiores que 0 recebem o valor 1, os bits menores que 0 recebem o valor 0, e os bits 0 recebem aleatoriamente 0 ou 1.

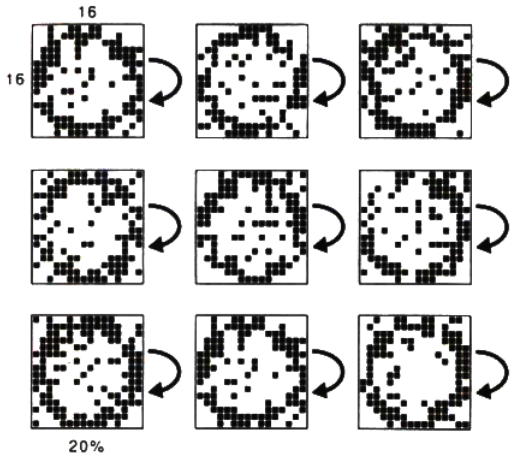


Figura 2.5: Exemplo de um processo de escrita em uma SDM [Denning, 1989].

A Fig.2.5 apresenta um exemplo de escrita em uma SDM, onde cada uma das nove imagens apresentadas foram escritas em uma SDM. Conforme explicado anteriormente, na SDM o dado é o próprio endereço de escrita, logo binarizando as imagens apresentadas na figura, obtém-se um vetor de bits onde cada bit representa um pixel da imagem, e este pode ser escrito na SDM no endereço especificado pelo próprio vetor. Nota-se que todas as imagens apresentadas na figura são diferentes, porém com um padrão similar, todas representam uma circunferência (com um ruído diferente em cada uma das imagens).

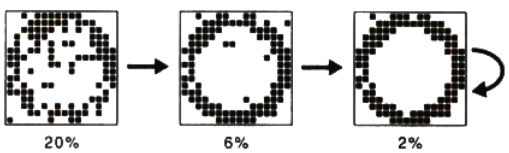


Figura 2.6: Exemplo de um processo de leitura iterativa em uma SDM [Denning, 1989].

A Fig.2.6 apresenta o processo de leitura de um padrão similar, onde a saída da memória é utilizada para uma nova pesquisa (pesquisa iterativa). A partir das associações realizadas

na memória pelas imagens escritas anteriormente, obteve-se uma figura distinta de todas as figuras que foram escritas, porém apresentando o mesmo padrão de similaridade entre elas, uma circunferência.

O armazenamento distribuído permite que a memória recupere um padrão armazenado quando o endereço de entrada corresponda apenas parcialmente com o padrão armazenado, permitindo assim a realização de associações com situações similares vistas anteriormente.

2.3.4 Memória Episódica Transiente

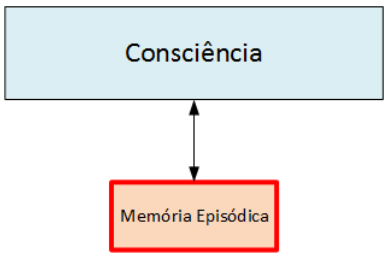


Figura 2.7: Memória Episódica Transiente

A memória episódica (Fig.2.7) é um sistema neuro-cognitivo que permite lembrar de experiências passadas [Tulving, 2002]. Esta é a memória responsável por armazenar eventos e suas características de local e tempo [Ramamurthy et al., 2004]. O aprendizado episódico extrai eventos e a história das experiências, enquanto o aprendizado semântico extrai fatos do contexto vivido [Tulving, 2002].

Seres humanos possuem uma memória episódica associativa, transiente, endereçável por conteúdo com uma taxa de decaimento da ordem de horas [Conway, 2001], que permite lembrarmos com grande detalhamento de eventos que ocorrem durante o dia. Estes detalhes de eventos ficam presentes apenas por uma curta duração de tempo (algumas horas) [Ramamurthy et al., 2004]. Por exemplo, quando lembramos o que comemos no almoço ou o que foi discutido com alguém há algumas horas. Temos grande riqueza de detalhes nessas memórias, porém esses detalhes são esquecidos com o passar do tempo.

A implementação da Memória Episódica Transiente no LIDA utiliza uma versão ligeiramente modificada da SDM, onde esta utiliza além dos bits 0's e 1's, um bit *don't care*. Esse bit possibilita que características desconhecidas possam ser representadas. A modificação da SDM para este contexto também envolve uma taxa de decaimento em horas [Ramamurthy et al., 2004]. No modelo cognitivo do IDA, a informação que não decaiu é consolidada na memória associativa de longo prazo em determinados intervalos.

2.3.5 Mecanismo de Consciência

O mecanismo para produzir a "consciência" é descrito em [Bogner, 1999] e consiste em quatro componentes: um gerenciador de coalizões, um controlador de foco de luz, um gerenciador de *broadcast* e um conjunto de *codelets* organizados como na Teoria do Pandemônio [Jackson, 1987] [Negatu, 2006]. Os componentes do módulo de "consciência" podem ser compreendidos como uma implementação de alguns aspectos do processo realizado pela metáfora do teatro [Baars, 1997].

Arena de Esportes

A Teoria do Pandemônio [Jackson, 1987] [Negatu, 2006] faz uma analogia com uma arena de esportes, composta por arquibancadas e um campo de jogo. *Codelets* inativos aguardam nas arquibancadas procurando uma condição relevante que possa acontecer na arena. Quando esta condição ocorre, o *codelet* é ativado e se junta ao campo de jogo [Negatu, 2006].

Codelets de Atenção

Codelets de atenção reconhecem situações novas ou problemáticas. A atenção, que pode ser automática ou voluntária, é o controle de acesso da consciência. A tarefa dos *codelets* de atenção é de levar a informação para a consciência. Os *codelets* de atenção se mantém observando um tipo particular de situação que possa ocorrer ou que precise requisitar uma intervenção da consciência. Ao encontrar tal situação, o *codelet* de atenção apropriado será associado ao pequeno grupo de *codelets* que carregam a informação, descrevendo assim a situação. Esta associação deve levar esse conjunto de *codelets* de informação, juntamente com o *codelet* de atenção que os coletou, formando assim uma coalizão. O *codelet* de atenção aumenta sua ativação em função de o quão bem ele é adequado para a situação atual para que então a coalizão formada possa competir pela consciência [Negatu, 2006].

Gerenciador de Coalizões

O gerenciador de coalizões é responsável por formar e monitorar coalizões de *codelets*. Um *codelet* é inicializado sem associações e em sua própria coalizão. *Codelets* ativos no campo de jogo constroem associações com outros *codelets*, seguindo a Teoria do Pandemônio. Associações crescem em co-atividade e resultam em coalizões de dois ou mais *codelets* [Capitanio, 2009].

Controlador de Foco de Luz

A relevância de uma coalizão é baseada na média do nível de ativação dos *codelets* que a compõem. O controlador de foco de luz realiza o cálculo da média de ativação das coalizões e escolhe a coalizão com a maior média para ir para a consciência e ser "iluminada pelo foco de luz", como na metáfora do teatro de Baars [Capitanio, 2009].

Gerenciador de *Broadcast*

O Gerenciador de *Broadcast* realiza a difusão do conteúdo da consciência para todos os codelets do sistema, reunindo o conteúdo de todos os *codelets* da coalizão escolhida pelo Gerenciador de Foco de Luz. Este conteúdo é então transmitido para todos os *codelets*, cada *codelet* então decide se essa informação é relevante para si, ou não [Negatu, 2006].

2.3.6 Seleção de Ação

O mecanismo de Seleção de Ação (Fig.2.8) da arquitetura Baars-Franklin utiliza a Rede de Comportamentos de [Maes, 1989] com aprimoramentos desenvolvidos em [Negatu and Franklin, 2002]. Este mecanismo, em conjunto com o mecanismo de consciência, são os principais elementos e formam o núcleo da arquitetura Baars-Franklin.

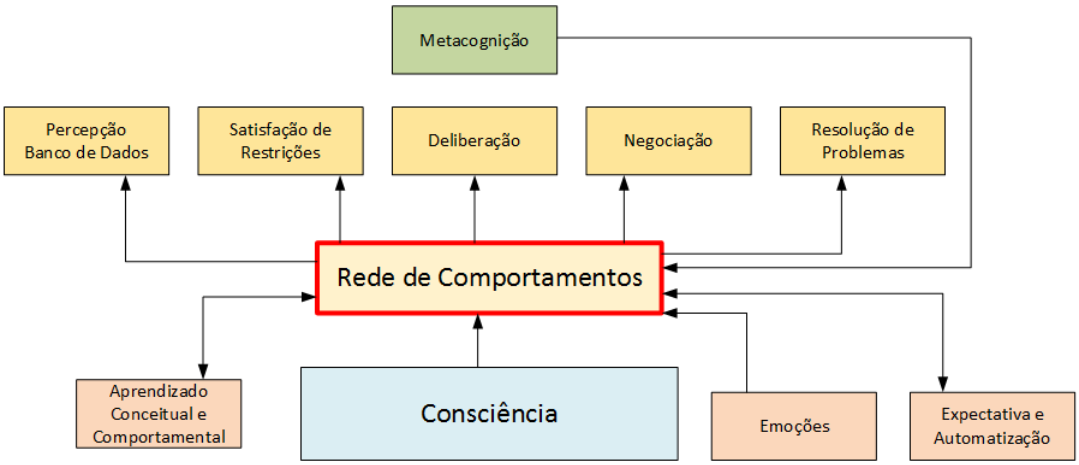


Figura 2.8: Mecanismo de Seleção de Ação

A arquitetura Baars-Franklin seleciona e executa as ações para atender a seus propósitos internos. Podem haver diversos propósitos operando em paralelo, que variam em urgência conforme o tempo e as mudanças no ambiente. Uma Rede de Comportamentos ativa é composta por estruturas chamadas de cadeias de comportamentos. Cadeias de comportamentos possuem nós de objetivo e nós de comportamento. Comportamentos e objetivos em uma rede de comportamentos são interconectados através de diversos *links*. Comportamentos são tipicamente ações de nível médio, podendo depender de diversos *codelets* de comportamento para sua execução. Objetivos são similares aos comportamentos, com a exceção de que sua ação é concluir objetivos sob condições específicas [Negatu, 2006]. Um comportamento na Arquitetura Baars-Franklin corresponde ao contexto de objetivo da Teoria do Workspace Global.

A Rede de Comportamentos atua em conjunto com o mecanismo de consciência para selecionar as ações [Negatu and Franklin, 2002]. Se um *codelet* de comportamento encontra informação relevante em um "broadcast consciente", uma cadeia de comportamentos é instanciada e esta cadeia se torna parte da Rede de Comportamentos ativa. Comportamentos cooperam e competem entre si na Rede de Comportamentos, a dinâmica em uma Rede de Comportamentos eventualmente seleciona o comportamento relevante para a ação. Um comportamento selecionado ativa seus *codelets* de comportamentos subjacentes, que por sua vez, entram no campo de jogo e executam suas ações apropriadas.

2.3.7 Satisfação de Restrições

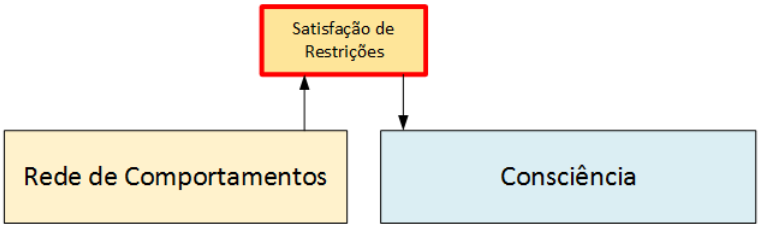


Figura 2.9: Módulo de Satisfação de Restrições

O problema da satisfação de restrições Fig.2.9 é definido por um conjunto de variáveis e um conjunto de restrições, onde cada variável possui um domínio não vazio de valores possíveis. Cada restrição envolve um subconjunto de variáveis e especifica as combinações aceitáveis de valores para o subconjunto. Uma associação que não viola nenhuma restrição é chamada de *consistente* ou associação legal. Uma associação completa é quando todas as variáveis são mencionadas, e a solução de um problema de satisfação de restrições é encontrada quando uma associação completa satisfaz todas as restrições [Russell and Norvig, 2009].

O módulo de Satisfação de Restrições do IDA foi projetado para prover uma medida numérica da aptidão de um trabalho em particular de um marinheiro [Kelemen et al., 2002]. Além das necessidades do marinheiro, IDA também deve levar em consideração as necessidades dos trabalhos individuais de aderirem às imposições da Marinha [Negatu and Franklin, 2002].

2.3.8 Deliberação

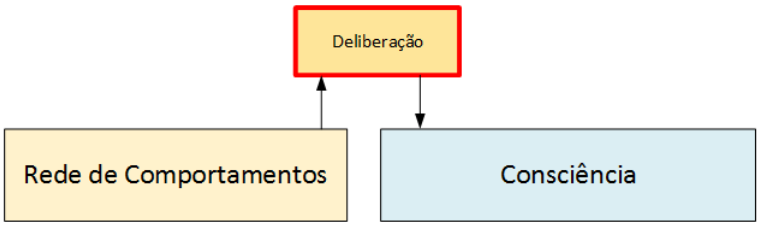


Figura 2.10: Módulo de Deliberação

O módulo de Deliberação Fig.2.10 da Arquitetura Baars-Franklin cria e avalia cenários para tomada de decisões [Franklin, 2000]. No processo de deliberação, os cenários são criados pela interação dos módulos de Rede de Comportamentos e da Consciência, bem como das ações dos *codelets* de atenção e comportamento que estão associadas ao eventos constituintes do cenário. Uma vez que um ou mais cenários aceitáveis existirem, uma decisão deve ser tomada. Esse processo é realizado utilizando uma implementação da *ideomotor theory* [James, 1890] [Franklin, 2000].

2.3.9 Negociação

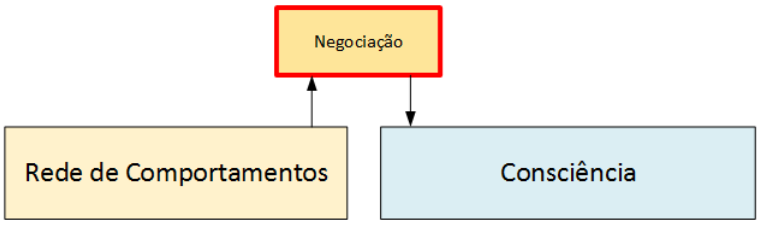


Figura 2.11: Módulo de Negociação

A Arquitetura Baars-Franklin utiliza um módulo de negociação Fig.2.11, que é utilizado pelo IDA após enviar uma mensagem de oferta para um marinheiro. Neste momento, o IDA deve lidar com a resposta do marinheiro, e é isso que o módulo de negociação faz [Franklin, 2001]. As respostas recebida podem ocasionar diversas situações que devem ser

tratadas de forma diferente, ocasionando a necessidade de tomadas de decisão. O módulo de negociação faz uso de outros módulos da arquitetura durante sua negociação. A negociação ocorre em múltiplos ciclos de interações do contexto de objetivo (seção 1.1.3) com mecanismo de consciência.

2.3.10 Emoções

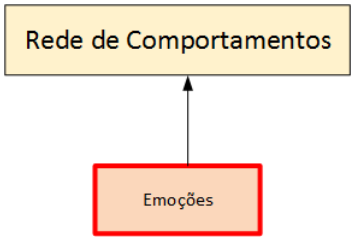


Figura 2.12: Módulo de Emoções

Uma emoção Fig.2.12 é normalmente causada por uma pessoa de forma consciente ou inconsciente, avaliando um evento como sendo relevante para um objetivo importante. A emoção é sentida como positiva quando há um avanço em relação a um objetivo e negativa quando algo entrava um objetivo. O centro de uma emoção é prontidão para agir. Uma emoção dá prioridade para alguns tipos de ações às quais ela dá uma sensação de urgência, assim ela pode interromper ou competir com outros processos [Oatley and Jenkins, 1996].

A importância das emoções para máquinas inteligentes é tão grande, que segundo [Minsky, 1985] a questão não é se máquinas inteligentes podem ter emoções, mas sim se máquinas podem ser inteligentes sem emoções.

No modelo conceitual da Arquitetura Baars-Franklin existe um mecanismo para emoções. Este mecanismo não reside em um módulo específico, mas existe afetando as ações de diversos *codelets*, principalmente os *codelets* de atenção. Ele também influencia na ativação interna do sistema de Seleção de Ação. Assim, as emoções afetam todas as ações do sistema [Negatu, 2006].

2.3.11 Metacognição

A Metacognição (Fig.2.13) é definida por [Flavell, 1976] como o conhecimento a respeito de seus próprios processos cognitivos, produtos, ou qualquer coisa relacionada a eles, como monitoramento, regulação e orquestração de processos. Nos seres humanos o monitoramento, a regulação e a orquestração podem ter a forma de checar, planejar, selecionar, autointerrogação, introspeção ou simplesmente o ato de realizar julgamentos sobre o que o ser sabe e o que não sabe a respeito de realizar uma tarefa. O projeto CMattie [Ramamurthy et al., 1998] implementou o módulo de metacognição, utilizando para tal controladores e classificadores fuzzy. O módulo de Metacognição é parte do modelo conceitual da Arquitetura Baars-Franklin, de forma que ele atue diretamente dentro do ciclo cognitivo da arquitetura, e não com um mecanismo a parte [Negatu, 2006].

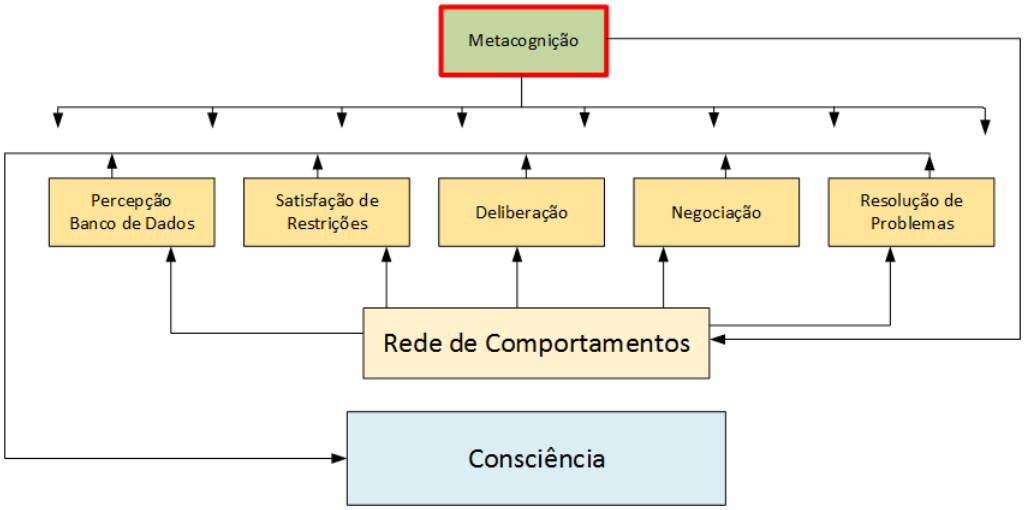


Figura 2.13: Módulo de Metacognição

2.3.12 Aprendizado

A Teoria do Workspace Global apoia a Hipótese do Aprendizado Consciente, onde o aprendizado significativo ocorre pela interação entre a consciência e os diversos sistemas de memória [Baars, 2003]. Ou seja, todos os sistemas de memória dependem da cognição consciente para sua atualização, tanto durante o curso de apenas um ciclo cognitivo, quanto durante o curso de diversos ciclos [Franklin et al., 2013]. De acordo com a Teoria do Worskpace Global, o fluxo de dados podem ser representado na forma de uma ampulheta, como pode ser visto na Fig. 2.14, onde os dados provenientes dos sensores entram pela esquerda, e fluem pelo cone esquerdo. O gargalo ao centro representa a capacidade limitada do Workspace Global, que atua como um filtro de relevância antes do *broadcast* consciente, que é representado pelo cone direito. A GWT é uma teoria sobre como as funções conscientes ocorrem cognitivamente, primeiro como um filtro e depois como um recrutador e modulador do aprendizado [Franklin et al., 2013].

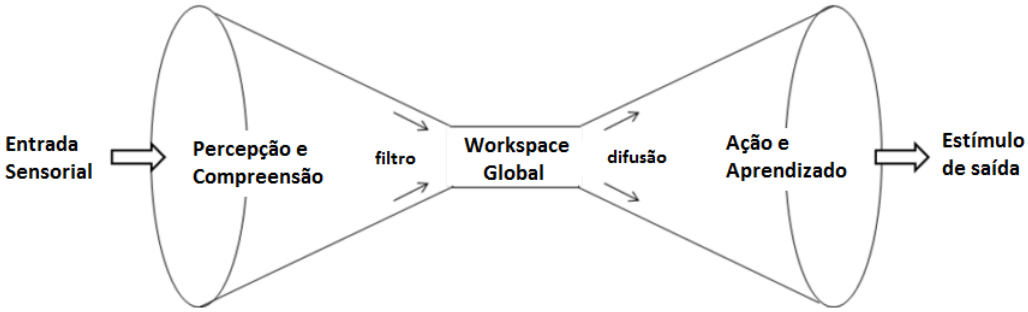


Figura 2.14: Ciclo de Ação-Percepção da Teoria do Workspace Global [Franklin et al., 2013]

2.4 Conclusão

Este capítulo efetuou um referencial bibliográfico apresentando uma síntese dos trabalhos correlatos ao tema deste trabalho, buscando apresentar os estudos mais recentes na área da consciência artificial aplicada a robótica. Um estudo foi apresentado abrangendo detalhes da Arquitetura Baars-Franklin com o sistema IDA. No próximo capítulo será apresentada a arquitetura sucessora do IDA, o LIDA (*Learning Intelligent Distribution Agent*) bem como sua implementação computacional, e a realização de alguns experimentos.

Capítulo 3

LIDA - Learning Intelligent Distribution Agent

LIDA - Learning Intelligent Distribution Agent é o sucessor do IDA apresentado no capítulo 2, sendo uma extensão que adiciona os mecanismos de aprendizado. Este capítulo descreve os mecanismos de aprendizado que diferem o modelo conceitual do LIDA de seu predecessor, bem como apresenta sua implementação computacional.

3.1 Aprendizado Perceptual

O Aprendizado Perceptual no modelo LIDA ocorre em duas formas: com o fortalecimento ou o enfraquecimento do nível de ativação dos nós existentes na PAM (*Perceptual Associative Memory*); com a criação de novos nós e links na PAM. Qualquer conceito ou relação que aparece no broadcast consciente (informação difundida pela coalizão vencedora) tem o nível de ativação de seu respectivo nó fortalecido em função da excitação do agente no momento do broadcast.

Um novo item que chega à consciência resulta em um novo nó criado. Juntamente a ele os *links* de suas características provenientes dos detectores de características, são também criados. Um novo item chega a consciência por meio de um *codelet* de atenção, quando este percebe uma coleção de novas características ativas no percepto, sem um objeto ao qual as características pertencem. Se este *codelet* de atenção obtém sucesso ao trazer o novo item para a consciência, um nó para ele é criado na PAM pelo mecanismo de aprendizado perceptual [Ramamurthy et al., 2006].

3.2 Aprendizado Episódico

O Aprendizado Episódico na arquitetura LIDA, ocorre a partir de eventos extraídos dos conteúdos da consciência e codificados para a SDM (*Sparse Distributed Memory*) que representa a Memória Episódica Transiente. Cada detector de característica na *slipnet* corresponde a um conjunto de dimensões dos vetores a serem armazenados na SDM. A dimensão do espaço da SDM é equivalente ao número de detectores de características primitivos. Os nós da *slipnet* que representam símbolos perceptuais que estão presentes em um evento consciente têm seus conteúdos rastreados pelos nós da *slipnet* até os detectores de características. Então um vetor

é formado e armazenado na Memória Episódica Transiente, assim realizando o aprendizado episódico em cada ciclo cognitivo. Além de codificar os conteúdos perceptuais, o aprendizado episódico inclui a codificação dos sentimentos, emoções e das ações realizadas pelo agente [Ramamurthy et al., 2006].

3.3 Aprendizado Procedural

3.3.1 Memória Procedural

A Memória Procedural da ABF é uma versão modificada e simplificada do mecanismo de Drescher [Drescher, 1991] chamado de *Scheme Net*. Assim como a *slipnet* da Memória Associativa Perceptual [Hofstader and Mitchell, 1994], a *scheme net* é um grafo direcionado, onde os nós são esquemas de ação e seus *links* representam a relação 'derivado de'. Um esquema consiste em uma ação juntamente com seu contexto e resultados. Nas bordas da *scheme net* é onde ficam os esquemas vazios (esquemas com uma ação primitiva), mas sem contexto ou resultados, enquanto esquemas mais complexos (que consistem em ações e sequências de ações) se localizam mais internamente. Para um esquema agir, inicialmente ele precisa ser instanciado e então selecionado para execução de acordo com o mecanismo de Seleção de Ação. *Codelet*s de comportamento, comportamentos e cadeias de comportamentos utilizam a mesma representação, consistindo em um esquema [Ramamurthy et al., 2006]. Cada esquema possui dois tipos de ativação, uma ativação de nível básico e uma ativação corrente. A ativação de nível básico mede a probabilidade de seu resultado ocorrer quando uma ação é tomada dentro de seu contexto. A ativação corrente mede a aplicabilidade do esquema para a situação atual.

3.3.2 Aprendizado

No Aprendizado Procedural, inicialmente a rede de comportamentos deve selecionar uma instância de um esquema vazio para execução. Como um esquema vazio não possui um contexto, assume-se que ele sempre será aplicável para a situação atual. Antes de executar a ação, o esquema instanciado (*codelet* de comportamento) invoca um novo *codelet* de expectativa. Após a ação ser executada, o *codelet* de expectativa aguarda mudanças no ambiente que sejam resultado da ação sendo executada e tenta levar esta informação para a consciência. Se o *codelet* de expectativa é bem sucedido em levar seu conteúdo para a consciência e um esquema apropriado já existe, este esquema é reforçado. Se o esquema não existe, um novo é criado. A informação consciente imediatamente antes da ação ser executada, se torna o conteúdo do novo esquema. O esquema inicia com um certo nível de ativação base e é conectado com o esquema vazio que o originou por um link. Coleções de *codelet*s de comportamento formam comportamentos [Ramamurthy et al., 2006].

3.4 Ciclo Cognitivo

A Arquitetura Baars-Franklin utiliza um ciclo cognitivo (Fig.3.1) para realizar seus processos, de forma que o sistema executa múltiplos ciclos de sentir o ambiente, decidir e agir. Assim o sistema processa continuamente sobre esse ciclo ativando seus módulos. A Fig.3.1

representa a interação entre os componentes da arquitetura durante seu ciclo cognitivo que será apresentado a seguir.

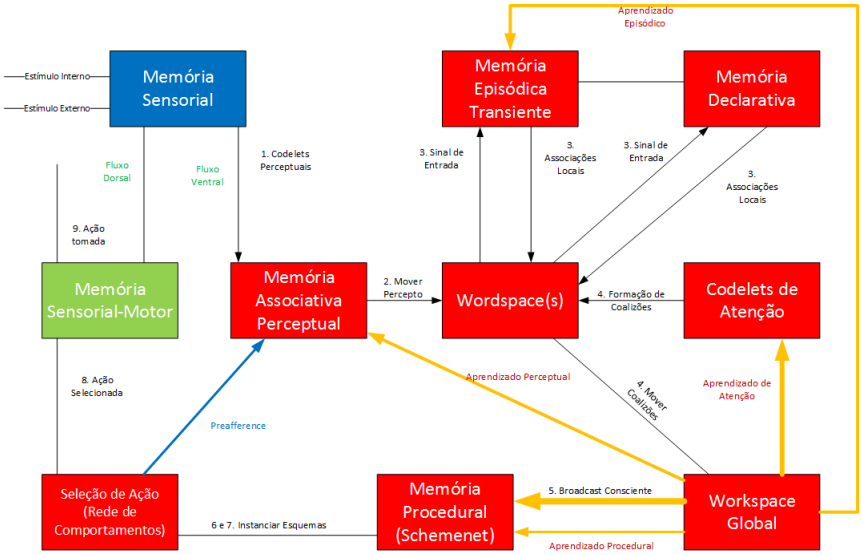


Figura 3.1: Ciclo cognitivo do LIDA - Adaptado de [Baars and Franklin, 2007]

O ciclo inicia com a ativação dos detectores de baixo nível, com as percepções de entrada na Memória Sensorial. A saída então é enviada aos detectores de caraterística de alto nível na Memória Associativa Perceptual, onde esta pode lidar com entidades mais abstratas, como objetos, eventos e ações. O resultado desta fase é um percepto que é enviado ao *workspace* onde são feitas associações locais através de informações provenientes da Memória Episódica e a Memória Declarativa. As associações locais combinadas com o perceto geram então o modelo situacional corrente, então o agente pode entender o que está acontecendo [Franklin et al., 2012] [Baars and Franklin, 2009].

A fase de consciência inicia quando os *codelets* de atenção formam coalizões, a partir de porções selecionadas do modelo situacional [Franklin, 2001][Baars and Franklin, 2009]. A competição pelo Workspace Global inicia selecionando a coalizão mais relevante, importante, saliente ou mais urgente para ir à consciência e realizar o broadcast consciente no sistema.

A próxima fase do ciclo cognitivo é a Seleção de Ação, onde diversos processos operam em paralelo. Novas entidades e associações são criadas, e as existentes são reforçadas quando o broadcast chega na Memória Associativa Perceptual. Os eventos do broadcast se tornam novas memórias na Memória Episódica. Os esquemas de ação, seus contextos e resultados são armazenados na Memória de Procedimento e esquemas antigos podem ser reforçados [Baars and Franklin, 2009].

Executando paralelamente com as funções da fase de Seleção de Ação, possíveis esquemas de ação são recrutados para a Memória de Procedimento. Uma cópia de cada um é instanciada com suas variáveis e enviados para a Seleção de Ação [Baars and Franklin, 2009], onde este compete para ser o comportamento selecionado para este ciclo cognitivo. O comportamento selecionado inicia sua execução, finalizando a execução do ciclo cognitivo.

3.5 Framework LIDA

O modelo conceitual da Arquitetura Baars-Franklin, conforme apresentado na seção 2.3, mostra a proposta de um modelo cognitivo que tenta cobrir uma grande porção de aprendizagem humana, combinando sistemas de seleção de ação, atenção, memória episódica, memória sensorial, memória perceptual e memória sensorial-motor. Este capítulo apresenta o LIDA Framework, que é a implementação computacional do modelo conceitual LIDA.

3.5.1 Estrutura

O Framework LIDA define diversas estruturas de alocação de dados e regras, sendo composto por diversos componentes. O principal componente de software são os módulos, que consistem em elementos interconectados que representam os módulos do modelo conceitual. Cada um dos módulos do modelo conceitual possui um correspondente no framework.

A maioria dos módulos do framework são independentes do domínio do problema. Para cada um destes módulos o framework possui uma implementação padrão. Porém alguns módulos são dependentes do domínio do problema e possuem apenas uma implementação básica, para que seja estendida com as funcionalidades específicas para o domínio. Como por exemplo o módulo do Ambiente (*Environment Module*), que é responsável pela interação do sistema cognitivo com ambiente.

Comunicação

Para que seja possível a comunicação entre os módulos, o Framework LIDA utiliza o padrão de projeto Observer [Gamma et al., 1995]. Um módulo ”listener” recebe informações de um módulo ”producer”. O ”listener” pode se registrar ao ”producer” para escutar o que o ”producer” tem a enviar. Cada vez que o ”producer” possui algo para enviar, ele transmite a informação para todos os seus ”listeners”. Existem diversas instâncias de ”listeners” no Framework LIDA, e um mesmo módulo pode estar registrado como ”listener” para diversos módulos. É possível também que um módulo seja um ”producer” e ”listener” de outros módulos ao mesmo tempo.

Tarefas

Visto que no modelo conceitual da Arquitetura Baars-Franklin todos os módulos e *codelets* executam tarefas em paralelo, o Framework LIDA implementa um sistema de gerenciamento de tarefas. As tarefas (Tasks) no LIDA encapsulam pequenos processos. Um módulo pode criar diversas tarefas para a realização de suas funções. Uma tarefa pode ser executada apenas uma vez ou executar suas funções repetidamente. O Gerenciador de Tarefas controla a execução de todas as tarefas e realiza a execução delas em threads separadas, a fim de alcançar a execução em paralelo de modo que fique transparente ao usuário.

Estruturas de Dados

Nós e *Links* são as principais estruturas de dados no Framework LIDA. Ambas possuem uma ativação que representa a medida da sua importância no contexto atual. Esta ativação pode ser excitada em diversas situações e decai ao longo do tempo. Um nó pode representar

características, objetos, conceitos, eventos, ações, etc. Cada nó é fundamentado em um nó na *PerceptualAssociativeMemory*. Os nós são instâncias dos nós da PAM (PamNode), e cada nó possui uma referência para o PamNode que o originou, bem como um *id* único que o identifica.

Um *Link* é responsável por conectar um nó a outro nó, ou a outro link. Um link que conecta dois nós é chamado de link simples. Por outro lado, um link complexo conecta um nó a outro link. Links possuem um atributo chamado *LinkCategory* que especifica a natureza do relacionamento representada pelo link, por exemplo um nó chamado ”bola” está conectado a outro nó chamado ”vermelho” através de um link com a *LinkCategory* ”característica”.

A *NodeStructure* é uma estrutura composta de nós e links que formam um grafo, e é a estrutura de dados padrão para comunicação entre a maioria dos módulos. A classe *NodeStructureImpl* possui métodos de adição, remoção e busca, além de realizar o gerenciamento dos nós e links presentes na estrutura.

Ativação

Nós, links e outros elementos como coalizões, *codelets*, *schemes* e *behaviors* possuem ativação. A ativação é representada por um número real entre 0.0 e 1.0. Em geral a ativação representa a relevância de um elemento. Os elementos citados possuem uma ativação atual, que mede a relevância no momento atual. Outros elementos possuem uma ativação adicional chamada *base-level activation*, que é utilizada para a implementação do aprendizado. O Framework utiliza duas interfaces para a implementação destas funcionalidades: *Activatible* e *Learnable*.

Um elemento que implementa a interface *Activatible* possui métodos de excitação e decaimento para regular a ativação. Um *Activatible* deve especificar duas estratégias para determinar como seu nível de ativação altera quando os métodos de excitar e decair são chamados. A interface *Learnable* possui adicionalmente a *base-level activation* para a implementação do aprendizado. A versão atual do Framework LIDA não possui a implementação de algoritmos de aprendizado. Para implementar o aprendizado atualmente, é possível desenvolver módulos customizados que implementam o método de aprendizado. Este método deveria modificar o *base-level activation* de elementos *Learnable*.

Interface Gráfica

O Framework LIDA possui uma interface com o usuário customizável, que permite a exibição em tempo real dos conteúdos dos módulos, valores dos parâmetros, tarefas em execução e outras variáveis de interesse durante a execução do LIDA. Um arquivo de propriedades permite a adição de painéis padrão ou customizados para configurar a aparência da interface gráfica.

3.5.2 Módulo de Ambiente

O módulo de ambiente (*Environment*) é responsável por realizar a integração do agente LIDA com o ambiente onde está inserido. Este módulo por ser dependente do problema, não possui uma implementação padrão, apenas uma classe base que deve ser customizada (através de herança) para a definição do problema, ou a integração com outros sistemas.

3.5.3 Memória Sensorial

A memória sensorial é um módulo do LIDA associado ao módulo *Environment* e é responsável por realizar a leitura dos sensores e disponibilizá-los para os demais módulos do sistema. Visto que os dados que serão adquiridos pelo sistema são um conteúdo dependente do problema, este módulo que requer uma implementação específica para o problema. Uma classe base para a customização deste módulo é definida pela classe *SensoryMemoryImpl*, é necessária a implementação de dois métodos para o funcionamento do módulo:

- *runSensors*: A cada passo da execução realiza a leitura dos sensores, utiliza a informação proveniente da classe Environment.
- *getSensoryContent*: Fornece acesso a outros módulos às informações da memória sensorial.

3.5.4 Memória Associativa Perceptual

A Memória Associativa Perceptual ou *Perceptual Associative Memory* (PAM) utiliza uma *NodeStructure*, para realizar a implementação da rede de conceitos (ver seção 2.3.2). Além da *NodeStructure* a PAM utiliza uma tabela de dispersão (*hashtable*) adicional que faz a relação entre os nós armazenados na NodeStructure e seus Labels e outra hashtable que relaciona os links e suas categorias.

A inicialização da PAM utiliza os seguintes parâmetros:

- *UPSCALE_FACTOR*: O dimensionamento da quantidade de ativação propagada de nós com baixa profundidade para nós com alta profundidade.
- *DOWNSCALE_FACTOR*: O dimensionamento da quantidade de ativação propagada de nós com maior profundidade para nós com menor profundidade.
- *PERCEPT_THRESHOLD*: A quantidade de ativação que um Nó ou um Link precisa ter para fazer parte do percepto (ser enviado ao Workspace).
- *EXCITATION_TICKS_PER_RUN*: O delay (em ticks) da excitação dos nós e links após receberem a ativação.
- *PROPAGATION_TICKS_PER_RUN*: O delay (em ticks) da propagação da ativação de um nó ou link.
- *PROPAGATE_ACTIVATION_THRESHOLD*: A quantidade de ativação necessária para ser propagada.

Excitação do nó

Para realizar a excitação da ativação de um nó na PAM é iniciada uma TASK pelo Gerenciador de Tarefas, que excita o nó com a quantidade indicada, se esta quantidade excede o threshold de percepção, uma nova TASK é criada com o intuito de levar o nó para o percepto. Em seguida é realizada a ativação dos nós ligados a este. Se a ativação for maior do que o *PROPAGATE_ACTIVATION_THRESHOLD* então a ativação a ser propagada é calculada utilizando a estratégia de propagação. A estratégia padrão é a UPSCALE onde o valor da ativação

é multiplicado pelo UPSCALE FACTOR e propagado em direção a seus filhos. A propagação é realizada iniciando uma nova TASK PropagationTask. Se a ativação for maior do que o threshold de percepção esta é adicionada ao percepto e sua ativação é propagada, realizando assim a propagação da ativação de forma recursiva.

Os *codelets* de Percepção (seção 3.5.5) são adicionados no sistema através da seguinte função da PAM:

- *addDetectionAlgorithm*: Ela valida o algoritmo de detecção (*codelet* de Percepção) para verificar se os nós utilizados pelo *codelet* estão na NodeStructure e então adiciona o *codelet* ao Gerenciador de Tarefas.

3.5.5 *Codelets* de Percepção

Os *codelets* de Percepção ou *codelets* de Detecção de Características (*Feature Detection Codelets*) são responsáveis por encontrar características relevantes a sua especialidade e ativar os nós correspondentes na *Perceptual Associative Memory*. Estes devem implementar a interface *DetectionAlgorithm*. Existem dois detectores padrão que podem ser estendidos para o domínio do problema, sendo um detector simples para excitar apenas um nó na PAM, chamado *BasicDetectionAlgorithm*. O outro chamado *MultipleDetectionAlgorithm* para quando for necessário que mais de um nó na PAM seja excitado. Ambos os detectores padrão possuem o método *detect* sendo um método abstrato, sendo necessário implementá-lo para o domínio específico do problema. Os *Codelets* de Percepção possuem como módulos associados a *PerceptualAssociativeMemory* e a *SensoryMemory*.

3.5.6 Workspace

O *Workspace* contém os buffers Perceptuais e Episódicos, bem como uma fila de broadcast e o Modelo Situacional Corrente (CSM - *Current Situational Model*). O acesso a todos esses buffers são realizados pela classe *Workspace* que funciona como um padrão de projeto ”facade” [Gamma et al., 1995].

Quando um nó da PAM excede o threshold de percepção uma *AddNodeToPerceptTask* é criada e esta invoca o método da PAM responsável por adicionar o nó ao Percepto. A PAM então realiza um broadcast com o percepto para quem estiver aguardando esta informação através de um *ReceivePerceptListener*, que é implementado pelo *Workspace*. O *Workspace* então copia o(s) nó(s) e o(s) link(s) ativados para o *PerceptualBuffer*. O *PerceptualBuffer* é uma instância do *WorkspaceBuffer*, que consiste em uma *NodeStructure* com alguns métodos de acesso.

A classe *CueBackgroundTask* é responsável por realizar a leitura do *PerceptualBuffer* e enviar o sinal de entrada para as memórias episódicas. Isto é realizado fazendo a leitura do Buffer e em seguida enviando um sinal para o *Workspace* por meio do método *cueEpisodicMemories* do *Workspace*. Este método realiza um broadcast para todos os ”listeners” registrados no workspace para a interface *CueListener*. A implementação atual da memória episódica só aceita Nós, portanto os links não são enviados.

Ao receber um broadcast do Workspace Global, o *Workspace* armazena o conteúdo do broadcast em uma fila, que possui um limite de ativação para os itens permanecerem na fila e um número limitado de espaços (por padrão o número de espaços é 20).

Quando o *Workspace* recebe uma associação local da memória episódica, os dados recebidos são escritos no *PerceptualBuffer*, através de uma operação de união da *nodeStructure*.

O Modelo Situacional Corrente ou *Current Situational Model* (CSM) também é implementado através de um *WorkspaceBuffer*. Seu estado é mantido pela TASK UpdateCsmBackgroundTask, e a cada iteração ela realiza a leitura do *PerceptualBuffer* e adiciona ao CSM. O modelo é construído baseado nas informações provenientes da PAM e das memórias episódicas.

3.5.7 Memória Episódica

A memória episódica se comunica com o *Workspace* de forma a receber os endereços e retornar associações locais. Ao receber um "broadcast consciente", ou seja, quando o ”listener” relacionado ao *Workspace Global* é ativado, a memória episódica armazena o conteúdo do broadcast.

O conteúdo a ser gravado proveniente do *Workspace Global* consiste em um conjunto de nós, os quais são ”traduzidos” para um vetor de bits. Esse processo de tradução consiste em utilizar o ID do nó como posição no vetor de bits. Desta forma o vetor de bits consegue representar toda a situação corrente em uma palavra, os nós que estão ativos são definidos como 1 e os inativos com 0.

Este resultado representa as associações realizadas pela memória episódica e é enviado a todos os ”listeners” que implementam a interface LocalAssociationListener. O Workspace implementa este ”listener” e, ao receber uma mensagem da memória episódica, ele junta as associações locais ao EpisodicBuffer, que posteriormente são enviadas ao Modelo Situacional Corrente. Através deste mecanismo é possível utilizar uma memória associativa endereçada por conteúdo, de forma que quando gravamos um modelo situacional na memória episódica ela irá realizar associações com experiências similares anteriores.

Os seguintes parâmetros são utilizados para a criação da memória episódica:

- *DEF_HARD_LOCATIONS*: O número de Hardlocations, por padrão é de 10000
- *DEF_ADDRESS_LENGTH*: O tamanho do endereço, por padrão é de 1000
- *DEF_WORD_LENGTH*: O tamanho da palavra armazenada, por padrão é 1000
- *DEF_ACCESS_RADIUS*: O raio de acesso de uma Hardlocation, por padrão é de 451

3.5.8 *Codelets* de Atenção

Os *codelets* de atenção são responsáveis por observar o modelo situacional corrente e identificar se o conteúdo desejado está presente, e se estiver ele cria uma coalizão, adicionando os nós do modelo e o próprio *codelet* a ela. Após a coalizão ser criada ela é adicionada ao *Workspace Global* para competir pela consciência.

O conjunto de nós que um *codelet* de atenção observa é chamado de *SoughtContent*. Ao criar a nova coalizão, os nós do *SoughtContent* presentes no CSM são copiados juntamente com os links, e os vizinhos que tem ativação maior do que o threshold.

O framework possui uma implementação padrão para *codelets* de atenção chamada *DefaultAttentioncodelet*, os seguintes parâmetros são necessários para a criação de um novo *codelet* de atenção:

- *NODES*: Quais nós estão no *SoughtContent* do *codelet*

- REFRACTORY PERIOD: O intervalo mínimo em *ticks* que o *codelet* deve esperar para criar uma nova coalizão

3.5.9 Workspace Global

O *Workspace Global* é a estrutura central do mecanismo de consciência, sendo ela responsável por receber as coalizões geradas pelos *codelets* de atenção e identificar qual coalizão deve realizar o broadcast, e então realizá-lo.

São necessários os seguintes parâmetros de inicialização:

- *COALITION_REMOVAL_THRESOLD*: A quantidade de ativação que uma coalizão deve ter para se manter no *Workspace Global*
- *COALITION_DECAY_STRATEGY*: O nome da estratégia de decaimento utilizada pelas coalizões no *Workspace Global*
- *REFRACTORY_PERIOD*: O tempo mínimo permitido entre broadcasts subsequentes (default 40)

Triggers

Os *Triggers* são classes responsáveis por identificar uma situação e disparar o broadcast do workspace global para os outros módulos. A seguir são apresentados os *Triggers* padrão do LIDA:

- *AggregateCoalitionActivationTrigger*: Dispara o broadcast quando a soma de todas as coalizões no *Workspace Global* excede um threshold.
- *IndividualCoalitionActivationTrigger*: Dispara o broadcast quando a ativação de uma coalizão no *Workspace Global* excede o threshold.
- *NoBroadcastOccurringTrigger*: Dispara quando não ocorreu nenhum broadcast durante um determinando número de ticks.
- *NoCoalitionArrivingTrigger*: Dispara quando nenhuma coalizão foi adicionada ao workspace global durante um determinado número de ticks.

Quando um *trigger* dispara, ele invoca o método *triggerBroadcast* do *Workspace Global*, que inicia o cálculo de qual é a coalizão com o maior nível de ativação (coalizão vencedora), em seguida envia o conteúdo da coalizão para todos os módulos que estão registrados como *broadcastListener* do *Workspace Global*. Após o broadcast todos os *triggers* são resetados. A cada passo no workspace global, é realizado o decaimento da ativação das coalizões através da estratégia de decaimento definida na inicialização da classe do *Workspace Global*.

O *Workspace Global* cria uma tarefa que controla a execução de todos os *Triggers* associados ao *Workspace Global*, chamada de *StartTriggersTask*.

3.5.10 Memória procedural

A memória procedural é implementa um grafo onde seus nós são esquemas de ação (*schemes*). Um *scheme* é uma estrutura composta por um contexto, uma ação e um resultado. O contexto possui os nós que precisam estar ativos para o esquema ser ativado. Se a ação é tomada, é esperado que os nós declarados no resultado se tornem ativos. Os nós presentes no contexto e no resultado são chamados de condições.

O módulo de memória de procedimento mantém uma *hashtable* contendo todos os *schemes*, e outra *hashtable* contendo todas as condições, tanto de contexto como de resultado. Duas outras tabelas também existem para relacionar as condições de contexto e resultado com seus respectivos *schemes*.

O *BroadcastBuffer* contém o conteúdo que é recebido dos broadcasts conscientes. Ao receber um broadcast, se um nó presente na lista de condições correntes da memória procedural já está no *BroadcastBuffer*, este tem sua ativação atualizada pelo conteúdo do broadcast, caso contrário o nó é adicionado ao buffer.

Após o broadcast consciente ser recebido e todos os nós serem atualizados ou adicionados ao buffer, verifica-se então quais são os *schemes* relevantes para esse broadcast (define-se como *scheme* relevante ao broadcast quando este possui em seu contexto nós presentes no broadcast). Para os schemes com maior ativação do que o *SCHEME_SELECTION_THRESHOLD* são instanciados *Behaviors*, que são instâncias dos *schemes* enviadas para o mecanismo de seleção de ação. Quando um *Behavior* é instanciado ele é enviado a todos os "listeners" registrados na interface *ProceduralMemoryListener*.

A ativação de um scheme é calculada da seguinte forma:

$$S = M * C_w + N * A_w$$

Sendo M a média de ativação dos nós do Contexto, C_w o peso das condições de contexto, N a média de ativação dos nós do resultado e A_w o peso das condições de resultado.

Os parâmetros para a criação da memória procedural são os seguintes:

- *SCHEME_SELECTION_THRESHOLD*: Determina a quantidade de ativação um *scheme* deve ter para ser instanciado.
- *CONTEXT_WEIGHT*: O peso das condições de contexto para realizar o cálculo da ativação de um *scheme*.
- *ADDING_LIST_WEIGHT*: O peso das condições de resultado para realizar o cálculo da ativação de um *scheme*.
- *DEFAULT_SCHEME_CLASS*: Nome da classe que implementa um *scheme*.
- *DECAY_STRATEGY*: Estratégia de decaimento utilizada pelas condições e pelo buffer de broadcast.

3.5.11 Seleção de Ação

O mecanismo de seleção de ação é realizado através de uma *Behavior Network*, baseada no modelo de Maes conforme descrito na seção 2.3.6. O módulo de seleção de ação é

registrado com um ”listener” do módulo da memória de procedural e recebe os *behaviors* que são instanciados no módulo da memória procedural.

A estrutura do grafo é implementada utilizando uma *hashtable* para realizar o mapeamento entre os comportamentos e as condições, e um outro é mantido para realizar o mapeamento entre os comportamentos e as condições de resultado.

Os seguintes parâmetros são utilizados para a inicialização do módulo de seleção de ação:

- *BROADCAST_EXCITATION_FACTOR*: A porcentagem de ativação que um broadcast envia aos *behaviors* cujo contexto ou o resultado possui intersecção com o elemento.
- *SUCCESSOR_EXCITATION_FACTOR*: A porcentagem da ativação que os comportamentos recebem de seus sucessores.
- *PREDECESSOR_EXCITATION_FACTOR*: A porcentagem de ativação que comportamentos recebem de seus predecessores.
- *CONFLICTOR_EXCITATION_FACTOR*: A porcentagem de ativação que comportamentos recebem de comportamentos em conflito.
- *CONTEXT_SATISFACTION_THRESHOLD*: A quantidade de ativação que uma condição de contexto deve possuir para ser satisfeita.
- *INITIAL_CANDIDATE_THRESHOLD*: O valor inicial do threshold de um candidado, o threshold de um candidato é resetado para o valor inicial a cada seleção de ação.
- *CANDIDATE_THRESHOLD_DECAY_NAME*: Nome da estratégia de Decay utilizada para o threshold do candidato.
- *BEHAVIOR_DECAY_NAME*: O nome da estratégia de Decay utilizado para decair os comportamentos.

Quando um comportamento é recebido através do broadcast da memória procedural, este é adicionado em uma *hashtable* responsável por manter os comportamentos recebidos e a indexação com as estruturas de dados citadas anteriormente é realizada.

A cada ciclo, a ativação dos comportamentos é calculada pelo produto da ativação do comportamento pelo *BROADCAST_EXCITATION_FACTOR*. Em seguida a ativação é propagada para os *behaviors* relacionados. Para cada *behavior* da rede, são verificadas se todas as condições daquele comportamento são satisfeitas (se seu nível de ativação é maior do que o *CONTEXT_SATISFACTION_THRESHOLD*), e caso isto ocorra a ativação é passada aos sucessores.

Em seguida é realizada a tentativa de selecionar uma ação. A seleção da ação ocorre escolhendo os comportamentos que possuem todas suas condições satisfeitas, em seguida são analisados os que possuem um nível de ativação maior que o *threshold*. Se mais de um comportamento tiver a ativação maior, o comportamento selecionado é escolhido aleatoriamente entre eles. O comportamento selecionado tem sua ação executada, e sua ativação é reinicializada com 0. Se nenhum comportamento for selecionado, o *threshold* do candidato é reduzido.

Quando um comportamento tem uma ativação abaixo do *REMOVAL_THRESHOLD* 0.0, ele é removido da rede, juntamente com suas ligações.

3.5.12 *Sensory Motor Memory*

A memória *Sensory Motor Memory*, escuta o módulo de seleção de ação pelo *ActionSelectionListener*, e chama o método *processAction* do módulo *Environment*, responsável por executar a ação no ambiente a cada ação selecionada.

3.6 Validação experimental - Labirinto

Foi desenvolvido um experimento para realizar uma avaliação inicial da capacidade de tomada de decisão de um robô móvel utilizando um modelo de consciência baseado no LIDA. O ambiente utilizado para o experimento foi um labirinto onde o robô pode se mover livremente. O labirinto possui um elemento objetivo, para onde o robô deve tentar ir. Foi definido um conceito de nível de bateria virtual, a qual decresce enquanto o robô navega pelo ambiente. Para a recarregar a bateria virtual o robô deve se locomover para pontos de recarga que foram distribuídos no labirinto. Ao se locomover em direção ao seu objetivo o robô deve decidir quando ele deve recarregar sua bateria e utilizando qual carregador, para que consiga chegar ao objetivo.

Após a realização deste experimento, um artigo entitulado "*Adding conscious aspects in virtual robot navigation through Baars-Franklin's cognitive architecture*", contendo o desenvolvimento e os resultados deste experimento foi publicado na conferência *IEEE International Conference on Autonomous Robot Systems and Competitions* (ICARSC), realizada em Portugal em 2015.

3.6.1 Definição do ambiente

O ambiente de simulação foi desenvolvido utilizando a plataforma de simulação *Virtual Robot Experimentation Platform* (V-REP)[1]. Foi desenvolvido um modelo do robô Turtlebot[2] para a realização da tarefa no ambiente V-REP além dos componentes de software necessários para a integração do robô virtual com o framework ROS, consistindo em um modelo cinemático do robô para o envio de comandos aos atuadores e um sistema de odometria. Os componentes implementados executam como dois nós no ROS. Sensores virtuais que identificam a posição dos carregadores no ambiente foram criados afim de simplificar o modelo, sendo assim o robô conhece as coordenadas cartesianas dos carregadores dentro no ambiente, o mesmo ocorrendo com o objetivo. A Fig.3.2 apresenta o ambiente virtual utilizado com o robô ao fundo, as esferas azuis representando os carregadores e a esfera vermelha representando o objetivo.

3.6.2 Sistema de controle "consciente"

Para a implementação do sistema de controle utilizando o *Framework LIDA* o módulo de memória sensorial foi implementado, visto que ele é um módulo dependente do problema e o *framework* não possui uma implementação padrão. A partir da classe base do lida *SensoryMemoryImpl*, a classe *ROSSensoryMemory* foi herdada, implementando dois principais métodos. O método *runSensors*, é chamado todas as vezes que o LIDA realiza a leitura de sensores, para

[1] `http://www.coppeliarobotics.com`
[2] `http://www.turtlebot.com`

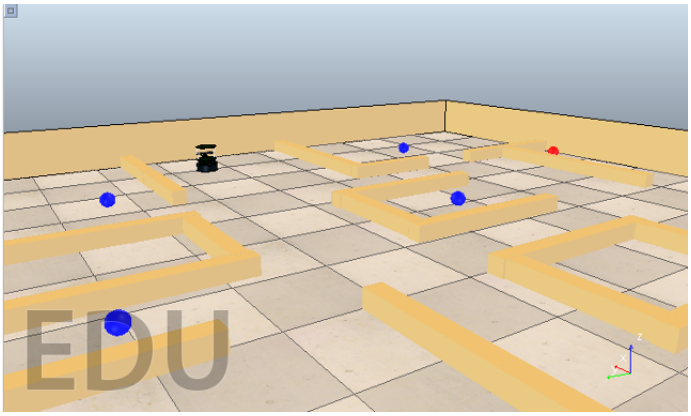


Figura 3.2: Ambiente virtual desenvolvido para o experimento.

tal a implementação consistiu em realizar a leitura de todos os tópicos do ROS registrados no LidaBridge e guardas seus valores em uma *Hashtable*. O segundo método implementado é o *getSensoryContent*, o qual busca na *Hashtable* o valor de determinado sensor.

Uma customização foi realizada no módulo de ambiente para realizar a integração dos algoritmos de controle do robô com o modelo de consciência, esta integração foi realizada a partir da leitura e publicação de mensagens em tópicos do ROS. Para a implementação desta integração, foi utilizado um pacote do ROS chamado *RosBridge*[3], que funciona realizando a leitura dos tópicos no sistema ROS e disponibilizando os resultados destas leituras via *websockets*. A aplicação Java por sua vez (LIDA), se conecta ao *RosBridge* e realiza a leitura dos dados.

Três detectores de baixo nível foram implementados para tratar das percepções provenientes da memória sensorial. O *HealthDetector* é responsável por monitorar o estado da bateria virtual e possui três possíveis estados que representam o nível de bateria que o robô possui no momento: *goodHealth*, *fairHealth* e *badHealth*. A Tabela 3.1 apresenta os valores associados a cada um destes estados, valores estes definidos a partir de diversos testes realizados com diferentes conjuntos de valores. Utilizando uma taxa de decaimento de 0.01 por *tick* (unidade de medida de execução do LIDA), o robô possui 800 *ticks* para encontrar um carregador quando está no estado *badHealth*. No ambiente utilizado esta foi uma configuração suficiente para o robô chegar em um carregador na maioria das situações.

Estado	**Nível de bateria**
goodHealth	Maior que 14
fairHealth	Maior que 8 e menor que 14
badHealth	Menor que 8

Tabela 3.1: Valores de referência para o *HealthDetector*

O detector *GoalDistanceDetector* monitora a distância do robô até o objetivo. Esta distância é medida utilizando o cálculo da Distância Euclidiana (como apresentada na Equação 3.1) entre o robô e o objetivo.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.1}$$

[3] `http://wiki.ros.org/rosbridge_suite`

O cálculo da distância não leva em consideração o caminho que será percorrido para desviar dos obstáculos mas somente a distância em linha reta visto que inicialmente o robô não possui o conhecimento do ambiente porque o mapeamento é realizado durante a navegação. Outro fator que levou a esta decisão foi manter um modelo simples para este primeiro experimento. A Tabela 3.2 apresenta os valores de distância associados aos estados possíveis.

Estado	**Distância**
goalFar	Objetivo está a mais de 4 metros
goalMedium	Objetivo está a mais de 2 metros e menos de 4 metros
goalNear	Objetivo está a menos de 2 metros

Tabela 3.2: Valores de referência para *GoalDistanceDetector*

O terceiro detector é o *RechargeDistanceDetector* que monitora a distância entre o robô e o carregador mais próximo. Este detector funciona de forma similar ao *GoalDistanceDetector*, sendo os estados possíveis: *rechargeFar*, *rechargeMedium* e *rechargeNear*. Os valores de referência utilizados para estes estados são os mesmos do detector de distância do objetivo.

Nove codelet*s* de atenção foram definidos no módulo de atenção: *BadHealthAttention*codelet, *FairHealthAttention*codelet, *GoodHealthAttention*codelet, *FarGoalAttention*codelet, *MediumGoalAttention*codelet, *NearGoalAttention*codelet, *FarRechargeAttention*codelet, *MediumRechargeAttention*codelet *e NearRechargeAttention*codelet. No módulo de procedimento foram definidos 20 esquemas de ação, que são apresentados na Tabela 3.3. A definição destes esquemas consistiu em um conjunto de regras abrangendo todas as combinações entre as variáveis de quantidade de bateria e distância dos carregadores. Nos experimentos posteriores serão apresentados esquemas de ação contendo conjunto de regras menos complexos, de forma a avaliar o comportamento do modelo em uma situação onde não existe uma modelagem de regras para todas as combinações de variáveis.

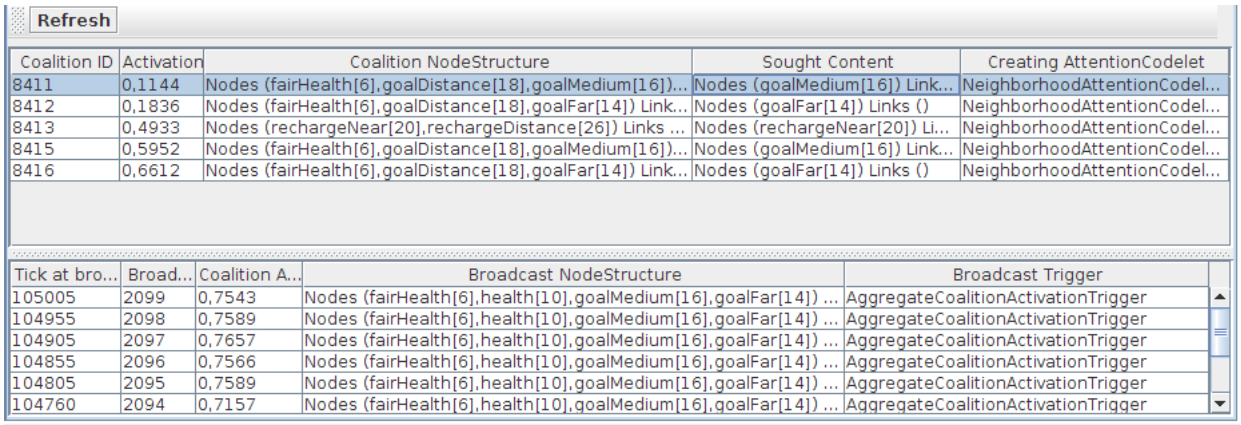


Figura 3.3: Interface gráfica do LIDA mostrando o estado do *Workspace Global* durante a execução da simulação.

A Figura 3.3 apresenta a interface do LIDA na aba que mostra o conteúdo do Workspace Global, composto por duas tabelas. A tabela superior mostra as coalizões que estão competindo pela consciência. A coluna *Coalition ID* apresenta o código único que representa a

Nós Ativados	**Ação**
goodHealth	ir para o objetivo
badHealth	ir para o carregador
rechargeNear, goodHealth	ir para o objetivo
rechargeNear, fairHealth	ir para o objetivo
rechargeNear, badHealth	ir para o carregador
goalNear, goodHealth	ir para o objetivo
goalNear, fairHealth	ir para o objetivo
goalNear, badHealth	ir para o objetivo
goalMedium, badHealth	ir para o objetivo
goalMedium, fairHealth	ir para o carregador
goalMedium, fairHealth	ir para o carregador
goalFar, goodHealth	ir para o objetivo
goalFar, fairHealth	ir para o carregador
goalFar, badHealth	ir para o carregador
rechargeMedium, goodHealth	ir para o objetivo
rechargeMedium, fairHealth	ir para o carregador
rechargeMedium, badHealth	ir para o carregador
rechargeFar, goodHealth	ir para o objetivo
rechargeFar, fairHealth	ir para o objetivo
rechargeFar, badHealth	ir para o carregador

Tabela 3.3: Esquemas de ação configurados no módulo de procedimento.

coalizão. A segunda coluna *Activation* apresenta um valor numérico que representa a ativação atual da coalizão. A coluna *Coalition NodeStructure* representa a coalizão em si, ou seja, possui uma lista de todos os nodos que formam a coalizão. A coluna *Sought Content* representa qual era o conteúdo que o nodo que criou a coalizão estava observando. A última coluna apresenta qual o tipo do codelet de atenção que criou a coalizão.

A tabela inferior da Figura 3.3 apresenta os broadcasts realizados, ou seja, quais das coalizões sendo apresentadas na tabela superior venceram a ”competição pela consciência”. A primeira coluna *Tick at Broadcast* mostra em qual momento (*tick* que representa uma unidade de tempo no LIDA) o broadcast foi realizado. A segunda coluna *Broadcast count* apresenta um número que representa o número de broadcasts realizados. A terceira coluna, *Coalition Activation* apresenta a ativação da coalizão no momento em que ela venceu a competição. A coluna *Broadcast NodeStructure* apresenta o conteúdo da coalizão que foi difundida pelo Workspace Global. A última coluna, *Broadcast Trigger* apresenta qual ”gatilho” foi disparado para que o *broadcast* consciente fosse realizado.

Para a execução desta tarefa de chegar ao objetivo sem deixar a “bateria virtual” acabar, o robô apresentou resultados positivos, tomando decisões de quando e onde ir de forma que em todos os testes executados onde o robô tinha bateria suficiente para chegar a um carregador, o robô conseguiu chegar ao objetivo. Houveram algumas dificuldades com relação ao desempenho do sistema. Devido a uma baixa taxa de renderização do simulador (entre 2 a 6 *frames* por segundo) alguns parâmetros tiveram que ser ajustados para realizar uma sincronização razoável entre os sistemas (V-REP e sistema de controle).

3.7 Validação experimental - Corredor

Foi desenvolvido um segundo experimento para realizar uma avaliação da capacidade de tomada de decisão de um robô móvel utilizando um modelo de consciência baseado no LIDA. O ambiente utilizado para o experimento foi uma simulação de um ambiente com obstáculos móveis, onde para chegar ao seu objetivo o robô deve passar por um corredor onde existe uma pessoa circulando. O robô deve decidir o que fazer quando encontrar a pessoa ou quando identificar a possibilidade da pessoa passar.

3.7.1 Ambiente de Simulação

Este ambiente de simulação também foi desenvolvido utilizando a plataforma V-REP. Foi desenvolvido um modelo do robô Pioneer 3AT[4] para a realização da tarefa. Também foram desenvolvidos outros componentes de software necessários para a integração do robô virtual com o ROS, consistindo em um modelo cinemático do robô para o envio de comandos aos atuadores e um sistema de odometria (os componentes implementados executam como dois nós no ROS). A simulação utiliza um modelo de pessoa virtual para a simulação de um obstáculo móvel, que segue uma trajetória predeterminada. A figura 3.4 apresenta o ambiente de simulação utilizado para a realização deste experimento.

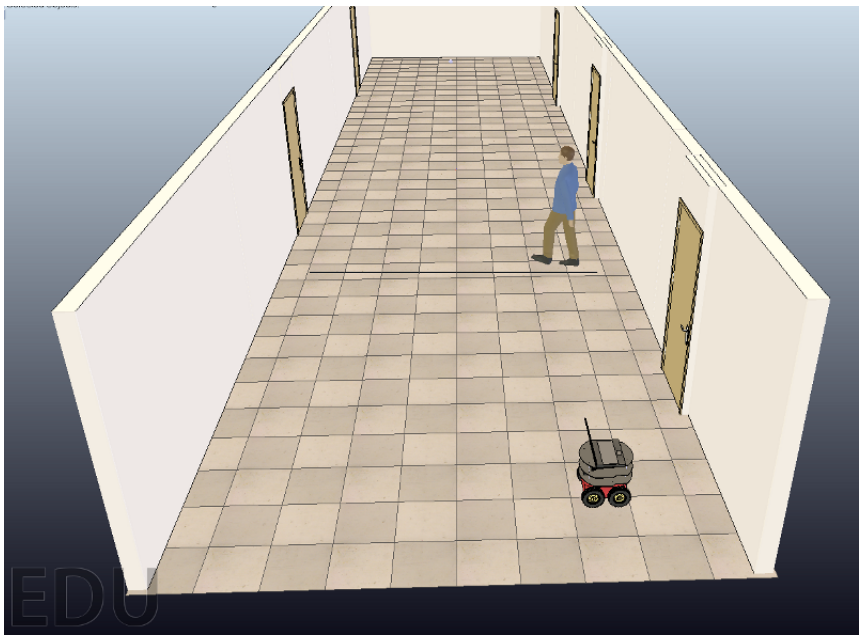


Figura 3.4: Ambiente de simulação utilizado para a realização do segundo experimento.

3.7.2 Definição do Agente

Para a implementação do controlador de alto nível, foi utilizada a linguagem Java e o *LIDA Framework*, sendo que o controlador de baixo nível consiste na utilização do ROS com a *Navigation Stack*, que é o conjunto de programas responsável por realizar a navegação de

[4] `http://www.mobilerobots.com/ResearchRobots/P3AT.aspx`

um robô móvel. Este conjunto de programas contém módulos que realizam o mapeamento e localização simultâneos, o planejamento e a execução da trajetória.

Os dois controladores se comunicam através da estrutura de *websockets*, o pacote *rosbridge_websocket* é responsável por realizar a comunicação entre o ROS e o controlador de alto nível, serializando e des-serializando os dados em formato *JavaScript Object Notation* (JSON) e comunicando os dois sistemas pelo protocolo HTTP.

O comando que o controlador de alto nível envia para o ROS consiste apenas na posição para onde o robô deve se mover, e a partir desta posição, o controlador de baixo nível realiza o planejamento e a execução da trajetória para atingir o objetivo.

Para este experimento, a posição do obstáculo móvel não é calculada utilizando os sensores do robô, a informação vem diretamente do simulador, que publica a posição na estrutura de comunicação do ROS.

3.7.3 Módulo de Ambiente

O módulo de ambiente é responsável por realizar a integração do agente com o problema em si. Para este experimento, foi desenvolvido um módulo que realiza a integração do agente LIDA com o controlador ROS. Este módulo consiste em uma classe chamada *ROSEnvironment*, e é configurado como um módulo na estrutura do *framework* LIDA. Esta classe realiza a integração do sistema através da leitura dos dados provenientes do ROS e o envio de comandos para o controlador.

3.7.4 Percepções

A estrutura utilizada para a realização das percepções é composta pelos seguintes componentes: memória sensorial, memória perceptual associativa e pelos *codelets* de atenção (conforme as sessões 2.3.2, 3.5.3, 3.5.4 e 3.5.5) respectivamente.

O módulo de memória sensorial não possui uma implementação padrão no *framework* LIDA e requer uma implementação específica para o domínio do problema. O módulo de memória sensorial implementado busca as informações sensoriais por meio do módulo de ambiente e as disponibiliza para o módulo de memória perceptual associativa. Os dados disponibilizados pela memória sensorial consistem na posição do robô, a posição do obstáculo, o movimento do robô e o movimento do obstáculo.

Na PAM as percepções possíveis são inseridas na *slipnet*. Cada percepção que pode ser recebida do ambiente possui um nó correspondente. Há duas formas diferentes de definição dos nós da PAM: nós criados durante a inicialização do agente e nós criados dinamicamente em tempo de execução. A Tabela 3.4 apresenta os nós criados durante a inicialização, e seu significado. A Tabela 3.5 apresenta as ligações entre os nós.

Os nós criados em tempo de execução representam a posição do robô no sistema de coordenadas cartesiano. Cada nó de posição representa um quadrado de $1m^2$ de área, e seu nome é composto pelos valores das coordenadas. Por exemplo, se o robô está na posição (1.0231, 2.9381) o nome do nó equivalente a esta posição é (1, 2). Estes nós são criados e adicionados na PAM pelo *codelet* de percepção *RobotPoseDetector*.

Os *codelets* de atenção são responsáveis por realizar a leitura da memória sensorial e estimular os nós da PAM que correspondem a leitura sensorial. A Tabela 3.6 apresenta os *codelets* implementados e quais nós são estimulados.

Nó	Significado
obstacle	Obstáculo.
onear	Obstáculo está perto.
ofar	Obstáculo está longe.
oidle	Obstáculo está parado.
omoving	Obstáculo está se movendo.
robot	Robô.
rmoving	Robô está se movendo.
ridle	Robô está parado.
rgoal	Última ação realizada foi seguir para o objetivo.
rstop	Última ação realizada foi parar.
rexcuse	Última ação realizada foi pedir licença.

Tabela 3.4: Nós da PAM criados durante a inicialização do sistema.

Nó	Nós ligados
obstacle	ofar, onear, oidle, omoving.
robot	rmoving, ridle, rgoal, rstop, rexcuse, (x,y)

Tabela 3.5: Os links da PAM que relacionam os nós de percepção.

O *RobotPoseDetector*, quando detecta a posição do robô, primeiramente verifica se o nó relativo a posição já existe, se existir ele é estimulado. Se o nó não existir, um novo nó é criado e adicionado na PAM, e tem sua ativação estimulada. Juntamente ao nó de posição é criado um link para relacionar o nó de posição com o nó *robot*.

O detector de distância realiza o cálculo para identificar a distância do obstáculo, e se esta distância for menor que $1,2m$ os nós *obstacle* e *onear* são estimulados, caso contrário, os nós *obstacle* e *ofar* são estimulados.

Para os detectores de movimento *ObstacleMovementDetector* e *RobotMovementDetector* a memória sensorial disponibiliza a informação que é proveniente de um nó ROS, que identifica se o obstáculo/robô está em movimento.

3.7.5 Memória Associativa

Foi utilizada uma memória associativa como memória episódica de curto prazo para a realização de associações das percepções com situações vistas anteriormente. A memória associativa possui uma implementação padrão no *framework* LIDA através de uma memória esparsa distribuída. Os parâmetros utilizados para a configuração da memória associativa, seguiram o padrão sugerido por Kaverna, como segue:

- Tamanho do endereço: 1000 bits;
- Tamanho da palavra: 1000 bits;
- Número de *Hard Locations*: 10000;

Codelet	**Nós**
ObstacleDistanceDetector	obstacle, onear, ofar.
ObstacleMovementDetector	obstacle, oidle, omoving.
RobotMovementDetector	robot, ridle, rmoving.
RobotStatusDetector	robot, rgoal, rstop, rexcuse
RobotPoseDetector	robot, (x,y)

Tabela 3.6: *Codelets* de percepção e os nós estimulados por eles.

- Raio de ativação: 451;

A implementação atual da SDM no *framework* LIDA não realiza a gravação nem a recuperação de links, apenas nós. Devido a esta característica, quando as associações são levadas ao modelo situacional corrente, os nós provenientes da associação só tem sua ativação reforçada quando uma associação carregar o nó novamente, visto que não existem links para propagar a ativação de outros nós.

3.7.6 *Codelets* de Atenção

Os *codelets* de atenção são responsáveis por monitorar se o conjunto de *codelets* a ele associados estão presentes no modelo situacional corrente. Se um codelet de atenção encontra este conjunto, uma nova coalizão é criada para competir pela consciência. A Tabela 3.7 apresenta os *codelets* de atenção utilizados para este problema. Todos os *codelets* de atenção utilizaram uma versão ligeiramente modificada do *codelet* padrão *NeighborhoodAttentioncodelet*. Esta modificação foi realizada para que quando um *codelet* adiciona um nó em sua coalizão todos os vizinhos diretos dos nós também sejam adicionados. Como os nós de posição são criados dinamicamente, não há como levá-los para a consciência utilizando a implementação padrão, onde os nós monitorados por um *codelet* são configurados antes da execução. Com a modificação realizada, todo o *codelet* que monitora o nó *robot* adiciona em sua coalizão também o nó de posição.

Codelet	**Nós Monitorados**
Attentioncodelet-01	robot, rstop, ofar.
Attentioncodelet-02	robot, rgoal, onear, omoving.
Attentioncodelet-03	robot, rstop, onear, oidle.
Attentioncodelet-04	robot, rexcuse, ofar.

Tabela 3.7: *Codelets* de atenção e os nós monitorados por eles.

3.7.7 Esquemas de ação

Foram definidos 4 esquemas de ação para a seleção de ação conforme apresentados na Tabela 3.8. Os esquemas são utilizados pelo mecanismo de seleção de ação para decidir qual ação será realizada, dado o contexto atual.

id	Nome	Contexto	Resultado	Ação
scheme.1a	Seguir	robot, ofar, rstop	rgoal	action.seguir
scheme.1b	Seguir	robot, rexcuse, ofar	rgoal	action.seguir
scheme.2a	Parar	robot, rgoal, onear, omoving	rstop, ofar, omoving	action.parar
scheme.3a	Pedir Licença	robot, rstop, onear, oidle	rstop, ofar, omoving	action.licenca

Tabela 3.8: Esquemas de ação, seu contexto, resultado e ação.

O *scheme.1a* realiza a ação de seguir em direção ao objetivo, se o robô estiver parado e o obstáculo estiver distante. O *scheme.1b* realiza a ação de seguir em direção ao objetivo se sua última ação foi a de pedir licença, e o obstáculo está longe. O *scheme.2a* faz o robô parar, se um obstáculo estiver se movendo próximo a ele. O *scheme3.a* faz o robô pedir licença, se ele estiver parado e o obstáculo estiver parado e próximo.

3.7.8 Resultados

A seguir são apresentados os resultados da realização do experimento descrito acima, apresentando as decisões tomadas pelo robô em face de diferentes situações. As medições realizadas dos parâmetros internos do agente foram realizadas em intervalos de 200 ticks do *framework* (200 unidades de tempo do LIDA).

Situação 1

A primeira situação consiste na seguinte configuração do ambiente: O robô está parado na coordenada (-0.2,5.2) do ambiente, o obstáculo (pessoa) encontra-se parado na posição (-1.3,1,3). A Figura 3.5 apresenta o ambiente de simulação com esta configuração.

Durante este intervalo foram realizados 4 broadcasts "conscientes" pelo Workspace Global, apresentados na tabela 3.9.

Conteúdo da coalizão	Ocorrências
ofar, rstop, obstacle, ridle, (0;5), robot	3
ofar, rstop, ridle, (0;5), robot	1

Tabela 3.9: Broadcasts "conscientes" realizados durante a situação 1.

A tabela 3.10 mostra a ativação dos esquemas de ação na memória procedural. A maior ativação (0.55) é da ação "Seguir", cujo contexto é o com mais semelhanças aos broadcasts conscientes ocorridos.

Dadas as posições do robô e do obstáculo descritas anteriormente, e como o cálculo da distância entre os pontos apontou que o obstáculo está longe do robô, e o robô estava parado, logo a ação escolhida foi de seguir em direção ao objetivo (pois não existem empecilhos para que o robô alcance o objetivo na situação atual). Durante a execução o mecanismo de seleção de ação selecionou uma ação, relativa ao esquema com maior ativação, seguir em direção ao objetivo. A figura 3.6 mostra o estado do modelo situacional corrente após a execução de 200 ticks do *framework* neste intervalo. A figura apresenta um grafo que representa os nós e os links

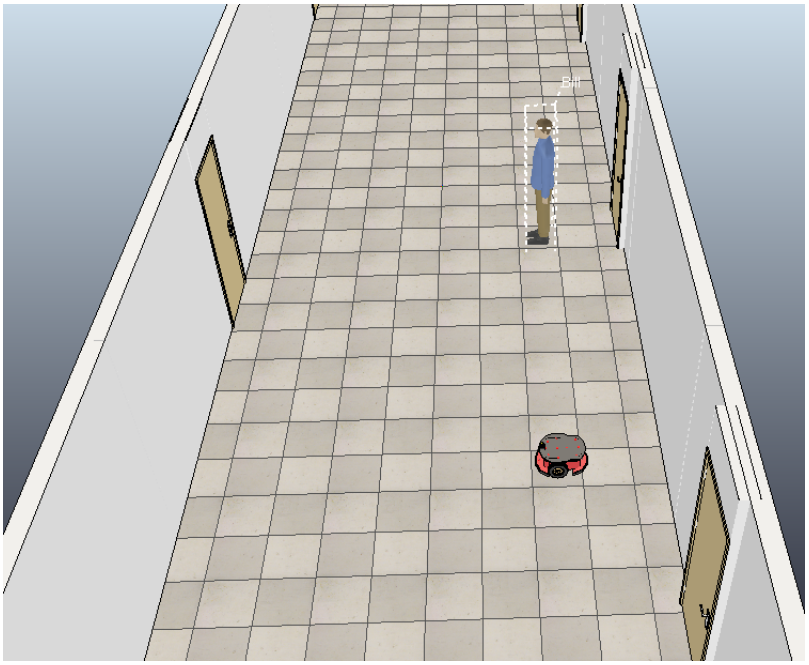


Figura 3.5: Ambiente de simulação apresentando a Situação 1.

Ação	**Contexto**	**Ativação**
Pedir licença	onear, rstop, odile, robot	0.3
Seguir	ofar, rexcuse, robot	0.36
Seguir	ofar, rstop, robot	0.55
Parar	onear, rgoal, omoving, robot	0.15

Tabela 3.10: Ativação dos esquemas de ação durante a situação 1.

entre eles. Cada nó é uma percepção presente no modelo situacional corrente, logo podemos observar os nodos *obstacle*, *oidle*, *ofar*, *ridle*, *0:5*, *robot*, *rstop* e *rgoal*. Como pode ser percebido o nó que representa o estado do agente *rstop* não possui mais o link para o nó *robot* pois está decaindo, já que o novo estado *rgoal* foi ativado após a ação de seguir em direção objetivo ser selecionada.

Situação 2

A segunda situação consiste na seguinte configuração do ambiente: o robô está parado, localizado na coordenada (-1.6,2.0) do ambiente, o obstáculo (pessoa) encontra-se parada na posição (-1.3,1,3). A Figura 3.7 apresenta o ambiente de simulação com esta configuração.

Durante este intervalo foram realizados 4 broadcasts "conscientes" pelo Workspace Global, apresentados na tabela 3.11.

A tabela 3.12 mostra a ativação dos esquemas de ação na memória procedural. A maior ativação (0.37) é da ação "Pedir licença", cujo contexto é o com mais semelhanças aos broadcasts conscientes ocorridos.

Durante a execução, o mecanismo de seleção de ação selecionou uma ação, relativa ao esquema com maior ativação, pedir licença. A figura 3.8 mostra o estado do modelo situacional

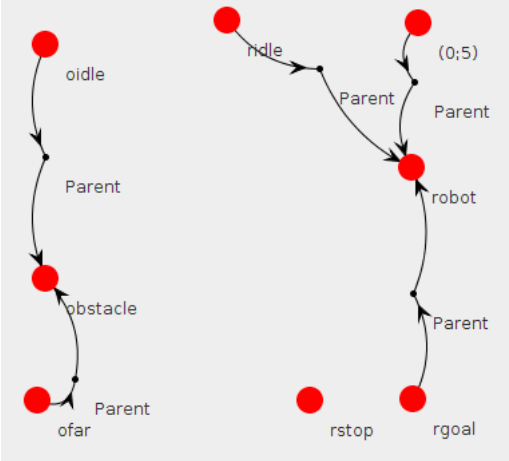


Figura 3.6: Modelo situacional corrente apresentado na interface gráfica do LIDA para a situação 1.

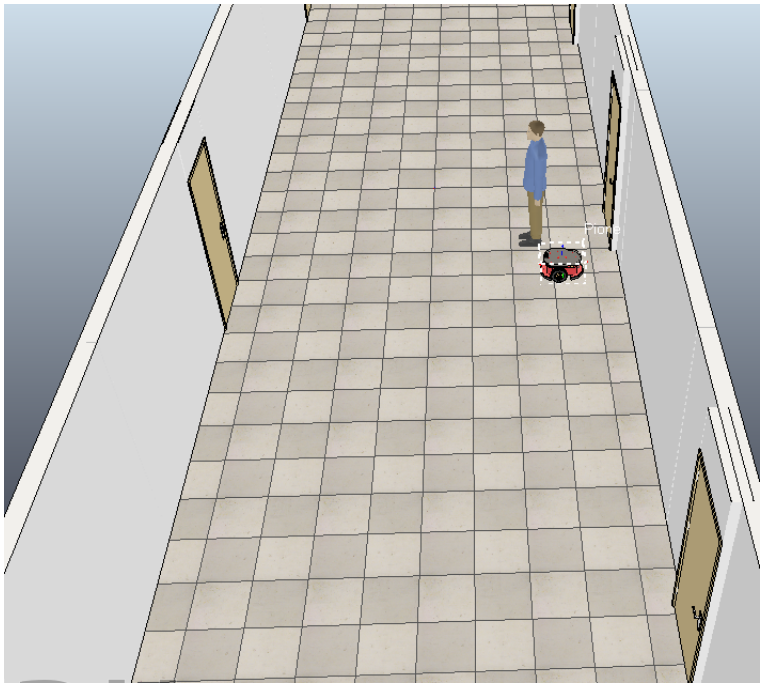


Figura 3.7: Ambiente de simulação apresentando a Situação 2.

corrente após a execução de 200 ticks do *framework* neste intervalo. A figura apresenta um grafo que representa os nós e os links entre eles, cada nó é uma percepção presente no modelo situacional corrente, logo podemos observar os nodos *obstacle*, *oidle*, *onear*, *ridle*, *(-1:2)*, *robot*, *rstop* e *rexcuse*. Como pode ser percebido o nó que representa o estado do agente *rstop* não possui mais o link para o nó *robot*, pois está decaindo, já que o novo estado *rexcuse* foi ativado após a ação de pedir licença. Em geral foi apresentada uma situação ao robô, onde um humano estava próximo a ele e a ação selecionada foi de pedir licença para o humano, para que o robô possa continuar seu trajeto sem colisões.

Conteúdo da coalizão	Ocorrências
onear, (-1,2), rstop, obstacle, ridle, rexcuse, oidle, robot, robot	2
onear, (-1,2), rstop, obstacle, ridle, oidle, robot	1
onear, (-1,2), rstop, ridle, oidle, robot	1

Tabela 3.11: Broadcasts “conscientes” realizados durante a situação 2.

Ação	Contexto	Ativação
Pedir licença	onear, rstop, odile, robot	0.37
Seguir	ofar, rexcuse, robot	0.33
Seguir	ofar, rstop, robot	0.17
Parar	onear, rgoal, omoving, robot	0.25

Tabela 3.12: Ativação dos esquemas de ação durante a situação 2.

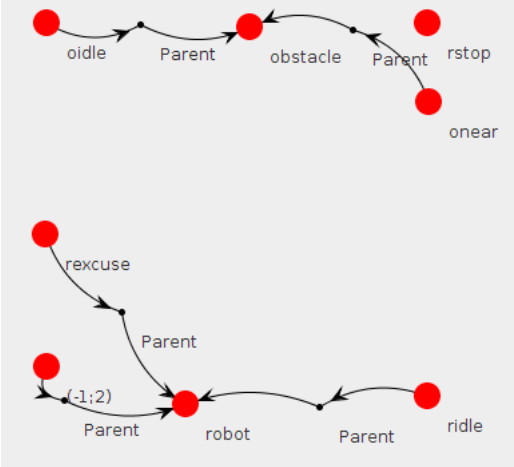


Figura 3.8: Modelo situacional corrente apresentado na interface gráfica do LIDA para a situação 2.

Situação 3

A terceira situação consiste na seguinte configuração do ambiente: O robô está em movimento indo em direção a região onde se localiza o obstáculo, também em movimento. A Figura 3.9 apresenta o ambiente de simulação com esta configuração.

Durante este intervalo foram realizados 4 broadcasts “conscientes” pelo Workspace Global, apresentados na tabela 3.13.

A tabela 3.14 mostra a ativação dos esquemas de ação na memória procedural, sendo que a maior ativação (0.59) é da ação parar, cujo contexto é o com mais semelhanças aos broadcasts conscientes ocorridos.

Durante a execução, o mecanismo de seleção de ação selecionou uma ação, relativa ao esquema com maior ativação, Parar. A figura 3.10 mostra o estado do modelo situacional corrente após a execução de 200 ticks do *framework* neste intervalo. Durante a execução desta situação, por ter ocorrido após algum tempo de simulação, algumas associações foram feitas pela memória episódica. A 3.11 apresenta o buffer de percepção após esta execução. O modelo

Figura 3.9: Ambiente de simulação apresentando a Situação 3.

Conteúdo da coalizão	**Ocorrências**
onear, rgoal, obstacle, ridle, omoving, robot, (0,1)	1
onear, rgoal, obstacle, (1,1), ridle, omoving, robot, (0,1)	1
rgoal, ofar, rstop, obstacle, rmoving, (1,1), (1,2), robot	1
rgoal, ofar, rstop, obstacle, rmoving, (1,2), robot	1

Tabela 3.13: Broadcasts "conscientes" realizados durante a situação 3.

situacional corrente contém os nós *ofar* e *rgoal* que não estão presentes na percepção atual, sendo resultado de uma associação da memória episódica.

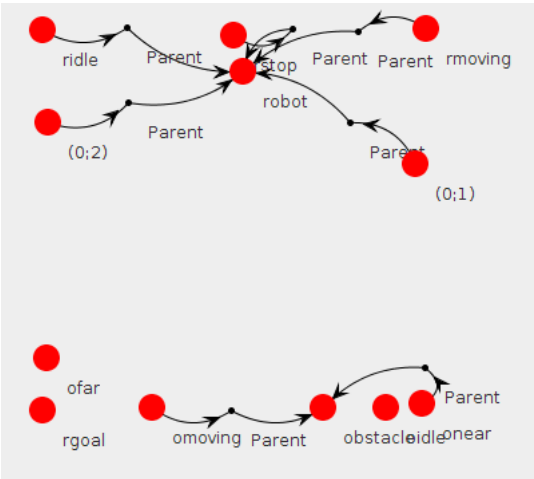


Figura 3.10: Modelo situacional corrente apresentado na interface gráfica do LIDA para a situação 3.

Nesta situação, o robô estava se movimentando em direção a um obstáculo (pessoa) que também estava em movimento. A ação do robô foi de parar, esperando que o obstáculo em movimento continue em movimento saindo da região que representa perigo ao robô.

Ação	Contexto	Ativação
Pedir licença	onear, rstop, odile, robot	0.3
Seguir	ofar, rexcuse, robot	0.2
Seguir	ofar, rstop, robot	0.2
Parar	onear, rgoal, omoving, robot	0.59

Tabela 3.14: Ativação dos esquemas de ação durante a situação 3.

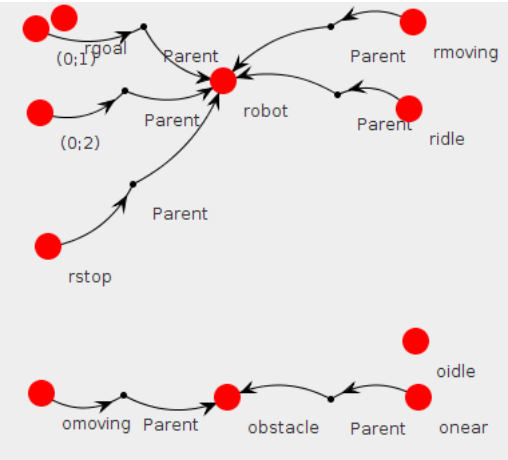


Figura 3.11: O *perceptual buffer* apresentado na interface gráfica do LIDA para a situação 3.

3.8 Conclusão

Com a execução destes experimentos foi possível constatar que é possível utilizar um sistema de controle para um robô móvel a partir de um agente construído com o *framework* LIDA. Os experimentos realizado ainda estão distantes de uma situação em ambiente real e com sensoriamento real, desta forma os próximos experimentos, descritos na seção 4.1 do capítulo 4, consistem em aprimorar os modelos do ambiente de esquemas de ação para a realização de testes em um ambiente real. As associações da memória episódica, embora presentes no experimento, não tiveram um resultado significativo, devido a simplicidade do modelo de percepções. Em um primeiro momento, poderia ser identificada alguma semelhança do modelo apresentado com um sistema baseado em regras *Fuzzy* [Dubois and Prade, 1996] [Pedrycz and Gomide, 2007], pois em ambos os casos temos um conjunto de variáveis linguísticas representando a situação atual, e um conjunto de regras as quais podem ser ativadas para obter uma tomada de decisão. Porém as semelhanças não vão muito além, pois diferente de um conjunto de regras *fuzzy* a ativação das regras (esquemas de ação no LIDA), dependem de competições que ocorrem no sistema pela ativação das coalizões, de forma que apenas a regra mais saliente contribui com o resultado final, não havendo uma contribuição de todas as regras como no caso de um sistema *fuzzy*. Um ponto interessante que podia ser explorado futuramente, seria uma utilização conjunta destas duas tecnologias, visto que no modelo utilizado neste trabalho, as variáveis linguísticas são obtidas através de uma conversão direta de valores escalares. A aplicação de regras *fuzzy* em conjunto com o modelo LIDA, poderia trazer uma maior adaptabilidade ao sistema na identificação das situações com base nos valores de entrada dos sensores. No próximo capítulo, são apresentados experimentos realizados em um ambiente com situações

mais próximas a realidade, visando avaliar o uso do modelo cognitivo LIDA em aplicações de navegação para robótica móvel.

Capítulo 4

Controle reativo e deliberativo

Este capítulo apresenta uma proposta de controlador híbrido (reativo e deliberativo) para a navegação de robôs móveis. Foi implementado um controlador para um robô móvel utilizando uma arquitetura em duas camadas conforme apresentado na seção 4.1.1. O robô deve realizar tarefas de navegação em um ambiente desconhecido previamente pelo mesmo, onde existem obstáculos estáticos (como paredes e outros objetos) e obstáculo móveis, como por exemplo pessoas ou outros robôs móveis. São apresentadas as especificações de software, hardware, e dos ambiente de testes e validação utilizados. Após o término da implementação da solução (incluindo software e hardware), foram realizados testes com o objetivo de comparar o comportamento do robô utilizando a arquitetura proposta, em relação ao mesmo robô utilizando uma abordagem tradicional, baseada apenas em um controlador reativo.

4.1 Estudo de caso

4.1.1 Software

A arquitetura proposta para o controlador do robô utiliza um modelo híbrido com duas camadas distintas de controle, sendo uma reativa e outra cognitiva/deliberativa.

O controlador reativo é responsável por tomadas de decisões imediatas de desvio de obstáculos e navegação, além de realizar as tarefas de auto-localização e mapeamento simultâneos, planejamento e execução de trajetória. Este controlador realiza comunicação direta com o robô, sendo responsável por enviar os comandos aos atuadores e receber as informações dos sensores. As informações provenientes dos sensores são utilizadas para identificar obstáculos e realizar o mapeamento e a localização do robô no ambiente. A realização do SLAM, planejamento e execução de trajetória é feita utilizando os algoritmos já implementados no ambiente ROS [Martinez and Fernández, 2013].

Outra característica do controlador reativo é que ele deve aceitar comandos do controlador cognitivo. Para isso, foi implementado um componente de software que atua de forma a receber os comandos e efetuá-los, enviando os devidos sinais para os componentes do ROS. Estes comandos consistem na definição de uma posição alvo para a navegação, o comando de replanejamento da trajetória, ou outras decisões, como: ou continuar seguindo a trajetória, ou parar de seguir a trajetória e aguardar por um tempo, ou executar um arquivo de áudio requisitando que pessoas que se encontrem paradas pelo caminho ”deem licença” para o robô passar,

bem como outros comandos que sejam necessários durante a implementação. Uma estrutura de dados específica para a troca de informações entre as duas camadas foi desenvolvida.

O controlador cognitivo (chamado também de controlador de alto nível), baseado em um modelo cognitivo, é responsável por identificar percepções de alto nível a partir das informações provenientes da camada inferior e, baseado nestas percepções, realizar a tomada de decisão relativa a próxima ação a ser realizada. As decisões são então enviadas para o controlador reativo (chamado também de controlador de baixo nível) para sua execução. O *framework* LIDA será utilizado para a implementação do modelo cognitivo, utilizando os mecanismos de memória perceptual, memória episódica, atenção e seleção de ação.

As percepções do ambiente são definidas em termos de posição do agente no ambiente, posição dos obstáculos presentes no ambiente, tipos de obstáculos presentes no ambiente, situação dos obstáculos, ação sendo executada atualmente e demais percepções que serão apresentadas na seção 4.3.2.

4.1.2 Hardware

O robô Pioneer 3-AT apresentado nas figuras 4.1 e 4.2 foi utilizado para o desenvolvimento dos experimentos deste trabalho. Para que esse robô ficasse adequado para para seu uso neste projeto, foram necessárias algumas modificações, como a instalação de alguns sensores necessários para a navegação. Foram adicionados quatro sensores ao robô original: um sensor *Kinect* (sensor de profundidade e câmera), um giroscópio, um acelerômetro e um GPS (o qual não foi utilizado, pois os experimentos realizados foram todos *indoor*, onde o sensor não possui um bom funcionamento). Além desses quatro sensores, foi instalado um sonar na parte frontal do robô. Sobre o suporte também foram colocados um microcontrolador Arduino Mega e um netbook. O Arduino é responsável por fazer a leitura dos sensores (com exceção do *Kinect*) e enviar os sinais para o ambiente ROS, que é executado no netbook. A Figura 4.3 apresenta a integração entre os sensores e o Arduino. O netbook é utilizado apenas para executar o controlador de baixo nível, baseado no ambiente ROS. O controlador cognitivo é executado em um computador externo ao robô, realizando comunicação via rede sem fio (via protocolo IEEE 802.11).

Figura 4.1: Robô Pioneer 3-AT com as modificações realizadas para a execução do projeto.

Figura 4.2: Visão frontal do Robô Pioneer 3-AT com o sensor Kinect e netbook.

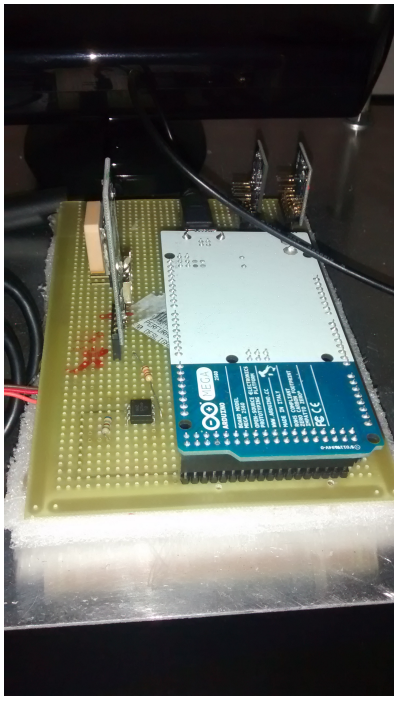


Figura 4.3: Placa universal contendo sensores acelerômetro, giroscópio e GPS, além de um Arduino Mega.

4.1.3 Ambiente de experimentação

Os experimentos para testes e validação da solução proposta foram realizados em um ambiente de piso plano, com circulação de pessoas e obstáculos como paredes e outros objetos estáticos (corredores da UTFPR). Pessoas e outros objetos que possuem características de movimento (como por exemplo outros robôs móveis), são considerados obstáculos dinâmicos. Ambientes onde existem estes tipos de obstáculos representam uma grande dificuldade da navegação de robôs móveis utilizando um controlador tradicional reativo, visto que o ambiente nunca permanece fiel ao mapa na presença destes obstáculos. O controlador apresentado neste

trabalho, propõe uma solução mais eficiente para estes casos, adicionando um controlador deliberativo(cognitivo) ao topo no controlador reativo de baixo nível, coordenando suas ações de forma que possa navegar com eficiência neste tipo de ambiente. Na seção seguinte (seção 4.2), é apresentado o controlador de baixo nível, e na seção imediatamente posterior (seção 4.3), é detalhado o controlador de alto nível proposto neste trabalho.

4.2 Desenvolvimento do controlador de baixo nível

Usando a *Navigation Stack* [Guimarães et al., 2016b], um pacote do ambiente ROS, foi possível configurar o robô para navegar e mapear autonomamente ambientes desconhecidos, usando o sensor de profundidade Kinect. O uso desse pacote possibilitou a navegação de forma que o robô recebe posições relativas a sua posição de partida e chega no ponto de destino, na direção pedida, desviando de qualquer obstáculo no caminho. Durante o percurso, o robô mapeia o ambiente, criando um mapa como o apresentado na figura 4.4. A figura 4.5 mostra o ambiente real e, como pode se ver, a topologia da sala, inclusive a maioria dos obstáculos, estão representados no mapa levantado autonomamente de forma fiel. A definição do controlador reativo resultou no desenvolvimento de um capítulo do livro [Guimarães et al., 2016a], onde é explicado em detalhes as configurações necessárias para a realização da navegação.

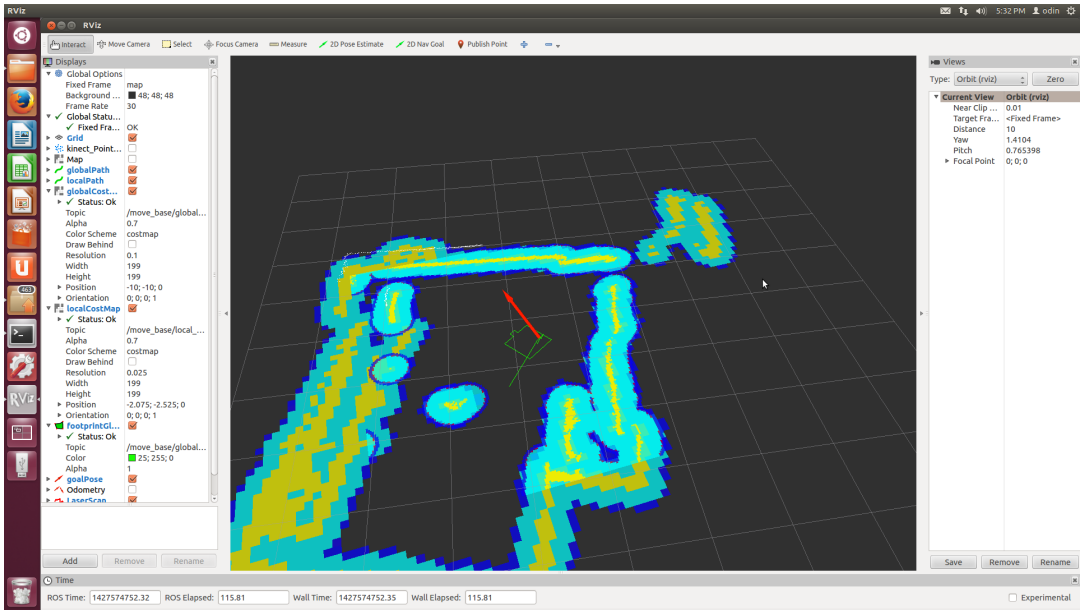


Figura 4.4: Mapa gerado autonomamente pelo robô ao navegar em uma sala do Laboratório Avançado de Sistemas Embarcados e Robótica.

4.3 Desenvolvimento do controlador de alto nível

Nesta seção é apresentado o controlador de alto nível, responsável por identificar percepções de alto nível provenientes da camada inferior e realizar o planejamento das próximas ações com base nas percepções. As ações definidas pelo controlador de alto nível são enviadas para o controlador de baixo nível para execução. Para o desenvolvimento deste controlador

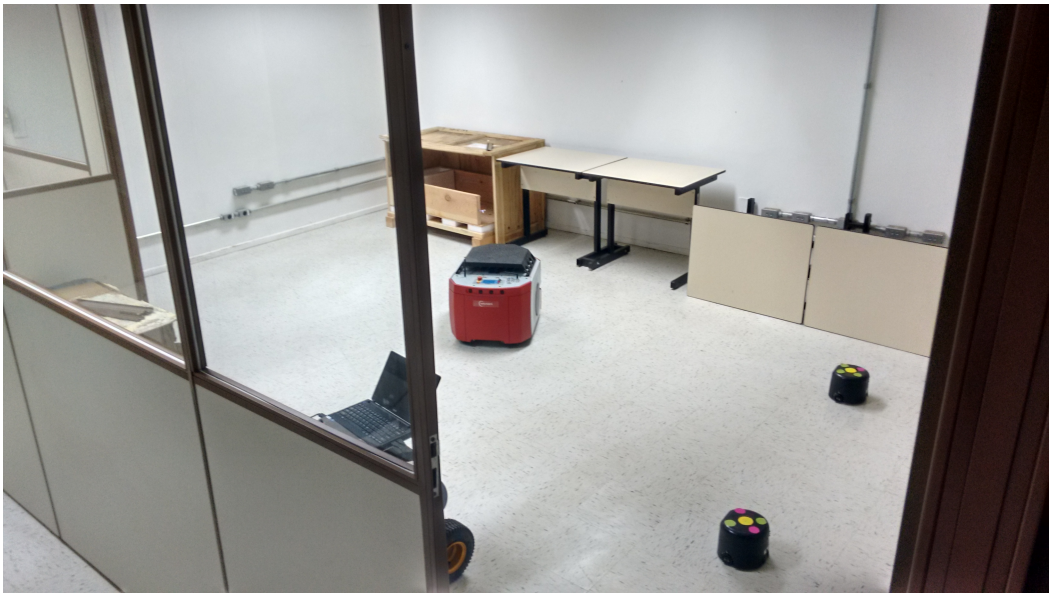


Figura 4.5: Foto do ambiente no momento em que o robô o mapeia, conforme mapa apresentado na figura anterior.

foi utilizado o *Framework* LIDA, utilizando os mecanismos de memória perceptual, memória episódica, atenção e seleção de ação para formar o modelo cognitivo. A arquitetura de software utilizada para o controlador de alto nível segue o padrão de arquitetura do *Framework* LIDA, uma vez que este possui uma arquitetura bem definida para o desenvolvimento de agentes baseados no modelo cognitivo de Baars-Franklin.

4.3.1 Arquitetura

Para o desenvolvimento do controlador de alto nível, foi desenvolvido um simulador, que é responsável por emular o envio das informações que na implementação nos experimentos com robô real, são obtidas a partir do controlador de baixo nível. O simulador consiste em uma série de parâmetros que definem quais serão as percepções enviadas aos detectores de características do controlador de alto nível. A Figura 4.7 apresenta a interface gráfica do simulador como um módulo da interface do *Framework* LIDA, contendo caixas de seleção para a definição de quais os sinais que estão sendo enviados da camada de baixo nível para o controlador de alto nível.

Além da interface gráfica, foi desenvolvida uma outra interface do simulador com o controlador que utiliza um arquivo de texto para a definição dos sinais enviados pelo controlador de baixo nível. No arquivo devem ser definidas diversas situações, onde os sinais para cada situação são atualizados para que seja possível simular de forma mais fluida a navegação do robô por um ambiente. A atualização das situações é realizada por uma tarefa no *Framework* LIDA, a qual permite que seja alterada a frequência de atualização da descrição do ambiente.

Inicialmente a simulação estava sendo realizada utilizando o V-REP (como apresentado nos experimentos anteriores), porém devido a maior complexidade do modelo desenvolvido nesta etapa, optou-se por utilizar uma simulação com um maior nível de abstração, onde ao invés dos dados provenientes dos sensores serem gerados por uma simulação realista do ambiente, eles poderiam ser definidos diretamente, para obter um maior controle de todos os

parâmetros do ambiente. Desta forma optou-se por primeiramente definir e implementar o controlador de alto nível, abstraindo os detalhes de implementação da camada de baixo nível. Nas etapas posteriores deste projeto, foi implementada a camada de baixo nível e realizada a integração dos dois níveis de controle.

A Figura 4.6 apresenta uma visão a nível de componentes da arquitetura desenvolvida para a implementação final do controlador deliberativo-reativo. São apresentados neste diagrama a camada de alto nível (controlador deliberativo), o módulo de integração chamado *LIDA Bridge*, e o controlador reativo e seus componentes.

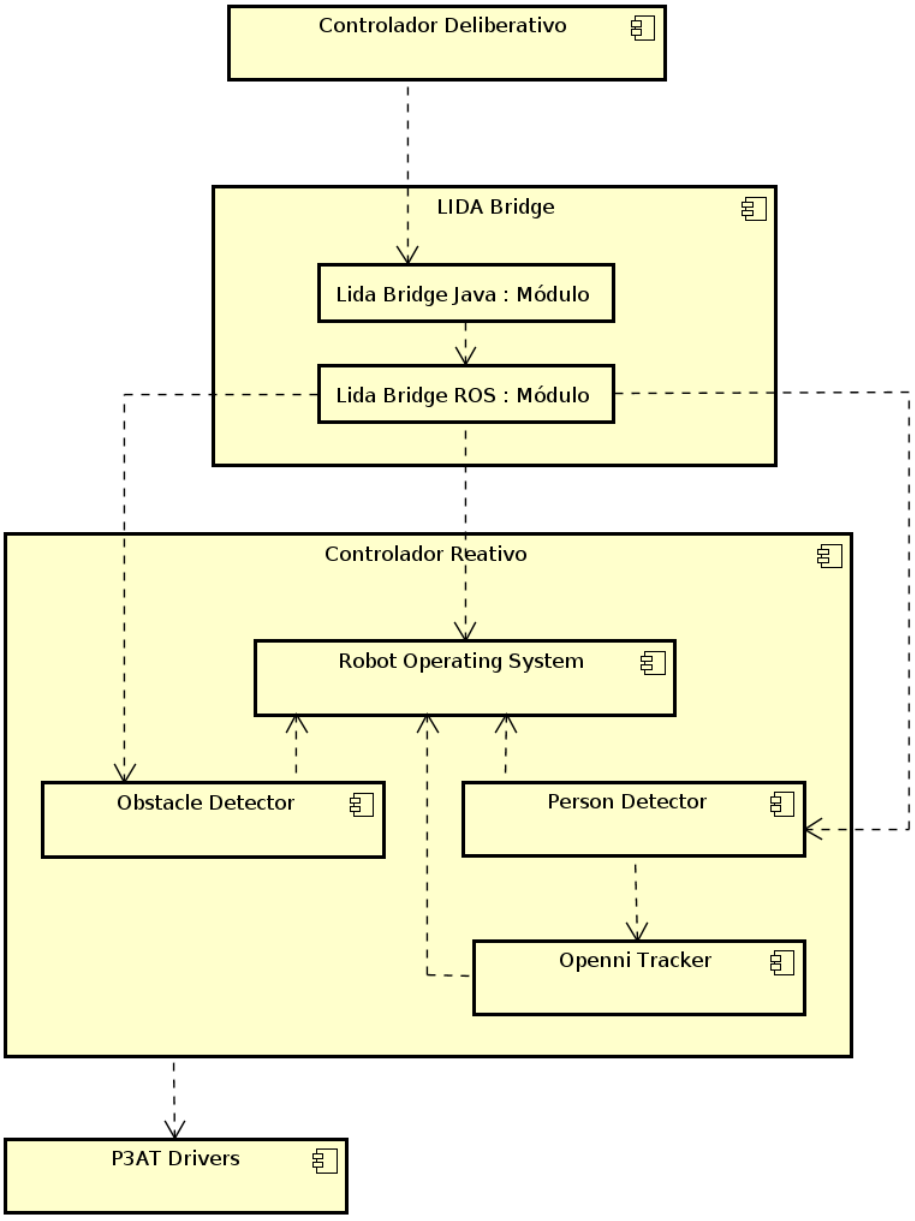


Figura 4.6: Diagrama de componentes apresentando a arquitetura do controlador desenvolvido.

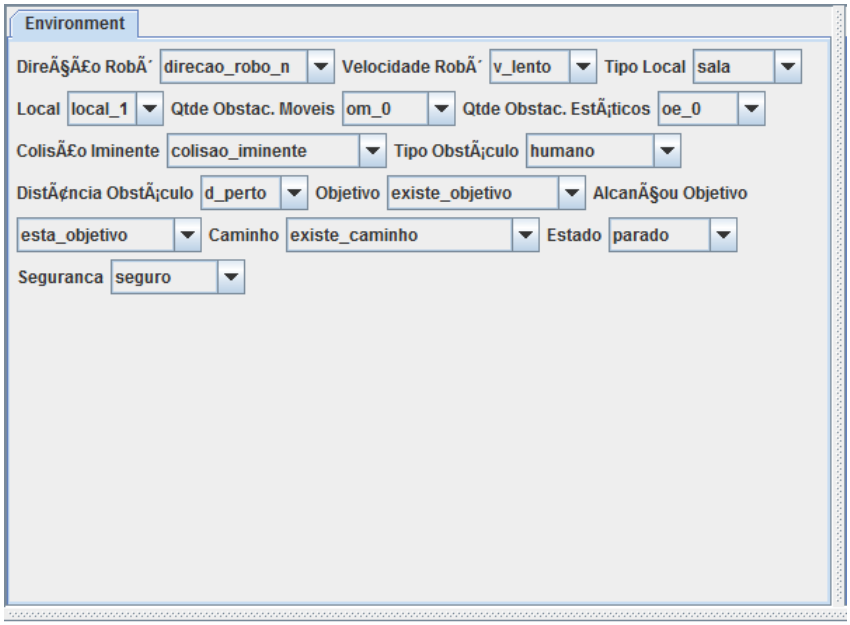


Figura 4.7: A interface gráfica do simulador de baixo nível desenvolvido para a validação do controlador de alto nível.

4.3.2 Percepções

A seguir é apresentada a estrutura de percepções de alto nível desenvolvida para o controlador, estas percepções definem os nodos da *slipnet*. A estrutura apresentada a seguir divide os nodos em categorias, porém estas categorias são apenas uma divisão lógica apresentada nesta seção para uma melhor compreensão da estrutura. A implementação destas não leva em consideração as categorias, mas apenas cada nodo individualmente.

Local: Categoria que contém as percepções que definem em qual o local o robô se encontra (representando uma distinção entre os possíveis ambientes que o robô possa se encontrar). Para o estudo de caso foram definidas três percepções possíveis para os locais onde o robô pode se encontrar, e os seguintes nodos representam os possíveis locais: *local_1*, *local_2*, *local_3*. Um codelet detector de características foi desenvolvido para identificar a presença do robô em determinado local. Durante o desenvolvimento do controlador de alto nível, e as características são identificadas com base nos valores apresentados pelo simulador.

Tipo de Local: Categoria que contém as percepções que definem qual o tipo de local o robô se encontra. O tipo de local está associado a percepção de local, e para o estudo de caso três percepções possíveis para os tipos de local foram definidas. Os seguintes nodos representam os possíveis locais: *corredor*, *sala*, *aberto*, que representam um corredor, uma sala fechada e um espaço aberto, respectivamente.

Obstáculos móveis: Categoria que contém as percepções que definem uma aproximação da quantidade de obstáculos móveis encontrados no ambiente. Para o estudo de caso cinco percepções possíveis para o número de obstáculos móveis foram definidas. Os seguintes nodos representam as possíveis quantidades de obstáculos:

- *om_0*: Não existem obstáculos móveis no ambiente;
- *om_1-2*: Existem 1 ou 2 obstáculos móveis no ambiente;
- *om_2-5*: O número de obstáculos móveis no ambiente está entre 2 e 5;
- *om_5-10*: O número de obstáculos móveis no ambiente está entre 5 e 10;
- *om_>=10*: O número de obstáculos móveis no ambiente é maior ou igual a 10.

Obstáculos estáticos: Categoria que contém as percepções que definem uma aproximação da quantidade de obstáculos estáticos encontrados no ambiente. Para o estudo de caso, cinco percepções possíveis para o número de obstáculos estáticos foram definidas. Os seguintes nodos representam as possíveis quantidades de obstáculos:

- *oe_0*: Não existem obstáculos estáticos no ambiente;
- *oe_1-2*: Existem 1 ou 2 obstáculos estáticos no ambiente;
- *oe_2-5*: O número de obstáculos estáticos no ambiente está entre 2 e 5;
- *oe_5-10*: O número de obstáculos estáticos no ambiente está entre 5 e 10;
- *oe_>=10*: O número de obstáculos estáticos no ambiente é maior ou igual a 10.

Colisão iminente: Categoria que contém as percepções que definem a existência ou inexistência de uma possível colisão do robô com um obstáculo. As percepções possíveis para esta categoria são apenas duas: *colisao_iminente* e *sem_colisao_iminente*. Nesta etapa estas informações vieram diretamente do simulador, nos teste com o robô real a detecção de uma possível colisão utilizou o controlador de baixo nível, que a partir dos sensores do robô pode detectar se um obstáculo está prestes a colidir com o robô.

Direção do robô: Categoria que contém as percepções que definem em qual direção o robô está se movendo. As percepções possíveis para esta categoria são:

- *direcao_robo_n*: Robô está se movendo em direção ao norte;
- *direcao_robo_s*: Robô está se movendo em direção ao sul;
- *direcao_robo_l*: Robô está se movendo em direção ao leste;
- *direcao_robo_o*: Robô está se movendo em direção ao oeste;
- *direcao_robo_ne*: Robô está se movendo em direção ao nordeste;
- *direcao_robo_no*: Robô está se movendo em direção ao noroeste;
- *direcao_robo_se*: Robô está se movendo em direção ao sudeste;
- *direcao_robo_so*: Robô está se movendo em direção ao sudoeste;

Velocidade do robô: Categoria que contém as percepções que definem com qual velocidade o robô está se movendo. As percepções de velocidade não representam valores numéricos de velocidade, mas uma discretização dos valores divididos em três possíveis percepções: *v_lento*, *v_medio*, *v_rapido*.

Distância do obstáculo mais próximo: Esta categoria contém as percepções que definem qual a distância entre o robô e o obstáculo mais próximo que pôde ser detectado. Assim como nas percepções de velocidade do robô, as percepções de distância não representam valores numéricos, mas sim uma discretização dos valores divididos em três possíveis percepções: *d_perto*, *d_medio* e *d_longe*.

Tipo do obstáculo mais próximo: Esta categoria contém as percepções que definem qual é o tipo do obstáculo mais próximo que pôde ser detectado pelo robô. Quatro possíveis percepções foram definidas para esta categoria: *humano*, *parede*, *outro_estatico*, *outro_movel*. nesta etapa as percepções de tipo de obstáculo são provenientes diretamente do simulador, porém durante os experimentos com o robô real, elas são provenientes do tratamento das informações da camada de baixo nível. A detecção de que o obstáculo é um humano é possível pois o principal sensor do robô utilizado é um Kinect, o qual possui seus próprios algoritmos de detecção de pessoas. A detecção de uma parede, outro_estatico e outro_movel, são realizadas a partir do processamento das informações do mapa gerado pelo robô.

Objetivo: Esta categoria contém as percepções que definem a existência de um objetivo para o robô (local ao qual ele deve se dirigir) e percepções que definem se ele conseguiu chegar a este objetivo. Quatro possíveis percepções foram definidas para esta categoria: *existe_objetivo*, *nao_existe_objetivo*, *esta_objetivo*, *nao_esta_objetivo*. As duas primeiras percepções definem a existência ou inexistência de um objetivo para o robô, as duas últimas percepções definem se o robô alcançou o objetivo ou ainda não alcançou o objetivo. Nesta etapa todas estas percepções são provenientes do simulador, porém durante os experimentos com o robô real estas informações são obtidas do controlador de baixo nível do robô da seguinte forma: o objetivo do robô é um local definido por suas coordenadas no mapa, e o processo de verificar se o robô está ou não no objetivo consiste em uma comparação das coordenadas definidas no objetivo com as coordenadas do robô (dado pelo algoritmo SLAM). A existência ou inexistência de um objetivo é uma entrada do usuário na interface de comunicação com o usuário.

Estado do robô: Esta categoria contém as percepções sobre o estado interno do robô, com base em suas últimas decisões. As seguintes seis percepções definem esta categoria:

- *parado*: A última ação do robô foi parar, ou ele acabou de ser inicializado;
- *seguindo*: A última ação do robô foi de seguir o caminho em direção ao objetivo;
- *recuando*: A última ação do robô foi de recuar, devido a alguma manobra para evitar uma colisão;
- *explorando*: A última ação do robô foi de explorar o ambiente, por falta de um objetivo ou falta de um caminho para o objetivo;

- *seguro*: Esta percepção é utilizada internamente pelo controlador de alto nível, ela é ativada quando o robô detecta que a manobra de evitar uma colisão está finalizada;
- *nao_seguro*: Esta percepção é utilizada internamente pelo controlador de alto nível, ela é ativada quando uma manobra para evitar uma colisão está sendo realizada, mas ainda não foi concluída.

Todas as percepções do estado do robô são provenientes do resultado das ações realizadas pelo controlador de alto nível e não percepções do exterior.

Caminho: Esta categoria contém as percepções sobre a existência ou inexistência de caminhos para o objetivo. Três possíveis percepções foram definidas para esta categoria:

- *existe_caminho*: Existe um caminho possível a partir da localização atual do robô até o objetivo;
- *existem_outros_caminhos*: Esta percepção define que além do caminho atual, existem outros caminhos que permitam o robô chegar ao objetivo, esta percepção só é ativada quando o robô procura por caminhos alternativos;
- *nao_existe_caminho*: Esta percepção define que não existem caminhos possíveis para o objetivo a partir da posição atual.

As percepções aqui apresentadas são representadas no controlador através de nodos que iniciam-se na *Perceptual Associative Memory* e formam as estruturas de dados que irão trafegar entre os módulos do controlador.

Para a detecção das características e ativação das percepções foram implementados 12 *codelets* de percepção, como pode ser visto na tabela 4.1. Cada *codelet* é responsável por identificar as percepções relacionadas a uma das categorias.

***Codelet* de percepção**	**Categoria de percepções**
CaminhoDetector	Caminho
ColisaoIminenteDetector	Colisão iminente
DirecaoRoboDetector	Direção do robô
DistânciaObstaculoDetector	Distância do obstáculo mais próximo
EstadoDetector	Estado do robô
LocalDetector	Local
ObjetivoDetector	Objetivo
QtdeOEDetector	Obstáculos estáticos
QtdeOMDetector	Obstáculos móveis
TipoLocalDetector	Tipo de local
TipoObstaculoDetector	Tipo do obstáculo mais próximo
VelocidadeRoboDetector	Velocidade do robô

Tabela 4.1: Codelets de percepção do controlador de alto nível.

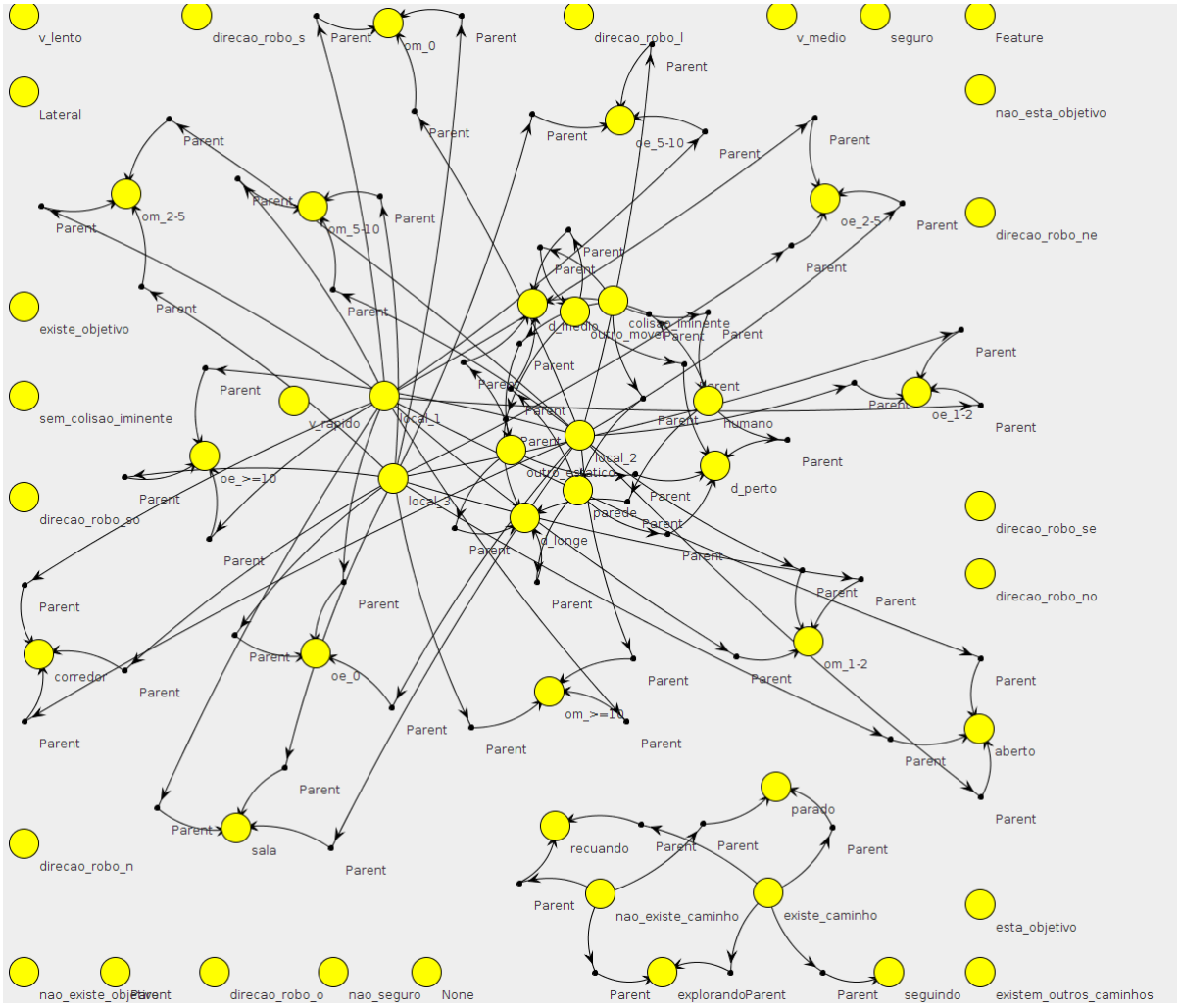


Figura 4.8: Grafo apresentando todas as percepções presentes no controlador de alto nível.

A figura 4.8 apresenta uma representação do grafo formado por todos os nodos que representam as percepções do controlador de alto nível conforme é apresentado pela interface gráfica do controlador de alto nível. Cada um dos círculos amarelos na imagem representam uma das percepções apresentadas nesta seção, as categorias de percepção fornecem apenas uma estrutura para apresentação das mesmas, não sendo utilizadas na implementação. As arestas que ligam cada percepção, representam a conexão entre as percepções, de forma que cada conexão tem a capacidade de propagar sua ativação para as outras percepções conectadas. Tomando como exemplo um dos nodos com o maior número de conexões que pode ser observado ao centro da figura, o *local_2*, nota-se que o mesmo possui conexões com todos os nodos que representam alguma característica do ambiente, como o tipo do local: *sala*, *corredor*, também os nodos representando a quantidade de obstáculos: *oe_1-2*, *oe_2-5*, entre outros. Estas conexões propagam e fortalecem a relação entre os nodos, logo quando uma percepção é encontrada uma outra percepção que está diretamente conectada a ela também recebe uma ativação, devido a sua relação. Quando esta já possui um valor de ativação, a soma de seu valor base com o valor propagado, pode levá-la ao modelo situacional corrente. Esta estrutura foi definida em tempo de projeto e não se altera durante a execução.

4.3.3 Atenção

O módulo de atenção do controlador de alto nível utiliza uma variação do *codelet* de atenção utilizado no segundo experimento. Nos estágios iniciais do desenvolvimento deste controlador, o mecanismo de atenção utilizado foi desenvolvido utilizando *codelets* do tipo *NeighborhoodAttentionCodelet*, porém esta abordagem levou a um problema neste modelo em relação à memória episódica (apresentada na próxima seção). No modelo inicial, cada categoria de nodos de percepção possuía um nodo associado a categoria e este era ligado com links pra cada percepção. O módulo de atenção então observava nodos relativos a categorias para que as percepções relativas as categorias fossem levadas a consciência. Por exemplo, os nodos de velocidade *v_lento*, *v_medio* e *v_rapido* eram ligados ao nodo *velocidade*, e o *codelet* de atenção que queria levar a velocidade do robô para a consciência então observava simplesmente o nodo *velocidade*. O problema com esta abordagem é que o nodo que representava a categoria, neste caso *velocidade* estava sempre ativo, e as associações geradas na memória episódica acabavam associando apenas os nodos que representam as categorias, devido a estes aparecerem muito mais que os outros.

Para resolver este problema um novo *codelet* de atenção foi desenvolvido. O *MultipleAttentionCodelet* é uma variação do *NeighborhoodAttentionCodelet*. Este *codelet* de atenção é capaz de observar uma categoria toda de percepções, eliminando assim a necessidade de possuir nodos que representam a categoria. No algoritmo que identifica a lista de percepções observadas pelo codelet, foi adicionado o operador *or*, quando na implementação original só utilizava-se o operador *and* para definir as combinações de percepções observadas pelo codelet. O exemplo da velocidade citado anteriormente, agora é resolvido de forma que o *codelet* de atenção, ao invés de observar o nodo *velocidade*, observa os três nodos da categoria velocidade: *v_lento*, *v_medio* e *v_rapido*.

Se um destes nodos estiver ativo, ele será identificado pelo *codelet* de atenção. Desta forma não existem nodos que estão sempre ativos, resolvendo o problema das associações da memória episódica citado anteriormente.

Durante o desenvolvimento do controlador de alto nível foram testadas diversas configurações distintas para os *codelets* de atenção, dentre elas a configuração apresentada na tabela 4.2 foi a configuração que apresentou os melhores resultados.

A tabela 4.2 apresenta os *codelets* de atenção definidos e qual situação este procura identificar. A situação é representada pela combinação dos nodos ativos. Por exemplo, no *AttentionCodelet-01* a situação identificada consiste nos nodos *nao_esta_objetivo*, *existe_caminho*, *existe_objetivo* e qualquer nodo pertencente às categorias *local*, *estado do robo*, *obstáculos móveis* e *obstáculos estáticos*.

4.3.4 Memória episódica

Foi utilizada uma memória associativa como memória episódica de curto prazo, para a realização de associações das percepções baseados em eventos passados. A memória associativa possui uma implementação padrão no *Framework* LIDA através de uma memória esparsa distribuída.

O padrão do *Framework* LIDA para a configuração da memória esparsa distribuída utiliza os seguintes parâmetros:

- Tamanho do endereço: 1000 bits;

Codelet de atenção	**Nodos**	**Categorias**
AttentionCodelet-01	*nao_esta_objetivo* *existe_caminho* *existe_objetivo*	Local, Estado do robô, Obstáculos móveis, Obstáculos estáticos
AttentionCodelet-02	*colisao_iminente*	Tipo de local, Distância Velocidade, Estado do Robô
AttentionCodelet-03	*colisao_iminente*	Local, Distância, Obstáculos móveis, Obstáculos estáticos
AttentionCodelet-04	*nao_esta_objetivo*, *nao_existe_caminho*, *existe_objetivo*, *sem_colisao_iminente*	Tipo de local, Direção do robô, Estado do robô
AttentionCodelet-05	*nao_existe_objetivo*, *sem_colisao_iminente*	Estado do robô, Direção do robô
AttentionCodelet-06	*esta_objetivo*	Local, Tipo de local, Colisão, Obstáculos móveis Obstáculos estáticos
AttentionCodelet-07	*esta_objetivo*, *parado*	nenhuma

Tabela 4.2: Codelets de atenção do controlador de alto nível.

- Tamanho da palavra: 1000 bits;
- Número de *Hard Locations*: 10000;
- Raio de ativação: 451.

Os testes iniciais deste modelo foram realizados utilizando esta configuração padrão. Durante os testes constatou-se que esta configuração não atendia o problema, visto que o número de características do modelo é de apenas 48, a diferença entre as situações apresentadas sempre era irrelevante para um raio de ativação de 451. Foram analisadas empiricamente uma série de configurações diferentes de forma que se obtivesse uma configuração mais ajustada para o problema, a configuração a seguir foi a que apresentou os melhores resultados:

- Tamanho do endereço: 54;
- Tamanho da palavra: 54;
- Número de *Hard Locations*: 10000;
- Raio de ativação: 13.

Nota-se que o tamanho do endereço e da palavra ainda são maiores que o número de características utilizadas pelo problema, isto ocorre pois existem caraterísticas internas do *Framework* LIDA que sempre são adicionadas ao modelo.

Com a intenção de persistir o conteúdo da memória associativa foi desenvolvido um módulo chamado *SaveSDMToFile*, que possui a tarefa de gravar o conteúdo da memória associativa em um arquivo no sistema. Uma alteração foi realizada no módulo de memória associativa no *Framework* LIDA para ler o conteúdo deste arquivo durante a inicialização do sistema. Desta forma é possível resgatar o conteúdo da memória criado em execuções anteriores do sistema. O módulo de persistência executa como uma tarefa no gerenciador de tarefas no *Framework* LIDA, e a cada 1000 unidades de tempo do sistema a rotina é executada.

Visto que a implementação atual do *Framework* LIDA não realiza a gravação e recuperação de links, uma alteração na implementação da memória episódica foi realizada. A alteração realizada consiste em adicionar os links pertencentes a um nodo recuperado da memória episódica e conceder uma pequena ativação para cada um, desta forma a ativação destes nodos pode ser reforçada no modelo situacional corrente por outros nodos presentes.

4.3.5 Seleção de ação

No módulo de seleção de ação, foram definidos os esquemas de ação como podem ser vistos na Tabela 4.3. Os nodos presentes na coluna *Contexto* representam quais são os nodos que devem estar ativados para que o esquema seja ativado. Os nodos presentes na coluna *Contexto esperado* representam os nodos que espera-se que se ativem depois que a ação seja realizada. A coluna *Ação* representa qual ação deve ser tomada quando o esquema em questão é ativado.

As ações descritas são interpretadas pelo módulo *SensoryMotorMemory* e executadas no ambiente. Neste caso, as ações definidas para execução são enviadas diretamente para o simulador, mas nos experimentos com o robô real, apresentados na continuação deste capítulo, onde há uma integração com o controlador de baixo nível, a *SensoryMotorMemory* envia os comandos para serem executados pelo controlador de baixo nível, que realiza então a ação no robô real.

Nome	**Contexto**	**Contexto esperado**	**Ação**
Ir para objetivo	*nao_esta_objetivo*, *parado*	*esta_objetivo*	action.seguir
Ir para objetivo	*nao_esta_objetivo*, *existe_objetivo*, *existe_caminho*, *sem_colisao_iminente*	*esta_objetivo*	action.seguir
Ir para objetivo	*nao_esta_objetivo*, *existe_objetivo*, *existem_outros_caminhos*	*esta_objetivo*	action.seguir
Aumentar velocidade	*v_lento*, *d_medio*, *seguindo*	*v_medio*	action.aumentar_vel
Aumentar velocidade	*v_medio*, *d_longe*, *seguindo*	*v_rapido*	action.aumenta_vel
Diminuir velocidade	*v_rapido*, *d_medio*, *seguindo*	*v_medio*	action.diminuir_vel
Diminuir velocidade	*v_medio*, *d_perto*, *seguindo*	*v_lento*	action.diminuir_vel
Parar	*colisao_iminente*, *outro_estatico*	*parado*, *seguro*	action.parar
Parar	*colisao_iminente*, *parede*	*parado*, *seguro*	action.parar
Parar	*colisao_iminente*, *humano*	*parado*, *nao_seguro*	action.parar
Parar	*recuando*, *sem_colisao_iminente*	*parado*, *seguro*	action.parar
Parar	*existe_objetivo*, *esta_objetivo*	*parado*, *seguro*	action.parar
Pedir licenca	*colisao_iminente*, *humano*, *parado*	*parado*, *seguro*	action.licenca
Recuar	*colisao_iminente*, *parado*, *d_perto*	*recuando*, *nao_seguro*	action.recuar
Recuar	*parado*, *nao_seguro*	*recuando*, *nao_seguro*	action.recuar
Calcular e seguir novo trajeto	*parado*, *seguro*	*esta_objetivo*	action.novo_trajeto
Explorar	*nao_esta_objetivo*, *nao_existe_caminho*	*existe_caminho*	action.explorar
Explorar	*nao_existe_objetivo*		action.explorar

Tabela 4.3: Esquemas de ação do módulo de seleção de ação do controlador de alto nível.

4.3.6 Validação experimental

Com o objetivo de validar experimentalmente o modelo cognitivo proposto neste trabalho, dois experimentos distintos foram realizados. O primeiro experimento consistiu em avaliar a solução em ambiente de simulação, conforme apresentada a seguir. O segundo experimento consistiu na avaliação da solução em um ambiente real, utilizando um robô móvel para realização dos experimentos, o qual é apresentado em detalhes posteriormente.

Experimento em simulação

O objetivo da validação em ambiente de simulação foi de avaliar preliminarmente o funcionamento do modelo cognitivo proposto, avaliando a capacidade de tomadas de decisão, o funcionamento das estruturas cognitivas como memória perceptual, memória associativa, codelets de percepção, codelets de atenção e a rede de comportamentos. Para tal, o simulador descrito na seção 4.3.1 foi utilizado.

O experimento no simulador foi realizado de forma que diversos cenários foram apresentados ao controlador cognitivo. Um módulo foi desenvolvido para o *Framework* LIDA onde, a partir de um arquivo texto, as percepções eram definidas no simulador. Uma série de combinações das percepções foram definidas neste arquivo, de forma que com o passar do tempo as percepções eram alteradas, simulando alterações no ambiente. No momento em que o LIDA executava uma tomada de decisão esta refletia diretamente nas percepções, desta forma simulando o comportamento de um robô em um ambiente desconhecido. O modelo apresentado anteriormente foi resultado de diversos ciclos de avaliações empíricas, que consistiram na apresentação de diversas situações para o controlador e a validação do comportamento do robô para cada situação. Como o problema proposto se trata de navegação do robô, a principal característica avaliada foi a capacidade do robô escolher o comportamento mais adequado para que ele siga o trajeto ao seu objetivo, evitando colisões. Durante os ciclos de avaliação do robô, foram avaliados quais os *codelets* de atenção e percepção se ativavam, e quais as percepções e os esquemas de ação melhor se adequavam para a situação apresentada, até se chegar no modelo apresentado acima.

Ao chegar em um modelo considerado ideal, um conjunto distinto de percepções foram definidas no simulador para avaliar se o processo de tomada de decisão utilizando o modelo proposto estava apresentando tomadas de decisão coerentes em relação ao ambiente. A seção 4.4 apresenta os resultados obtidos nesta validação.

Experimento com robô real

A validação experimental com um robô real, foi realizada utilizando o robô descrito na seção 4.1.2. Para a realização do controle de baixo nível foi utilizada a plataforma ROS, conforme apresentado na seção 4.2.

O ambiente definido para a realização dos testes foi um corredor da UTFPR com uma pessoa circulando no ambiente. A figura 4.10 apresenta o ambiente em que os experimentos foram realizados.

O ambiente definido para a realização dos experimentos é ilustrado na figura 4.9. O experimento consiste em, dada uma posição inicial, o robô se movimentar e seguir um caminho até o objetivo (o objetivo é o ponto marcado na ilustração com um *X*). O hexágono representa uma pessoa no ambiente, que está em movimento seguindo a trajetória marcada pela linha verde.

Desta forma, para chegar até o objetivo o robô irá se deparar com a pessoa. A avaliação realizada neste experimento consistiu em analisar o comportamento entre a execução do controlador deliberativo-reativo proposto neste trabalho com o comportamento de um controlador reativo, de forma a identificar quais as vantagens de se utilizar o controlador deliberativo-reativo em vista a uma abordagem tradicional, identificando os momentos de tomada de decisão de cada um e qual o esforço empenhado pelos mesmos até atingir o objetivo.

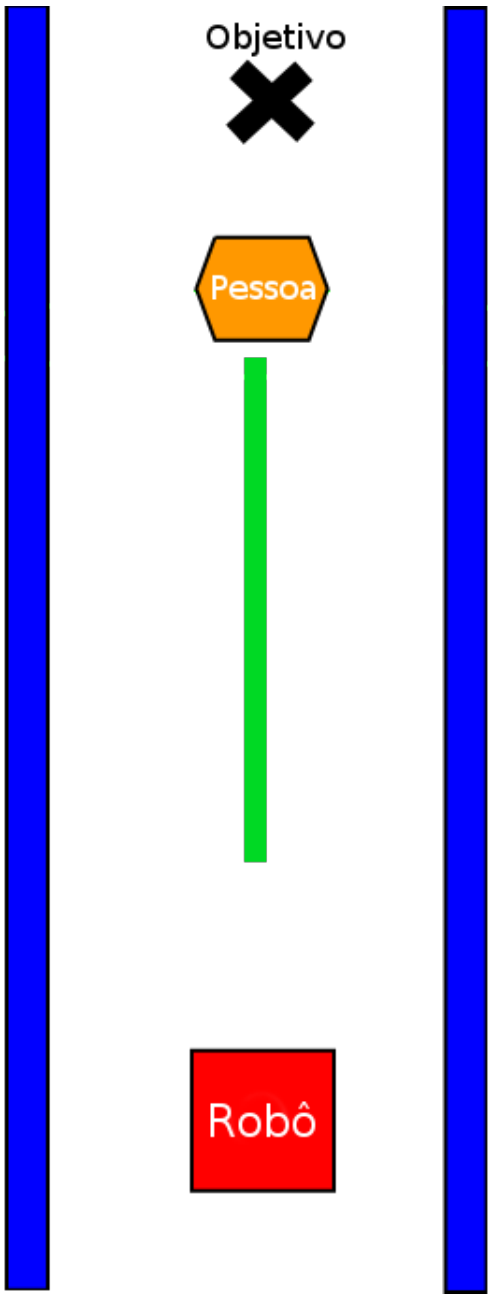


Figura 4.9: Ilustração do ambiente proposto para o experimento.

Para a realização da integração entre o controlador de alto nível com o de baixo nível, primeiramente foi desenvolvido em Java um componente de software que permite a comunicação do LIDA com o sistema ROS. A comunicação é realizada via *WebSockets* e utiliza o

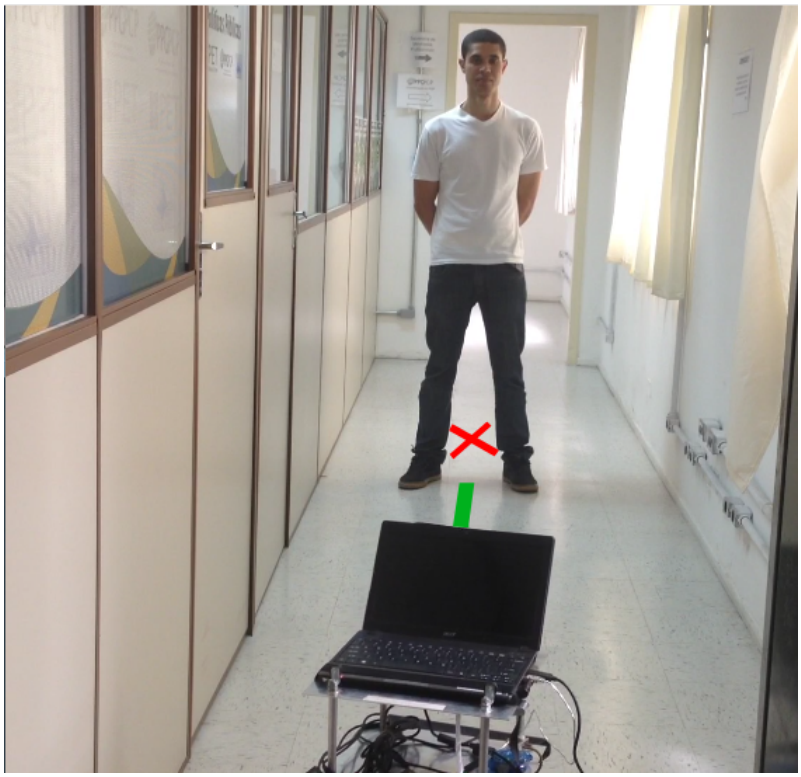


Figura 4.10: Ambiente utilizado para experimentos como robô real.

formado JSON para a troca de informações. Um outro componente foi desenvolvido no lado do ROS, para converter dados sensoriais de baixo nível para que pudessem ser interpretados pelo controlador de alto nível. Este componente, desenvolvido em Python, atua como um nodo do ROS e coleta as informações provenientes do sensoriamento do robô e apresenta para o controlador de alto nível no formato apresentado na seção 4.3.2. Por realizarem uma espécie de ponte entre os dois controladores, eles foram chamados de *Lida Bridge Java* e *Lida Bridge ROS*, respectivamente.

Os experimentos iniciais foram realizados utilizando de forma completa as percepções apresentadas na seção 4.3.2. Durante estes testes identificou-se que as percepções da categoria *Direção robô* não colaboravam para a tomada de decisão pelo controlador de alto nível e apenas estavam gerando uma sobrecarga computacional no sistema, visto que estas percepções foram adicionadas apenas para agregar mais informações para as associações realizadas pela memória episódica. Como estas informações não se mostraram relevantes para a tomada de decisão do controlador de alto nível, as mesmas foram removidas do modelo. Além da direção do robô, outra categoria de percepções que foi removida do modelo nesta fase foram os nodos relacionados a velocidade do robô, bem como as ações relativas a aumentar e diminuir a velocidade. O motivo desta decisão foi porque o controlador de baixo nível já realiza o planejamento de trajetória e movimento do robô, tornando-se redundante um controle de velocidade nos dois módulos. Os nodos das categorias *Tipo Local* e *Local* foram definidos manualmente, dado que o experimento foi realizado em apenas um local e estes parâmetros não sofriam alterações.

Os esquemas de ação também foram simplificados do modelo inicial. Os esquemas de aumentar e diminuir a velocidade foram retirados conforme explicado anteriormente. Os esquemas *Recuar*,*Explorar* e Calcular e seguir novo trajeto foram removidos pois o primeiro

já é executado pelo controlador de baixo nível, e o segundo tornou-se desnecessário para o experimento, visto que o ambiente utilizado para os experimentos foi um corredor fechado e o ato de explorar o ambiente não iria adicionar benefício algum à navegação. A ação de *calcular e seguir novo trajeto* tornou-se implícita dentro da ação de *Ir para objetivo*, na forma em que o sistema foi implementado, a qual ocorre cada vez que um objetivo é enviado para o controlador. A tabela 4.4 apresenta os esquemas de ação que foram utilizados nos experimentos com o robô real.

Nome	**Contexto**	**Contexto esperado**	**Ação**
Ir para objetivo	*nao_esta_objetivo*, *parado*	*esta_objetivo*	action.seguir
Ir para objetivo	*nao_esta_objetivo*, *existe_objetivo*, *existe_caminho*, *sem_colisao_iminente*	*esta_objetivo*	action.seguir
Ir para objetivo	*nao_esta_objetivo*, *existe_objetivo*, *existem_outros_caminhos*	*esta_objetivo*	action.seguir
Parar	*colisao_iminente*, *outro_estatico*	*parado*, *seguro*	action.parar
Parar	*colisao_iminente*, *parede*	*parado*, *seguro*	action.parar
Parar	*colisao_iminente*, *humano*	*parado*, *nao_seguro*	action.parar
Parar	*recuando*, *sem_colisao_iminente*	*parado*, *seguro*	action.parar
Parar	*existe_objetivo*, *esta_objetivo*	*parado*, *seguro*	action.parar
Pedir licenca	*colisao_iminente*, *humano*, *parado*	*parado*, *seguro*	action.licenca

Tabela 4.4: Esquemas de ação do módulo de seleção de ação do controlador de alto nível utilizadas no experimento com robô real.

Algumas das percepções apresentadas na seção 4.3.2, são interpretações de alto nível das informações geradas pelos sensores. Para a realização destas interpretações, técnicas distintas foram utilizadas para a extração de informações a partir dos dados brutos gerados pelos sensores do robô. Para gerar as informações que foram utilizadas pela categoria de percepções *Obstáculos Estáticos*, foi utilizado um algoritmo de segmentação de imagens sobre o mapa gerado pelo controlador de baixo nível, para identificar o tamanho e posição destes obstáculos. Desta forma, cada vez que o controlador de baixo nível atualiza o mapa, o nodo *Lida Bridge ROS* realiza a interpretação dos dados utilizando as técnicas citadas e disponibiliza a informação da quantidade de obstáculos existentes no ambiente para o controlador de alto nível. As percepções da categoria *Obstáculos Móveis* foram geradas utilizando a API OpenNI, que é um *driver* livre para o Kinect, o qual possui ferramentas de detecção e rastreamento de pessoas.

O nodo *Lida Bridge ROS* monitora constantemente as informações provenientes do Kinect, e quando uma pessoa entra no campo de visão do sensor, este a identifica e disponibiliza a informação da quantidade e movimento de obstáculos móveis existentes no ambiente para o controlador de alto nível. Um obstáculo móvel também é identificado como obstáculo estático, pois o mapa gerado pelo controlador de baixo nível e que é utilizado pelo detector de obstáculos estáticos não leva em consideração o movimento dos obstáculos. Quando um obstáculo estático e um móvel estão na mesma posição, apenas o obstáculo móvel é levado em consideração.

A partir das informações dos obstáculos estáticos e móveis, utiliza-se o obstáculo mais próximo do robô para extrair as informações *Tipo do obstáculo mais próximo* e *Distância do obstáculo mais próximo*. O Tipo do obstáculo é identificado de duas formas: se o obstáculo é móvel este é classificado como *outro_movel*, do contrário utiliza-se o seu tamanho para classificar entre *outro_estatico* e *parede*. A percepção *humano* foi removida classificando assim todos os obstáculos móveis da mesma forma.

Durante os primeiros testes com o robô identificou-se uma limitação no Kinect, onde quando uma pessoa estava à uma distância de menos de 1 metro do robô, o mesmo não conseguia realizar a detecção e fornecer estes dados em tempo para o robô. Para contornar este problema, a distância mínima de um obstáculo móvel, para ser considerado mais próximo do robô, foi alterada para 1 metro.

4.4 Resultados

Os experimentos realizados em ambiente de simulação, tiveram como propósito validar o modelo cognitivo em um ambiente mais simples e controlado do que uma aplicação robótica real, a fim de identificar se o comportamento do sistema cognitivo seria coerente com o comportamento esperado. Os resultados destes experimentos apresentaram um resultado satisfatório, pois as tomadas de decisão realizadas pelo sistema em face das situações apresentadas foram coerentes com cada situação, permitindo que o robô realizasse ações que poderiam satisfazer as condições impostas pelo ambiente.

Neste experimento foram avaliadas as capacidades do sistema cognitivo, em seus mecanismos de percepção, atenção, aprendizado episódico e seleção de ação. Avaliando a Tabela 4.3 em relação às percepções definidas na seção 4.3.2, pode-se perceber que existem algumas percepções que não influenciam diretamente na seleção de ação, como é o caso das percepções relacionadas a local, tipo de local, direção do robô e quantidade de obstáculos. Estas percepções foram adicionadas no modelo para que se fosse possível realizar associações mais complexas na memória episódica, inferindo conhecimentos a respeito do ambiente que possam alterar as tomadas de decisão.

Um exemplo da realização destas associações ocorre na situação a seguir, onde a configuração do simulador neste momento define que não há uma colisão iminente e o obstáculo mais próximo é uma parede que está longe do robô (o nó *d_perto* está ativo devido a uma percepção anterior, e seu decaimento ainda não foi total). O estado da PAM pode ser observado na figura 4.11, onde cada círculo representa um nó ativo da PAM, a área destacada apresenta os nós relativos às categorias de tipo de obstáculo e distância do obstáculo mais próximo. A figura 4.13 apresenta os nós que são provenientes da associação encontrada na memória episódica a partir de uma *cue* (consulta realizada na SDM a partir do estado corrente das percepções do ambiente) realizada com o conteúdo da PAM. A área destacada mostra uma associação referente aos nodos *d_perto*, *parede* e *colisao_iminente*. Esta associação não é apenas resultado

da associação dos nós *d_perto* e *parede*, visto que cada *broadcast consciente* produz associações na memória episódica e as associações são buscadas da memória episódica a partir do conteúdo formado por todos os nós ativos na PAM (*percepto*). A união das associações da memória episódica com as percepções da PAM (modelo situacional corrente) pode ser observada na figura 4.12. Como pode ser observado na figura, o nó *colisao_iminente* está ativo, sendo resultado da associação da memória episódica. Esta união do *percepto* com as associações da memória episódica que são identificadas pelos *codelets* de atenção e irão formar coalizões para competir pelo *workspace* global, neste caso os *codelets* de atenção: *AttentionCodelet-01*, *AttentionCodelet-02* e *AttentionCodelet-03* (ver tabela 4.2) irão ativar e criar as coalizões para competir pela consciência.

Os *codelets* de atenção *AttentionCodelet-02* e *AttentionCodelet-03* só irão se ativar devido à associação local que está ativando o nó relativo a uma colisão iminente, se uma coalizão formada por um destes dois *codelets* for a vencedora, a ação *Parar* (ver tabela 4.3) receberá uma ativação maior, aumentando sua chance de escolha dentro do processo de seleção de ação.

Desta forma o controlador desenvolvido utiliza todos os mecanismos da arquitetura cognitiva de Baars e Franklin para a aplicação na navegação de um robô móvel: a percepção, a atenção, o aprendizado episódico e a seleção de ação trabalhando em conjunto no ciclo cognitivo a fim de ter uma melhor eficiência no controle da navegação do robô.

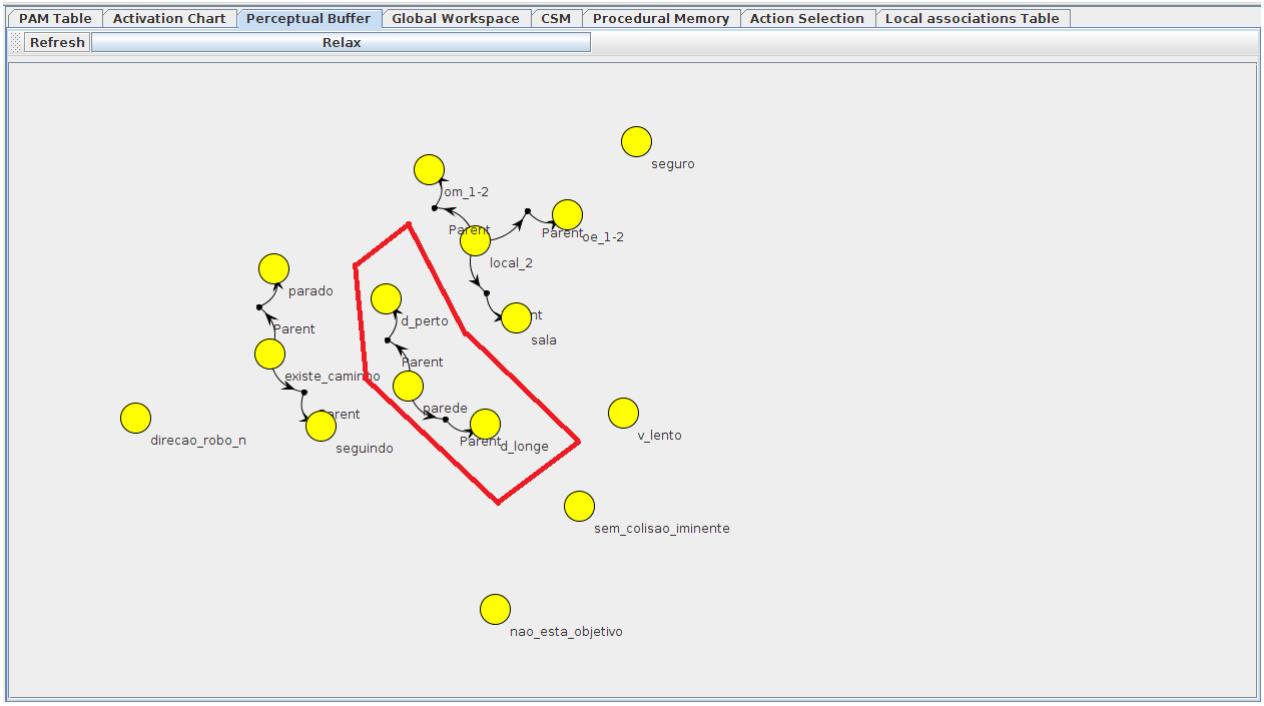


Figura 4.11: Grafo apresentando as percepções presentes no controlador de alto nível em determinada situação.

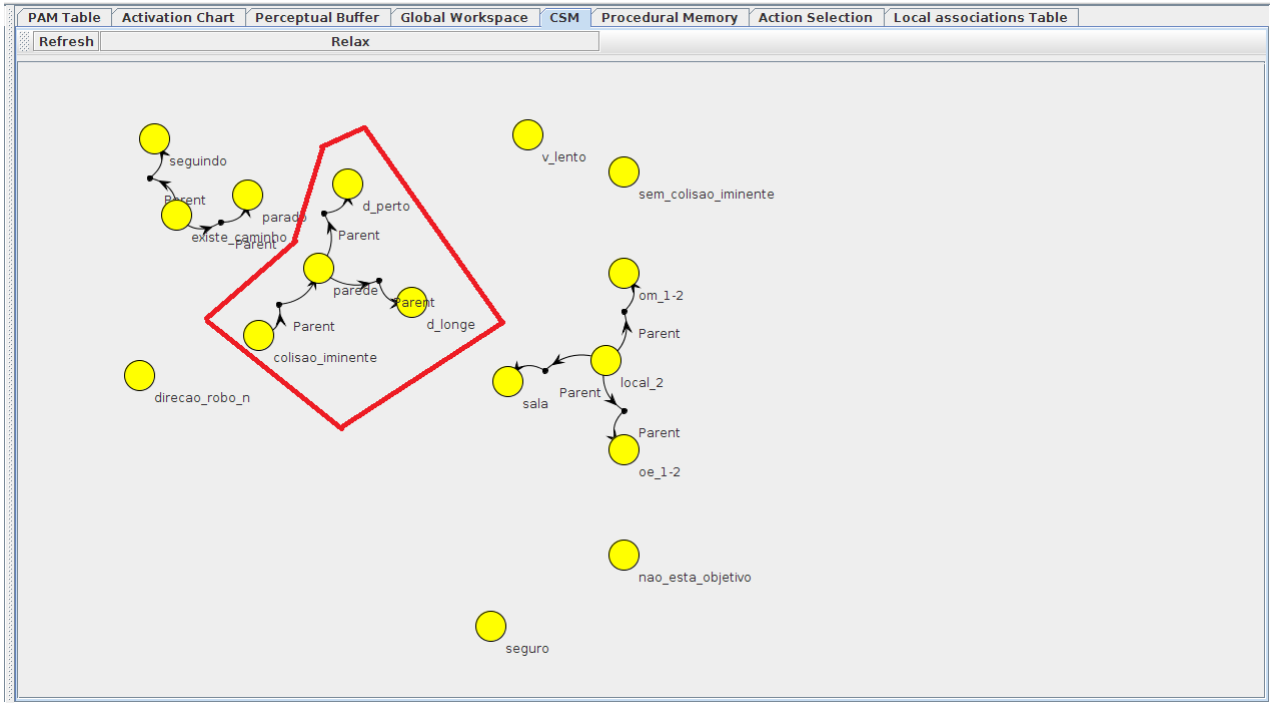


Figura 4.12: Grafo apresentando as percepções presentes no CSM do controlador de alto nível em determinada situação.

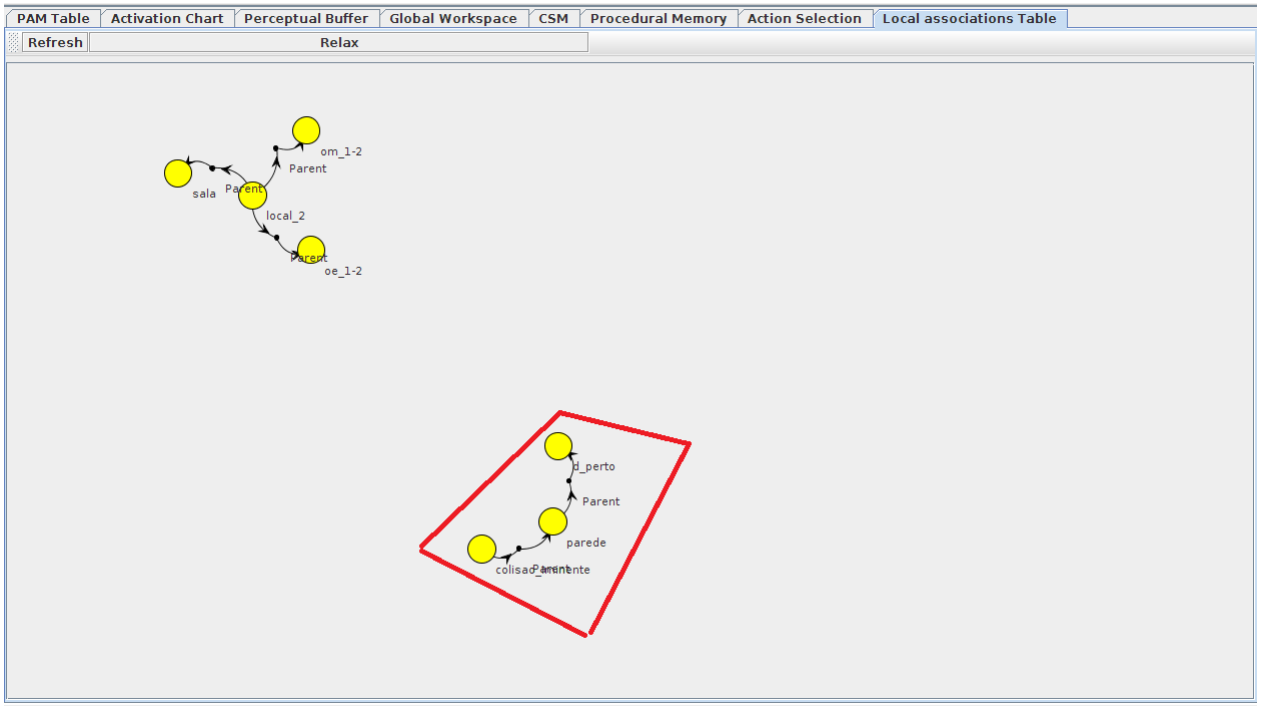


Figura 4.13: Grafo apresentando as associações da memória episódica do controlador de alto nível em determinada situação.

Os experimentos realizados utilizando o robô real, tiveram como propósito avaliar não somente o modelo cognitivo, mas sim a solução híbrida e sua capacidade de atuar em um

ambiente dinâmico real. Este experimento inicialmente utilizou o mesmo modelo utilizado nos testes em simulação, porém no início dos testes foram identificadas alterações necessárias para um melhor funcionamento, conforme descrito na seção 4.3.6. A primeira etapa da avaliação experimental com um robô real, foi a utilização de um controlador de baixo nível, realizando as tarefas de mapeamento e navegação simultâneas, para que fosse possível avaliar as diferenças entre a navegação de um robô móvel por um método tradicional e a navegação utilizando o modelo cognitivo proposto. A segunda etapa desta avaliação experimental, consistiu na utilização do controlador de baixo nível executando em conjunto com o controlador de alto nível, onde o controlador de alto nível possui a responsabilidade de identificar os componentes do ambiente em um formato de mais alto nível e tomar decisões para guiar o controlador de baixo nível.

O comportamento do robô executando a primeira etapa do experimento, realizou-se conforme previsto nas etapas de projeto, de forma que ao se deparar com um obstáculo móvel o algoritmo de navegação identificou o obstáculo como um outro obstáculo qualquer e o inseriu no mapa. Quando o obstáculo ofereceu uma barreira para que se continuasse na trajetória inicial, o controlador alterou a trajetória para uma nova, de forma que evitasse o obstáculo. Nos casos em que a pessoa se opôs ao robô se movimentando de forma que sua trajetória fosse bloqueada, o robô apresentou dois comportamentos distintos em diferentes ambientes: em um ambiente amplo, conforme apresentado na figura 4.14, o robô ficou alterando sua trajetória até que conseguisse encontrar um caminho válido e chegou ao objetivo; no experimento realizado em um ambiente menor, conforme a figura 4.10, o robô não conseguiu desviar e acabou por colidir com o obstáculo.

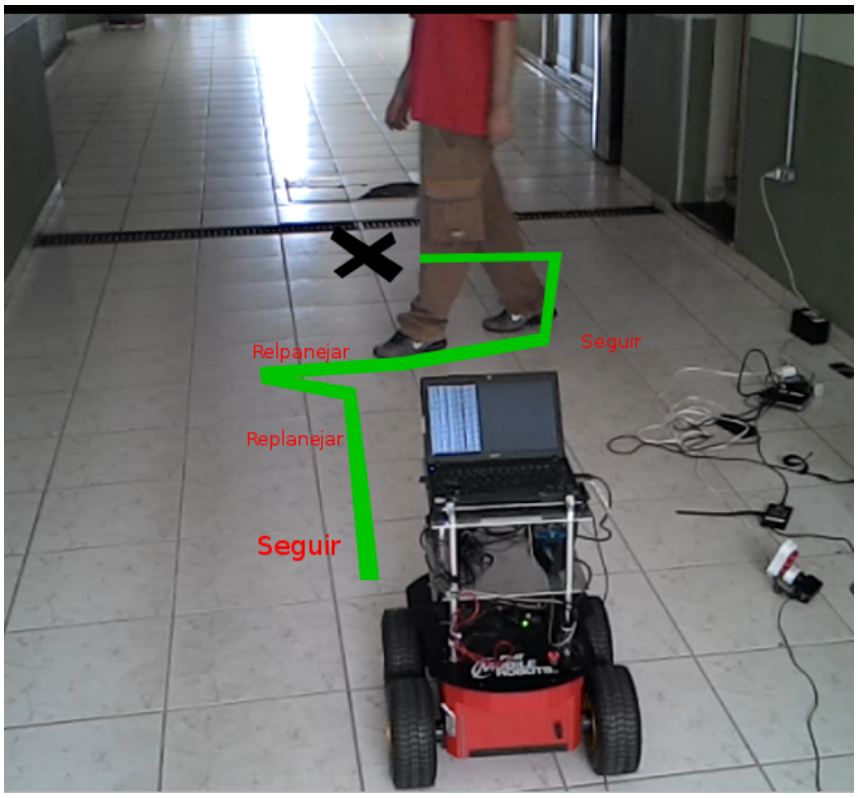


Figura 4.14: Ilustração das ações tomadas pelo controlador reativo.

Com a utilização do modelo cognitivo de alto nível coordenando as ações do controlador de baixo nível, o comportamento ótimo era que, ao se deparar com um obstáculo móvel, o robô deveria parar sua trajetória (o controlador de alto nível envia uma nova trajetória ao robô, sendo esta o próprio lugar onde o robô está) e em seguida pedir licença para a pessoa em sua frente, quando a pessoa se afastasse do robô, o mesmo recalcularia a trajetória e retomaria o caminho. A execução dos testes apresentou o resultado esperado, porém não em todos os casos. Tendo-se executado diversos testes, em 70% dos casos o robô realmente executou todo o comportamento ótimo, quando este comportamento não foi executado, o robô continuou o trajeto de forma que as ações tomadas foram as mesmas do controlador reativo.

Duas situações ocorreram para o robô não executar o comportamento ótimo: a falta de desempenho computacional do sistema, e o processo de detecção de pessoas realizado pelo Kinect. No ambiente de simulação utilizado, a carga computacional no sistema também foi alta, porém para a realização dos experimentos de avaliação do modelo cognitivo o tempo de execução de todo o sistema foi compartilhado, desta forma as ações do controlador de alto nível ocorreram de forma sincronizada com o controlador de baixo nível e as simulações do ambiente. Com o robô real isto não ocorre, visto que em um ambiente real a dinamicidade do ambiente e a movimentação do próprio robô permanecem constantes, enquanto o processamento de tomada de decisão é dependente do sistema computacional. Nos casos onde a falta de desempenho influenciou diretamente o resultado das tomadas de decisão, as ações do mecanismo de seleção de ação foram coerentes com o modelo do ambiente que estava no modelo situacional corrente, porém algumas vezes estas não estavam mais coerentes com o ambiente real em que o sistema estava inserido. As tomadas de decisão são influenciadas diretamente pelo processo de competição que ocorre pela consciência, e novamente pelo processo de competição que ocorre na rede de comportamentos. Estes processos de competição com frequência atrasaram os processos de tomada de decisão, que não conseguiu realizar em tempo hábil a decisão correta. Quando a decisão foi tomada incorretamente e o robô não parou ou pediu licença para a pessoa em frente, o ambiente já havia se alterado e outro processo de tomada de decisão iniciado, muitas vezes repetindo a problemática. Durante os testes, quando esta situação ocorreu, o robô não alterava sua decisão para parar ou pedir licença, e acabava seguindo o caminho e tomando as mesmas ações que o controlador de baixo nível. O principal motivo da carga computacional excessiva, foi devido aos acessos à SDM (o cálculo para recuperar as associações deveria ser feito em cada um dos 54 bits da palavra e em 10000 *hardlocations*, totalizando 540000 cálculos a cada acesso). Outro ponto que acarreta atrasos no processo de tomada de decisão foi a complexidade do modelo em relação aos esquemas de ação. Presume-se que em um modelo cognitivo mais minimalista em todos os processos (codelets de atenção, nodos da PAM e esquemas de ação), a competição pela consciência e pelas tomadas de decisão poderia ser menos custosa.

O segundo motivo que acarretou falhas no processo de decisão foi o processo de detecção de pessoas realizado pelo Kinect. Para a identificação e rastreamento de pessoas foi utilizado o driver OpenNI em conjunto com o pacote ROS openni_tracker. Esta solução, de modo geral, realizou a detecção de pessoas as quais estavam dentro da área de visualização do Kinect. Porém, em alguns casos, a detecção não ocorreu ou o rastreamento não ocorreu de forma correta. Uma das limitações encontradas nesta solução de rastreamento de pessoas ocorre quando uma pessoa sai do campo de visão do Kinect pelas laterais. O módulo openni_tracker continua enviando ao sistema ROS a última posição na qual a pessoa foi detectada, de forma que, quando uma pessoa sai pela lateral, mesmo que esta já esteja longe do robô, a interpretação do sistema é de que ainda existe um humano próximo. Para contornar este problema, nos

testes evitamos que a pessoa saísse do campo de visão do Kinect ao se afastar do robô. Outra limitação é o problema da iluminação. Quando utilizado em um ambiente exposto a luz solar, o funcionamento do Kinect torna-se algumas vezes imprevisível (criando obstáculos inexistentes ou não detectando obstáculos), pois como seu funcionamento baseia-se em luz infra-vermelha, a luz infra-vermelha proveniente do sol atrapalha o funcionamento do dispositivo.

Em alguns casos, o robô parou e deixou de pedir licença, ou pediu licença sem parar seu movimento. Foram casos em que ”lembranças passadas” provenientes das associações realizadas pela SDM vieram à tona e ocasionaram um erro nas tomadas de decisão. O comportamento nestes casos era esperado, também ocorrendo no ambiente de simulação, porém quando uma associação da SDM apresentava uma percepção antiga que não refletia a realidade da situação, ela logo era recuperada e a ação correta era tomada, ou reforçada em caso a memória fosse coerente com a situação. Porém com o baixo desempenho da solução, esta demora para se recuperar e realizar a ação correta foi muito grande, acarretando uma alteração no ambiente antes da tomada de decisão, que por sua vez quando era tomada o ambiente já havia se modificado, tornando assim a decisão errada para a situação.

4.4.1 Comparação dos resultados dos experimentos

Esta seção apresenta uma análise dos resultados entre o comportamento do controlador cognitivo com o comportamento onde o robô executou apenas o controlador de baixo nível (reativo). A figura 4.15 apresenta as tomadas de decisão e o caminho seguido pelo controlador reativo durante o experimento. A figura 4.17 apresenta as tomadas de decisão e o caminho seguido pelo controlador deliberativo na situação em que o mesmo realizou as tomadas de decisão nas condições ótimas.

Analisando as duas imagens, nota-se claramente que a trajetória utilizada pelo controlador reativo é mais custosa, pois o mesmo a altera diversas vezes ao se deparar com um obstáculo em movimento, pois o mapa utilizado para a realização do planejamento de trajetória não leva em consideração a movimentação de um obstáculo. No ambiente específico, como o robô não possuía muito espaço disponível para uma trajetória onde um desvio era possível, o mesmo colidiu com o obstáculo e em alguns casos, colidiu com a parede, como mostra a figura 4.16, devido a limitações na distância mínima de detecção do sensor (Kinect). Em contrapartida, o controlador deliberativo, sendo executado no topo do controlador reativo, altera suas políticas de tomada de decisão e, mesmo mantendo os mesmos algoritmos de planejamento de trajetória, as ações tomadas de parar e interagir com a pessoa presente no ambiente acarreta uma trajetória muito mais simples e com uma menor possibilidade de colisão. A figura 4.18 apresenta a interface do *Framework* LIDA, apresentando as decisões tomadas durante a interação com uma pessoa. Nos casos onde o robô não tomou as decisões esperadas, o planejamento e a execução da trajetória foram no pior dos casos iguais aos do controle reativo. A figura 4.19 apresenta uma representação do mapa gerado pelo SLAM, e a linha em verde representa o planejamento da trajetória a ser executada. A figura 4.20 apresenta o momento onde uma colisão com a parede ocorre durante a execução dos experimentos com apenas o controlador reativo, e nota-se que neste momento a pessoa na frente do robô é representada no mapa como um obstáculo estático, desta forma o robô tenta desviar do mesmo gerando a colisão. Estas imagens foram geradas pelo visualizador *rviz*, o qual é parte integrante do ROS.

O controlador deliberativo-reativo desenvolvido apresentou problemas quanto ao desempenho da solução, tomando decisões diferentes das esperadas em alguns momentos. Porém

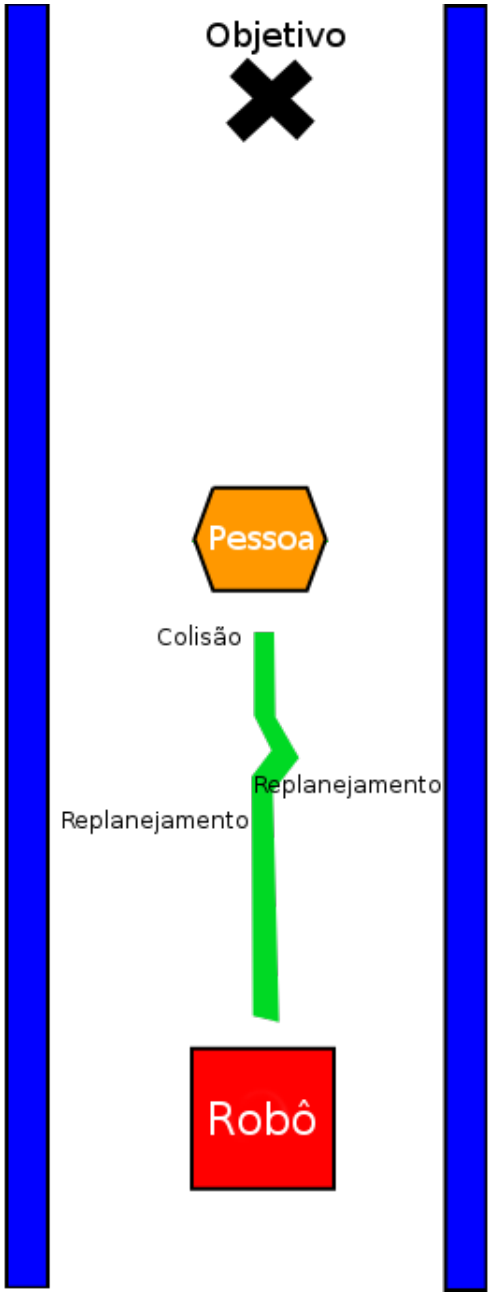


Figura 4.15: Ilustração das ações tomadas pelo controlador reativo.

quando o controlador de alto nível tomou decisões aquém do esperado, o controlador de baixo nível executou o trajeto de forma equivalente aos experimentos com apenas o controlador reativo. A proposta de um controlador em dois níveis visa justamente a possibilidade de o nível mais baixo ter a capacidade de tomar decisões mais rápidas do que o de alto nível, existindo desta forma uma solução mais flexível e robusta.

Nos experimentos realizados um ambiente com espaço suficiente para que o robô pudesse realizar mais manobras e alterações na trajetória para desviar do obstáculo com facilidade, o controlador deliberativo-reativo em comparação com o controlador reativo apresentou vantagens em relação a realizar uma trajetória mais fluída, visto que ao invés de realizar o processo

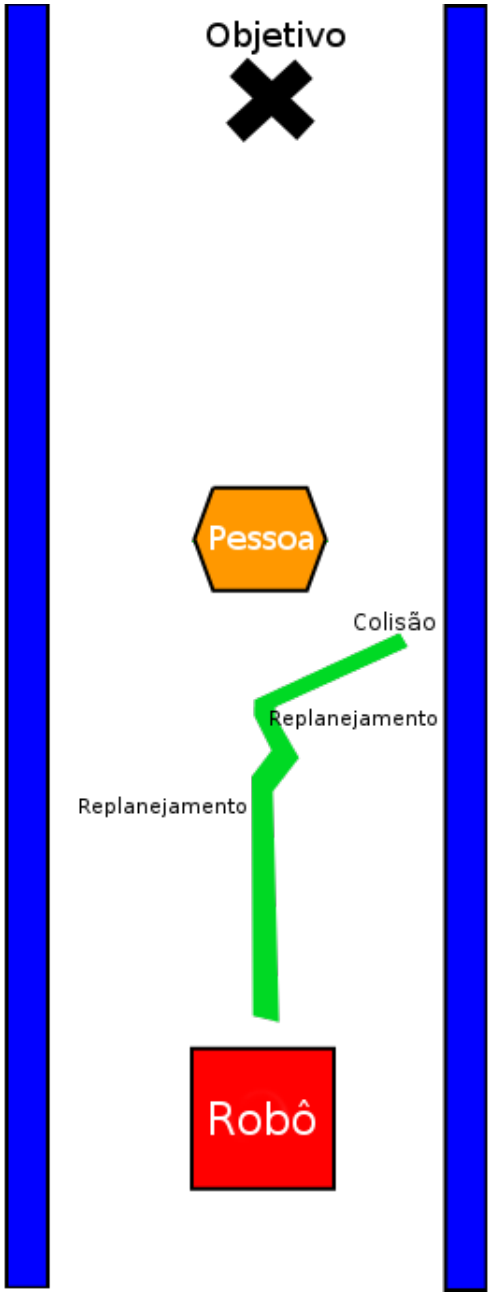
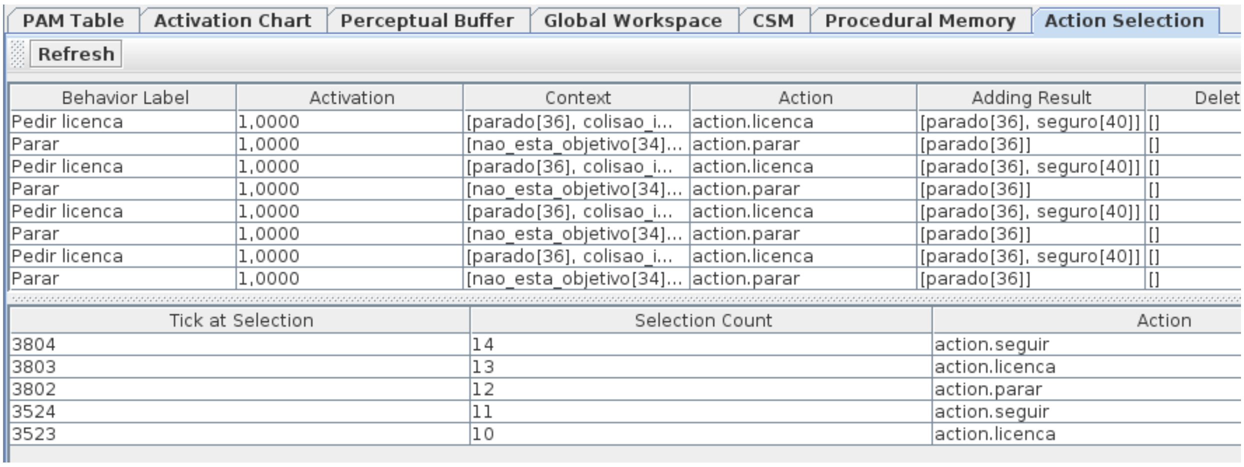


Figura 4.16: Ilustração das ações tomadas pelo controlador reativo.

de buscar uma nova trajetória, o mesmo interage com o obstáculo (se for uma pessoa) assim liberando o caminho para o objetivo.

A solução deliberativo-reativo apresentou maior vantagem em relação a solução reativa no caso onde o ambiente de experimentação possuía um espaço mais limitado, de forma que quando uma ação do obstáculo é essencial para que o robô pudesse chegar ao objetivo, a abordagem cognitiva foi capaz de interagir com o obstáculo para que este pudesse tomar uma ação. Neste cenário a solução com o controlador reativo, que não possui a capacidade de interação e identificação de percepções de alto nível no ambiente, não conseguiu chegar ao objetivo, colidindo com o obstáculo móvel ou a parede em todos testes.

Figura 4.17: Ilustração das ações tomadas pelo controlador deliberativo.



Behavior Label	Activation	Context	Action	Adding Result	Delet
Pedir licenca	1,0000	[parado[36], colisao_i...	action.licenca	[parado[36], seguro[40]]	[]
Parar	1,0000	[nao_esta_objetivo[34]...	action.parar	[parado[36]]	[]
Pedir licenca	1,0000	[parado[36], colisao_i...	action.licenca	[parado[36], seguro[40]]	[]
Parar	1,0000	[nao_esta_objetivo[34]...	action.parar	[parado[36]]	[]
Pedir licenca	1,0000	[parado[36], colisao_i...	action.licenca	[parado[36], seguro[40]]	[]
Parar	1,0000	[nao_esta_objetivo[34]...	action.parar	[parado[36]]	[]
Pedir licenca	1,0000	[parado[36], colisao_i...	action.licenca	[parado[36], seguro[40]]	[]
Parar	1,0000	[nao_esta_objetivo[34]...	action.parar	[parado[36]]	[]

Tick at Selection	Selection Count	Action
3804	14	action.seguir
3803	13	action.licenca
3802	12	action.parar
3524	11	action.seguir
3523	10	action.licenca

Figura 4.18: Interface do LIDA Framework no momento de tomada de decisão de interação com a pessoa.

Com base nos resultados obtidos nestes experimentos, nota-se que o comportamento do controlador deliberativo-reativo desenvolvido neste trabalho demonstra-se superior a uma abordagem tradicional do ponto de vista de possuir uma maior capacidade de se adaptar a situações mais complexas, como no caso do corredor estreito. Por outro lado, o custo computacional

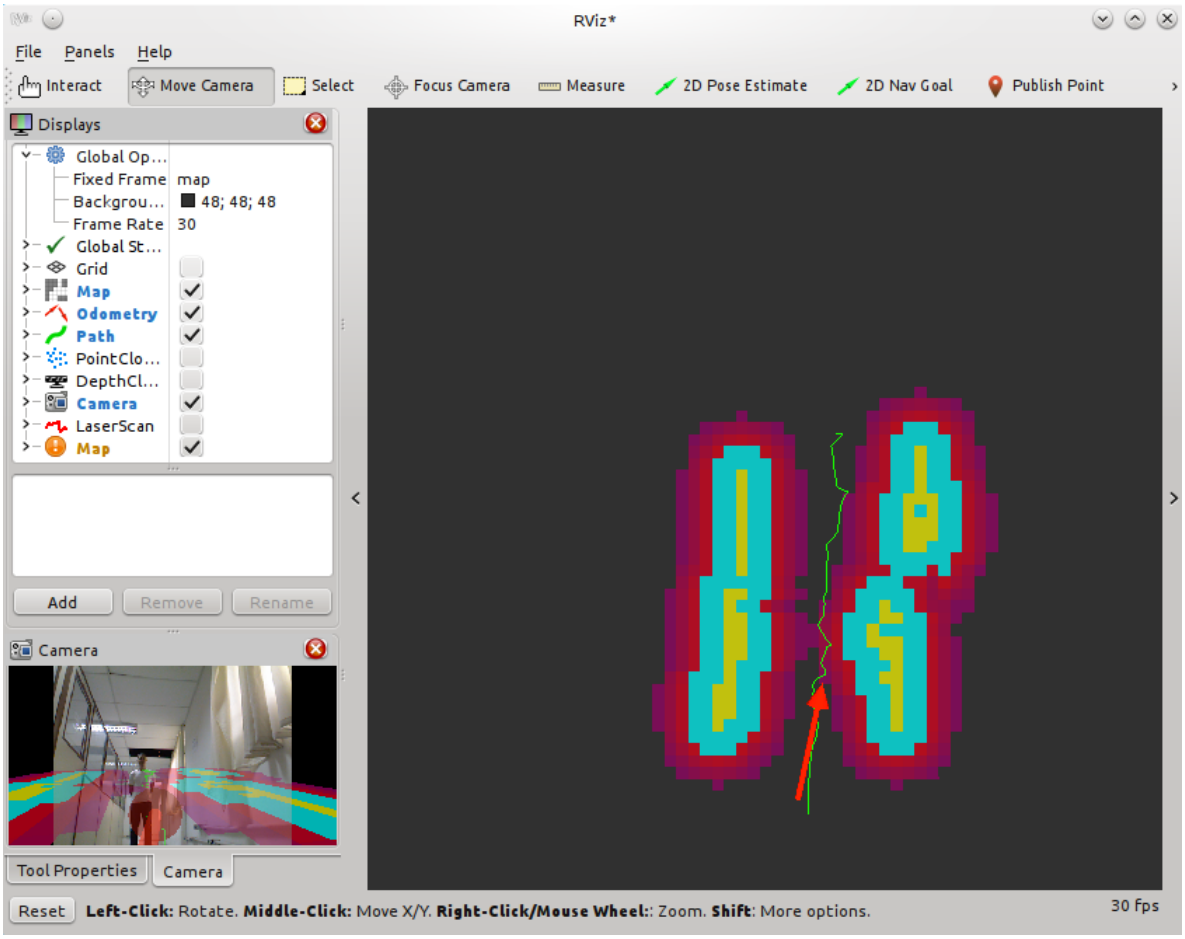


Figura 4.19: Mapeamento e trajetória do robô durante a execução do experimento.

da solução apresentada foi muito superior ao necessário para a execução de uma solução reativa, de forma que surge a necessidade de hardware mais potente para a execução desta solução. Outro ponto é a necessidade de uma modelagem concisa do problema, por que o *Framework* LIDA, apesar de ter diversos mecanismos de aprendizado em seu modelo conceitual, ainda não possui uma implementação dos mesmos, ficando a cargo do projetista uma modelagem fina do problema. Neste trabalho o foco foi a interação com uma pessoa presente no ambiente, porém para uma aplicação mais complexa este modelo tende a crescer e necessitar de um esforço ainda maior do projetista em comparação a abordagem tradicional.

4.5 Conclusão

Neste capítulo foi explicado o modelo de percepções, *codelets* de atenção e esquemas de ação do sistema cognitivo proposto. Nele também foram apresentados os experimentos propostos para a avaliação da solução, bem como os resultados obtidos durante a realização destes experimentos. No próximo capítulo, será apresentada a conclusão da pesquisa e os trabalhos futuros.

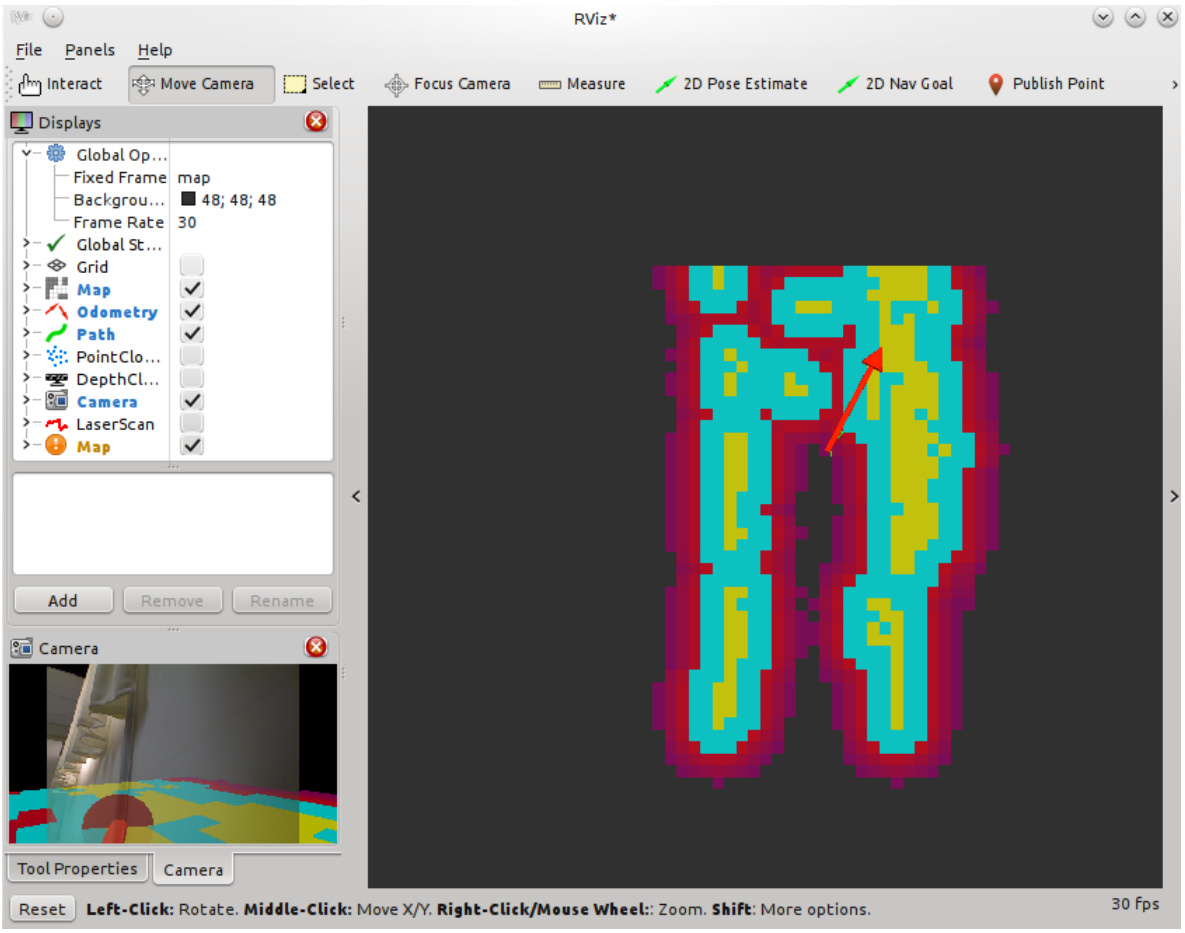


Figura 4.20: Mapeamento e trajetória do robô durante a execução do experimento no momento em que ocorre uma colisão com a parede.

Capítulo 5

Conclusão e Trabalhos Futuros

Este capítulo apresenta as conclusões obtidas, contribuições realizadas com o desenvolvimento deste trabalho, as limitações encontradas durante o desenvolvimento, bem como a identificação de possíveis trabalhos futuros tendo em vista os resultados obtidos.

5.1 Considerações finais

Este trabalho apresentou um estudo sobre modelos de consciência artificial e suas aplicações no problema da navegação de robôs móveis. Inicialmente realizou-se a discussão sobre o problema da navegação de robôs móveis em ambientes dinâmicos, delimitando quais são as abordagens tradicionais de resolução deste problema. Definiram-se os conceitos fundamentais sobre consciência artificial, os quais foram a base para o desenvolvimento do agente cognitivo.

O abordagem proposta para a navegação cognitiva baseia-se na arquitetura Baars-Franklin e sua implementação computacional (o *Framework* LIDA). O modelo de consciência artificial desenvolvido constitui o controlador deliberativo, o qual consiste na utilização do *Framework* LIDA para definir um agente cognitivo com a capacidade de interpretar os sinais provenientes dos sensores de um robô móvel (em termos de um modelo de percepções de alto nível do ambiente), e a partir destas percepções de alto nível tomar decisões relativas ao comportamento do robô. Executando em conjunto com o controlador deliberativo, foi utilizado um controlador reativo, responsável por receber os comandos do controlador deliberativo e executar as ações no robô móvel, bem como disponibilizar os dados provenientes do sistema de sensoriamento do robô para o controlador deliberativo. Este sistema de controle em dois níveis forma assim um controlador híbrido, que foi aqui denominado de controlador deliberativo-reativo.

Este sistema foi posto em contraste com um sistema de controle sem a camada deliberativo/cognitiva, e assim apenas sendo executado o mapeamento do ambiente e controle de trajetória baseado nas informações sensoriais, o qual é uma abordagem tradicional para controle de robôs móveis autônomos. Nos experimentos apresentou-se a mesma situação para um robô executando cada um dos controladores, e a partir dos resultados obtidos nestes experimentos identificou-se que a abordagem cognitiva (controlador deliberativo-reativo) apresentou comportamentos superiores ao controlador reativo. O principal comportamento implementado no controlador deliberativo-reativo foi o de interagir com uma pessoa que estava obstruindo o caminho do robô. No ambiente utilizado para o experimento esta obstrução de caminho ocasionou que o controlador reativo colidia com o obstáculo durante a tentativa de encontrar uma

trajetória para chegar ao objetivo. Em contrapartida, o controlador proposto neste trabalho, utilizando as percepções do ambiente, foi capaz de selecionar o comportamento de interagir com A pessoa, até que a mesma desobstruísse o caminho, conseguindo passar e chegar ao objetivo.

Algumas limitações foram encontradas durante a realização dos experimentos, tanto em relação ao sistema cognitivo quanto em relação ao hardware utilizado no dispositivo. O *Framework* LIDA é uma implementação do modelo conceitual LIDA, implementação esta que ainda está incompleta em relação ao seu modelo conceitual. Um dos pontos chave que ainda está faltando é a questão do aprendizado procedural e perceptual, de forma que as ações e as percepções devem ser definidas manualmente durante o projeto do agente cognitivo. Outro ponto é a questão do desempenho computacional. O LIDA possui, desde sua concepção, uma natureza paralela de execução de seus componentes, incluindo a competição dos codelets e o módulo de memória esparsa distribuída, o que é implementado computacionalmente por meio de *threads*. Entretanto, o grande volume de dados sendo tratados durante a execução dos experimentos acabou acarretando alguns atrasos no tempo de resposta do sistema de controle deliberativo. Algumas vezes, estes atrasos inclusive acabaram por afetar o processo de tomada de decisão do agente. Porém, como o controlador foi desenvolvido em duas camadas, sendo a de baixo nível comandada pela de alto nível, em situações críticas e sem uma decisão tomada pelo controlador deliberativo em tempo hábil, o controle reativo teve a capacidade de continuar a tarefa através dos algoritmos do controle reativo.

A limitação identificada em relação ao hardware do robô foi relativa ao sensor Kinect, onde foi utilizado o pacote openni_tracker para rastrear pessoas no ambiente, e esta funcionalidade apresentou alguns problemas, muitas vezes não identificando as pessoas no ambiente devido a interferência causada pelas variações de iluminação (incidência de luz solar, sombras e outras variações). Para que a aplicação possa ser executada plenamente em ambientes com interferências, considera-se após este trabalho que seja mais adequada a utilização de um sensor mais robusto, como por exemplo um sensor laser.

O desenvolvimento deste trabalho resultou em um controlador híbrido para navegação de um robô móvel, o qual apresentou resultados positivos de modo a validar que a abordagem utilizando técnicas de consciência artificial pode ser aplicada e oferece benefícios frente a navegação tradicional utilizando *SLAM*. Algumas limitações foram determinadas durante o desenvolvimento e sugestões de melhorias identificadas para trabalhos futuros. A área da consciência artificial é um campo recente na pesquisa da inteligência artificial e tem ainda um grande caminho a ser percorrido até seu amadurecimento, mas conclui-se que a robótica móvel tem muito a se beneficiar destas técnicas, visando o aprimoramento dos processos de tomada de decisão por meio de sistemas cognitivos.

5.2 Contribuições

A arquitetura Baars-Franklin foi aplicada no desenvolvimento de um controlador em dois níveis para um robô móvel autônomo. O desenvolvimento deste controlador fez uso de ambientes de simulação e de um ambiente físico com um robô real.

As principais contribuições desse trabalho são:

- realização de uma revisão bibliográfica sobre modelos cognitivos conscientes e suas implementações no âmbito da robótica móvel;

- realização de um estudo teórico sobre a arquitetura Baars-Franklin e suas implementações IDA e LIDA;
- proposta e desenvolvimento de um controlador em dois níveis para a navegação de um robô móvel, utilizando um paradigma de consciência artificial associado às técnicas tradicionais de navegação autônoma;
- realização de ensaios experimentais de um robô utilizando este controlador, tanto em ambiente de simulação como em ambiente real;
- execução de um estudo em uma área ainda pouco explorada, que ofereceu um ganho de experiência e aprendizado para o PPGCA/DAINF/UTFPR, de forma que abriu-se caminho para que possam ser realizadas mais pesquisas no campo da computação cognitiva e robótica móvel.
- publicação de um artigo no congresso internacional ICARSC 2015 - The IEEE International Conference on Autonomous Robot Systems and Competitions, entitulado "Adding Conscious Aspects in Virtual Robot Navigation through Baars-Franklin's Cognitive Architecture", realizado em Vila Real, Portugal.
- publicação de um capítulo no livro "Robot Operating System (ROS): The Complete Reference (Volume 1) (Studies in Computational Intelligence) 1st Edition", 2016, entitulado "LIDA Bridge-A ROS Interface to the LIDA (Learning Intelligent Distribution Agent) Framework".

5.3 Limitações

A arquitetura Baars-Franklin apresenta um modelo de consciência artificial, o qual tem a capacidade de solucionar problemas computacionais de naturezas distintas. Um dos processos cruciais de seu mecanismo de tomada de decisão é a competição pela "consciência" de diversos processos internos. Esta competição possui natureza paralela, entretanto sua implementação em um ambiente computacional sequencial (baseado em arquiteturas tradicionais sequenciais) pode acarretar problemas de desempenho. Em uma aplicação puramente computacional (como é o caso das simulações) esta limitação não acarreta em perdas consideráveis, pois com a solução e o problema estando inseridos completamente no mesmo ambiente, os atrasos são sincronizados, e a solução é coerente, o que não ocorre em um ambiente onde parte do modelo está em ambiente computacional e parte não está (como o mundo real). Em um ambiente de robótica móvel, onde a solução computacional interage fisicamente com o ambiente real, onde os "processos' ou ações do mundo real ocorrem de forma completamente paralela, esta limitação se mostra de forma muito mais importante.

Em relação à capacidade de solução de problemas da arquitetura, ela apresenta um alto grau de generalização. O LIDA *Framework*, sendo uma implementação computacional genérica para o desenvolvimento de agentes utilizando o modelo cognitivo LIDA, herda estas características de seu modelo conceitual. Sendo um modelo genérico para resolução de problemas, isto acarreta a exigência de um grande esforço na fase de projeto do agente cognitivo, sendo que as especificidades do projeto devem ser ajustadas aos moldes de um problema geral. Em relação ao controlador desenvolvido durante este trabalho, os principais problemas em sua

execução estiveram ligados diretamente com o problema de desempenho acima descrito. Uma evolução deste sistema pode dar-se através da utilização de um hardware que permita um maior grau de paralelismo, e por um refinamento do projeto do agente cognitivo (codelets, percepções e esquemas de ação).

O processo de aprendizado do modelo conceitual LIDA prevê questões de aprendizado de máquina nos mecanismos de memória episódica, memória procedural e memória perceptual, ainda que os dois últimos não possuem uma implementação computacional já desenvolvida. A implementação dos algoritmos necessários para a realização destes mecanismos de aprendizado apresentará um avanço imenso nas capacidades da arquitetura, visto que realizando um processo de aprendizado na definição de percepções e ações do agente cognitivo, o projeto do agente pode ser altamente simplificado e a evolução do agente com suas experiências se aproximará muito mais dos processos biológicos, nos quais a arquitetura foi inspirada.

O sensor Kinect utilizado no robô apresentou uma grande limitação, pois a interferência causada pela luz solar atrapalha seu funcionamento, visto que o funcionamento do Kinect se dá pela emissão e recaptura de luz infra-vermelha. O *driver* utilizado na aplicação (OpenNi) em conjunto com o pacote openni_tracker do ROS, não são eficientes no rastreamento de pessoas, muitas vezes perdendo a referência da pessoa presente, e não removendo as mensagens de detecção quando uma pessoa sai do alcance do sensor.

5.4 Trabalhos futuros

Com base nos resultados e realizações deste estudo, pode-se sugerir alguns trabalhos futuros:

- o desenvolvimento dos mecanismos de aprendizado perceptual e aprendizado procedural, os quais fazem parte do modelo conceitual LIDA, porém ainda não possuem uma implementação em seu *framework* computacional;
- a aplicação do *Framework* LIDA em um ambiente que suporte paralelismo real, para que os codelets possam realizar sua execução da forma a qual foi concebida, ao invés da utilização de *threads* para sua execução;
- o desenvolvimento de uma "Memória Esparsa Distribuída" para ser executada em um *Field Programmable Gate Array* (FPGA), de forma que seu acesso se realize de forma paralela;
- a implementação dos módulos de meta-cognição e emoções, propostos no modelo conceitual LIDA;
- a evolução do agente desenvolvido neste trabalho, por meio do desenvolvimento de mais codelets, percepções e ações para execução em outros ambientes.

Referências Bibliográficas

[Arkin and Balch, 1997] Arkin, R. C. and Balch, T. (1997). Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189.

[Baars, 1988] Baars, B. J. (1988). *A cognitive theory of consciousness*. Cambridge University Press.

[Baars, 1997] Baars, B. J. (1997). *In the Theater of Consciousness: The Workspace of the Mind*. Oxford University Press.

[Baars and Franklin, 2007] Baars, B. J. and Franklin, S. (2007). An architectural model of conscious and unconscious brain functions: Global workspace theory and ida. neural networks. *Elsevier Neural Networks*, pages 955–961.

[Baars and Franklin, 2009] Baars, B. J. and Franklin, S. (2009). Consciousness is computational: The lida model of global workspace theory. *International Journal of Machine Consciousness*.

[Baars, 2003] Baars, Bernard J. Franklin, S. (2003). How conscious experience and working memory interact. *Trends in Cognitive Sciences*, 7(4):166–172.

[Barsalou, 1999] Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences*, (22):577–660.

[Belbachir et al., 2013] Belbachir, A., Boutteau, R., Merriaux, P., Blosseville, J.-M., and Savatier, X. (2013). From autonomous robotics toward autonomous cars. In *Intelligent Vehicles Symposium*, pages 1362–1367. IEEE.

[Bennewitz, 2004] Bennewitz, M. (2004). *Mobile Robot Navigation in Dynamic Environments*. PhD thesis, Fakultät für Angewandte Wissenschaften der Albert-Ludwigs-Universität.

[Block, 2002] Block, N. (2002). Some concepts of consicousness. In CHALMERS, D., editor, *Philosophy of Mind: Classical and Contemporary Readings.* Oxford University Press.

[Bogner, 1999] Bogner, M. B. (1999). *Realizing "Consciousness"in Software Agents*. PhD thesis, The University of Memphis. AAI9949957.

[Brooks, 1985] Brooks, R. A. (1985). A robust layered control system for a mobile robot. Technical report, Cambridge, MA, USA.

[Brooks, 1991] Brooks, R. A. (1991). Intelligence without reason. In *COMPUTERS AND THOUGHT, IJCAI-91*, pages 569–595. Morgan Kaufmann.

[Bryson, 2007] Bryson, J. J. (2007). Mechanisms of action selection: Introduction to the special issue. *Adaptive Behaviour*, 15(1):5–8.

[Burch et al.,] Burch, C. et al. A survey of machine learning. Technical report, Tech. report, Pennsylvania Governor's School for the Sciences, 2001. 4.

[Burgard et al., 1998] Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1998). The interactive museum tour-guide robot. In *Fifteenth National Conference on Artificial Intelligence*.

[Capitanio, 2009] Capitanio, R. (2009). Análise da arquitetura baars-franklin de consciência artificial aplicada a uma criatura virtual. Master's thesis, Faculdade de Engenharia Elétrica e de Computação - Universidade Estadual de Campinas, Campinas - SP.

[Conway, 2001] Conway, M. A. (2001). Sensory-perceptual episodic memory and its context: autobiographical memory. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 356:1375–1384.

[Crick, 1994] Crick, F. (1994). *The Astonishing Hypothesis: The scientific search for the soul*. Simon and Schuster.

[Crick and Koch, 2003] Crick, F. and Koch, C. (2003). A framework for consciousness. *Nature Neuroscience*, 6(2).

[da Silva and Gudwin, 2009] da Silva, R. C. M. and Gudwin, R. R. (2009). Developing a consciousness-based mind for an articial creature. In *Advances in Artificial Intelligence – SBIA 2010*, pages 122–132. Springer Berlin Heidelberg.

[Daniela et al., 2011] Daniela, L., Barrera, D., and Franklin, S. (2011). Robot localization using consciousness. *Journal of Pattern Recognition Research*, 1:96–119.

[Dawkins, 1976] Dawkins, R. (1976). *The Selfish Gene*. Oxford University Press.

[Dennet, 1991] Dennet, D. (1991). *Consciousness Explained*. Allen Lane, The Penquim Press.

[Denning, 1989] Denning, P. J. (1989). Sparse distributed memory. *American Scientist*, 77:333–335.

[Drescher, 1991] Drescher, G. (1991). *Made Up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press.

[Dubois and Prade, 1996] Dubois, D. and Prade, H. (1996). What are fuzzy rules and how to use them. *Fuzzy Sets and Systems*, 84(2):169–185.

[Durrant-Whyte and Bailey, 2006] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006.

[Faria et al., 2014] Faria, Mónica, B., Reis, L. P., and Lau, N. (2014). Adapted control methods for cerebral palsy users of an intelligent wheelchair. *Journal of Intelligent & Robotic Systems*, pages 1–14.

[Flavell, 1976] Flavell, J. H. (1976). Metacognitive aspects of problem solving. In Resnick, L. B., editor, *The nature of Intelligence*, pages 231–236. Erlbaum, Hillsdale, NJ.

[Fountas et al., 2013] Fountas, Z., Gamez, D., and Fidjeland, A. K. (2013). A neuronal global workspace for human-like control of a computer game character. *IEEE Transactions on computational intelligence and AI in games*, 5.

[Franklin, 2000] Franklin, S. (2000). Deliberation an voluntary action in "conscious"software agents. *Neural Network World*, 11:505–521.

[Franklin, 2001] Franklin, S. (2001). Automating human information agents. In Chen, Z. and Jain, L. C., editors, *Practical Applications of Intelligent Agents*. Springer-Verlag, Berlim.

[Franklin et al., 1996] Franklin, S., Graesser, A., Brent, O., Song, H., and Aregahegn, N. (1996). Virtual mattie- an intelligent clerical agent.

[Franklin et al., 1998] Franklin, S., Kelemen, A., and Mccauley, L. (1998). IDA: a cognitive agent architecture. *Systems*.

[Franklin et al., 2013] Franklin, S., Strain, S., McCall, R., and Baars, B. (2013). Conceptual commitments of the lida model of cognition. *Journal of Artificial General Intelligence*, 4:1–22.

[Franklin et al., 2012] Franklin, S., Strain, S., Snaider, J., McCall, R., and Faghihi, U. (2012). Global workspace theory, its lida model and the underlying neuroscience. *Biologically Inspired Cognitive Architectures*, 1:32–43.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Goldsmiths, 2009] Goldsmiths, M. V. (2009). How to define consciousness – and how not to define consciousness. *Journal of Consciousness*, 5:139–156.

[Guimarães et al., 2016a] Guimarães, R. L., Becker, T., Fabro, J. A., and Schneider de Oliveira, A. (2016a). Ros navigation: Concepts and tutorial. In Koubaa, A., editor, *Robot Operating System (ROS) The Complete Reference (Volume 1)*, chapter 5, pages 121–160. Springer International Publishing.

[Guimarães et al., 2016b] Guimarães, R. L., de Oliveira, A. S., Fabro, J. A., Becker, T., and Brenner, V. A. (2016b). Chapter 5: Ros navigation: concepts and tutorial. In Koubaa, A., editor, *Robot Operating System (ROS): The Complete Reference (Volume 1) (Studies in Computational Intelligence) 1st ed.* Elsevier.

[Haikonen, 2000] Haikonen, P. (2000). An artificial mind via cognitive modular neural architecture. In *Proceedings Symposium on How to Design a Functioning Mind*. AISB00 Convention.

[Hofstader and Mitchell, 1994] Hofstader, D. R. and Mitchell, M. (1994). *The Copycat Project: A model of mental fluidity and analogy-making*, pages 31–112.

[Huber, 2000] Huber, M. (2000). *A hybrid architecture for adaptative robot control.* PhD thesis, Department of Computer Science - University of Massachusetts Amherst.

[Hélie et al., 2008] Hélie, S., Sun, R., and Wilson, N. (2008). The clarion cognitive architecture: A tutorial.

[Jaafar et al., 2007] Jaafar, J., McKenzie, E., and Smaill, A. (2007). A fuzzy action selection method for virtual agent navigation in unknown virtual environments. In *FUZZ-IEEE*, pages 1–6. IEEE.

[Jackson, 1987] Jackson, J. V. (1987). idea for a mind. *ACM SIGART Bulletin*, xx(101):23–26.

[James, 1890] James, W. (1890). *The Principles of Psychology*. Harvard University Press.

[Junior, 2006] Junior, V. G. (2006). *Arquitetura Híbrida para Robôs Móveis Baseada em Funções de Navegação com Interação Humana*. PhD thesis, Escola Politécnica da Universidade de São Paulo, São Paulo.

[Kanerva, 1988] Kanerva, P. (1988). *Sparse Distributed Memory*. MIT Press.

[Kelemen et al., 2002] Kelemen, A., Liang, Y., Kozma, R., and Franklin, S. (2002). *Optimizing Intelligent Agent's Constraint Satisfaction with Neural Networks.*

[Kim et al., 2007] Kim, H.-D., Chung-Ang Univ., S., Seo, S.-W., Jang, I.-H., and Sim, K.-B. (2007). Slam of mobile robot in the indoor environment with digital magnetic compass and ultrasonic sensors. In *Control, Automation and Systems, 2007. ICCAS '07. International Conference on*, pages 87 – 90.

[Lee et al., 2013] Lee, D., Kim, D., Lee, S., Myung, H., and Choi, H.-T. (2013). Experiments on localization of an auv using graph-based slam. In *URAI*, pages 526–527. IEEE.

[Maes, 1989] Maes, P. (1989). How to do the right thing. *Connection Science Journal*, 1.

[Martinez and Fernández, 2013] Martinez, A. and Fernández, E. (2013). *Learning ROS for Robotics Programming*. Packt Publishing.

[Minsky, 1985] Minsky, M. (1985). *The Society of Mind*. Simon and Schuster.

[Negatu, 2006] Negatu, A. S. (2006). *Cognitively Inspired Decision Making for Software Agents: Integrated Mechanisms for Action Selection, Expectation, Automatization and Non-Routine Problem Solving*. PhD thesis, The University of Memphis.

[Negatu and Franklin, 2002] Negatu, A. S. and Franklin, S. (2002). An action selection mechanism for "conscious"software agents. *Cognitive Science Quarterly*, 2:363–386.

[Newman et al., 1997] Newman, J., Baars, B. J., and Cho, S.-B. (1997). A neural global workspace model for conscious attention. *Neural Networks*, 10(7):1195–1206.

[Oatley and Jenkins, 1996] Oatley, K. and Jenkins, J. M. (1996). *Understanding Emotions*. Blackwell Publishers Ltd.

[Pedrycz and Gomide, 2007] Pedrycz, W. and Gomide, F. (2007). *Fuzzy Systems Engineering: Toward Human-Centric Computing*. Wiley-IEEE Press.

[Ramamurthy et al., 1998] Ramamurthy, Uma, Bogner, Myles, and Franklin, S. (1998). Conscious learning in an adaptative software agent. In *Proceedings of The Second Asia Pacific Conference on Simulated Evolution and Learning*, pages 24–27.

[Ramamurthy et al., 2006] Ramamurthy, U., Baars, B. J., S. K., D., and Franklin, S. (2006). Lida: A working model of cognition. In Eds: Danilo Fum, F. D. M. and Stocco, A., editors, *Proceedings of the 7th International Conference on Cognitive Modeling.*, pages 244–249, Trieste, Italy.

[Ramamurthy et al., 2004] Ramamurthy, U., Mello, Sidney, K. D., and Franklin, S. (2004). Modified sparse distributed memory as transient episodic memory for cognitive software agents. In *2004 IEEE International Conference on Systems, Man and Cybernetics.*, volume 6, pages 5858 – 5863.

[Russell and Norvig, 2009] Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

[Sariff et al., 2006] Sariff, N., Alam, M. S., and Buniyamin, N. (2006). An overview of autonomous mobile robot path planning algorithms. In *Research and Development, 2006. SCOReD 2006. 4th Student Conference on*, pages 183–188. IEEE.

[Selfridge, 1959] Selfridge, O. (1959). Pandemonium: A paradigm for learning. National Physics Laboratory.

[Shwail et al., 2013] Shwail, S. H., Karim, A., and Turner, S. (2013). Article: Probabilistic multi robot path planning in dynamic environments: A comparison between a* and dfs. *International Journal of Computer Applications*, 82(7):29–34.

[Sperry, 1969] Sperry, R. W. (1969). A modified concept of counsciousness. *Psycological Review*, 76(6):532–536.

[Sun, 1996] Sun, R. (1996). Learning, action, and consciousness: A hybrid approach toward modeling consciousness.

[Takahashi et al., 2009] Takahashi, M., Suzuki, T., Cinquegrani, F., Sorbello, R., and Pagello, E. (2009). A mobile robot for transport applications in hospital domain with safe human detection algorithm. pages 1543–1548.

[Thrun et al., 1998] Thrun, S., steffen Gutmann, J., Fox, D., Burgard, W., and Kuipers, B. J. (1998). Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *In Proceedings of the AAAI Fifteenth National Conference on Artificial Intelligence*.

[Tulving, 2002] Tulving, E. (2002). Episodic memory: From mind to brain. *Annual Review of Psychology*, 53:1–434.

[Vaščák, 2007] Vaščák, J. (2007). Navigation of mobile robots using potential fields and computational intelligence means. *Acta Polytechnica Hungarica*, 4(1).