UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

KLEBER ROBERTO DA SILVA SANTOS

SISTEMA DE NAVEGAÇÃO AUTÔNOMA PARA ROBÔS MÓVEIS BASEADO EM ARQUITETURA HÍBRIDA: TEORIA E APLICAÇÃO

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica com como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Área de Concentração: Automação e Sistemas Elétricos Industriais

Orientador: Prof. Dr. Luiz Edival de Souza.

Co-orientador: Prof. Dr. Leonardo de Mello Honório.

ITAJUBÁ

MAIO DE 2009

Dedicatória

Dedico este trabalho aos meus pais, José Ivo e Marilda, pelo exemplo de vida, carinho e apoio com que eu e meu irmão fomos criados. Saibam que tudo que sou hoje é graças a vocês.

Agradecimentos

Aos meus pais, por todo o apoio e incentivo para meus estudos.

Ao meu irmão, que apesar de todas as desavenças, sempre foi e será uma pessoal muito importante em minha vida.

A Aline, pelo companheirismo demonstrado durante o tempo em que estamos juntos, dedicação e afeto. Sem seu apoio eu não chegaria a onde estou hoje. Você continua sendo meu futuro, apesar das falsas incertezas do passado.....

A Zélia e João, pais da Aline, por me acolherem como membro da sua família.

A todos os amigos, pelo incentivo e apoio demonstrados durante este tempo.

Aos amigos do CRTi, pelo apoio e camaradagem demonstrados no grupo de pesquisa.

Ao Prof. Dr. Luiz Edival de Souza, pelo convite, orientação e todo tempo disponibilizado para realização deste trabalho.

A todos aqueles que, direta ou indiretamente, colaboraram para que este trabalho conseguisse atingir os objetivos propostos.

"A paciência é amarga, mas seu fruto é doce."

Jean Jacques Rousseau

"Deus, em sua infinita misericórdia, manda de vez em quando um missionário gordinho para alimentar canibais pobres subnutridos."

Oscar Wilde

Resumo

Este trabalho apresenta a implementação de um sistema simulado de navegação autônoma para robôs móveis baseado na arquitetura híbrida denominada AuRA (*Autonomus Robot Architecture)*(Arkin, 1987). Esta arquitetura utiliza elementos deliberativos e reativos para o planejamento e execução das tarefas que levarão o robô a cumprir seu objetivo final. Os elementos deliberativos são responsáveis por todo o planejamento das atividades e escolha das trajetórias de movimentação. Para realizar esta tarefa é utilizada uma modelagem do domínio a qual está inserido. Nesta modelagem estão descritas as relações entre os diversos componentes do ambiente, as ações que podem ser realizadas, suas pré-condições e os efeitos provocados por cada ação. Através desta modelagem, que foi implementada em uma linguagem de planejamento denominada PDDL (*Planning Domain Definition Language*), o sistema pode selecionar o conjunto de ações que o levará a atingir o objetivo pré-estabelecido. Com as ações selecionadas, o elemento deliberativo escolhe a melhor forma para realizar aquela atividade, configurando a execução da ação que é realizada de forma reativa. Portanto o elemento reativo é responsável por realizar a ação escolhida que é configurada pelo deliberador. Quando todas as atividades forem cumpridas o robô alcança seu objetivo final. Caso o elemento reativo não consiga realizar alguma ação selecionada, ele solicita um replanejamento das atividades, modificando sua base de conhecimento do mundo a qual está inserido e o conjunto de ações a serem realizadas. A aplicação foi implementada utilizando-se a linguagem C++ e as bibliotecas do software *ARIA* (*Advanced Robot Interface for Applications*), biblioteca de desenvolvimento e controle dos robôs móveis fabricados pela empresa *ActiveMedia/MobileRobots*. O sistema foi testado e validado em um simulador chamado *MobileSim* do mesmo fabricante da biblioteca de *software*.

Palavras-chave: Arquitetura AuRA, Navegação de Robôs Móveis, Inteligência Artificial, Planejamento Automático, PDDL.

Abstract

This work presents the implementation of a navigation system for autonomous mobile robots based on hybrid architecture called AuRA (Autonomous Robot Architecture) (Arkin, 1987). This architecture uses deliberative and reactive elements for planning and execution of tasks that will lead the robot to meet its final goal. The decision factors are responsible for all the planning of activities and choosing the path of movement. To perform this task is used in modeling the domain to which he belongs. This modeling are described relations between the various components of the environment, actions that can be performed, their pre-conditions and the effects caused by each action. Through this modeling, which was implemented in a planning language called PDDL (Planning Domain Definition Language), the system can select the set of actions that will achieve the predetermined goal. With the selected actions, the deliberative system chooses the best way to perform that activity, setting the implementation of the action that is performed in a reactive. Therefore the reactive element is responsible for performing the action that is set by the chosen deliberately. When all activities are completed the robot reaches its goal. If the reactive element can not perform any selected action, it calls for new planning activities, changing its base of knowledge of the world which is inserted and all the actions to be undertaken. The application was implemented using the C++ language and libraries of software ARIA (Advanced Robot Interface for Applications), library development and control of mobile robots manufactured by the company ActiveMedia/MobileRobots. The system was tested and validated in a simulator called MobileSim the same manufacturer of the library software.

Keywords: AuRA Architecture, Navigation of mobile robots, Artificial Intelligence, Automated Planning, PDDL.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ARIA – *Advanced Robotics Interface for Applications*
AuRA – *Autonomous Robot Arquitecture*
CASE – *Computer Aided Software Engineering*
DAMN – *Distributed Architecture for Mobile Navigation*
DD – *Dual Dynamics*
EURON – *European Robotics Research Network*
FF – *Fast-Foward*
IA – Inteligência Artificial
IPC – *International Planning Competition*
ISO – *International Organization for Standardization*
JIT – *Just-in-Time*
LPS – *Local Perceptual Space*
NHC – *Nested Hierarchical Controller)*
PC – *Personal Computer*
PDDL – *Planning Domain Definition Language*
RAP – *Reactive Action Packages*
RCS – *Realtime Control System*
RIA – *Robotics Institute of Association*
SFX – *Sensor Fusion Effects*
SIP – *Session Initiation Protocol*
UML – *Unified Modeling Language*

Sumário

1 Introdução

A robótica é um mercado que movimenta bilhões de dólares anuais, sendo uma área de atuação em franca expansão, movida por avanços tecnológicos e abertura de novos mercados. Ao longo da história a robótica foi se desenvolvendo pelo advento do avanço tecnológico, dês dos chamados *autômatos,* bonecos mecânicos acionados por sistemas de cordas no século XVI (Pieri, 2002), até os robôs atuais de exploração espacial, manipuladores industriais e humanóides que a cada ano melhoram sua capacidade de controle de movimentação e agilidade (Figura 1, Figura 2 e Figura 3).

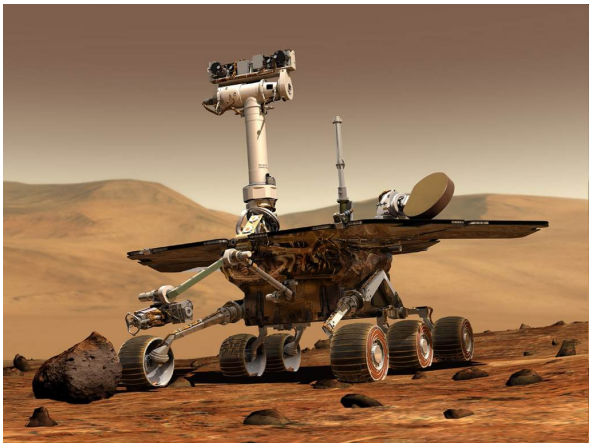

Figura 1 – Robô de exploração espacial

Figura 2 – Robôs manipuladores industriais

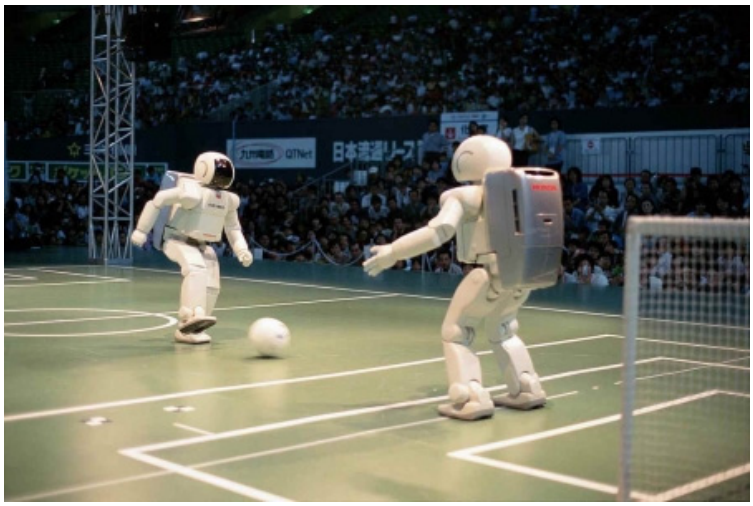

Figura 3 – Robôs humanóides em competição de futebol de robôs

Há diversas definições sobre o que seria um robô, segundo a *Robotics Institute of Association* (RIA) “Um robô, é um manipulador multifuncional e reprogramável, projetado

para movimentar materiais, partes, ferramentas ou peças especiais, mediante movimentos programáveis e variáveis para a realização de uma variedade de tarefas". Já a *International Organization for Standardization* (ISO) dá uma definição de robô mais completa na norma ISO-10218 como sendo "uma maquina manipuladora, com vários graus de liberdade, controlada automaticamente, reprogramável, multifuncional, que pode ter base fixa ou móvel para utilização em aplicações de automação industrial" (Romano, 2002). Esta definição é utilizada pela Federação Internacional de Robótica, a *European Robotics Research Network* (EURON), e muitas outras comissões.

Um robô móvel é um dispositivo mecânico montado sobre uma base não fixa, que age sob o controle de um sistema computacional, equipado com sensores e atuadores que o permitem interagir com o ambiente (Beket, 2005). O modo com ele interage com o ambiente está intimamente ligado a forma com seu sistema computacional foi desenvolvido para interpretar as variações neste ambiente. Esta interação pode ser caracterizada pela arquitetura utilizada para criação do sistema.

Esta dissertação visa apresentar a integração entre diversas técnicas de representação do conhecimento utilizadas para gerar um sistema de navegação autônoma para robôs móveis. O sistema criado tem a capacidade de se auto-planejar, executando as tarefas geradas de forma autônoma. Para isso foi utilizada como base a arquitetura híbrida AuRA (*Autonomous Robot Arquitecture*)(Arkin, 1987).

Para realização desta proposta o sistema criado foi dividido em quatro camadas como sugere a arquitetura adotada. A primeira camada chamada "Planejador da Missão" interagem com o usuário para gerar os objetivos a serem alcançados e as restrições impostas pelo ambiente. A segunda camada chamada "Raciocinador Espacial" é responsável por selecionar as tarefas que devem ser realizadas para que seja possível cumprir a missão imposta pelo usuário. A terceira camada chamada de "Sequenciador de Planos" faz o detalhamento das tarefas para que o robô possa executá-las. Sendo a quarta e última camada chamada de "Controlador Esquema" responsável por gerenciar as execuções das tarefas que são realizadas de forma reativa.

Para apresentação desta proposta o trabalho foi estruturado da seguinte forma: no capítulo 2 será abordado um referencial teórico sobre arquiteturas de robôs móveis, principais arquiteturas e suas definições. No capítulo 3 serão detalhadas as técnicas de planejamento automático, conceitos e aplicabilidade. No capítulo 4 será detalhado os softwares utilizados para a criação da proposta de trabalho, detalhando suas funcionalidades e modo de operação.

No capítulo 5 será definido o sistema de navegação autônomo criado, com todo o sistema de planejamento, suas execuções de forma reativa, capacidade dele se replanejar mediante erros nas execuções dos planos e a integração de todos estes módulos no software Navegador Autônomo. No capítulo 6 é apresentado um exemplo de utilização, mostrando todo o detalhamento das execuções, além dos resultados obtidos da aplicação. Por fim, no capítulo 7 é apresentada a conclusão com relação ao sistema e resultados, além das propostas de trabalhos futuros.

2 Arquitetura de Robôs Móveis

Arquitetura de um robô móvel descreve uma maneira de se construir o software de controle inteligente do robô, apresentando quais módulos devem estar presentes no sistema, e como estes módulos interagem entre si. A descrição de uma arquitetura pode ter um nível razoável de abstração permitindo várias implementações diferentes, ou instâncias, de uma mesma arquitetura (Grassi Jr, 2006).

Esta definição de (Grassi Jr, 2006) é compartilhada por (Arkin, 1998) "uma arquitetura para robôs está mais relacionada a uma arquitetura de software, e não tanto à parte de hardware do sistema de controle".

Entre as diversas características de implementação de uma arquitetura de robôs móveis, o parâmetro que mais se diferencia e, portanto caracteriza uma arquitetura pela definição adotada, é a forma de raciocínio.

O raciocínio dentro da robótica móvel está ligado à forma com a qual o sistema age mediante um estímulo, seja ele uma entrada de dados especificando um objetivo ou uma leitura de seus sensores. O raciocínio dentro da robótica móvel pode ser dividido entre as classificações denominadas: Reativa, Deliberativa ou Hibrida.

2.1 Arquiteturas Deliberativas

Esta abordagem transcreve o processo de planejamento e tomada de decisão do homem para que o robô execute uma determinada atividade (Grassi Jr, 2006)(Pieri, 2002). O sistema de controle do robô utiliza-se do conhecimento armazenado em um modelo interno do mundo que pode ser construído a partir do conhecimento prévio do ambiente e das informações sensoriais do robô.

Um robô empregando raciocínio deliberativo exige o conhecimento completo do mundo e usa esse conhecimento para prever o resultado de suas ações, uma habilidade que lhe permite otimizar o seu desempenho relativamente ao seu modelo do mundo. O raciocínio deliberativo muitas vezes requer um forte modelo de hipóteses sobre o mundo, principalmente, que o conhecimento sobre o qual se baseia seja coerente, confiável e seguro. Se a informação é imprecisa, confusa ou mudou, o resultado do raciocínio pode causar erros sérios. Em um mundo dinâmico, onde os objetos podem rapidamente se mover (por exemplo,

em um campo de batalha ou em um corredor lotado), é potencialmente perigoso invocar o passado já que as informações podem não serem mais válidas (Arkin, 1998).

O robô pode ter modelos de mundo representados de diferentes maneiras: simbólico e geométrico. O modelo simbólico é baseado em lógica utilizando, geralmente, a inteligência artificial. No modelo geométrico o ambiente é representado espacialmente por espaços livres e regiões com obstáculos. Este último é mais utilizado para que o robô execute a atividade de mover-se de forma autônoma de uma região com espaços livres e sem colidir com os obstáculos representados neste modelo (Grassi Jr, 2006)

Existem outras formas de representar o modelo, mas o que importa é definir, através do modelo, o objetivo da tarefa a ser executada pelo robô. Após o objetivo ser estabelecido, o robô deve se planejar para que suas ações o levem ao cumprimento de sua tarefa de forma eficiente e ótima. Para isto este analisa seu modelo interno, por isso o mesmo deve ser extremamente confiável, completo, preciso, consistente para refletir o máximo possível a realidade do mundo. As arquiteturas deliberativas são mais adequadas à ambientes estáticos e muito bem controlados, devido à dependência do mundo interno para execução das ações. Existe também a questão que em caso de mudanças no ambiente, que devem ser percebidas pelo robô durante sua ação e que impeçam a realização da tarefa, o modelo interno do robô tem que ser atualizado, e então, um novo planejamento é feito.

Alguns exemplos de arquiteturas deliberativas são: NHC (*Controlador Hierárquico Aninhado)* desenvolvida por (Meystel, 1991) e a arquitetura RCS (*NIST Sistema de Controle em Tempo Real*) desenvolvida por (Albus, 1996) explicadas abaixo. Dentre as existentes, essas são as mais utilizadas atualmente dentro do princípio puramente deliberativo.

2.1.1 NHC- Controlador Hierárquico Aninhado

Na arquitetura NHC (*Nested Hierarchical Controller*) (Meystel, 1991) o robô coleta informações proveniente de seus sensores e as combina em uma estrutura de dados que representa o modelo do mundo. Através deste modelo o planejamento das suas ações pode ser realizado. Nesta arquitetura o planejamento é composto por três níveis hierárquicos ou níveis de abstração: Planejador da missão, Navegador, e Piloto. O Planejador da missão tem como função enviar trechos da missão para o Navegador, e este, envia trechos de uma trajetória de navegação para o Piloto, que então, determina ações que devem ser enviadas ao controlador de baixo nível para que o robô percorra o trecho da trajetória.

Quando o robô se move, o modelo de mundo é atualizado, mas o ciclo de planejamento não é repetido. Se necessário, o Piloto realiza a correção do movimento local do robô contornando obstáculos percebidos pelo mesmo. Caso isto não seja suficiente o Navegador gera outra trajetória, e apenas como última alternativa, o Planejador da missão replaneja a missão.

2.1.2 RCS – NIST Sistema de Controle em Tempo Real

A arquitetura RCS NIST (*Realtime Control System*) (Albus, 1996) é um auxílio para fabricantes de robôs que querem ampliar a inteligência dos mesmos. Na arquitetura RCS, o modelo do mundo também está organizado de forma hierárquica em vários níveis de abstração, que acompanham a hierarquia de planejamento. Existem módulos de percepção sensorial que são responsáveis pela atualização do modelo do mundo, e seguem a divisão hierárquica da arquitetura. A abordagem RCS possibilita a simulação dos planos de ação para verificação se os mesmos satisfazem os requisitos da tarefa. Também existem módulos que conferem um valor às informações no modelo do mundo e aos planos de ação gerados, decidindo quais informações são mais confiáveis e o que tem maior prioridade.

2.2 Arquiteturas Reativas

A arquitetura reativa é composta por uma série de comportamentos que relacionam condições sensoriais a um conjunto de ações do robô, além de ser dotada de métodos que possibilitam a correta coordenação destes comportamentos (Pieri, 2002). Esta arquitetura evita a utilização de um modelo interno do mundo. Para (Brooks, 1991) “o mundo é a melhor representação dele mesmo”. E por não utilizar um modelo interno do ambiente, este tipo de arquitetura, normalmente interpreta as características do ambiente que podem ser do interesse do robô.

O principal objetivo das arquiteturas reativas é possibilitar a implementação de sistemas de controle que respondam rapidamente a uma variedade de ocorrências ou situações no ambiente, fazendo com que robôs operem em ambientes altamente dinâmicos. Esta rapidez se deve a simplicidade no tratamento das informações sensoriais e a forma direta pela qual a percepção, ou estímulo, está associado com uma ação, ou resposta (Grassi Jr, 2006).

A coordenação em uma arquitetura reativa pode ser feita de forma competitiva ou cooperativa. Na coordenação competitiva, dos comportamentos ativos em um dado momento, apenas um deles prevalece, dentro de uma hierarquia ou arbitragem, determinando a ação que o robô deve realizar. Já na coordenação cooperativa, todos os comportamentos ativos contribuem para determinar a ação do robô. Na Tabela 1 é possível ver, resumidamente, o mecanismo de coordenação utilizado em alguns exemplos de arquiteturas reativas:

Tabela 1 – Mecanismo de coordenação de arquiteturas reativas (Grassi Jr, 2006)

Arquitetura Reativa	Método de Coordenação
Subsunção	Competitivo, supressão e inibição
Esquema Motor	Cooperativo, soma vetorial
Circuito	Competitivo, arbitração com abstração
Seleção e Ação	Competitiva, nível de ativação
Colônia	Competitiva, supressão

2.2.1 Arquitetura de Subsunção

A abordagem de Subsunção (*Subsumption*), proposta por (Brooks, 1986) é uma arquitetura reativa, organizada em camadas de competência através de uma hierarquização de comportamentos. Os níveis mais altos correspondem à execução dos objetivos das tarefas especificadas, enquanto as de níveis mais baixos correspondem às tarefas básicas, como as que garantem a sobrevivência e integridade do robô. O nível superior suprime o fluxo de dados da camada inferior. As camadas decidem quando e como agir, sem o uso de sub-rotinas de outras camadas, no que se convencionou chamar de sistemas de atividades. (Grassi Jr, 2006).

2.2.2 Esquema Motor

Proposta por (Arbib, 1992), os comportamentos são módulos que mostram a relação entre o controle motor e a percepção que agem paralelamente e concorrentemente no sistema, cooperando uns com os outros para determinar a resposta geral do referido sistema.

As respostas motoras de cada comportamento devido a estímulos são representadas na forma de um vetor com amplitude e orientação gerado a partir de um método de campos potenciais artificiais. A somada destes vetores implica na ação a ser executada pelo robô.

Assim, não existe arbitragem de um comportamento em relação ao outro, pois todos os comportamentos contribuem para a resposta do sistema (Grassi Jr, 2006).

2.2.3 Arquitetura de Circuito

A arquitetura de Circuito é uma hibridização dos princípios da reatividade como tipificado pela arquitetura de subsunção, utilizando abstrações e formalismos lógicos (Arkin, 1998). Desenvolvida por (Rosenschein, 1986), suas reações são formados pelo agrupamento de comportamentos mais primitivos, que por sua vez pode ser formado por outros comportamentos, permitindo grande abstração e a formação de um grupo mais complexo de reações comportamentais.

2.2.4 Seleção de Ação

Esta arquitetura, desenvolvida por (Maes, 1989), utiliza um mecanismo dinâmico para selecionar os comportamentos que devem ser ativados. A abordagem de seleção de ação faz uso de um nível de ativação para determinar em tempo de execução qual comportamento deve ser selecionado. O comportamento com maior nível de ativação é escolhido entre um conjunto de todos os comportamentos cujas condições prévias também estejam satisfeitas. Nesta abordagem não existe uma organização clara dos comportamentos em camadas pré-definidas (Grassi Jr, 2006).

2.2.5 Arquitetura de Colônia

A arquitetura de colônia (Connell, 1992) é descendente da arquitetura de subsunção. No entanto, utiliza somente a supressão como estratégia de coordenação e permite especificar as relações entre os comportamentos de maneira mais flexível. A prioridade dos comportamentos na arquitetura de colônia é definida na forma de árvore ao invés de camadas como é feita na arquitetura de subsunção(Arkin, 1998).

2.3 Arquiteturas Híbridas

Este tipo de arquitetura é a que predomina atualmente, ela utiliza deliberação para planejar as ações do robô a partir de uma representação interna do conhecimento do mundo, de forma que os objetivos do robô possam ser atingidos eficientemente. Quando as ações já foram planejadas, a execução do plano gerado é feita utilizando-se reatividade, que responde em alta velocidade a mudanças dinâmicas no ambiente. Assim, a arquitetura híbrida busca ser apropriada para solução de problemas complexos atingindo objetivos de maneira ótima e eficiente, através do uso da deliberação, e em ambientes dinâmicos que exigem rapidez na resposta através de comportamentos reativos.

Outra definição em (Ribeiro, 2001), a arquitetura híbrida corresponde a uma arquitetura reativa controlada por um plano de execução e de sequenciamento de comportamentos. Através do agrupamento da habilidade de raciocínio que se baseiam em modelos internos do mundo (deliberação), estas arquiteturas permitem a reconfiguração dinâmica de sistemas de controle reativo.

Em (Davoren, 1995) são citados alguns exemplos de aplicações da arquitetura híbrida na engenharia, como: controle de tráfego aéreo, controle automotivo, automação da manufatura, controle de processos químicos e robótica.

A arquitetura híbrida é responsável por definir a função da parte deliberativa e reativa dentro do sistema de controle inteligente do robô. Também define como cada uma destas partes estão organizadas, onde e como é feita a interface de coordenação entre deliberação e reação dentro do sistema (Grassi Jr, 2006).

Em (Arkin, 1998) são enumeradas algumas estratégias principais de como deliberação, ou planejamento, interage com a reação, ou execução do plano de ação, são elas:

I. Seleção: o planejamento atua na configuração do sistema de execução.
II. Conselho: o planejamento provê conselhos para o sistema de execução.
III. Adaptação: o planejamento realiza contínuas adaptações no sistema de execução.
IV. Adiamento: o planejamento é realizado somente quando necessário.

Algumas arquiteturas híbridas apresentam uma divisão clara da funcionalidade do sistema em módulos ou componentes funcionais (Murphy, 2000), são eles:

- Sequenciador: gera um conjunto de comportamentos para serem usados na execução de um plano de ação. Também determinam a sequência de ativação e/ou os parâmetros de ativação destes comportamentos.

- Gerenciador de Recursos: aloca recursos para os comportamentos. Por exemplo, se um robô possui uma câmera estéreo, sonares, e sensores de infravermelho disponíveis, o gerenciador de recursos pode determinar qual destes sensores é mais adequado para cada situação.

- Cartógrafo: responsável por criar, armazenar, e manter atualizados os mapas e informações espaciais. Também é responsável por métodos de acesso a esses dados.

- Planejador da Missão: interage com o usuário humano para definir e construir planos para realizar a missão.

- Monitor de Desempenho e Solucionador de Problemas: permitem que o robô perceba se está tendo progresso na realização de sua tarefa.

Algumas arquiteturas empregam estes componentes em camadas da arquitetura mais voltada às atividades deliberativas e de interface com a camada reativa de execução das tarefas que faz uso comportamentos. Mas, pode haver arquiteturas nas quais algumas das funções apresentadas se encontram distribuídas, embutidas em vários dos comportamentos presentes na arquitetura. Nestas arquiteturas pode ser difícil identificar um módulo específico para cada uma destas funções (Grassi Jr, 2006).

Alguns exemplos de arquiteturas consideradas híbridas são: AuRA, SFX, Arquitetura de Agente (*Animated Agent*), Planejador-Reator, DDeP, e arquiteturas de três camadas, tais como, SSS, Atlantis, arquitetura 3T, e a arquitetura genérica do LAAS-CNRS.

A arquitetura AuRA será definida na seção 2.4 com maior grau de detalhes, já que é utilizada com base do trabalho.

2.3.1 Efeito de Fusão de Sensorial

A arquitetura de Efeito de Fusão Sensorial (*SFX - Sensor Fusion Effects*) é uma extensão da arquitetura AuRA para introduzir módulos que tivessem a capacidade de tratar com maior robustez a informação sensorial. Com isso, introduziu-se os módulos de fusão sensorial e módulos que atuem no caso de uma falha eventual nos sensores, que possibilitam a recuperação do robô quando estas falhas ocorrem. Ou seja, a arquitetura SFX surge de uma reorganização dos componentes reativos e deliberativos da arquitetura AuRA.

A parte deliberativa da arquitetura SFX é dividida em módulos ou agentes especializados em uma área de competência: planejador de missão, planejador de tarefa, gerente de sensores, e gerente de atuadores. Estes módulos atuam de forma paralela, cooperando uns com os outros para encontrar um conjunto de comportamentos satisfatórios que possam realizar a missão atendendo às restrições dadas pelo planejador de missão. Nesta camada deliberativa inferior também se encontra o cartógrafo e agentes para monitorar o desempenho do sistema.

Já a parte reativa da arquitetura SFX é dividida em duas camadas: uma de comportamentos estratégicos e outra de comportamentos táticos. Os comportamentos estratégicos definem qual deve ser o comportamento geral do robô, como, por exemplo, qual a direção deve ser seguida para se chegar a uma posição de destino. Os comportamentos táticos se preocupam com a situação imediata do robô, como, por exemplo, desvio de um obstáculo; também atuam como filtros que garantem que o robô opere de maneira segura sem se desviar dos objetivos da missão apresentada pelos comportamentos estratégicos (Grassi Jr, 2006).

2.3.2 Arquiteturas de Três Camadas

Nas arquiteturas de três camadas, deliberação (camada superior) e reação (camada inferior) estão geralmente divididas em duas camadas distintas, e que são coordenadas por uma camada intermediária. A camada reativa controla o robô com o uso de comportamentos, a camada deliberativa é responsável pelo planejamento, e a camada intermediária faz a conexão entre as duas camadas por meio de um mecanismo de sequenciamento que habilita os comportamentos para cada caso que o robô execute a tarefa seguindo o planejamento. Estas três camadas normalmente operam de forma paralela Alguns exemplos de arquiteturas de três camadas são: a arquitetura SSS, a arquitetura Atlantis, a arquitetura genérica do LAAS-CNRS e a arquitetura 3T.

Em (Grassi Jr, 2006), embora possuam três camadas, estas arquiteturas são diferentes entre si pela maneira como cada uma das camadas funciona, e também pela forma como acontece a comunicação entre as camadas da arquitetura. Por exemplo: a arquitetura SSS se baseia no mecanismo de subsunção (Brooks, 1986) para realizar o seqüenciamento. A arquitetura 3T começou com a utilização de um sistema de sequenciamento chamado REX/GAPPS e posteriormente passou a utilizar um sistema chamado RAP (*Reactive Action*

Packages). Já a arquitetura Atlantis utilizava RAP para fazer o sequenciamento, e então partiu para um novo sistema ou linguagem chamada ESL.

2.3.3 Arquitetura de Agente

A arquitetura de agente (*Animated Agent Architecture*), proposta por (Firby, 1998) foi criada com o objetivo de ser aplicada em robôs que trabalham em um ambiente com seres humanos.

A arquitetura possui duas camadas principais. A camada de nível mais baixo possui as habilidades de percepção e atuação. Estas correspondem a processos contínuos, ou seja, controladores de malha fechada implementados na forma de módulos com o intuito de controlar os sensores e atuadores do robô. A camada de nível mais alto na arquitetura é responsável pela execução das tarefas. Esta camada superior é composta principalmente por um executor de planos reativos que seleciona uma seqüência de ações e programa o nível inferior quando o sistema está em execução. Este nível de sequenciamento para execução de tarefa utiliza o sistema RAP (*Reactive Action Packages*), que se compõe de tarefas em um alto nível de abstração na camada de nível mais alto. Quando este sistema encontra-se em operação o RAP detalha as tarefas em níveis de abstração cada vez menores, até atingir o ponto em que as referidas tarefas são primitivas e podem ser realizadas por habilidades no nível inferior da arquitetura. Este detalhamento das tarefas utiliza uma biblioteca hierárquica, o estado atual do robô e as informações do ambiente dadas pelo modelo de mundo.

2.3.4 DDeP

A arquitetura DDeP é uma arquitetura híbrida de duas camadas que trabalham de forma paralela. A camada reativa é definida utilizando uma estrutura chamada DD (*Dual Dynamics*). Seus comportamentos são especificados como sistemas dinâmicos contínuos expressos na forma de equações diferenciais ordinárias. Os comportamentos DD são organizados em níveis hierárquicos. Cada nível influencia o outro, e o nível 0 age diretamente nos atuadores do robô.

Já a camada deliberativa da arquitetura DDeP tem por função o planejamento que age influenciando qualquer nível da camada DD. O planejamento de ação é feito de forma

semelhante aos métodos clássicos de Inteligência Artificial. Mas, o método utilizado é capaz de gerar rapidamente um plano bastante curto e abstrato.

Na arquitetura DDeP, realizar o plano de ação constitui-se de interferir no nível de acionamento dos comportamentos de modo que estes que contribuem para a execução do plano predominem, e que os comportamentos que agem no sentido contrário ao plano de execução sejam contidos.

2.3.5 Planejador-Reator

A arquitetura Planejador-Reator associa planejamento e reatividade. Esta arquitetura possui dois componentes principais que trabalham de forma paralela: o sistema reativo, ou Reator; e o sistema de planejamento, ou Planejador. Diferentemente de um executor de planos de ação, o Reator age independentemente do Planejador, podendo produzir um comportamento útil mesmo sem planejamento. O Planejador ajusta o Reator continuamente, de forma incremental, para que este resulte em um comportamento geral apropriado, capaz de alcançar os objetivos da aplicação. Assim, nesta arquitetura, comportamentos totalmente novos podem ser criados, pois o Planejador modifica de forma incremental a estrutura do Reator.

2.4 Arquitetura AuRA

A arquitetura AuRA – Arquitetura de Robôs Autônomos (*Autonomous Robot Arquitecture*) foi desenvolvida na metade da década de 80 como sendo uma abordagem híbrida para navegação de robôs móveis (Arkin, 1987). A abordagem híbrida surge da junção entre a arquitetura deliberativa ou planejador hierárquico, baseada em técnicas tradicionais de inteligência artificial e na abordagem reativa.

Este planejador hierárquico se utiliza da informação sobre o ambiente e o conhecimento sobre os comportamentos disponíveis no sistema para configurar a parte reativa que tem por função a execução da missão ou tarefa do robô. Conforme mostrado a Figura 4.

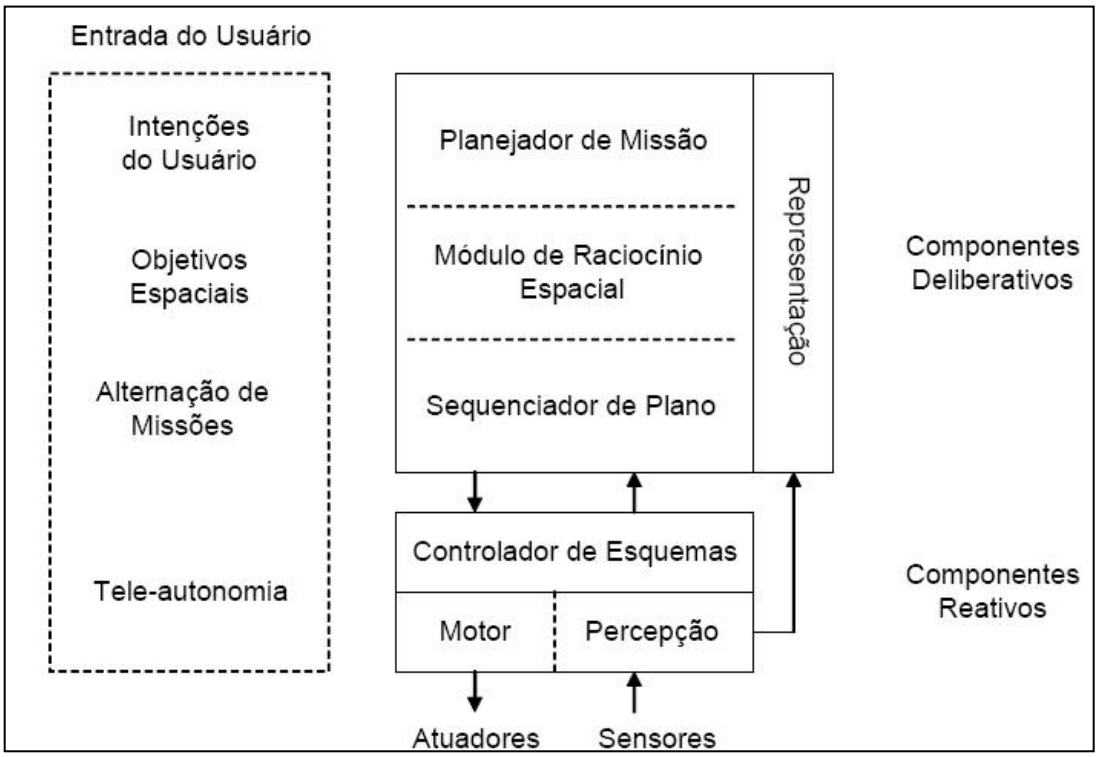

Figura 4 – Esquema da Arquitetura AuRA (Grassi Jr, 2006).

O planejador hierárquico é composto de: Planejador da Missão, Raciocinador Espacial e Sequenciador de Planos. Estes elementos serão explicados abaixo:

- Planejador da Missão: nível mais elevado da hierarquia, tem por função estabelecer objetivos de alto-nível para o robô e restrições dentro das quais o robô deve operar.

- Raciocínador Espacial (Navegador): nível intermediário da hierarquia e que utiliza as informações obtidas a partir do mapa do ambiente, armazenado em uma memória de longa duração, para construir uma seqüência de trechos de navegação que devem ser executadas pelo robô para que a missão possa ser completada.

- Sequenciador de Planos (Piloto): nível mais baixo na hierarquia, onde para cada trecho de navegação gerado pelo módulo de raciocínio espacial, o piloto aponta um conjunto de comportamentos motores que devem ser enviados para execução.

Então, o conjunto de comportamentos especificados pelo Sequenciador de Planos é enviado para execução no robô. Aqui, termina a parte correspondente à deliberação e começa a execução reativa.

A função do controlador de esquemas é o controle e monitoramento dos comportamentos reativos durante a execução. Todo comportamento motor (ou esquema) está associado a um esquema perceptivo que possui a capacidade de prover o estímulo demandado para um determinado comportamento (Grassi Jr, 2006).

A referida percepção voltada para a ação é a base para esta forma de navegação fundamentada em comportamento. Cada comportamento resulta em um vetor de resposta de forma semelhante ao encontrado em campos potenciais. Os esquemas operam de forma assíncrona transmitindo seus resultados para um processo (mover-robô) que realiza o somatório dos vetores e normaliza estas entradas, e transmitindo o resultado para um sistema de controle de baixo nível responsável pela execução (Arkin, 1997).

Quando se inicia a execução da parte reativa, a parte deliberativa não é reativada a não ser que seja detectada uma falha na parte reativa, que normalmente é indicada por uma ausência de evolução na execução da tarefa. Neste ponto, o planejador hierárquico é reativado uma camada por vez, com início na camada inferior chegando à camada superior, até que o problema seja solucionado. Inicialmente o sequenciador de planos é convocado para que forneça uma nova configuração de comportamentos. Esta configuração é determinada pela informação local sobre o ambiente armazenada em uma memória de curto prazo. Se isso não for suficiente, ou seja, a rota está completamente obstruída, o módulo de raciocínio espacial é chamado e tenta instituir um novo caminho que desvie da região onde há um problema. Se isto não for suficiente, o planejador de missão é chamado e o usuário é avisado da dificuldade, podendo desistir da missão ou realizar alterações na mesma (Arkin, 1997).

A arquitetura AuRA é uma arquitetura bastante modular, flexível e geral, além de ser híbrida. Como os componentes da arquitetura são modulares, estes podem ser trocados de acordo com a aplicação e evolução das tecnologias utilizadas para implementar cada um desses módulos. A arquitetura AuRA também admite que o sistema possa se adaptar utilizando métodos de aprendizado. A adaptação é feita alterando-se a importância ou peso dado a cada um dos esquemas motores empregados na realização da tarefa (Grassi Jr, 2006).

A arquitetura AuRA pode ser utilizada em diversas aplicações, incluindo domínios como:

- Ambientes de manufatura
- Navegação tridimensional como as encontradas em domínios aéreos e aquáticos
- Navegação em ambientes abertos e fechados
- Competições de robôs
- Cenários militares
- Manipulação móvel
- Equipes de robôs

3 Planejamento Automático

3.1 Introdução

A tarefa de chegar com uma sequência de ações que irão atingir uma meta é chamado planejamento (Russell, 2003). Segundo (Ghallab, 2004), planejamento é o lado do raciocínio agindo. É um resumo explícito do processo deliberativo que escolhe e organiza ações antecipando os seus resultados esperados. Esta deliberação tem em vista atingir da melhor forma possível, os objetivos pré-estabelecidos.

O ato de planejar faz necessário o conhecimento prévio dos requisitos para a realização de cada ação e dos efeitos causados por ela. Através deste conhecimento pode-se selecionar as ações que executadas de forma organizada gerem o objetivo pretendido.

O planejamento automático é uma área da Inteligência Artificial (IA) que estuda o processo de escolhas de ações de forma computacional. Esta crescente área da IA está presente em cenários como: planejamento de trajetória e movimentação de sistemas automáticos móveis; planejamento de percepção envolvendo ações de sensoriamento para captação de informação do ambiente; planejamento de navegação que combina sensoriamento e definições de trajetórias; planejamento de manipulação relacionado com movimentação de objetos como, por exemplo, montagem de peças, organização de containers, entre outros (Ghallab, 2004).

No planejamento de trajetória e movimentação se está mais preocupado com a síntese do caminho geométrico entre uma posição inicial e final, e um controle da trajetória durante o caminho encontrado. Planejamento de movimentação leva em conta o modelo do ambiente e as restrições cinemáticas e dinâmicas do sistema móvel. Ela é uma área em destaque nos tempos atuais, já bastante madura sendo oferecida uma gama de métodos eficientes e confiáveis (Ghallab, 2004).

No planejamento de percepção se está mais preocupado com a geração de planos envolvendo ações de sensoriamento para captação de informações. Surge em tarefas com modelagem de ambientes ou objetos, localização através de sensoriamento de sistemas móveis ou identificação do estado atual do mundo (Ghallab, 2004).

O planejamento de navegação combina os dois planejamentos anteriores, de movimentação e de percepção a fim de se chegar a um objetivo ou explorar uma área. O

objetivo do planejamento é a navegação que sintetiza uma política que combina localização e sensoriamento de movimentos.

Uma abordagem natural para estas diversas formas de planejamento é resolver cada problema com as representações e técnicas adequadas para o problema. O conceito de planejamento automático vem pregar e desenvolver soluções para os problemas de forma independente do domínio de utilização, pois o estudo foca no processo de planejamento em si, que é comum a todos os domínios específicos, gerando abordagens gerais que também podem ser aplicados a estes domínios.

Dentro dos estudos em planejamento automático, uma de suas áreas enfoca a pesquisa nos chamados planejadores (*plannes*) que são softwares capazes de selecionar uma sequência de ações a serem aplicadas no domínio para atingir a meta estipulada, partindo como base de um determinado estado inicial. Eficiência e desempenho são características desejadas a estes sistemas. A eficiência está ligada a qualidade do plano gerado e a execução com o mínimo esforço. Enquanto que o desempenho está ligado à velocidade com que é gerado o conjunto de soluções.

Para que o planejador possa planejar é necessário que este raciocine sobre uma descrição (representação), um modelo do domínio. Este modelo possui, de um modo geral, as ações (com suas precondições e efeitos), restrições, recursos disponíveis, os objetos, os objetivos a serem atingidos (eventualmente com critérios de otimização) e o estado inicial de um problema de planejamento. Utilizando esta informação como entrada, o planejador pode fornecer uma coleção organizada das ações de forma a atingir os objetivos pré-estabelecidos, ou seja, uma solução aceitável para o problema. É possível perceber que a especificação do problema e o conhecimento sobre os domínios fornecidos ao planejador possuem um papel fundamental no processo de planejamento automático (Vaqueiro, 2007).

Em (Ghallab, 2004), é mostrado um modelo conceitual do planejamento automático (Figura 5), nele um plano é gerado pelo planejador a partir de um modelo do domínio de execução, do estado inicial e dos objetos existentes no mundo. Este plano é enviado ao controlador que observa e executa as ações em um sistema de estados, isto é, o mundo real é discretizado em um conjunto de estados e transições, onde uma ação pode causar a mudança das variáveis do mundo, portanto um estado diferente.

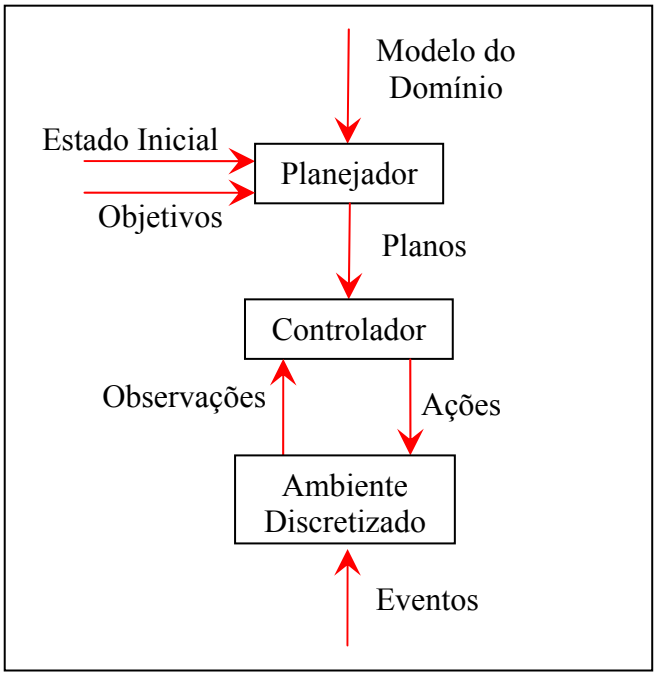

Figura 5 – Modelo conceitual de planejamento (Ghallab, 2004)

Este modelo pode ser estendido, fornecendo para o planejador as informações executadas e observadas pelo controlador, com isso existe a capacidade do sistema se replanejar caso algum fato não considerado no planejamento ocorra alterando o resultado final. Esta modificação amplia as funcionalidades existentes, permitindo alterações de forma dinâmica (Figura 6).

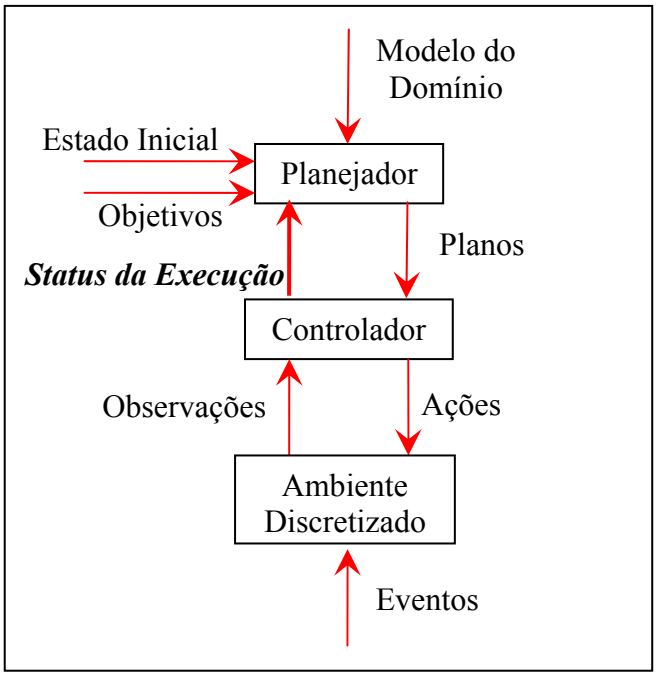

Figura 6 – Modelo conceitual dinâmico de planejamento (Ghallab, 2004)

Dentro das pesquisas em planejamento, para que fosse possível sua expansão e desenvolvimento, surgiram as linguagens padronizadas para descrição de domínios e problemas de planejamento, como é o caso de STRIPS (Fikes, 1970) e ADL (Pednault, 1987) Com isso o desenvolvimento de planejadores independente de domínio foi encorajado. Com o surgimento da primeira Competição Internacional de Planejamento em 1998 - (IPC 98), surgiu também a linguagem PDDL (*Planning Domain Definition Language*). Baseada nas linguagens anteriores, esta linguagem fez com que todos os desenvolvedores de planejadores a utilizassem como linguagem de entrada para o algoritmo de planejamento (planejador).

3.2 A linguagem PDDL

Como comentado, foi através da IPC (*International Planning Competition*) realizada pela primeira vez em 1998, que surgiu a linguagem PDDL. Ela surgiu da união entre a STRIPS (Fikes, 1970) e ADL (Pednault, 1987) para ser a linguagem base da competição.

O IPC é um evento bi-anual. Os objetivos da competição são fornecer um fórum para comparação empírica de sistemas de planejamento, destacando os desafios para a comunidade cientifica e a atual capacidade dos projetos apresentados. Propor novos rumos para as pesquisas na área, fornecer um núcleo comum de problemas formais e uma representação que possa auxiliar na comparação e avaliação dos sistemas de planejamento. Embora o evento tenha um estilo competitivo (sistemas com excepcionais desempenhos são identificados durante a competição), o foco é coletar os dados apresentados, com a interpretação dos resultados, trazendo o máximo de informação possível para a comunidade cientifica (IPC6, 2008).

A linguagem PDDL está atualmente na versão 3.1, onde foram inseridos novos recursos como variáveis de custo de uma ação e variáveis com novos estados.

De forma simplificada os componentes da linguagem PDDL são apresentados a seguir (Vidigal, 2007):

- Objetos: todos os elementos do “Domínio/Mundo” que são pertinentes ao problema;
- Predicados: propriedades e relações entre os objetos;
- Estado Inicial: é a descrição do “Mundo” em seu estado inicial. São os objetos instanciados;

- Descrição dos Objetivos: o estado desejado do “Mundo”, ou os predicados que devem ser verdadeiros para se alcançar o objetivo;
- Ações/Operadores: Descrição dos meios de como modificar o “Mundo”;

Para demonstrar a linguagem, abaixo encontra-se um exemplo simples de uma domínio criado em PDDL, este exemplo foi retirado de (Vidigal, 2007).

```
(1)   (define (domain Viagem)
(2)   (:requirements :strips :typing)
(3)   (:types cidade - object )
(4)   (:predicates
(5)         (Existe-Estrada-Entre ?x - cidade ?y - cidade )
(6)         (Estou-em ?x - cidade ) )
(7)   (:action Viajar
(8)         :parameters (?a - cidade ?b - cidade )
(9)         :precondition
(10)              (and (Estou-em ?a )
(11)              (Existe-Estrada-Entre ?a ?b ))
(12)        :effect (and (Estou-em ?b )
(13)              (not (Estou-em ?a ) ) ) ))
```

Neste exemplo é modelado um domínio chamado de Viagem, nele são descritos os comportamentos necessários para a execução de uma viagem entre dois lugares. Em (1) é definido o nome do domínio, em (2) os requisitos da linguagem utilizada, em (3) os tipos objetos que podem ser criados.

A partir de (4) começa as definições dos predicados e das ações possíveis no domínio de execução. Em (5) e (6) são definidos dois predicados que são usados no domínio, um chama “Existe-Estrada-Entre” e o outro chama “Estou-em”. Os predicados são usados para informar ao sistema as condições presentes no ambiente. O predicado “Existe-Estrada-Entre” informa que entre duas localidades existe uma estrada que liga os dois pontos. Já o predicado “Estou-em”, informa em qual cidade uma pessoa ou objeto está.

A partir de (7) é definida uma ação de Viajar, a ação necessita de algumas pré-condições para poder ser executada, isso pode ser visto em (9), (10) e (11). Caso as pré-condições forem

atendidas e a ação fosse aplicada em um planejamento, o efeito causado por ela seria mudar a localização de um objeto através do predicado “Estou-em”, isso pode ser visto em (12) e (13).

Após a modelagem do domínio, deve ser criado um problema de planejamento para que o programa planejador possa produzir uma sequência de ações que resulte no objetivo desejado, a partir do domínio e do problema de planejamento.

Um problema de planejamento é modelado abaixo:

```
(1)   (define (problem ViagemLimeiraCampinas)
(2)   (:domain Viagem)
(3)   (:objects
(4)         Campinas - cidade
(5)         Americana - cidade
(6)         Limeira - cidade )
(7)   (:init
(8)         (Existe-Estrada-Entre Limeira Americana )
(9)         (Existe-Estrada-Entre Americana Campinas )
(10)        (Estou-em Limeira ))
(11)  (:goal (Estou-em Campinas ) ))
```

Neste problema de planejamento é instanciado os objetos existentes no mundo de aplicação, neste caso existem três objetos que são cidades, como mostrado em (3),(4), (5) e (6). O problema fornece o estado atual do mundo nas diretivas (7) (8) (9) e (10), informando a existência de uma estrada entre Limeira e Americana, e de Americana a Campinas. O objetivo do problema é mostrado em (11), sendo ele chegar a Campinas.

Utilizando um programa planejador e fornecendo como entrada o domínio e o problema criado, o planejador deverá fornecer a sequência de ações que deve ser aplicada para atingir a solução desejada.

No caso deste exemplo, a solução será primeiro viajar de Limeira a Americana e depois viajar de Americana a Campinas.

Este exemplo é bem simples, mas as possibilidades de aplicação da linguagem são enormes, pois podem ser modelados domínios e problemas bem mais complexos, na qual a solução para o mesmo não seria tão trivial e muito menos intuitiva.

4 Ferramentas de Software

Para implementação do sistema propostos por este trabalho, foi utilizado um conjunto de softwares para criar, validar e testar a metodologia sugerida.

O sistema de Navegação Autônomo foi desenvolvido na ferramenta de programação Visual Studio 2008 da Microsoft em linguagem C++ nativa, esta linguagem foi escolhida, pois proporciona desempenho completo da plataforma (hardware e software) já que lida diretamente com o sistema operacional e seus drivers de hardware. Por este motivo ela é a linguagem mais utilizada para a programação de sistemas para robôs moveis, sendo considerada de execução mais rápida em comparação as linguagens gerenciadas por *frameworks*.

Além do software de programação foram utilizadas bibliotecas de funções, sendo as principais as bibliotecas *ARIA e ARNetworking*; simuladores de interação entre robôs e o ambiente (*MobileSim);* criador de mapas de simulação *Mapper3Basic*; ferramenta de criação de domínio e problemas de planejamento (*itSIMPLE*) e software planejador (*FF)*. A utilização e integração de todas estas ferramentas de software propiciaram a concretização deste trabalho, sendo eles detalhados neste capitulo.

4.1 ARIA e ARNetworking

ARIA (*Advanced Robotics Interface for Applications*) é uma base de desenvolvimento de fonte aberta que oferece uma interface robusta, do lado do cliente, para uma variedade de sistemas integrados aos robôs da família *Pioneer*, ver Apêndice C, incluindo comandos e comunicação com o microcontrolador, além de diversos acessórios conectados ao sistema (MobileRobots A, 2009).

A interface de comunicação pode ser feita pela biblioteca ARNetworking, ela fornece o desenvolvimento da camada TCP/IP base para a comunicação com a plataforma através da rede.

Este pacote de software (ARIA e ARNetworking) auxilia no desenvolvimento do sistema, pois o usuário pode se preocupar em realizar o desenvolvimento das rotinas de alto nível, não se preocupando com as interfaces de mais baixo nível de detalhes como as interações cliente-servidor, incluindo a criação de redes e comunicações seriais, transformação de comandos em pacotes a serem transmitidos, ciclo temporal, *multithreading*,

bem como a comunicação e controle de diversos dispositivos que podem ser integrados ao sistema, como: sonares, *bumpers*, câmeras, atuadores, e muitos outros. Na Figura 7 um detalhamento da estrutura da biblioteca ARIA pode ser vista.

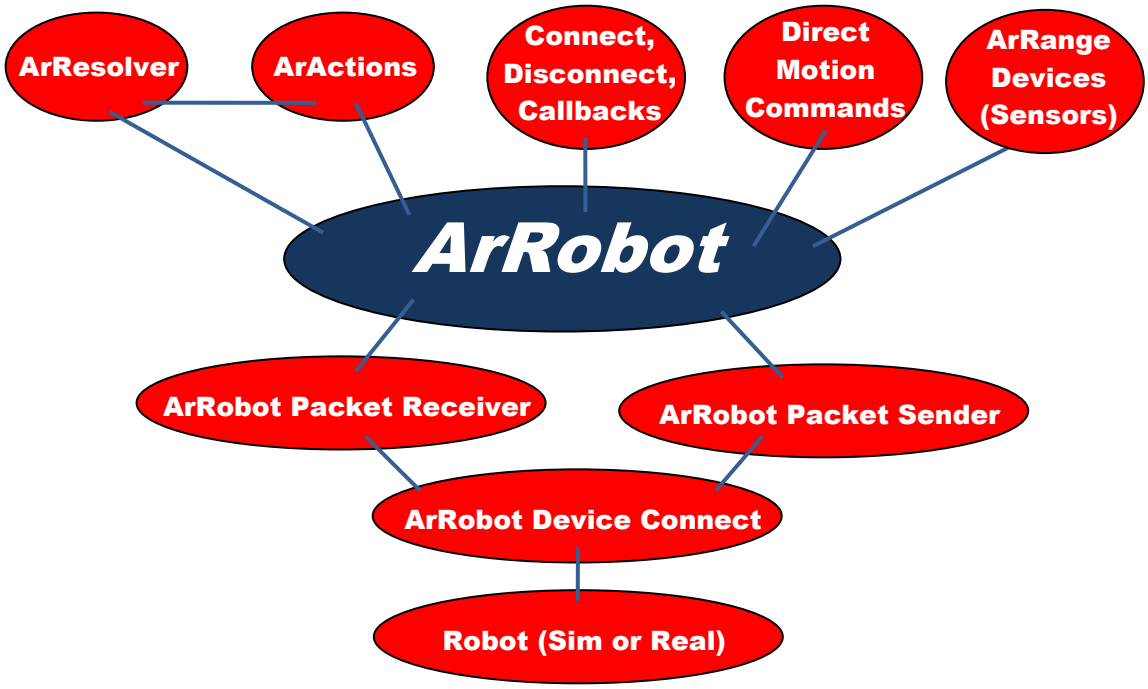

Figura 7 – Arquitetura da Biblioteca ARIA (MobileRobots, 2006)

Neste trabalho a biblioteca ARIA e ARNetworking foram utilizadas para fazer a comunicação entre o programa cliente desenvolvido e o simulador utilizado. Vale ressaltar que o mesmo programa poderia ser enviado a robôs físicos da família *Pioneer*, para seu controle sem que seja necessária a realização de nenhuma alteração. Também foram utilizadas desta biblioteca algumas funções de controle de movimentação e desvio de obstáculos de mais baixo nível.

Nos próximos capítulos será discutida mais profundamente a utilização da biblioteca ARIA e ARNetworking no desenvolvimento do sistema de navegação proposto.

4.2 Ambiente de Simulação e Criação de Mapas

Como comentado anteriormente, para a validação do sistema foi utilizado um simulador chamado MobileSim em sua versão 0.4.0. Este programa simula robôs móveis em seus ambientes, realiza *debug* e experimentos em conjunto com programas baseados no *ARIA*.

MobileSim é um software de código aberto (*Open Source)* para simulação de robôs móveis e seus ambientes (MobileRobots B, 2009), baseado no simulador Stage, pertencente ao projeto Player/Stage (Player, 2009), com modificações para ser usado em plataformas MobileRobots/ActiveMedia, podendo ser usado em conjunto com o *ARIA* para testar os algoritmos implementados no *ARIA*.

O software MobileSim converte um mapa criado no *software* Mapper3Basic para o ambiente de simulação, colocando um modelo de robô simulado neste ambiente. Também provém uma conexão com o *Pioneer* simulado via porta TCP 8101 (similar à conexão via porta serial com o *Pioneer* real). A maioria dos programas baseados no ARIA conecta automaticamente na porta TCP caso ela esteja disponível. Na Figura 8 pode ser observado o ambiente de simulação.

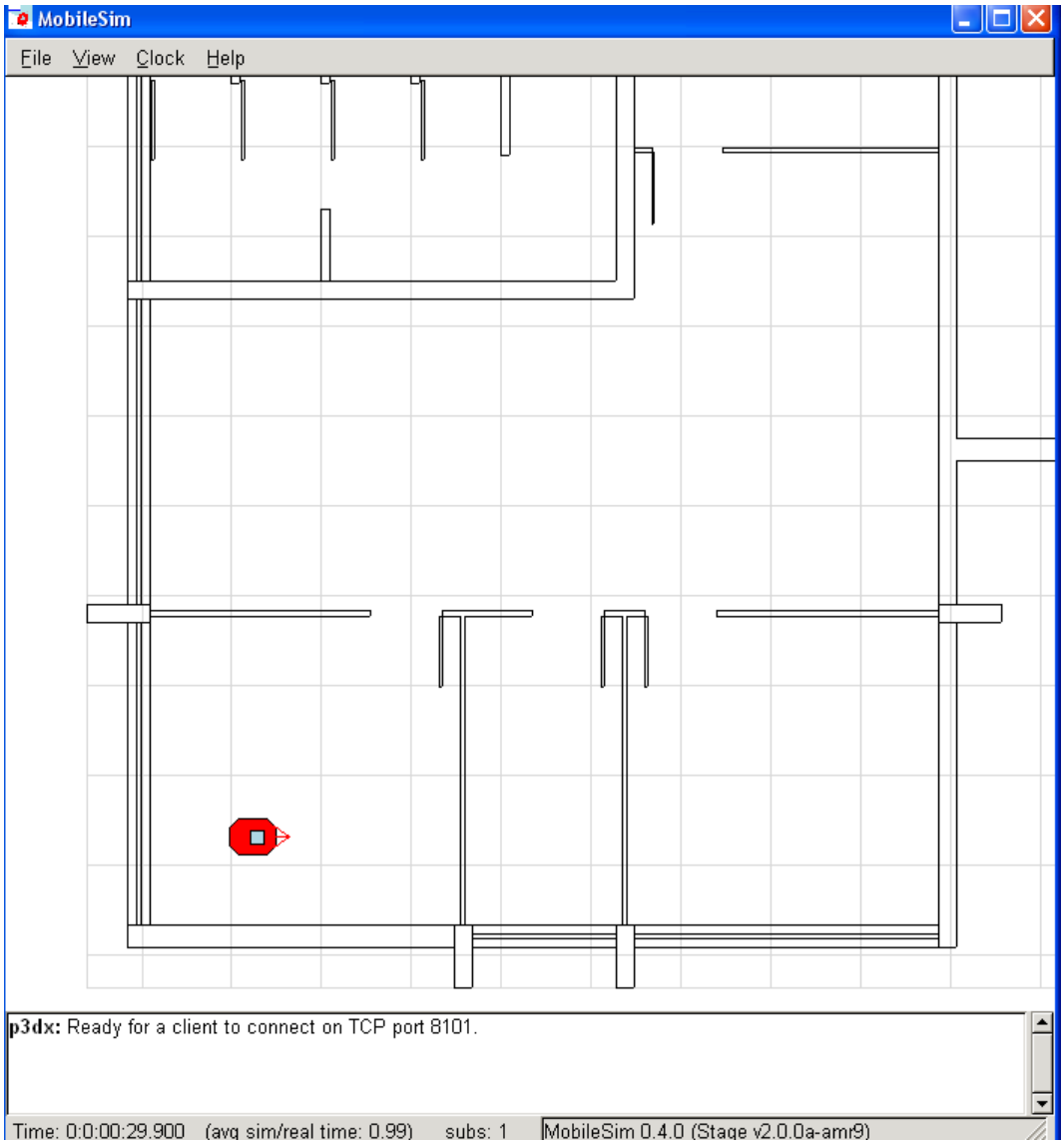

Figura 8 – Ambiente de Simulação MobileSim

O mapa do ambiente foi criado a partir de uma planta baixa das dependências do andar térreo do prédio da Engenharia Elétrica da UNIFEI – Universidade Federal de Itajubá, com dimensões reais, utilizando o *software* Mapper3Basic, ver Figura 9.

O simulador utiliza um banco de dados dos robôs da família *Pioneer*, com toda a configuração do robô, dimensional, aceleração e velocidade máxima de movimentação, disposição dos sensores, resolução e erros dos *encoders*, e demais parâmetro que modelem o robô. Esta parametrização aproxima mais a simulação do mundo real.

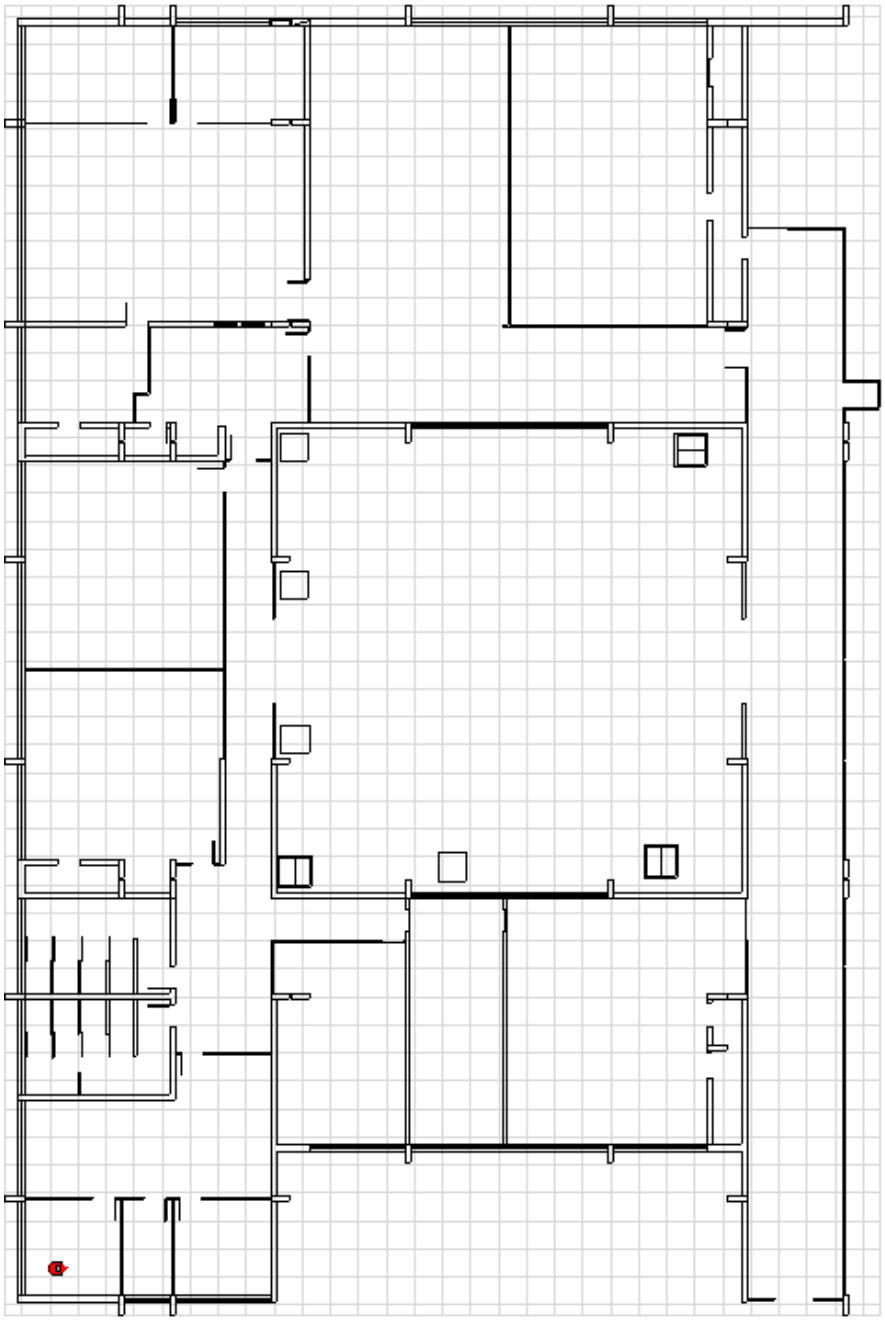

Figura 9 – Mapa utilizado nas simulações

4.3 Criador de Domínio e Problema de Planejamento (Itsimple)

O itSIMPLE, acrônimo para *Integrated Tools Software Interface for Modeling PLanning Environments* (Ferramenta de Software Interface para Modelagem de Ambientes de Planejamento) é uma ferramenta CASE - *Computer Aided Software Enginnering* (Desenvolvimento de Software Auxiliado por Computador) para o auxilio à elaboração de domínios e problemas de planejamento em linguagem PDDL, ver capítulo 3.2.

O software itSIMPLE, desenvolvido na linguagem Java™ por um grupo de pesquisados do departamento de mecatrônica da Faculdade de Engenharia Industrial (FEI), é um software de código aberto (*Open Source)* que possibilita a criação de modelos, ricos em conhecimento, de diversos domínios de planejamento reais através de diagramas UML(UML, 2009), ver Figura 10.

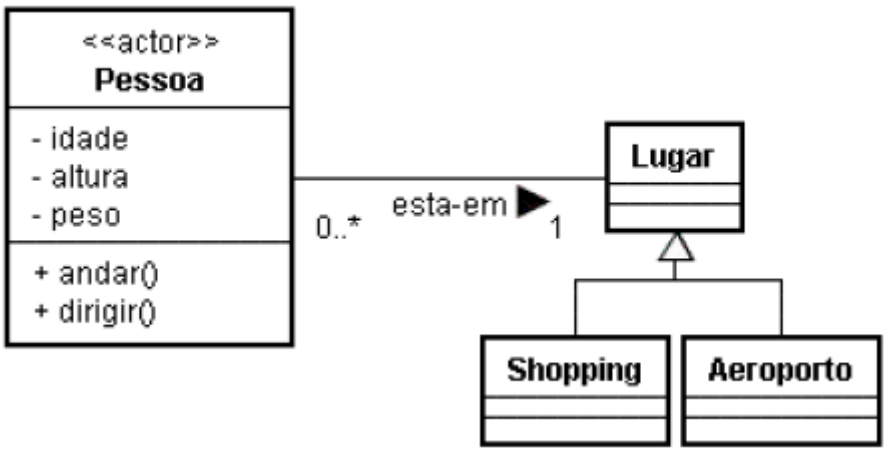

Figura 10 – Exemplo de Diagrama de Classes em UML (Vaqueiro, 2007).

A ferramenta desenvolvida é independente de qualquer técnica ou algoritmo de planejamento existente, isto é, os processos de modelagem e verificação não dependem de uma técnica específica de planejamento (planejador). De fato, os algoritmos de planejamento são vistos como componentes presentes neste ambiente que podem ser escolhidos (automaticamente ou não) livremente(Vaqueiro, 2007).

Através do itSIMPLE o projetista pode especificar, modelar e analisar domínios utilizando linguagens conhecidas sem a necessidade de profundos conhecimentos na área de Planejamento Automático. Linguagens e formalismos como, inicialmente, a UML, XML, Redes de Petri e PDDL são integradas em uma única ferramenta (Ambiente) de modo a proporcionar ao usuário uma grande flexibilidade e facilidade na mudança de contexto (modelagem, análise, entre outros), possibilitando o uso do potencial de cada linguagem e enriquecendo assim os modelos e as respectivas análises. Linguagens visuais como a UML e Redes de Petri propiciam uma maior uniformidade nos conceitos do modelo perante os participantes (*stakeholders*, especialistas em planejamento, especialistas do domínio, usuários, etc.), com diferentes pontos de vista, devidos aos diagramas e as animações (Vaqueiro, 2007).

Outro fato importante para a escolha da utilização deste software em comparação a programação direta em linguagem PDDL é a questão da documentação gerada ser de fácil entendimento mesmo para usuários que não estão acostumados com os formalismos da linguagem PDDL.

Neste software o modelo do domínio de planejamento é criado através de relações entre classes UML, (UML, 2009), como a generalização, associação, agregação, composição e dependência. Conseguindo desta forma, modelar as interações entre os diversos elementos existentes do domínio de planejamento.

A partir da modelagem em UML, o software é capaz de gerar o código em linguagem PDDL que represente o domínio e o problema de planejamento que se deseje desenvolver, podendo o programa em PDDL ser carregado em um software de planejamento (Planejador), para obtenção da solução do problema em questão.

4.4 Planejador de Ações (FF)

Planejadores de tarefas, ou somente Planejadores, são softwares que geram uma lista de tarefas a serem realizadas para se atingir um objetivo pré-estabelecido, utilizando-se para isso de um modelo do domínio e problema de planejamento geralmente desenvolvidos em uma linguagem de planejamento como a PDDL, ver capitulo 3.

O planejador utilizado neste trabalho foi o FF, acrônimo para *Fast Forward*, desenvolvido por Jörg Hoffmann (Hoffmann, 2001). O FF é um planejador independente do domínio que efetua uma busca local pelo espaço de estados. Esta busca é orientada por uma função heurística que estima a distância da solução em relação ao estado inicial. O espaço de

estado em qualquer problema de planejamento não é trivial e na maioria dos casos é muito grande, ou seja, computacionalmente difícil. Por esta razão, planejadores que fazem busca com esta técnica são dependentes de boas funções heurísticas para guiar a busca.

Um planejador de espaço de estado progressivo executa busca a partir do estado inicial (busca de encadeamento progressivo) até encontrar um estado no qual o objetivo especificado é satisfeito. A Figura 11, ilustra uma busca progressiva no espaço de estados. Na figura, os estados são representados pela letra 'S', sendo S0 o estado inicial, Sg o estado objetivo e Sn os demais estados possíveis. As ações são representadas pela letra 'a'. Através da aplicação de ações (a1, a2, an), novos estados vão sendo gerados, e a árvore de busca vai sendo montada até uma solução ser encontrada (Vidigal, 2007).

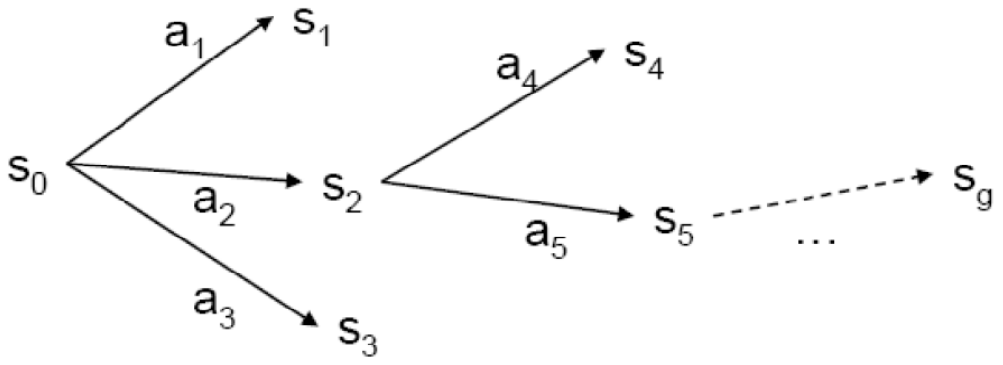

Figura 11 – Busca Progressiva no espaço de estados (Vidigal, 2007)

A busca no espaço de estado pode ser feita de maneira regressiva (busca de encadeamento regressivo). Ao invés de fazer a busca adiante, a partir do estado inicial, faz-se regressivamente a partir do estado objetivo. A Figura 12 ilustra a busca regressiva no espaço de estados, onde a partir do estado objetivo g0, e da aplicação das ações de forma reversa, novos estados vão sendo gerados até que o estado inicial S0 seja encontrado, (Vidigal, 2007).

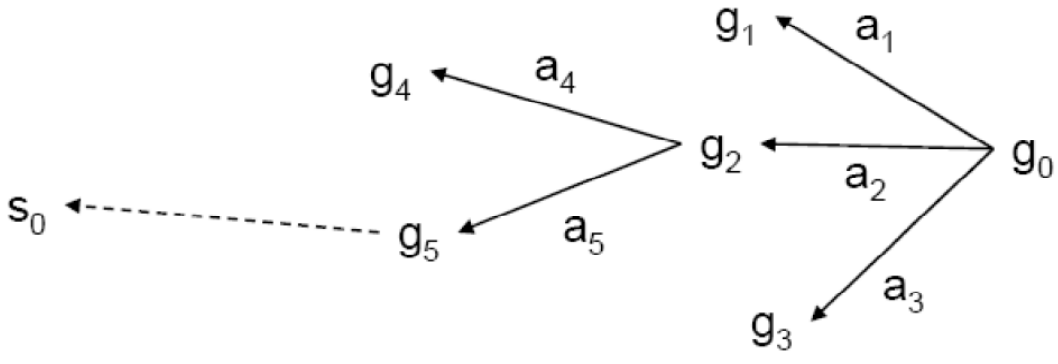

Figura 12 – Busca Regressiva no espaço de estados (Vidigal, 2007)

No planejador FF os problemas são atacados por uma busca progressiva no espaço de estados, e esta busca é guiada por uma função heurística que é automaticamente extraída da descrição do domínio de planejamento. Para chegar a tal função, o planejador "relaxa" o problema de planejamento. Quando se aplica uma ação no problema ela gera a exclusão e a adição de certas condições no espaço de estados, o relaxamento consiste em não se excluir condições existentes quando se aplica uma ação. Isto faz com que se possa chegar a uma estimativa do resultado final de forma mais rápida ou se possa constatar que o problema não tem solução.

O resultado da função heurística do planejador FF é obtido computando o comprimento da linearização desta solução para a tarefa relaxada. Com isso, cada estado criado durante a busca em árvore, recebe um valor de custo estimado pela heurística. Como a busca é feita em largura pelo FF, o estado que possuir um menor custo é o escolhido para a expansão do nó. Isso é feito até que a solução seja encontrada, ou até que estados comecem a se repetir (Vidigal, 2007).

O FF ao encontrar um resultado para o problema gera uma lista de tarefas que pode ser impressa na tela do computador ou armazenada em um arquivo em formato texto, como pode ser visto na Figura 13.

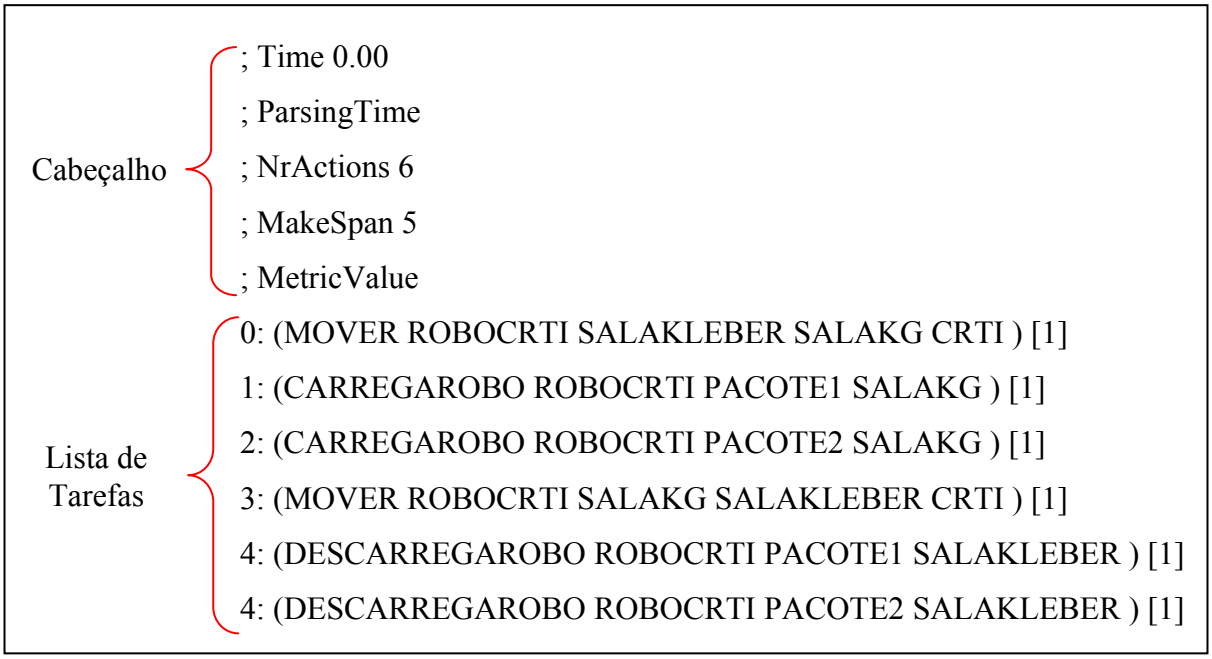

Figura 13 – Resultado gerado pelo FF de um problema de planejamento

5 Sistema de Navegação Autônoma

O sistema de navegação autônomo desenvolvido foi baseado na arquitetura AuRA (Arkin, 1997). Esta arquitetura utiliza elementos deliberativos e reativos para a execução das suas funcionalidades. Por apresentar estas duas formas de raciocínio ela é considerada uma arquitetura híbrida, onde o raciocínio deliberativo é utilizado para a execução do planejamento das atividades que devem ser realizadas, e o raciocínio reativo é utilizado para executar as tarefas planejadas, sendo associado a ele um conjunto de sensores que realizam a percepção do ambiente, guiando as execuções das ações pelos estímulos captados do ambiente a sua volta.

A arquitetura criada pode ser vista na Figura 14, sendo todos os seus módulos explanados neste capítulo. Basicamente, se tem duas estruturas, uma de planejamento e outra de execução.

A estrutura de planejamento utiliza técnicas de inteligência artificial, como o planejamento automático, para gerar uma lista de atividades a serem feitas para alcançar uma meta estipulada. Isso é feito utilizando-se uma modelagem do mundo a qual se deseja interferir, com seus comportamentos, possíveis ações, estado inicial e estado final ("Especificação do Domínio de Planejamento" e "Especificação do Problema de Planejamento"). Esta modelagem foi feita utilizando-se o software ItSimple (Vaqueiro, 2007).

Através da modelagem e de um software planejador independente do domínio, é gerado uma lista de ações a serem executadas ("Gerador de Planos"), sendo esta armazenada e direcionada para a execução correta ("Decisão e Armazenagem da Lista de Tarefas"). No "Gerador de Rotas" os planos são detalhados para que o robô possa executá-los, este detalhamento utiliza o conhecimento que o robô tem do ambiente a qual está inserido.

Ainda na estrutura de planejamento, existe um módulo que possibilita a execução do replanejamento das tarefas, ele deverá ser habilitado caso seja identificado uma falha de execução, esta pode ser feita de forma global ou local (replanejamento de trajetória).

Terminado o planejamento, as tarefas são executadas de forma reativa através de um conjunto de ações configuradas. Estas por sua vez realizam suas execuções sendo guiadas pelos sensores que captam o ambiente. No caso, foi utilizado um ambiente simulado através do software MobileSim, ele interage com o sistema do robô fornecendo as respostas para as ações executadas. O ambiente foi criado através do software Mapper3Basic, sendo baseado

em um ambiente real. Todas as interligações dos módulos funcionais podem ser vistas pela Figura 14 e entendidas com maior detalhe através das próximas seções deste capítulo.

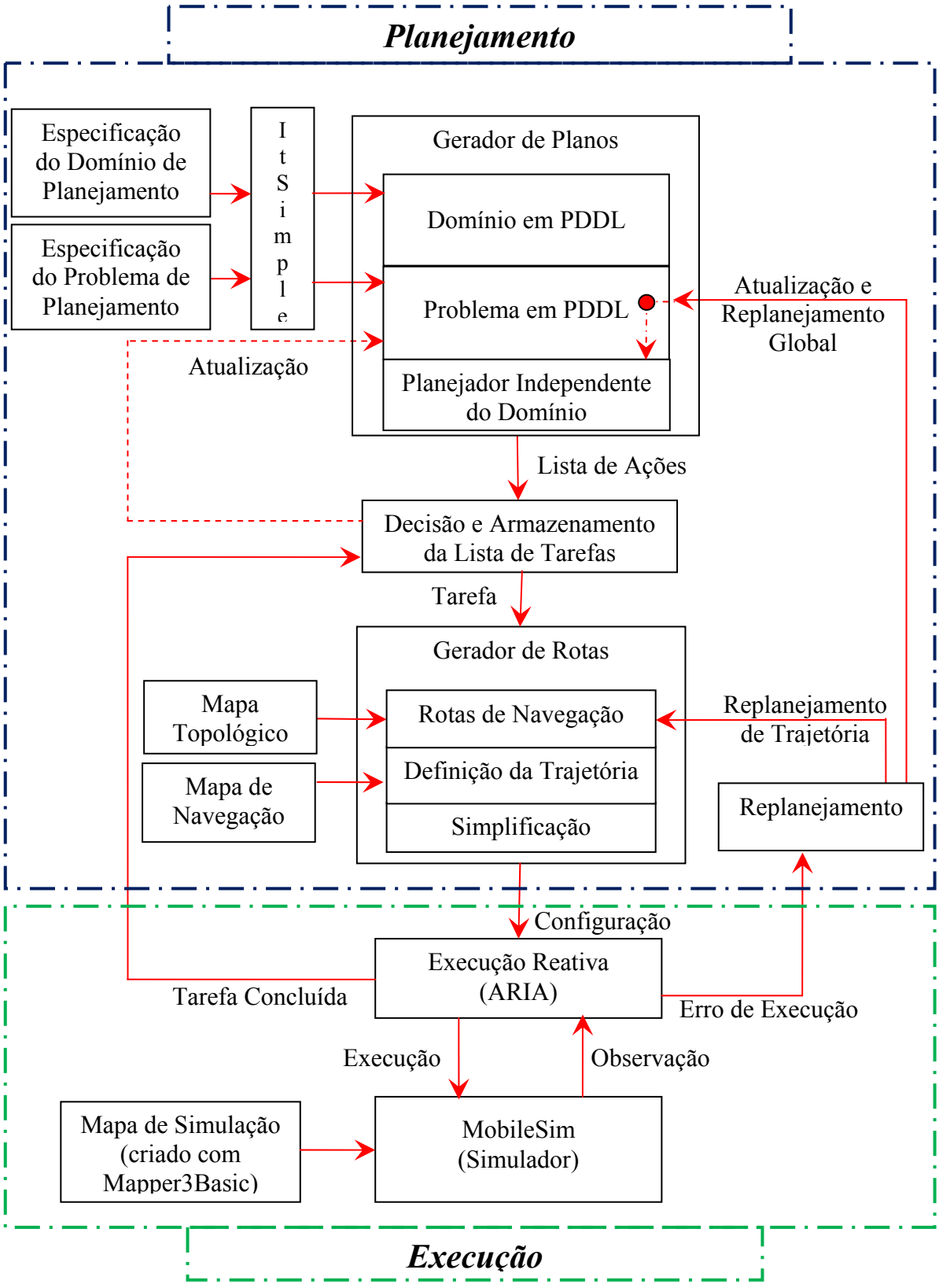

Figura 14 – Diagrama de funcionamento do Sistema de Navegação Autônomo

5.1 Planejamento – Componentes Deliberativos

No sistema desenvolvido, o planejamento da missão é o ponto fundamental para tornar o sistema inteligente, proporcionando a ele a capacidade de solucionar problemas diversos dentro do domínio modelado na sua base de conhecimento.

O planejamento na arquitetura AuRA(Arkin, 1997), utilizada como base para este trabalho, é criado nas camadas deliberativas que fazem parte da arquitetura. As camadas que se enquadram nesta categoria são: o "Planejador da Missão", o "Raciocinador Espacial" e o "Sequenciador de Planos".

O "Planejador da Missão" é o nível mais alto da arquitetura. Ela é responsável por estabelecer metas a serem alcançadas e restrições a serem seguidas, além de ser uma interface entre o usuário e o sistema.

A segunda camada deliberativa da arquitetura seria o "Raciocinador Espacial", esta camada definido inicialmente como navegador, utiliza o conhecimento cartográfico armazenado em memória para criar os passos que o robô deve seguir para completar a missão (Faria, 2006).

Já no "Sequenciador de Planos", é detalhado o plano gerado pelas camadas superiores e traduzido em um conjunto de comportamentos a serem executados. Esta é a última camada das funções deliberativas executadas pelo sistema, já que o conjunto de comportamentos gerados é executado de forma reativa.

O planejamento é o módulo de maior complexidade no sistema implementado, sendo utilizadas diversas técnicas para criação do mesmo. Além de gerar ações, este módulo precisa detalhá-los a ponto de o robô conseguir executar as missões planejadas. Para isso foram utilizadas técnicas de planejamento automático com a modelagem do domínio de execução na linguagem PDDL. O resultado deste planejamento foi detalhado através de mapas do ambiente em estruturas topológicas e celulares, sendo o resultado simplificado para adequar ao tipo de movimentação do robô. Cada técnica será comentada nas próximas seções desse capítulo.

5.1.1 Especificação do Domínio de Planejamento

O domínio de planejamento foi criado a partir de uma ferramenta de modelagem CASE (*Computer Aided Software Engineering*) chamada itSIMPLE na sua versão 2.0.30 (Vaqueiro, 2007), esta ferramenta permite a geração do código em PDDL a partir de diagramas UML (*Unified Modeling Language*). Nestes diagramas são criadas classes de elementos que compõem o domínio de planejamento, suas relações, ações e efeitos podem ser aplicadas para gerar modificações no estado atual do sistema.

O domínio criado foi baseado no domínio clássico conhecido como *Logística,* neste o objetivo é transportar pacotes entre localidades através de veículos (aviões e caminhões). No caso, os caminhos só podem se locomover dentro da mesma cidade enquanto os aviões podem se locomover entre cidades que tenham aeroportos. A Figura 15, mostra um ilustração deste domínio.

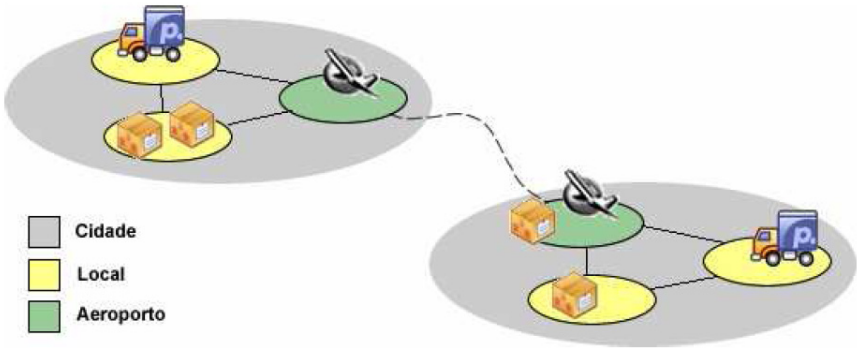

Figura 15 – Domínio clássico de planejamento Logística - (Vaqueiro, 2007)

No domínio criado os veículos utilizados são robôs e carros, onde os carros podem se locomover entre territórios e os robôs podem ser locomover entre lugares dentro do mesmo território. Esta restrição foi imposta aos robôs, pois foi considerado que seu ambiente de movimentação deveria ser interno e controlado. A Figura 16 mostra o diagrama UML de Classes que modela o domínio.

Na modelagem foi criado o agente "Carro" para permitir que entre dois territórios diferentes fosse possível o envio de pacotes, simulando um ambiente logístico mais complexo, onde, por exemplo, fábricas separadas fisicamente, mas que tenham o mesmo sistema autônomo de movimentação de carga, pudessem ser integradas no mesmo ambiente. A movimentação real do agente "Carro" não foi implementada neste trabalho, sendo inclusive, uma proposta para trabalhos futuros.

Como mostrado pelo diagrama de classes da Figura 16, existe uma classe denominada "Lugar" que pode ser instanciada como sendo um "Local" ou uma "Garagem". Esta classe é

utilizada para informar aos veículos os lugares onde podem se movimentar, sendo que os robôs têm acesso liberado a todos os lugares e os carros só podem ocupar lugares que são garagens. A localização dos veículos é informada pela relação “veiculoEstaNo”.

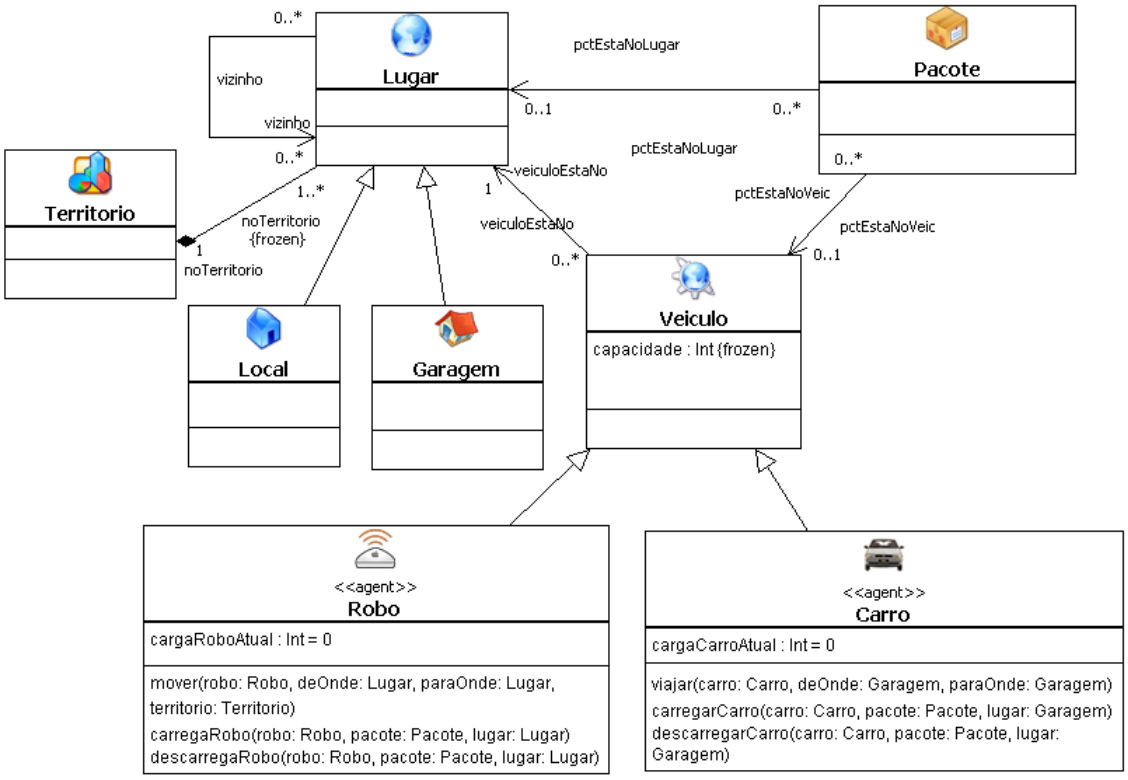

Figura 16 – Diagrama de Classes do Domínio Mundo CRTi

Uma modificação mais profunda em relação ao domínio *Logística,* mas que é essencial para o funcionamento do mundo criado, é a relação chamada de “vizinho” entre os elementos da classe “Lugar”. Esta relação permite a criação de um mapa de vizinhança dentro da modelagem, informando ao robô todos os lugares que ele precisa passar para chegar a um determinado local alvo.

Os pacotes são criados através da classe “Pacote”. Esta classe apresenta uma relação de “pctEstaNoLugar” com a classe “Lugar” e uma relação de “pctEstaNoVeic” com a classe “Veiculo”, estas relações informam ao mundo qual é a localização do pacote no ambiente de movimentação.

Pela modelagem é necessária a criação de um território através da classe “Territorio”, para informar ao sistema qual mapa ele deve se referenciar. O território de cada lugar é informado ao sistema pela relação “noTerritorio”.

Como mencionado anteriormente, os integrantes da classe “Veiculo” podem ser instanciados como sendo um “Robo” ou um “Carro”. Todos os veículos apresentam um atributo de “capacidade” do tipo inteiro que informam a quantidade máxima de pacotes permitida ao embarque.

Os robôs além de apresentarem um atributo de “cargaRoboAtual”, informando a quantidade de pacotes que estão embarcados naquele momento, ainda dispõem de três operações que podem ser realizadas, modificando o estado atual do mundo, são elas:

- mover: esta ação realiza a movimentação de um robô que está em um lugar denominado “deOnde”, para um lugar “paraOnde”, isso caso estes lugares estejam no mesmo território, forem vizinhos e o robô realmente se encontrar no lugar de origem. Esta ação provoca a mudança do local onde se encontra o robô.

- carregaRobo: esta ação realiza o carregamento de um pacote no robô, desde que a capacidade máxima não tenha sido atingida e o robô e o pacote estejam no mesmo lugar. Ela provoca o incremento da carga atual, e a mudança do local a onde está o pacote, de um “Lugar” para um “Robo”.

- descarregaRobo: esta ação realiza o descarregamento de um pacote do robô, desde que o pacote esteja realmente carregado no robô. Ela provoca o decremento da carga atual, e a mudança do local a onde está o pacote, do “Robo” para um “Lugar”.

Os carros também apresentam um atributo informando a quantidade de pacotes que estão embarcados chamado “cargaCarroAtual” e também dispõem de três operações:

- viajar: esta ação realiza a movimentação de um carro que está em uma garagem denominado “deOnde”, para uma garagem “paraOnde”, isso caso o carro realmente se encontrar na garagem de origem. Esta ação provoca a mudança da garagem onde se encontra o carro.

- carregarCarro: esta ação realiza o carregamento de um pacote no carro, desde que a capacidade máxima não tenha sido atingida e o carro e o pacote estejam no mesmo lugar. Ela provoca o incremento da carga atual, e a mudança do local a onde está o pacote, de um lugar para um carro.

- descarregarCarro: esta ação realiza o descarregamento de um pacote do carro, desde que o pacote esteja realmente carregado no carro. Ela provoca o decremento da carga atual, e a mudança do local a onde está o pacote, do carro para um lugar.

No software utilizado para a modelagem do sistema a descrição das pré-condições e das alterações provocadas por cada ação é descrita por Diagramas de Estados, conforme se pode observar nas Figura 17, Figura 18 e Figura 19.

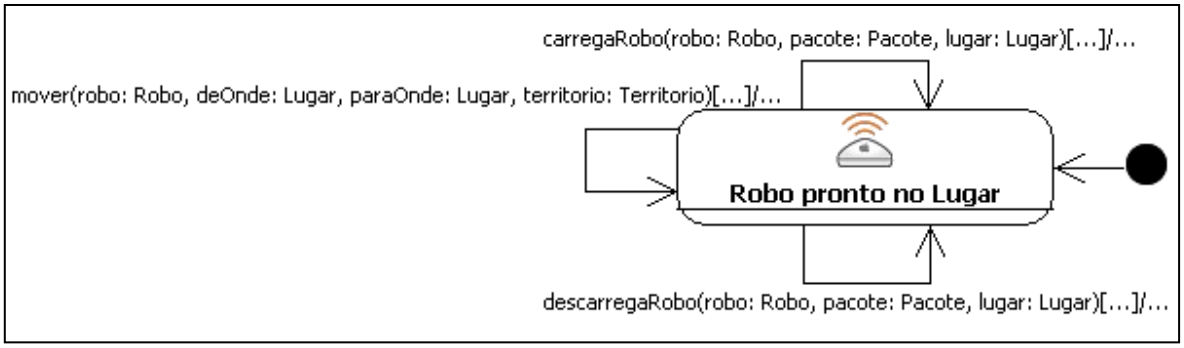

Figura 17 – Diagrama de Estado da Classe Robô

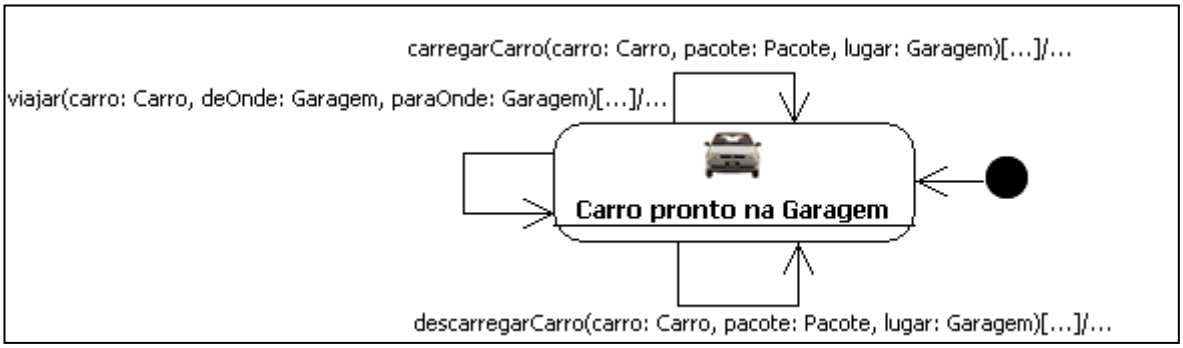

Figura 18 – Diagrama de Estado da Classe Carro

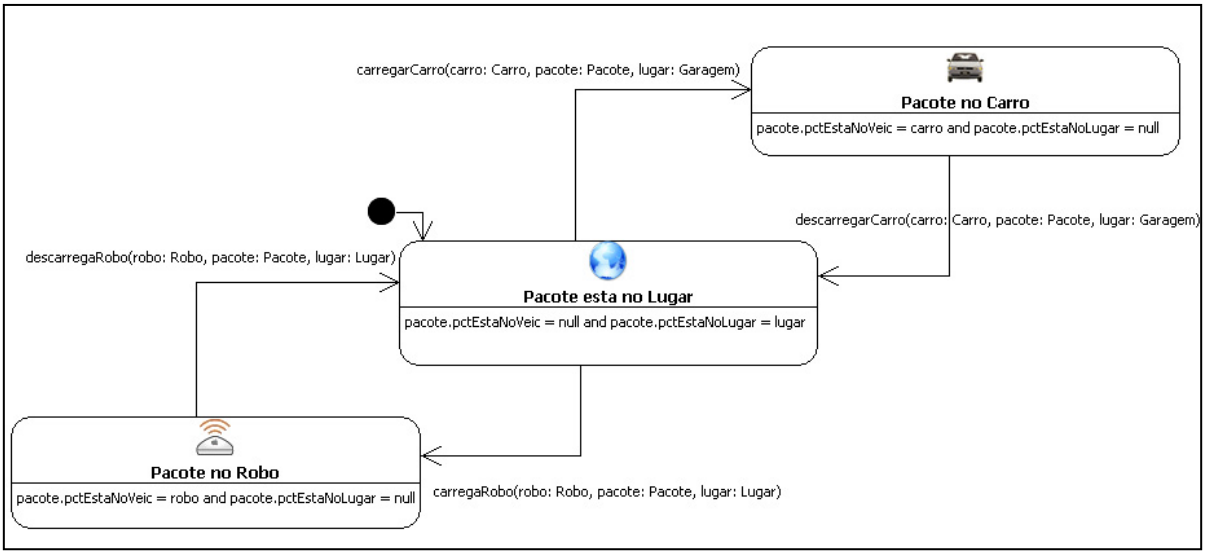

Figura 19 – Diagrama de Estado da Classe Pacote

5.1.2 Especificação do Problema de Planejamento

Após a modelagem do domínio na qual foram definidas todas as entidades pertencentes ao mundo, suas inter-relações, condições e operações, é necessário instanciar os elementos pertencentes a ele, isto é, é necessário criar os elementos que fazem parte deste mundo. Para isso o software itSIMPLE (Vaqueiro, 2007), apresenta um modulo específico para criação de problemas, neste módulo é possível criar elementos das classes definidas e suas interligações. Como pode ser observado no exemplo da Figura 20.

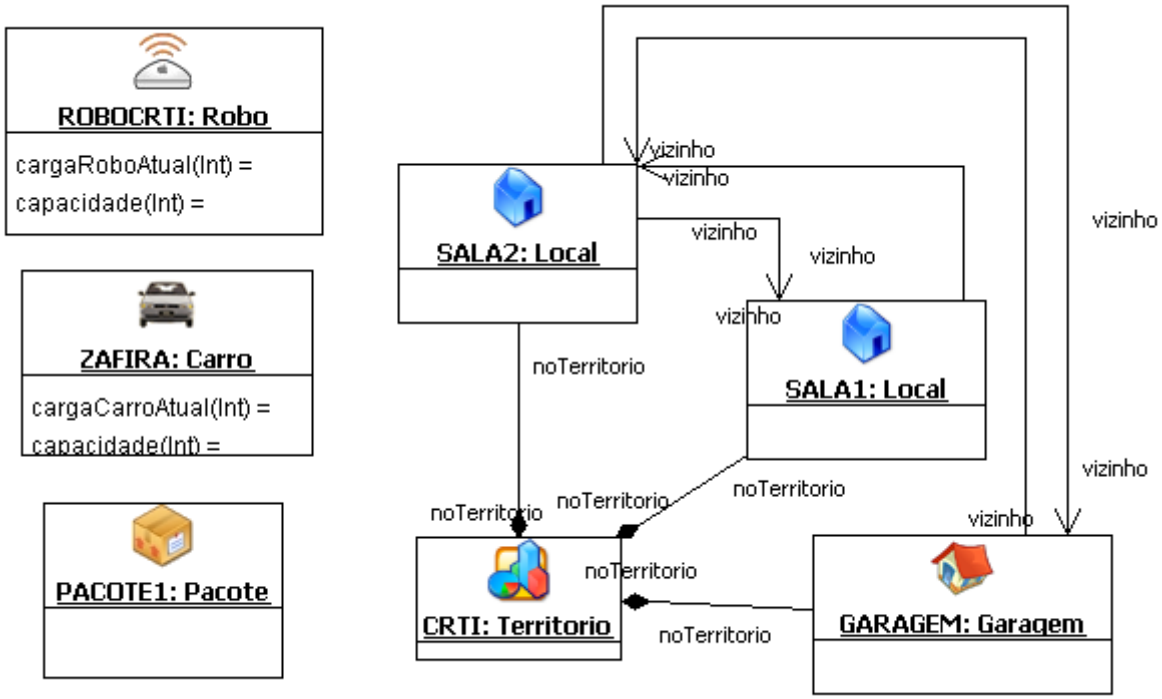

Figura 20 – Diagrama com os objetos criados no problema exemplo

Neste exemplo foram instanciados um robô chamado “RoboCRTi”, um carro chamado “Zafira”, um pacote (“Pacote1”), três lugares sendo dois locais (“Sala1” e “Sala2”) e uma garagem (“Garagem”), um território chamado “CRTi” e diversas relações entre os elementos. Por exemplo, existe uma relação de “noTerritorio” entre os lugares e o território “CRTi”, existem relações de vizinhança entre a “Sala2” e a “Garagem” sendo que uma delas informa que a “Sala2” é vizinha da “Garagem” e outra informando que a “Garagem” é vizinha da “Sala2”, isto permite que seja possível uma ação de “mover” do robô em ambos os sentidos, saindo da “Sala2” para a “Garagem” e da “Garagem” para a “Sala2”. A relação de vizinhança

foi definida desta forma para permitir que, caso necessário, fosse restringida a passagem do robô em um sentido de uma sala a outra.

No diagrama da Figura 20, foram colocadas apenas as relações que não se alteram no decorrer do problema, formando as condições para a execução do mesmo.

Para que seja possível a realização do planejamento das ações é preciso definir um estado inicial para o restante dos elementos e um estado final para os elementos que fazem parte do objetivo final a ser alcançado.

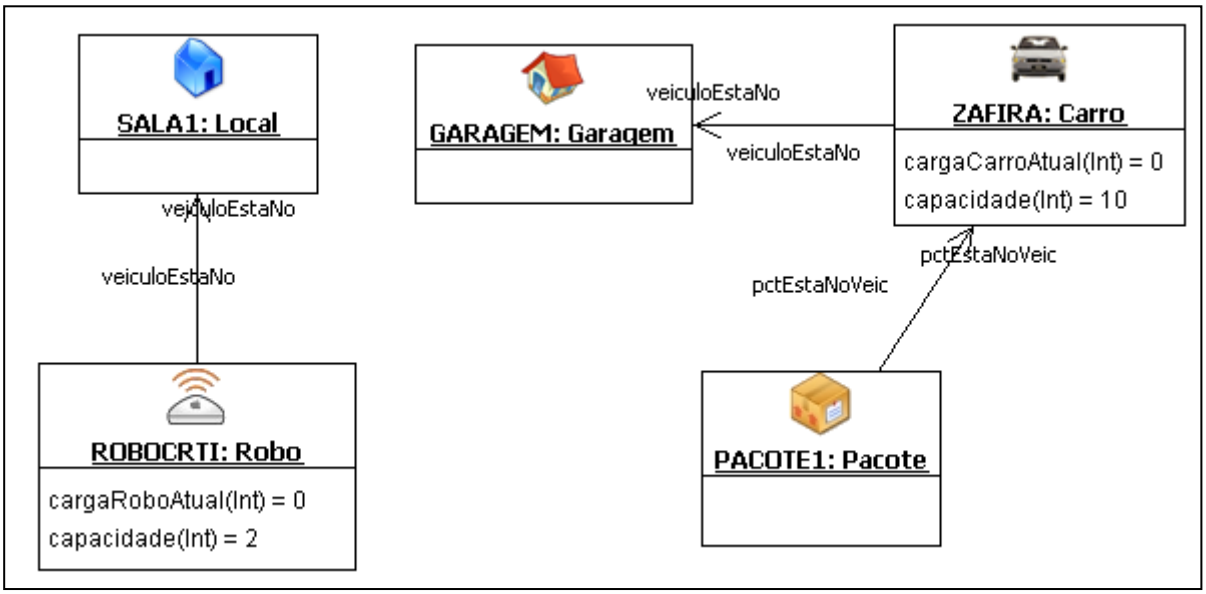

Figura 21 – Diagrama com os estados iniciais do problema

Na Figura 21 acima pode-se observar o restante das condições iniciais do problema, que seriam: o “RoboCRTi” está na “Sala1”, o “Zafira” está na “Garagem” e o “Pacote1” esta embarcado no “Zafira”.

Como mostra a Figura 22, o objetivo deste problema exemplo é levar o “Pacote1” para a “Sala1”.

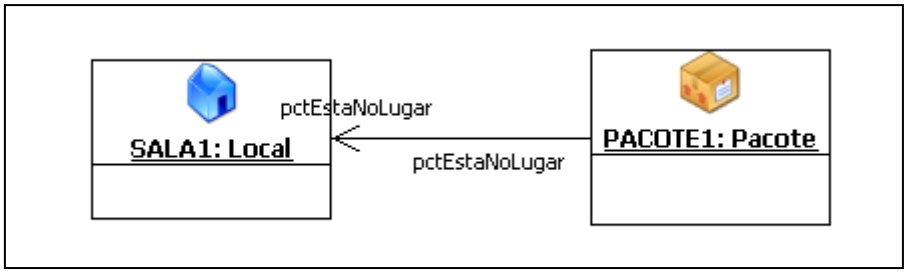

Figura 22 – Diagrama com o objetivo a ser alcançado

5.1.3 Gerador de Planos

Com o término das definições do domínio e do problema de planejamento, comentados nas seções anteriores, o *software* itSimple (Vaqueiro, 2007) traduz os diagramas UML para a linguagem de planejamento PDDL. As traduções dos diagramas gerados pelo exemplo do capítulo 5.1.2 encontram-se no Apêndice A.

A versão do software utilizada consegue traduzir os diagramas para a versão 2.1, 2.2 e 3.0 da linguagem PDDL, neste trabalho foi utilizada a conversão para a linguagem 2.2, pois ela se mostrou mais estável com o *software* itSimple, gerando os resultados esperados. Ocorreram eventos durante os testes onde o mesmo domínio e problema não geraram resultados quando foi utilizada a linguagem PDDL na versão 3.0. Este fato não ocorreu quando utilizada a versão 2.2.

Após a tradução dos diagramas é necessário escolher um programa planejador para executar a busca por uma lista de ações, um plano, que realize o objetivo esperado mediante as condições iniciais. O planejador escolhido foi o *FF* (*Fast-Forward*) (Keyder, 2008), pois dentre as escolhas dos planejadores contidos no software *itSimple*, com o sistema operacional utilizado, foi o de melhor desempenho.

Rodando o domínio gerado com o problema exemplo foi obtida a seguinte lista de ações a serem tomadas:

0: (MOVER ROBOCRTI SALA1 SALA2 CRTI)[1]
1: (MOVER ROBOCRTI SALA2 GARAGEM CRTI)[1]
2: (DESCARREGARCARRO ZAFIRA PACOTE1 GARAGEM)[1]
3: (CARREGARROBO ROBOCRTI PACOTE1 GARAGEM)[1]
4: (MOVER ROBOCRTI GARAGEM SALA2 CRTI)[1]
5: (MOVER ROBOCRTI SALA2 SALA1 CRTI)[1]
6: (DESCARREGAROBO ROBOCRTI PACOTE1 SALA1)[1]

Nesta lista estão contidas todas as ações a serem tomadas para a realização do objetivo desejado, cabe agora ao restante do sistema desenvolvido detalhar melhor a lista de ações, para que o ambiente de simulação possa executar as ações.

5.1.4 Decisão e Armazenamento da Lista de Tarefas

Esta estrutura tem a responsabilidade de armazenar a lista de ações geradas enviando para o próximo estágio, "Gerador de Rotas de Navegação", as que necessitem ser detalhadas para execução no simulador (movimentação do robô). As demais tarefas, modeladas no domínio de planejamento, não são utilizadas para a execução no simulador, sendo somente enviada uma mensagem de confirmação da realização da mesma.

Além disso, esta estrutura tem a função de otimizar o planejamento das execuções, pois só um tarefa é enviada de cada vez para o "Gerador de Rotas de Navegação". Este fato é importante, pois podem ocorrer eventos de replanejamento global durante a execução, descartando o restante das tarefas não realizadas. Isso impede que sejam detalhadas tarefas que podem ser descartadas posteriormente.

Cada vez que é finalizada uma tarefa, ela é apagada da lista e é atualizado o arquivo "problema de planejamento" com os efeitos causados por ela. Isso mantém o sistema sempre com as informações mais atuais. Além disso, começa o tratamento da próxima tarefa da lista.

5.1.5 Gerador de Rotas de Navegação

Esta estrutura é responsável por detalhar as tarefas que serão executadas no simulador, tarefas de movimentação do robô, configurando as execuções reativas utilizadas para realização das mesmas.

A lista de ações gerada pelo planejador é genérica demais para que o robô consiga executá-la, sendo necessário o detalhamento da mesma fornecendo objetivos claros para o robô. Para isso o sistema se utiliza do conhecimento que ele tem do ambiente em que está inserido para traduzir a lista de tarefas em informações úteis para a execução do plano.

5.1.5.1 Rotas de Navegação

Através das relações de vizinhança atribuídas às entidades do tipo "Local" pela diretiva chamada de "vizinho", ver capítulo 5.1.2, pode-se retirar um mapa de rotas que represente o ambiente utilizado, este mapa é conhecido como modelo topológico do ambiente.

Continuando o exemplo do capítulo 5.1.2, existem neste exemplo quatro relações de vizinhança, como mostra a Figura 23.

Figura 23 – Diagrama com as relações de vizinhança do exemplo do capítulo 5.1.2

Estas relações de vizinhança no domínio criado e convertido na linguagem PDDL seriam representadas pelas seguintes afirmações:

(vizinho SALA1 SALA2) (1)
(vizinho SALA2 SALA1) (2)
(vizinho GARAGEM SALA2) (3)
(vizinho SALA2 GARAGEM) (4)

A afirmação em (1), "(vizinho SALA1 SALA2)", significa que existe uma rota que saindo da "Sala1" chega a "Sala2". Para que o robô possa entender o que deve fazer com esta informação de vizinhança é necessário atribuir uma coordenada do mapa do ambiente a esta relação, esta coordenada indicará o ponto de transição entre os locais, por exemplo:

SALA1 => SALA2 x= 5030 y= 41775 (5)
SALA2 => SALA1 x= 3030 y= 42475 (6)

Com esta relação entre coordenadas do mapa e vizinhança, o robô consegue interpretar o seu objetivo, por exemplo, em um comando de movimentação:

0: (MOVER ROBOCRTI SALA1 SALA2 CRTI)[1] (7)

Este comando informa ao robô que ele vai se movimentar do ponto em que está dentro da "Sala1" (ponto conhecido pelo robô) para a "Sala2". Aplicando-se a afirmação em (5) temos que o objetivo dele é sair do ponto em que está para chegar ao ponto com coordenadas x = 5030 e y = 41775 do mapa do ambiente. Como isso o robô consegue extrair seu objetivo dentro de um comando de movimentação.

Este modelo topológico captura as relações de adjacências do ambiente compilando-as em uma estrutura de grafo, facilitando sua manipulação e procura por informações.

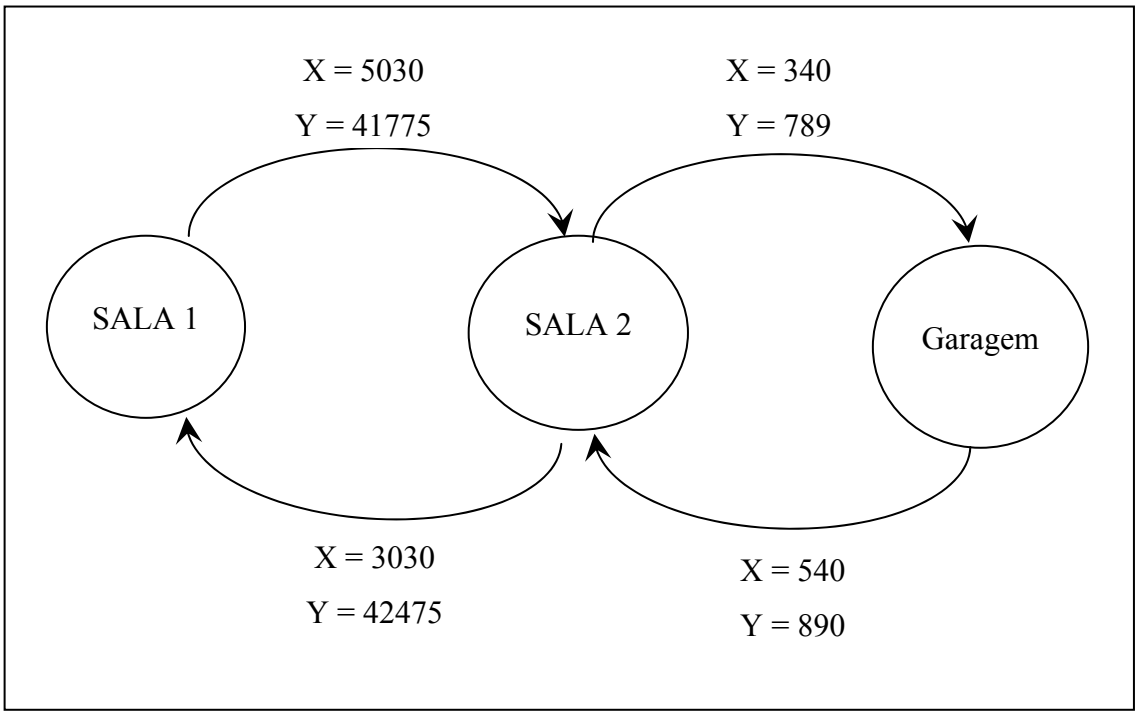

Figura 24 – Grafo de Rotas do exemplo do capítulo 5.1.2

Para os comandos de carregar e descarregar robô, foram criados dentro do banco de dados de "Mapas de Rotas" pontos que seriam "zonas de carga e descarga" em uma sala. Exemplo:

GARAGEM x= 2873 y= 43878 (8)

Isto significa que para realizar o comando do exemplo:

3: (CARREGARROBO ROBOCRTI PACOTE1 GARAGEM)[1] (9)

O robô teria que se movimentar até o ponto de carregamento para executar o embarque do pacote. A execução do comando em (9) com a afirmação em (8) contida no seu banco de dados seria: o robô está em um ponto conhecido na sala "Garagem" e precisa se deslocar até o ponto contido em (8), x= 2873 y= 43878, para realizar o embarque do "Pacote1".

Com esta metodologia o sistema consegue gerar um objetivo de movimentação válido para o robô, mas ainda não consegue realizar a ação de mover de forma satisfatória, pois o plano ainda não foi detalhado o suficiente. Para isso será utilizada uma técnica para detalhamento da trajetória a ser executada, descrita na próxima seção.

5.1.5.2 Definição da Trajetória de Movimentação

Com as informações que o robô tem até o presente momento, ele até consegue se movimentar pelo ambiente, mas de forma caótica e não otimizada, por exemplo:

Considere que o robô está em um lugar como mostrado pela Figura 25 e que ele precise executar uma ação de movimentação para alcançar uma sala adjacente. Ele irá obter a posição de saída através da verificação do mapa de rotas e poderá realizar uma movimentação até aquele ponto alvo.

Caso ele não tenha nenhuma informação de como é este ambiente, um mapa que indique os obstáculos do lugar, ele irá calcular uma trajetória, provavelmente retilínea, que poderá colidir com obstáculos presentes no ambiente, impedindo sua movimentação.

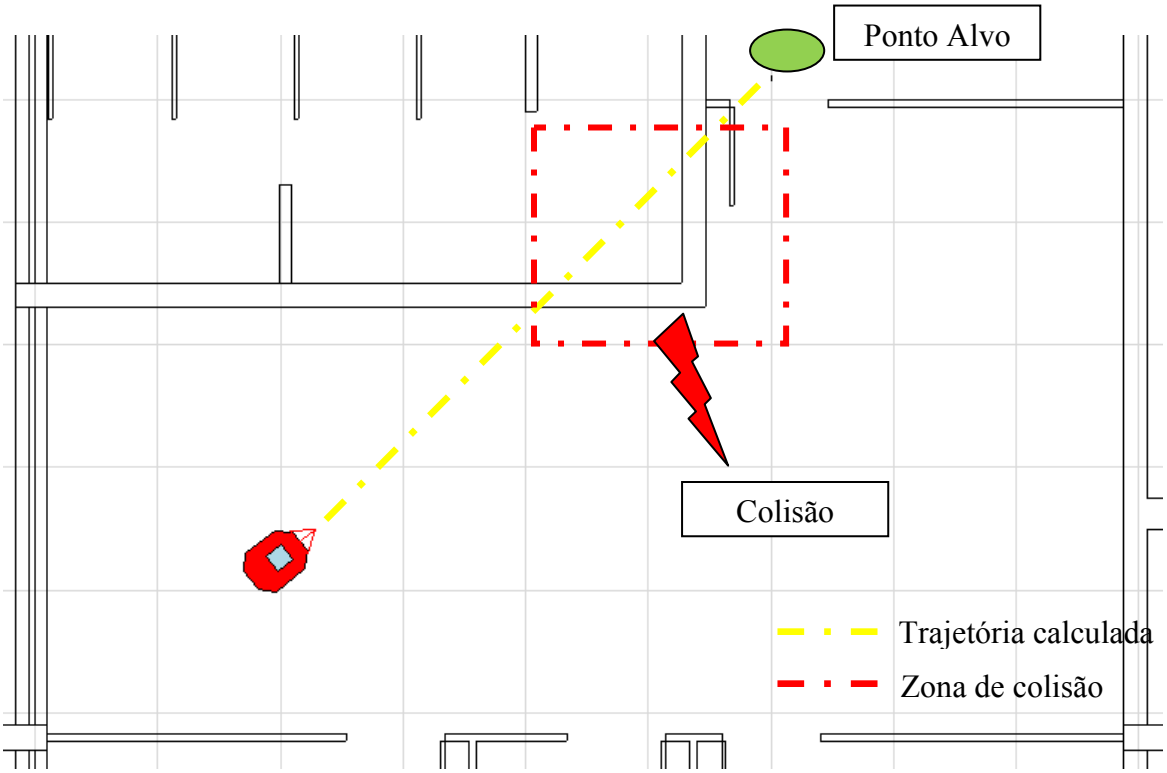

Figura 25 – Exemplo de movimentação sem determinação da trajetória

Caso o robô tenha configurado em seu programa uma rotina de desvio de obstáculos, mesmo assim ele irá perder muito tempo até que consiga desviar dos obstáculos não planejados e termine a execução desta ação.

Mediante estas dificuldades, um mapa do ambiente e uma estratégia para descobrir uma trajetória de movimentação adequada e otimizada são necessários para possibilitar a execução das tarefas de forma mais inteligente.

- *Mapa de navegação*

Para a representação do mapa de navegação foi utilizado o método de decomposição em células fixas (Siegwart, 2004), conhecida como modelagem por enumeração.

A modelagem por enumeração consiste na criação de uma matriz de células de tamanho fixo. Este é um método aproximado, onde a resolução está diretamente relacionada com o tamanho da célula. Células grandes reduzem a complexidade, mas promovem a perda de espaço livre, enquanto que células pequenas permitem uma modelagem mais acurada e, consequentemente, o aumento da complexidade (Pieri, 2002).

Segundo (Siegwart, 2004), a desvantagem da decomposição é a imediata perda de fidelidade entre o mapa e o mundo real. Tanto qualitativamente, em termos de estrutura global, quanto quantitativamente, em termos de precisão geométrica. Apesar desta desvantagem, a decomposição pode ser útil, já que capta as características relevantes do ambiente, enquanto rejeita todas as outras características menos relevantes, minimizando a complexidade da representação. Computacionalmente o desempenho do tratamento de um mapa decomposto é muito superior ao planejamento em um modelo de mundo totalmente detalhada.

Esta representação transforma um ambiente real contínuo em uma aproximação discreta. Essa transformação é demonstrada na Figura 26, que mostra o que acontece com uma área ocupada por um obstáculo e uma área livre durante esta transformação. A principal desvantagem desta abordagem resulta na sua natureza inexata. É possível que passagens estreitas possam ser perdidas durante essa transformação, como mostrado a Figura 26. Formalmente, isto significa que decomposição fixa é boa, mas não completa (Siegwart, 2004).

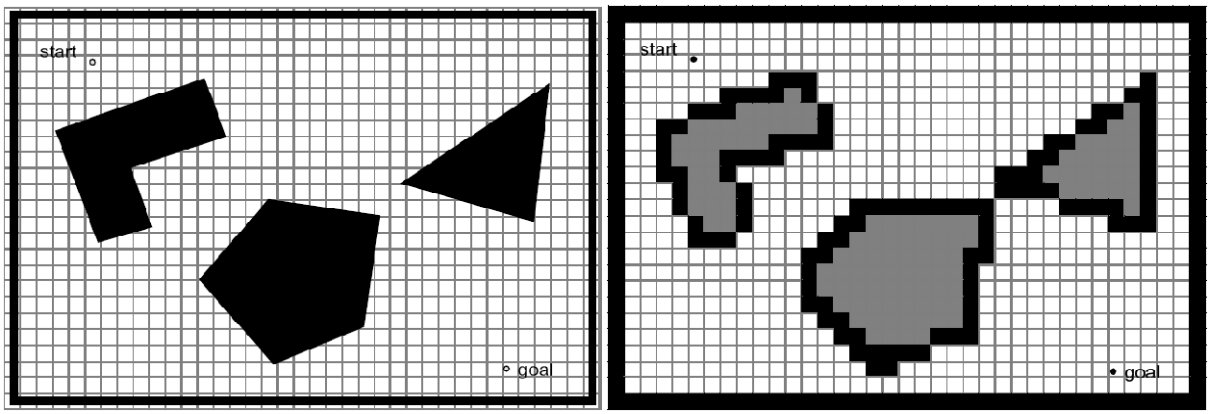

Figura 26 – Exemplo de mapa decomposto por célula - (Siegwart, 2004)

O método é bastante simples, principalmente para acessar um determinado ponto no ambiente. A cada célula é atribuída valor binário para a ocupação (1 para células com obstáculos e 0 para células vazias). Os diferentes locais do ambiente são distinguidos baseando-se na posição geométrica do robô, com relação ao sistema de coordenadas global. A posição do robô é estimada incrementalmente, baseando-se na informação dos sensores de odometria. A Figura 26 ilustra esta metodologia. O robô é considerado posicionado em uma célula caso seu centro esteja dentro da área da célula.

A Figura 27, mostra o mapa do ambiente utilizado para as simulações. Ele apresenta um tamanho de célula de 350x350mm e um tamanho total de 88x133 células. Este tamanho de célula foi escolhido considerando que as portas das salas do ambiente deveriam ser capazes de armazenar pelo menos uma célula inteira, caso contrário poderia existir portas que não seriam representadas no mapa por suas células estarem parcialmente ocupadas. Deste modo garante-se que existirá pelo menos uma célula livre em cada passagem a qual o robô conseguiria entrar.

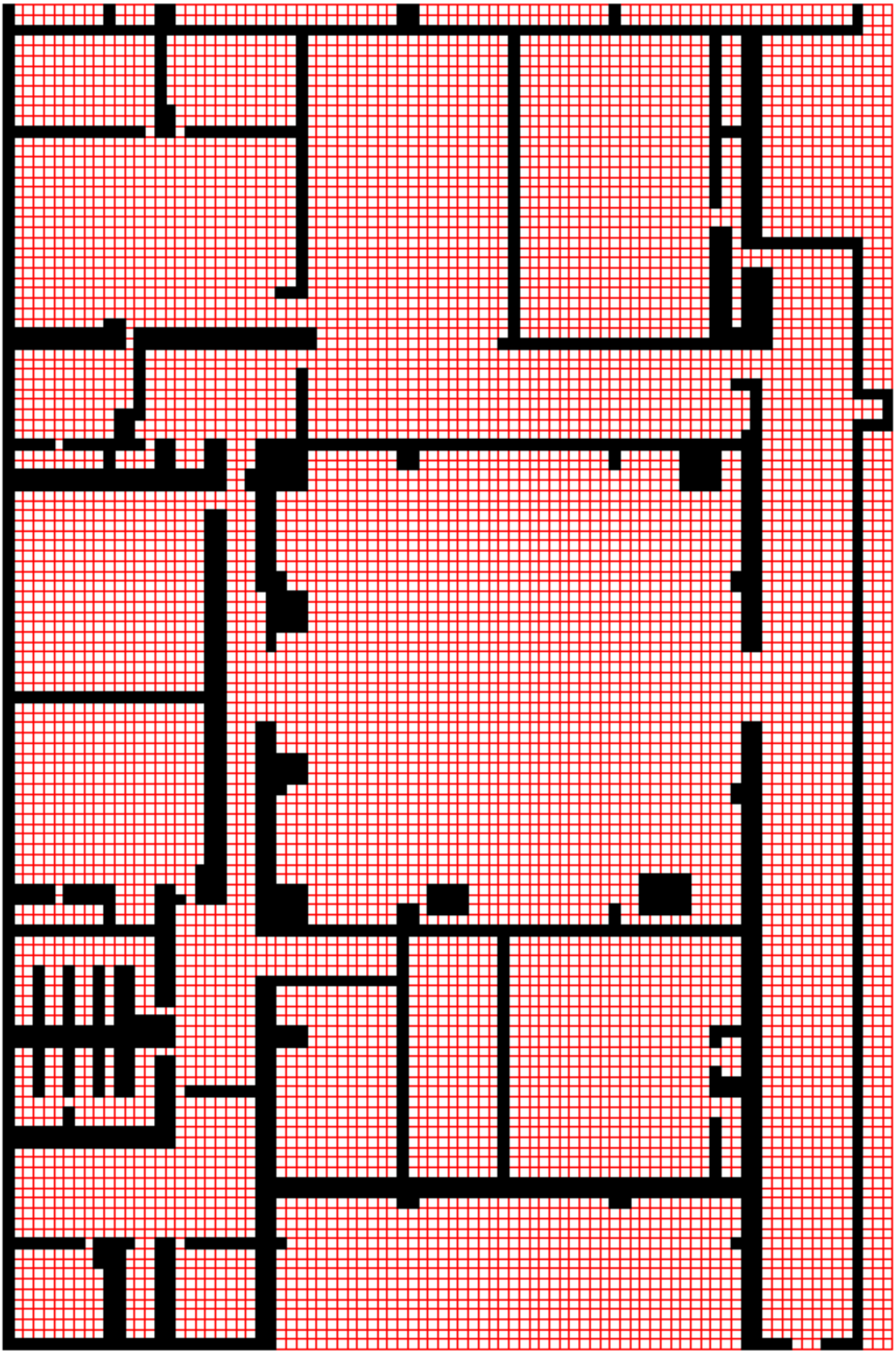

Figura 27 – Mapa Enumerado do ambiente de simulação

- *Escolha da melhor trajetória*

Tendo um mapeamento das informações contidas no ambiente, pelo menos uma boa estimativa delas, o problema agora é escolher um conjunto de células do mapa do robô que o leve a seu destino final de forma segura e otimizada. Para isso foi utilizado o algoritmo de busca conhecido como A* pronunciado (A-estrela), que se utiliza de heurísticas para o direcionamento das buscas.

Para se trabalhar com este algoritmo é necessário a discretização do mapa do ambiente, isto é, é necessário dividi-lo em células, reduzindo a área de busca a uma ordem bi-dimensional. Esta etapa já foi realizada e descrita na seção anterior.

Outra necessidade do algoritmo é o conhecimento das localizações de origem e destino, além de uma relação entre elas para gerar a heurística de otimização de busca.

O A* é um dos mais famosos algoritmos de busca *Best-First Search* (Russell, 2003). Nesta abordagem um nó de busca (célula do mapa) é selecionado para expansão com base em uma função de avaliação *f(n).* Tradicionalmente, o nó com o menor valor de função de avaliação é selecionado para a expansão, pois a avaliação mede a distância do nó atual a meta, este fato é o que gerou o nome da metodologia que em português seria "o melhor primeiro".

No caso do A* a função de avaliação é composta por:

$$f(n) = g(n) + h(n) \qquad \textbf{Eq. 1}$$

Onde: $g(n)$ fornece o custo do deslocamento do nó inicial ao nó corrente n.

$h(n)$ fornece o custo estimado através de uma heurística do menor caminho do nó corrente n ao objetivo final

Segundo (Russell, 2003), o A* é uma metodologia que gera o caminho ótimo, isto é, o de menor custo possível, desde que a heurística adotada para a estimação da função $h(n)$ seja admissível. Uma heurística $h(n)$ é admissível caso seu custo nunca seja superestimado em relação ao custo real, isto é, caso o custo estimado pela heurística nunca seja maior que o custo real para se alcançar o objetivo. Caso contrário, a metodologia poderá gerar trajetórias mais longas que as necessárias.

Para garantir que o caminho gerado pelo sistema criado fosse ótimo, foi utilizado à distância em linha reta da célula corrente a célula alvo como heurística da função $h(n)$, desconsiderando possíveis desvios na trajetória por obstáculos no percurso. Com isso é

garantido que a trajetória real sempre será maior ou pelo menos igual ao valor estimado por *h(n),* já que a menor distância entre dois ponto é uma reta.

A metodologia A*, segundo (Lester, 2005), consiste em se verificar as células do mapa para encontrar o melhor caminho entre duas posições distintas. A verificação é guiada por uma heurística para otimização da quantidade de células a serem observadas e começa a partir da célula inicial, sendo verificada sua vizinhança (células adjacentes). Para cada célula passível de observação é armazenada a célula anterior a ela ("célula pai") e é calculada a função de custo, sendo estas células armazenadas em uma lista de verificação ("lista aberta"). A que tiver o menor valor de custo será a próxima a ter a vizinhança verificada, sendo esta adicionada a uma lista de células que não precisarão mais ser observadas ("lista fechada"). Esse processo termina quando é armazenada na "lista fechada" a célula alvo (célula final), sendo o caminho encontrado determinado pelo parentesco, a partir da célula alvo, entre as células observadas. O parentesco é determinado pelo dado "célula pai" contido em todas as células verificadas.

Para que o robô consiga se movimentar de uma posição inicial para uma posição objetivo, basta ele se mover de centro a centro das células do caminho encontrado.

Caso em algum momento a "lista aberta" fique vazia, não existirá caminho que ligue a célula inicial a final, sendo terminado o processo de verificação.

Em (Lester, 2005), há um algoritmo resumo da metodologia mostrando cada operação a ser realizada de forma passo a passo, este algoritmo segue abaixo:

Algoritmo 1 - Método A * segundo (Lester, 2005)

1) *Adicione o quadrado inicial à lista aberta.*
2) *Repita o seguinte:*
    a. *Procure o quadrado que tenha o menor custo de F na lista aberta. Este quadrado é chamado de quadrado corrente*
    b. *Mova-o para a lista fechada.*
    c. *Para cada um dos quadrados adjacente a este quadrado corrente:*

        i. *Se não é passável ou se estiver na lista fechada, ignore-o. Caso contrário faça o seguinte:*

1. Se não estiver na lista aberta, acrescente-o à lista aberta. Faça o quadrado atual o pai deste quadrado. Grave os custos F, G, e H do quadrado.

2. Se já estiver na lista aberta, confira para ver se o caminho para aquele quadrado é melhor, usando o custo G como medida. Um valor G mais baixo mostra que este é um caminho melhor. Nesse caso, mude o pai do quadrado para o quadrado atual, e recalcule os valores de G e F do quadrado. Caso a lista aberta esteja sendo ordenada pelo valor de F, ela necessitará ser reordenar.

d. Pare quando:

i. Acrescentar o quadrado alvo à lista fechada o que determina que o caminho foi achado, ou

ii. Não ache o quadrado alvo, e a lista aberta está vazia. Neste caso, não há nenhum caminho.

3) Salve o caminho. Caminhando para trás do quadrado alvo, vá de cada quadrado a seu quadrado pai até que seja alcançado o quadrado inicial. Este é o caminho encontrado pelo algoritmo.

Fim do Algoritmo.

No apêndice B encontra-se um exemplo do algoritmo A* baseado na metodologia descrita em (Lester, 2005).

Como visto nesta seção do trabalho, através do A* é possível encontrar o melhor caminho a se seguir para alcançar um objetivo de movimentação, este caminho é ótimo se a heurística adotada para computar os custos é admissível. Ainda, para que seja possível a utilização da metodologia, é necessário discretizar o mapa do ambiente em células e se ter o conhecimento da localização inicial e final do robô.

Utilizando-se esta metodologia, o problema ilustrado pela Figura 25 pode ser solucionado. O robô com um conhecimento prévio do ambiente pode planejar a realização de

desvios de obstáculos otimizando o tempo de execução das tarefas e diminuindo o índice de falhas (colisões e não cumprimento das missões). Isso pode ser visto pela Figura 28.

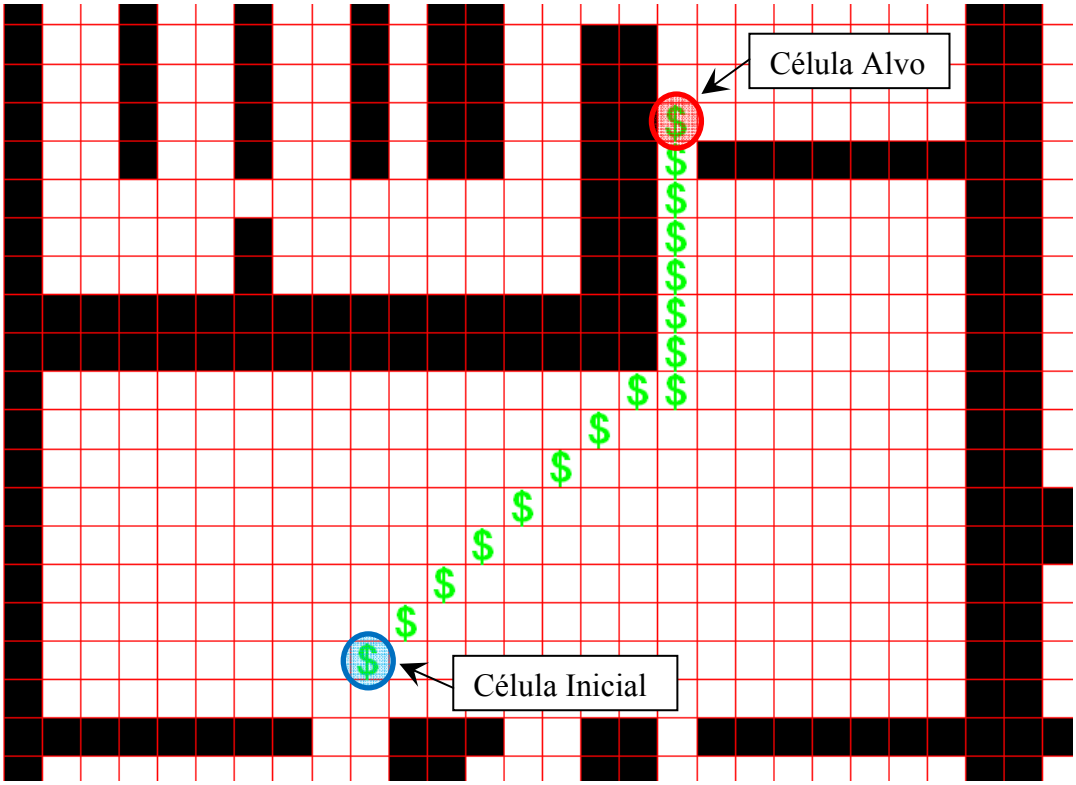

Figura 28 – Caminho encontrado pelo A* em um teste realizado

Na Figura 28 é mostrado o caminho encontrado pelo algoritmo A* para movimentação do robô em um ambiente do mapa criado, o símbolo $ indica as células selecionadas como caminho a ser seguido, o círculo azul a célula inicial e o vermelho a célula alvo da movimentação.

5.1.5.3 Simplificação do Caminho Encontrado

Com a adição da metodologia de busca A* as técnicas utilizadas e comentadas por este trabalho, conseguiu-se detalhar o plano gerado. Esse sofreu a adição de informações cartográficas do "*Mapa de Rotas*", gerando um conjunto de ponto ou células, que se seguidas fazem com que o robô cumpra uma tarefa de movimentação planejada.

Apesar de conseguir se movimentar, o robô ainda não o faz de maneira satisfatória, pois seguindo o resultado gerado pelo A* ele iria se movimentar de célula a célula, tendo que

calcular toda sua trajetória (acelerações e velocidades) para pequenos trechos, tornando tal tarefa muito custosa e complicada. A Figura 29 ilustra este fato.

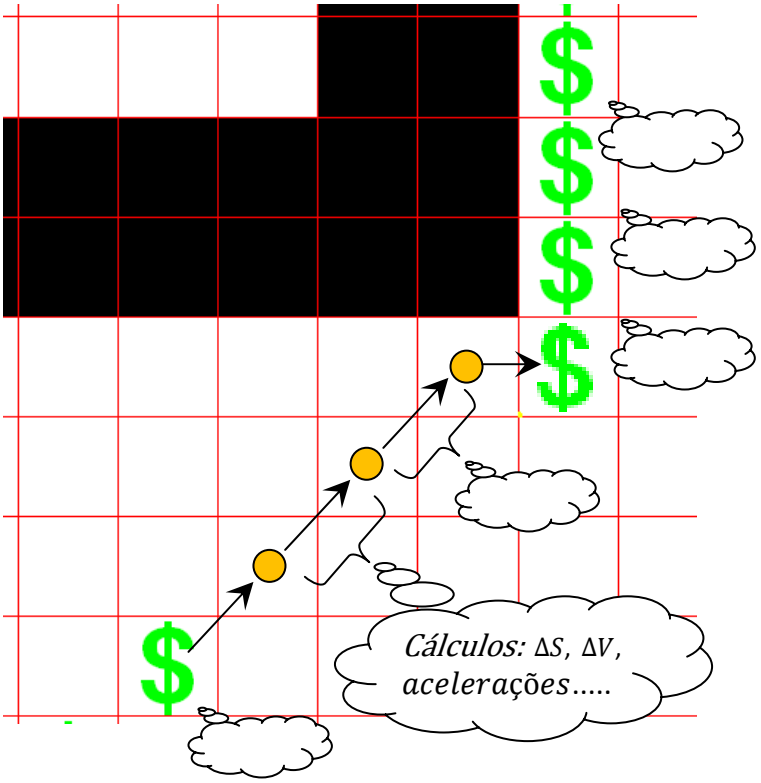

Figura 29 – Movimentação sem utilização de simplificação

Para simplificar a operação de movimentação, foi desenvolvida uma rotina de simplificação do caminho encontrado pelo A*, esta rotina se baseia no axioma geométrico que diz que "dois pontos distintos determinam uma única reta".

O conceito desta simplificação é encontrar pontos consecutivos que estejam dentro de uma mesma reta, com isso esta trajetória retilínea com múltiplos pontos poderia ser simplificada por apenas dois pontos, o ponto inicial e o final da reta. Conforme mostra a Figura 30, onde se encontra a mesma trajetória da Figura 28 de forma simplificada.

A implementação desta rotina de simplificação é bem simples, o mapa do ambiente é representado por uma matriz de ocupação, sendo que a posição de cada célula é apresenta por dois valores, X e Y (posição do valor da célula na matriz). Estes valores de X e Y de cada célula da trajetória são armazenados em um vetor de números chamado de "Vetor de Posição" (ver Tabela 2). Outro vetor de números chamado "Vetor de Coeficientes" é criado para armazenar os valores dos seguintes cálculos:

$$X_c - X_{c-1} \quad \text{e} \quad Y_c - Y_{c-1} \qquad \textbf{Eq. 2}$$

Sendo c o índice das células nos vetores.

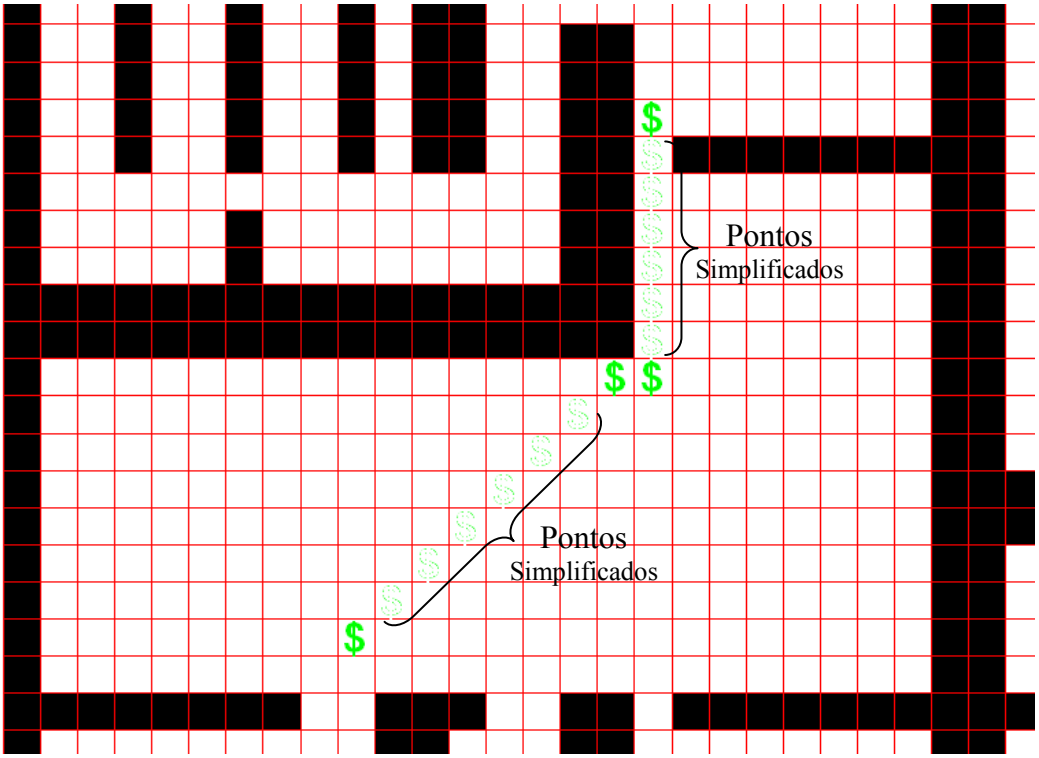

Figura 30 – Movimentação com utilização de simplificação

Com isso foi criado um vetor que armazena a variação entre a posição de duas células adjacentes, sendo que como a primeira célula do "Vetor de Posição" não apresenta uma célula anterior, o primeiro valor do "Vetor de Coeficientes" é nulo.

Para determinação das células que podem ser simplificadas, basta comparar os valores contidos no "Vetor de Coeficientes" a partir da segunda célula (c = 2) entre células adjacentes, caso os valores sejam iguais ($X_c = X_{c+1}$ e $Y_c = Y_{c+1}$), isso significa que eles fazem parte da mesma reta e a célula com o menor valor de c pode ser excluída.

A Tabela 2 mostra os cálculos realizados para a simplificação da trajetória da Figura 29, sendo possível visualizar o resultado na Figura 30. A coluna "Células depois da Simplificação" mostra as células que permaneceram na trajetória simplificada na cor verde e as que foram eliminadas na cor vermelha.

A forma como foi implementada a rotina de simplificação deixa a execução da tarefa muito simples e rápida, pois não precisa de cálculos mais complexos como raiz quadrada e

potenciação. Estas operações seriam necessárias caso a rotina fosse criada utilizando-se o coeficiente angular entre a reta formada por dois pontos consecutivos, outra forma de implementação desta rotina.

Tabela 2 – Exemplo do cálculo da simplificação da trajetória da Figura 30

	Vetor de Posição		Vetor de Coeficientes		Células depois da Simplificada	
Célula	X	Y	$X_c - X_{c-1}$	$Y_c - Y_{c-1}$	X	Y
1	10	13	-	-	10	13
2	11	14	-1	-1	-----	-----
3	12	15	-1	-1	-----	-----
4	13	16	-1	-1	-----	-----
5	14	17	-1	-1	-----	-----
6	15	18	-1	-1	-----	-----
7	16	19	-1	-1	-----	-----
8	17	20	-1	-1	17	20
9	18	20	-1	0	18	20
10	18	21	0	-1	-----	-----
11	18	22	0	-1	-----	-----
12	18	23	0	-1	-----	-----
13	18	24	0	-1	-----	-----
14	18	25	0	-1	-----	-----
15	18	26	0	-1	-----	-----
16	18	27	0	-1	18	27

Esta simplificação diminui muito a complexidade da movimentação e permite que o robô utilize velocidades diferentes de movimentação de acordo com o tamanho do trecho reto que ele precise percorrer.

Esta é a ultima tarefa de planejamento necessária para que o robô consiga executar suas tarefas de movimentação de forma adequada e otimizada.

5.2 Execução Reativa

Com todas as técnicas apresentadas até o momento o robô consegue planejar inteiramente uma rotina de movimentação de forma bastante detalhada. É possível planejar desvios de obstáculos conhecidos, configurar sua velocidade de acordo com o trecho que esteja realizando, cumprindo teoricamente suas tarefas. Mas após o planejamento e seu detalhamento é necessário executar os resultados deste trabalho.

Para realizar a execução das tarefas, na arquitetura AuRA existe uma última camada chamada de “Controlador Esquema”. Esta camada é responsável por controlar e monitorar os

processos comportamentais em tempo de execução. Segundo (Faria, 2006) cada comportamento motor (esquema) está associado com um esquema de percepção capaz de fornecer o estímulo necessário para um comportamento específico.

Estes comportamentos executam as ações de forma reativa, baseando suas atividades nos estímulos externos captados pelos sensores instalados no robô. No presente trabalho esta camada é implementada por uma classe da biblioteca ARIA chamada de ArRobot.

5.2.1 Biblioteca ArRobot

Segundo (Bastos, 2007), esta é a classe mais importante do ARIA. Ela é usada para fazer a comunicação com o robô, para receber dados a respeito de sensores do robô e também para controlar diversos dispositivos (posicionamento, sonares). Além disso, é usada para realizar determinadas ações pré-definidas e controlar a execução das mesmas.

A comunicação entre o ARIA e o robô ou simulador é realizada através do envio de pacotes encapsulados de acordo com um protocolo, neste contexto o cliente da comunicação é um software usando a biblioteca ARIA para operar o robô, sendo o servidor de comunicação o *firmware* do robô.

A Figura 31 mostra a forma com que a classe ArRobot executa suas tarefas. Neste ciclo de execução o robô manda a cada 100ms um pacote de informações para o "Manipulador de dados" contendo os *status* atuais dos seus dispositivos. Estas informações são, por exemplo: a posição atual do robô, estimativas da velocidade de translação e rotação, leituras dos sonares, tensão de bateria, valores dos pontos de entrada e saída do robô. Esses dados são utilizados pela ArRobot na etapa de "Reflexão dos estados" e são acessíveis através de métodos da classe ArRobot.

Para controlar a plataforma robótica, um programa cliente envia pacotes de comandos através da conexão de comunicação com o robô. Este controle pode ser feito utilizando comandos do método *Motion* da classe ArRobot ou através de comandos diretos. Deve-se tomar cuidado na utilização de comandos diretos, pois podem conflitar com comandos realizados pela classe ArRobot.

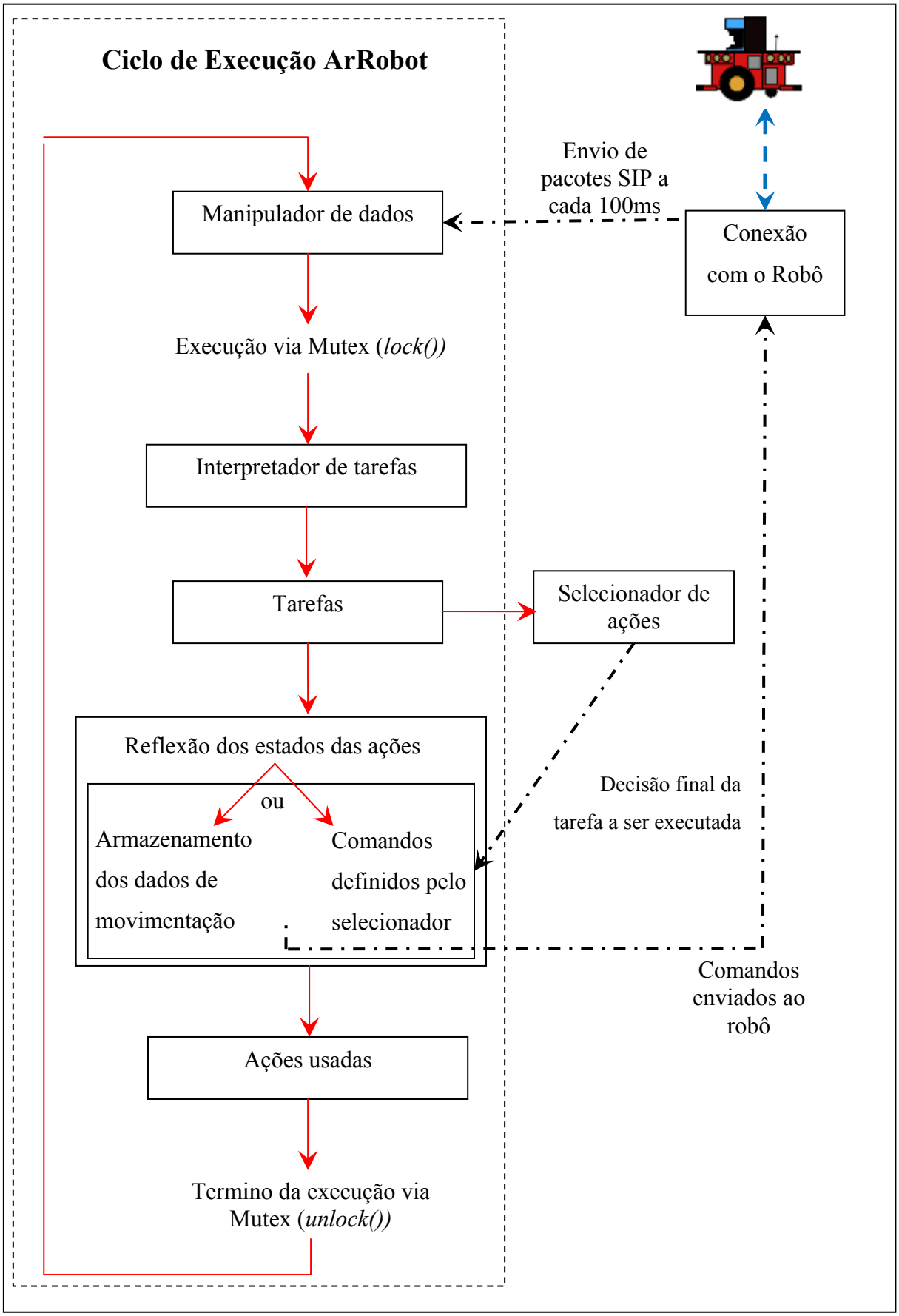

Figura 31 – Ciclo de execução das tarefas da classe ArRobot

O ciclo de execução pode ser feito de forma síncrona ou assíncrona, na forma síncrona a cadeia de ações mostrada pela Figura 31 deve ser feita de forma mais rápida possível, pois a chegada de um pacote SIP (*Session Initiation Protocol*) desencadeia a execução do ciclo, sendo que todas as etapas devem ser terminadas antes da chegada de um novo pacote. Nesta execução caso ocorra uma fila de pacotes SIP para serem tratados, os pacotes mais antigos são descartados e são executados os mais novos. Já na forma assíncrona as execuções são realizadas na forma de *threads,* com isso dois pacotes podem ser tratados de forma simultânea.

Como comentado no inicio deste capítulo, a classe ArRobot controla e realiza a execução de ações. Estas ações são conjuntos de comandos que podem ser agrupados e executados de forma a gerar um comportamento desejável pelo sistema. Elas podem ser invocadas quantas vezes forem necessárias, sendo que o usuário pode desenvolver uma ação especifica para atender suas necessidades agrupando ações já pré-desenvolvidas na biblioteca.

Uma ação é adicionada a uma lista de ações que devem ser executadas pelo robô, com sua respectiva prioridade. A prioridade, a princípio, pode ser qualquer número inteiro, mas por convenção usa-se números entre 0 e 100. Vale ressaltar que as ações são executadas em ordem decrescente de prioridades, assim, uma ação com prioridade 100 será executada antes de uma ação com prioridade 50 (Bastos, 2007).

O bloco de "Interpretador de tarefas" da cadeia de execução da classe ArRobot, (Figura 31), é responsável por verificar os valores dos sensores que estão sendo utilizados pelas tarefas presentes no programa, as enviando para o bloco "Tarefas". Este bloco verifica se houve alguma mudança na ordem de execução das tarefas pela mudança do valor de algum sensor, caso haja mudança, a escolha da ação que deverá ser realizada é feito pelo *"*Selecionador de ações*"* a qual invoca cada ação, combinando seus comandos de movimentação e as enviando para armazenamento no bloco "Reflexão dos estados das ações", este por sua vez envia para o robô a ação que precisa ser desempenhada.

Por exemplo, uma ação pré-definida pelo sistema chamada de "*ArActionAvoidFront"* tem a função de alterar o comportamento do robô quando este está perto de um obstáculo, tentando desviá-lo do mesmo. Esta ação é realizada caso os sensores de proximidades ultra-sônicos da parte frontal do robô detecte a presença de um obstáculo a uma distância configurável. Portanto caso haja mudança dos valores dos sensores frontais de proximidade, elas serão detectadas e se forem menores que a distância configurada para desvio de obstáculos, esta ação será habilitada, mas só será executada caso não haja nenhuma outra ação habilitada com valor de prioridade maior.

5.2.2 Execução das Tarefas

Diferentemente da execução proposta pela arquitetura AuRA, a tarefa a ser executada não é uma somatória das ações habilitadas, mas sim a seleção da ação habilitada com maior prioridade, isso se deve pois no sistema implementado existem apenas quatro ações reativas criadas, sendo que uma de execução normal e as restantes de ajuste de falhas eventuais no planejamento de rotas, que são executadas por motivo de segurança, não justificando a criação desta estrutura. Apesar disso, conforme a camada "Controlador Esquema" da arquitetura AuRA, as ações utilizadas são realizadas de acordo com a necessidade imposta pelo ambiente e pelo plano gerado nas camadas deliberativa da arquitetura.

As ações são configuradas pelas camadas deliberativas, mas se comportam de forma reativa, pois seus comportamentos são baseados nas leituras dos sensores que interpretam o meio de navegação, reagindo a estes estímulos.

As ações pré-definidas pela biblioteca e utilizadas neste trabalho são:

- ArActionGoTo: esta ação faz o robô se mover para uma posição determinada, sempre com relação à posição inicial. Ela é referenciada através dos sensores *encoder* instalados nas rodas do robô, permitindo o controle da odometria e a estimação de sua posição. Deve-se especificar a distância mínima que o robô poderá considerar como tendo atingido o alvo, a velocidade de movimentação quando este se movimenta linearmente e a velocidade enquanto este está fazendo curvas. O método "*setGoal(ArPose goal)*" determina a posição para a qual o robô deve se mover, lembrando que esta posição é relativa à localização inicial do robô.

- ArActionAvoidFront: esta ação tem a função de alterar o comportamento do robô quando este está perto de um obstáculo, tentando desviá-lo do mesmo. A proximidade dos obstáculos é detectada pelos sensores ultra-sônicos instalados no robô (ver Figura 51). Deve-se especificar a distância em milímetros (mm) que um obstáculo deve estar do robô para que este execute a ação de desvio, assim como a velocidade (mm/s) do robô no momento em que este está desviando de algum obstáculo e o ângulo em graus do desvio a ser aplicado.

- ArActionBumpers: esta ação basicamente responde ao acionamento das chaves de contato (*Bumpers*) instalados na lateral do robô (ver Figura 52). Se o robô está indo em frente e se choca com algum obstáculo com sua parte frontal, este método irá alterar o comportamento do robô fazendo com que ele volte ("de uma ré") e gire se direcionando para outro sentido. Se o robô está indo para trás e bate em um obstáculo com a parte traseira, este método irá fazer com que ele se movimente para frente e gire.

- ArActionStallRecover: esta ação muda o comportamento do robô quando alguma de suas rodas está parada ("travada"), quando não deveria estar.

Todas estas ações são utilizadas para executar o plano gerado pelas camadas deliberativas, sendo a ação "*ArActionGoTo*" utilizada para que o robô se movimente pelo ambiente realizando o plano criado. As demais ações são utilizadas para corrigir possíveis erros de planejamento, como desviar de obstáculos não previstos e erros de localização. Os erros de localização podem ser gerados por erros de odometria e/ou imprecisão do mapa do robô.

Para gerenciar a ordem de execução de cada ação, foram adotados os seguintes valores de prioridade, conforme a Tabela 3.

Tabela 3 – Prioridade das ações utilizadas pelo sistema

Ação	Prioridade
ArActionStallRecover	100
ArActionBumpers	75
ArActionAvoidFront	70
ArActionGoTo	50

Com a adoção da ordem de prioridade mostrada na Tabela 3, o sistema irá executar as ação de movimentação de um ponto a outro (ArActionGoTo) desviando de obstáculos não previstos através da ação de desvio (ArActionAvoidFront) e tendo a possibilidade de mudar seu comportamento de movimentação no caso de choques com o ambiente e travamento de rodas.

Com isso, além de planejar o robô agora consegue executar o plano gerado.

5.3 Replanejamento

As tarefas são selecionadas pelo sistema através do conhecimento que ele tem do mundo a qual está inserido. Mas como este ambiente, no caso real, é dinâmico, podem ocorrer eventos não conhecidos que interferem diretamente na execução das tarefas. Portanto o sistema deve ser capaz de procurar alternativas que solucione o problema imposto mediante eventuais mudanças no ambiente.

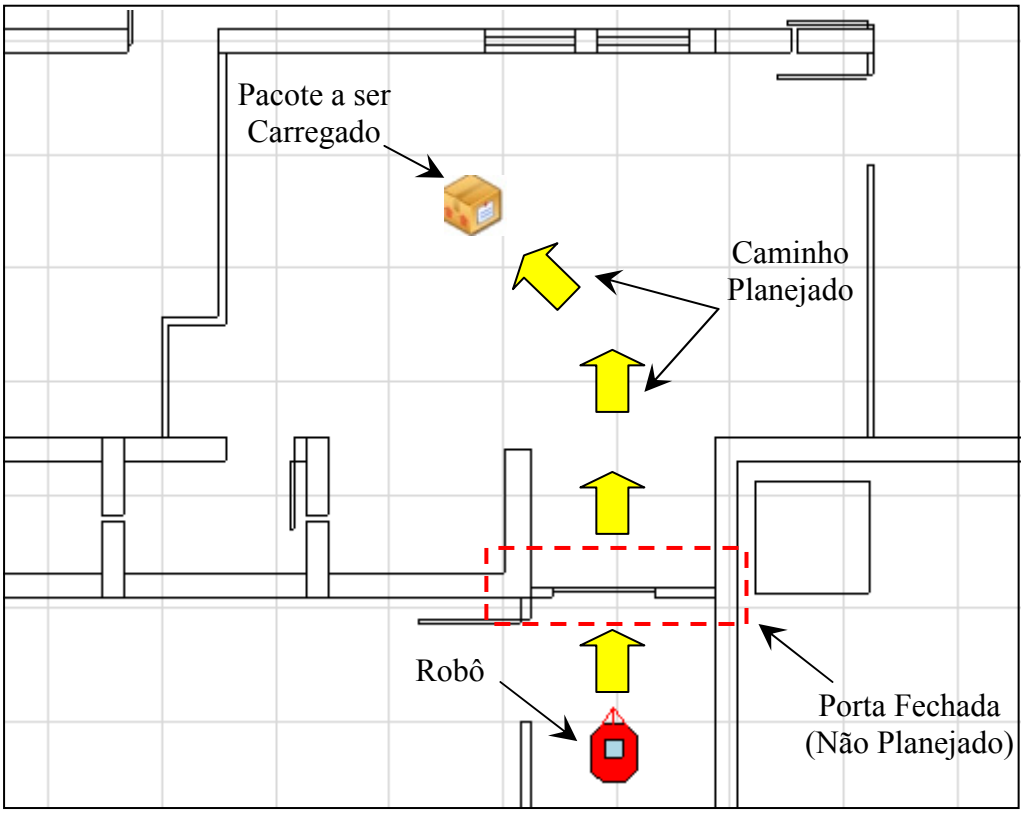

Figura 32 – Exemplo replanejamento

Para ilustrar melhor este fato, considere o seguinte problema (Figura 32):

O robô precisa buscar um pacote que está em uma sala qualquer, para isso o sistema de planejamento planeja e detalha todas as execuções que ele precisa fazer baseado no conhecimento cartográfico que ele dispõe do mundo naquele momento. Os planos gerados dizem que para chegar ao pacote ele precisa passar por um corredor e atravessar uma porta. O robô executa os planos conforme programado, mas ao chegar a uma sala encontra a passagem que ele precisa atravessar fechada, não conseguindo evoluir.

Neste caso ele deverá constatar esta impossibilidade de execução do plano corrente, identificando o motivo pelo qual não é possível sua execução, pois todas as informações que o

robô dispõe e todo o conhecimento das tarefas que ele deve executar posteriormente estão baseados no fato de que é possível atravessar a porta que está fechada, sendo necessário um replanejamento das ações a serem executadas.

Segundo (Pieri, 2002) na arquitetura AuRA, uma vez iniciada a execução reativa, o componente deliberativo não é ativo pelo menos até que se detecte uma falha em tal execução. Uma falha típica é detectada pela falta de progresso, evidenciando uma velocidade zero ou sinal gerador de *time-out*. Neste momento o planejador hierárquico é reinvocado um estágio por vez, de baixo para cima na hierarquia, até o problema ser resolvido. Primeiro, o "Sequenciador de Planos" tenta refazer a rota do robô baseado nas informações que foram obtidas durante a navegação e armazenadas na memória de curto prazo. Se por algum motivo isto mostra-se insatisfatório, por exemplo, a rota está completamente bloqueada dentro deste contexto local, o "Raciocinador Espacial" é novamente invocado. Ele tenta gerar uma nova rota global que sobrepõe a região inteira afetada. Se ainda assim houver falha, o "Planejador de Missão" é reinvocado, informando o operador da dificuldade e perguntando pela reformulação ou abandono da missão.

No caso do exemplo citado, para resolver este problema a única solução é refazer o planejamento global de todas as ações a serem executadas, pois o caminho está totalmente bloqueado. Em outras situações menos drásticas, pode ser que apenas um replanejamento de trajetória solucione o problema.

5.3.1 Replanejamento Global

Para que seja possível a realização do replanejamento de forma adequada é preciso que a base de conhecimento do sistema vá se modificando enquanto o robô executa as tarefas designadas, caso contrário este replanejamento pode chegar a soluções erradas ou pode repetir a solução descartada anteriormente.

No sistema criado, o planejamento global é feito baseado em um "*domínio de conhecimento*" que informa ao sistema as relações existentes entre os tipos de elementos que podem existir e as ações possíveis de cada elemento que modificam este mundo. Ele também é baseado em um "*problema de planejamento*" que informa ao sistema os elementos que realmente existem no mundo (elementos instanciados), os valores de seus atributos e as relações existentes entre os objetos naquele instante inicial. Além dos objetivos a serem alcançados.

A partir destes conhecimentos prévios o sistema gera um conjunto de tarefas a serem executadas para se alcançar o objetivo final. O cumprimento de todas as tarefas implica em conseguir cumprir o objetivo estipulado.

Portanto, para que o sistema possa se replanejar, ao término da execução de uma tarefa é necessário a atualização do "*problema de planejamento*", pois as condições iniciais do problema já não existem mais.

Utilizando-se novamente o exemplo comentado nas seções 5.1.2 e 5.1.3 deste trabalho, onde o resultado do planejamento gerou o seguinte conjunto de tarefas:

0: (MOVER ROBOCRTI SALA1 SALA2 CRTI)[1] (1)
1: (MOVER ROBOCRTI SALA2 GARAGEM CRTI)[1] (2)
2: (DESCARREGARCARRO ZAFIRA PACOTE1 GARAGEM)[1] (3)
3: (CARREGARROBO ROBOCRTI PACOTE1 GARAGEM)[1] (4)
4: (MOVER ROBOCRTI GARAGEM SALA2 CRTI)[1] (5)
5: (MOVER ROBOCRTI SALA2 SALA1 CRTI)[1] (6)
6: (DESCARREGAROBO ROBOCRTI PACOTE1 SALA1)[1] (7)

A realização da tarefa em (1) implica em eliminar da base de conhecimento do sistema ("*problema de planejamento*") a informação que diz que o robô chamado *"RoboCRTi"*está na *"Sala1"* e colocar na base de conhecimento a informação que o *"RoboCRTi"* está na *"Sala2"*.

Na prática, isso seria substituir a diretiva em PDDL: *"(veiculoEstaNo ROBOCRTI SALA1)"* por *"(veiculoEstaNo ROBOCRTI SALA2)"* no programa mostrado no Apêndice A – Código do Exemplo do Capítulo 5.1.2 em PDDL versão 2.2.

Desta forma, atualizando as informações a cada término da execução de uma tarefa, o sistema pode gerar um replanejamento de forma correta, mas ainda não é garantido que este replanejamento não seja a continuação do plano anterior.

Para que seja gerado um planejamento diferente é preciso modificar a base de conhecimento retirando a informação que foi modificada no mundo real que impede a execução do plano anterior.

No caso do exemplo citado, se a tarefa em (2) não pudesse ser realizada, pois a passagem entre a *"Sala2"* e a *"Garagem"* está impedida, para que o sistema pudesse gerar um resultado diferente seria necessário a retirada da informação que diz que as duas salas são vizinhas e estão conectadas. Esta informação no programa em PDDL seria a diretiva *"(vizinho SALA2 GARAGEM)"*.

Com isso se garante que não será repetido o plano anterior, pode ser que nem exista outro plano possível, neste caso o sistema deve informar ao usuário a impossibilidade de realizar o objetivo estipulado.

5.3.2 Replanejamento de Trajetória

O replanejamento de trajetória deve ser feito antes do replanejamento global, pois como há erros provocados pela imprecisão do mapa do robô e os erros acumulados provocados pela odometria, o robô pode-se perder na execução da tarefa.

Este replanejamento no sistema criado consiste em se rodar novamente o algoritmo de busca A* considerando o objetivo final da tarefa e a posição atual do robô dentro do ambiente. Maiores informações sobre a metodologia A* ver tópico 5.1.5.2 deste trabalho.

5.4 Integração de Softwares e Sistemas

Mediante a descrição de todos os subsistemas presentes no referido trabalho, além de todos os softwares utilizados, cabe agora mostrar a forma como cada um se comunica e troca informações para gerar o Navegador Autônomo. Na Figura 33 encontra-se o fluxograma que modela o funcionamento do software, cada passo será explicado nesta seção do trabalho.

Primeiramente tem-se a modelagem do domínio e do problema de planejamento criados no software itSIMPLE, conforme descrito no capitulo 5.1.1 e 5.1.2, eles são convertidos pelo próprio itSIMPLE em linguagem PDDL e armazenados em arquivos de texto.

No inicio da execução, o Navegador Autônomo procura os arquivos em formato texto do domínio e problema de planejamento, dos mapas topológicos e de navegação. Sendo os arquivos encontrados, eles são armazenados na memória do programa para proporcionar maior velocidade de execução. Caso algum dos arquivos não seja encontrado durante um tempo de busca, o programa é finalizado com uma mensagem de erro informando qual arquivo não foi localizado, como é mostrado na Figura 33.

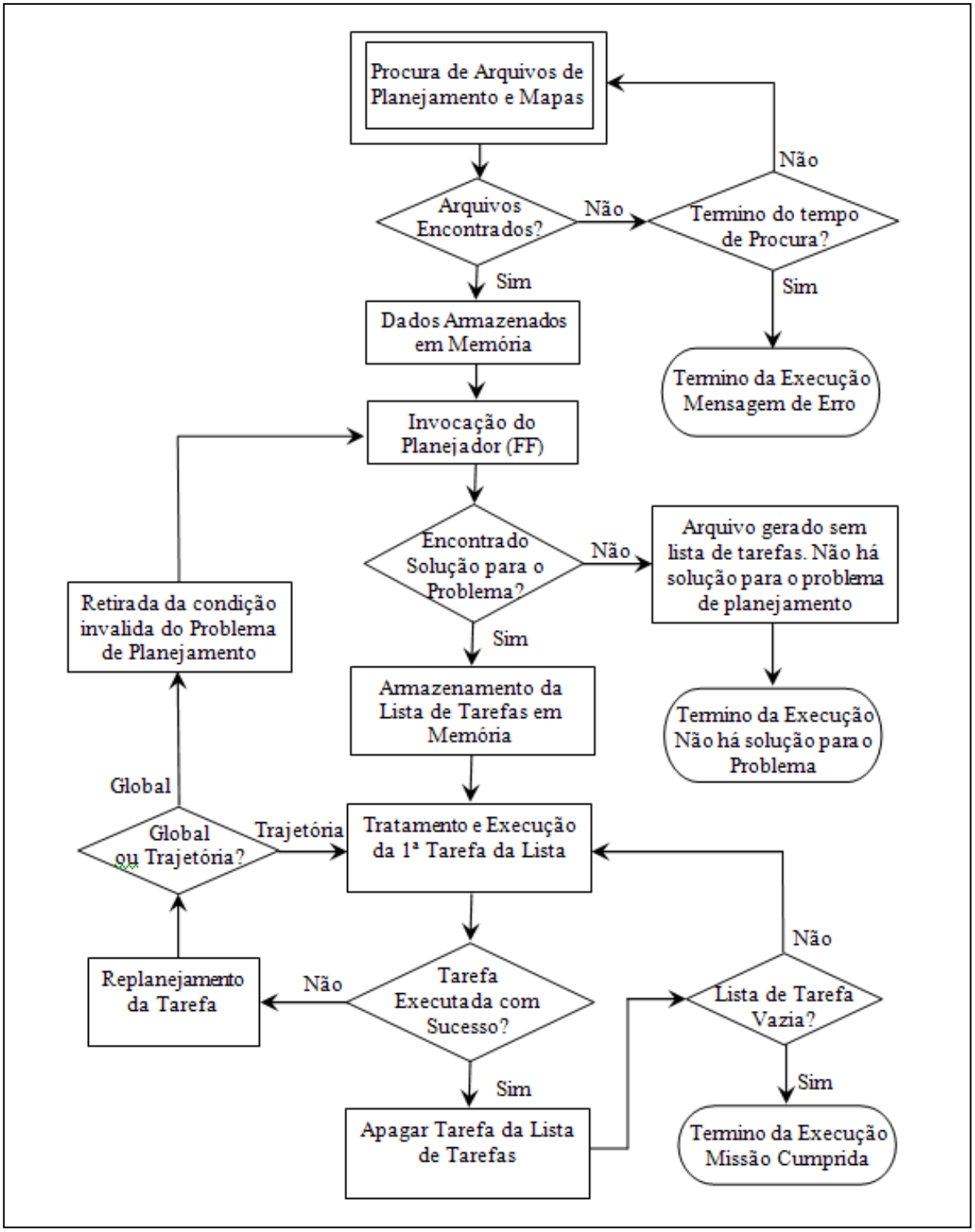

Figura 33 – Fluxograma de Execução do Software Navegador Autônomo

Havendo um problema e um domínio de planejamento, o planejador FF pode ser invocado pelo Navegador Autônomo, a fim de verificar a existência de uma solução para o problema. Sendo encontrada uma solução ou não o FF gera um arquivo em formato texto após ser invocado. Caso exista solução e ela foi encontrada, o arquivo terá uma lista de tarefas que

deverão ser executadas para se atingir o resultado final. Caso a solução não seja encontrada, tal arquivo terá somente um cabeçalho de informações sem nenhuma lista de tarefas, ver Figura 13 para mais detalhes sobre o arquivo gerado pelo planejador FF.

Supondo que uma solução foi encontrada e o arquivo texto apresente uma lista de tarefas, então elas serão armazenadas na memória do programa através de uma lista de palavras. Cada tarefa irá ocupar um índice dentro da lista, sendo a lista ordenada pela ordem de execução das mesmas. A primeira tarefa a ser executada será a que estiver no começo da lista.

Para que o robô possa executar as tarefas planejadas é necessário que o Navegador Autônomo conheça todas as ações modeladas no domínio de planejamento, interprete o que deve ser realizado na tarefa e a associe a um conjunto de eventos a serem executados. Isto é feito no processo "Tratamento e Execução da 1ª tarefa da Lista" do fluxograma da Figura 33.

Durante este processo, o Navegador Autônomo lê a primeira tarefa a ser executada na lista e a classifica de acordo com o tipo de tarefa a ser feita. As ações modeladas no software podem ser: Mover, CarregarRobo, DescarregarRobo, Viajar, CarregarCarro e DescarregarCarro. Como comentado no capítulo 5.1.1, além do veiculo "Robô" foi modelado no domínio de planejamento um veiculo chamado de "Carro", sendo o mesmo utilizado para possibilitar a integração de "Territorios" diferentes, simulando um ambiente logístico mais complexo. As ações referentes ao "Carro" ("Viajar", "CarregarCarro" e "DescarregarCarro"), são executadas no software Navegador Autônomo através do envio de mensagens pedindo a confirmação ou não da execução da ação. No capítulo 6 será mostrado um exemplo de execução do "Carro" e o envio da mensagem de confirmação.

As ações referentes ao veículo "Robô" são executadas e simuladas através do software *MobileSim* (ver capítulo 4.2), utilizando as bibliotecas ARIA e ARNetworking (ver capítulo 4.1) . A execução das tarefas do robô é feita da seguinte maneira:

- MOVER: conforme descrito no capítulo 5.1.1, esta ação consiste em movimentar o robô de um lugar de origem para um lugar de destino dentro de um mesmo território. Os dados de origem, destino e território são especificados na tarefa planejada. O Navegador Autônomo irá, mediante estes dados, encontrar no mapa topológico armazenado em memória a coordenada de destino da movimentação (capítulo 5.1.5.1), após isso ele irá gerar uma trajetória de movimentação para o robô com auxilio do mapa de navegação e da execução do Algoritmo A* (capítulo 5.1.5.2). Após gerada a trajetória, ela será simplificada (capítulo 5.1.5.3) e seus pontos serão armazenados em uma lista para serem enviados para a execução

no robô. A lista de pontos apresenta todas as coordenadas da trajetória que o robô deverá atingir para realizar a movimentação planejada, sendo enviada para o robô simulado um ponto de cada vez utilizando-se o comando *ArActionGoTo* da biblioteca ARIA (capítulo 5.2.2), este comando faz com que o robô se movimente do ponto em que ele está para a coordenada de destino. A implementação do *ArActionGoTo* na biblioteca ARIA configura uma malha de controle que gera, mediante a distância do robô ao ponto de destino, as velocidades que devem ser empregadas nas rodas do robô para que seja alcançado o ponto desejado. Uma configuração importante antes do envio do comando *ArActionGoTo* é a velocidade máxima permitida ao robô, uma velocidade grande irá proporcionar maior velocidade de atuação mas menor precisão nos movimentos do robô, em caso de pequenas movimentações uma velocidade alta poderá ocasionar instabilidade, podendo a malha de controle de movimentação nunca conseguir convergir para o ponto desejado. Por este motivo, antes do envio do comando de movimentação para o robô simulado é calculada a distância entre pontos consecutivos da trajetória, sendo a velocidade máxima de movimentação configurada de acordo com a distância encontrada. Enviando o comando *ArActionGoTo* para o robô simulado pelo *MobileSim* ele tentará realizar a movimentação para o ponto desejado de acordo com sua malha de controle de movimentação. Quando o ponto de destino da movimentação for alcançado, o robô simulado envia uma mensagem informando que o comando *ArActionGoTo* foi realizado com sucesso. Caso exista outro ponto dentro da lista de pontos da trajetória, ele é enviado para execução, caso não exista, a tarefa de movimentação foi cumprida com sucesso.

- CARREGARROBO: esta tarefa é utilizada para possibilitar o embarque de um pacote dentro do robô. Ela consiste de uma movimentação do robô para o ponto de carregamento do local e do carregamento do próprio pacote. Portanto nesta tarefa é configurada inicialmente uma ação de MOVER e quando esta ação é completada é emitida uma mensagem pedindo que o pacote seja embarcado no robô. Quando é confirmado o embarque do pacote a tarefa é terminada.

- DESCARREGARROBO: esta tarefa é utilizada para possibilitar o desembarque de um pacote do robô para o local de destino. Ela consiste de uma movimentação do robô para o ponto de descarregamento do local e do descarregamento do próprio pacote. Portanto nesta tarefa é configurada inicialmente uma ação de MOVER e quando esta ação é completada é

emitida uma mensagem pedindo que o pacote seja desembarcado no robô. Quando é confirmado o desembarque do pacote a tarefa é terminada.

Para que o Navegador Autônomo tenha sempre o status real do ambiente onde está inserido, cada vez que uma tarefa é cumprida, antes de sua eliminação da lista e do começo da execução da próxima tarefa, é atualizado o arquivo texto do problema de planejamento com as modificações ocasionadas pela tarefa, desta maneira, caso seja necessário um replanejamento da missão o Navegador pode realizá-lo imediatamente. Para mais detalhes sobre as modificações ocasionadas pelas ações do domínio de planejamento, ver capítulo 5.1.1.

Após atualização do problema de planejamento, a tarefa realizada da lista é eliminada começando a execução da próxima, caso não haja uma próxima tarefa, a missão planejada foi cumprida sendo terminada a execução do programa Navegador Autônomo, Figura 33.

Durante a execução da tarefa, caso seja identificada a não realização da mesma, é solicitado ao usuário o replanejamento da missão. Ela pode ser feita de maneira global ou somente de trajetória. A forma global realiza o replanejamento da missão voltando ao passo de Invocação do Planejador (FF), fluxograma da Figura 33, desta maneira o planejador irá tentar emitir novo plano. Para que o planejador não emita o mesmo plano que não pôde ser realizado antes, é retirada do arquivo do problema de planejamento a precondição que foi identificada como falsa, que impossibilitou a execução do plano anterior, no capítulo 6.1 será mostrado um exemplo com replanejamento global de missão. Caso seja encontrado outro plano para realizar a missão, o Navegador Autônomo continua sua execução, caso contrário o programa é finalizado com uma mensagem de erro.

Já no replanejamento de trajetória é apenas re-executado o Algoritmo A* para as ações de movimentação, trançando novos pontos de trajetória para serem tratados pelo programa e executados pelo robô.

6 Aplicação e Resultados

Através de todos os módulos criados e comentados nos capítulos anteriores, é possível a criação de um sistema de transporte de cargas que trabalha de forma autônoma utilizando-se veículos para fazer a movimentação dos pacotes.

O domínio de planejamento foi implementado considerando a existência de dois veículos de movimentação, um robô para a movimentação de pacotes em ambientes internos (*indoors)* e um automóvel para transportes em ambientes externos (*outdoors)*, sendo apenas criado neste trabalho o sistema de execução das tarefas para o robô. Para o automóvel, o sistema somente emitirá mensagens perguntando sobre a execução das tarefas relacionadas a ele.

6.1 Ambiente de Aplicação

Para validar o sistema proposto, um mapa de simulação foi criado baseado em um ambiente real. Este ambiente trata-se do andar térreo do prédio da Engenharia Elétrica da UNIFEI – Universidade Federal de Itajubá, sendo criado a partir de uma planta baixa das dependências com dimensões reais, conforme mostra a Figura 34 e comentado no capítulo 4.2.

As diversas salas do ambiente de teste foram nomeadas, sendo estes nomes utilizados na modelagem do "problema de planejamento" e no "mapa de rotas". O "mapa enumerado" foi criado considerando um tamanho de célula de 350x350mm, sendo o referencial zero do sistema de coordenada colocado no canto inferior esquerdo do mapa.

No "problema de planejamento" foram criadas todas as relações de vizinhança entre as salas, estipulando no "mapa de rotas" as coordenadas dos pontos de carga e descarga de pacotes, além dos pontos de transição entre as salas.

O mapa foi carregado no simulador MobileSim sendo colocado no ambiente um robô modelo "Pioneer3-DX" com dois semi-aneis de sonares (somando-se 16 sonares distribuídos em 360º), um conjunto completo de *bumpers* em toda a lateral do robô, além dos dois *encoder* instalados (um em cada roda).

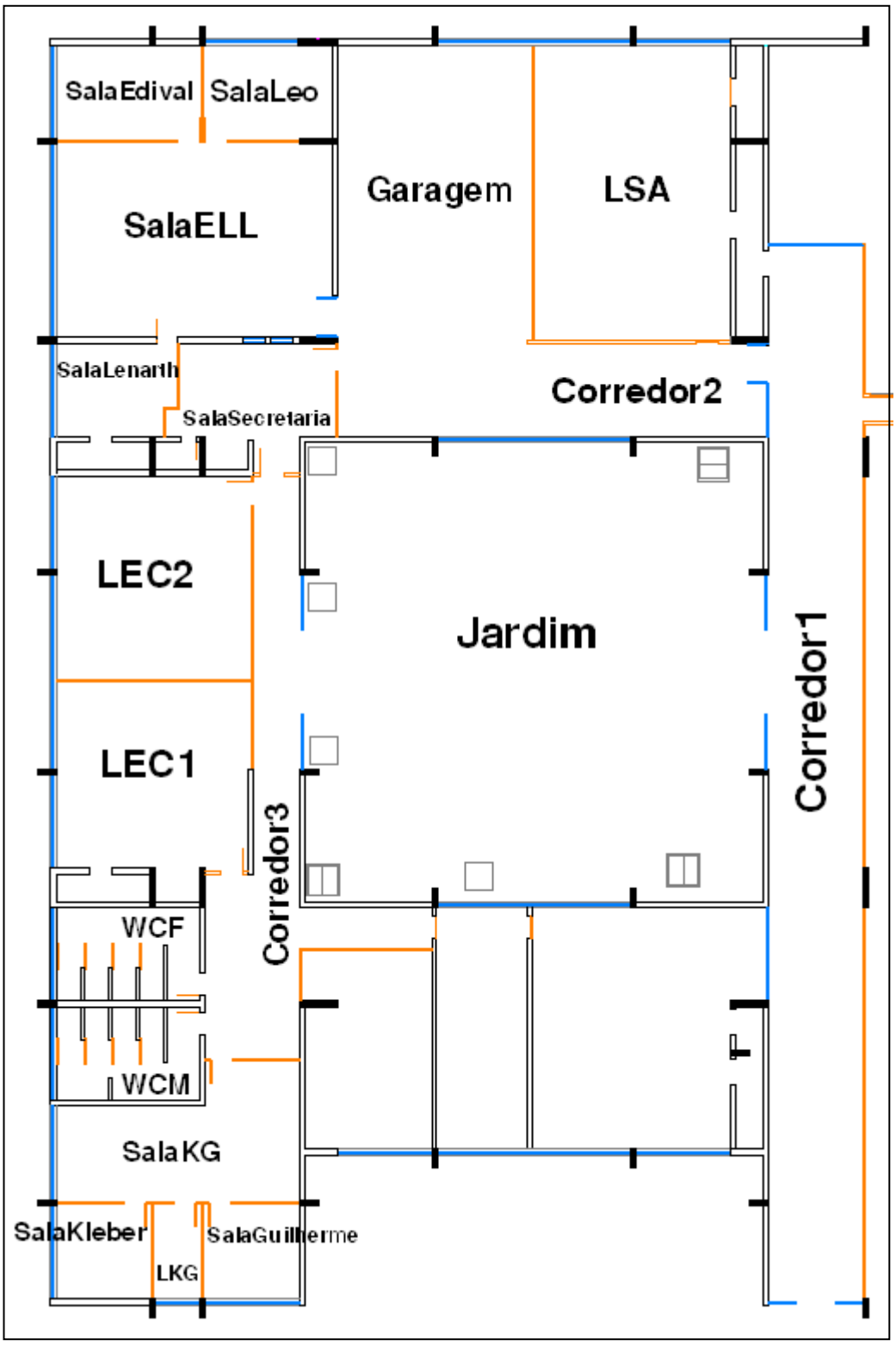

Figura 34 – Planta do ambiente de simulação com a nomenclatura dos lugares

Foram realizados diversos testes com problemas de planejamento variados. Dentre eles será demonstrado nessa seção do trabalho um que foi considerado mais completo, pois envolve a necessidade de replanejamento já que uma passagem entre salas foi propositalmente fechada, não sendo informado ao sistema este fato.

O simulador utilizado não permite o carregamento de mapas de forma dinâmica, com isso o mapa só pode ser carregado no inicio da simulação. Portanto, para executar o exemplo a ser detalhado, a passagem obstruída teve que ser criada em um novo mapa e carregada no inicio da simulação.

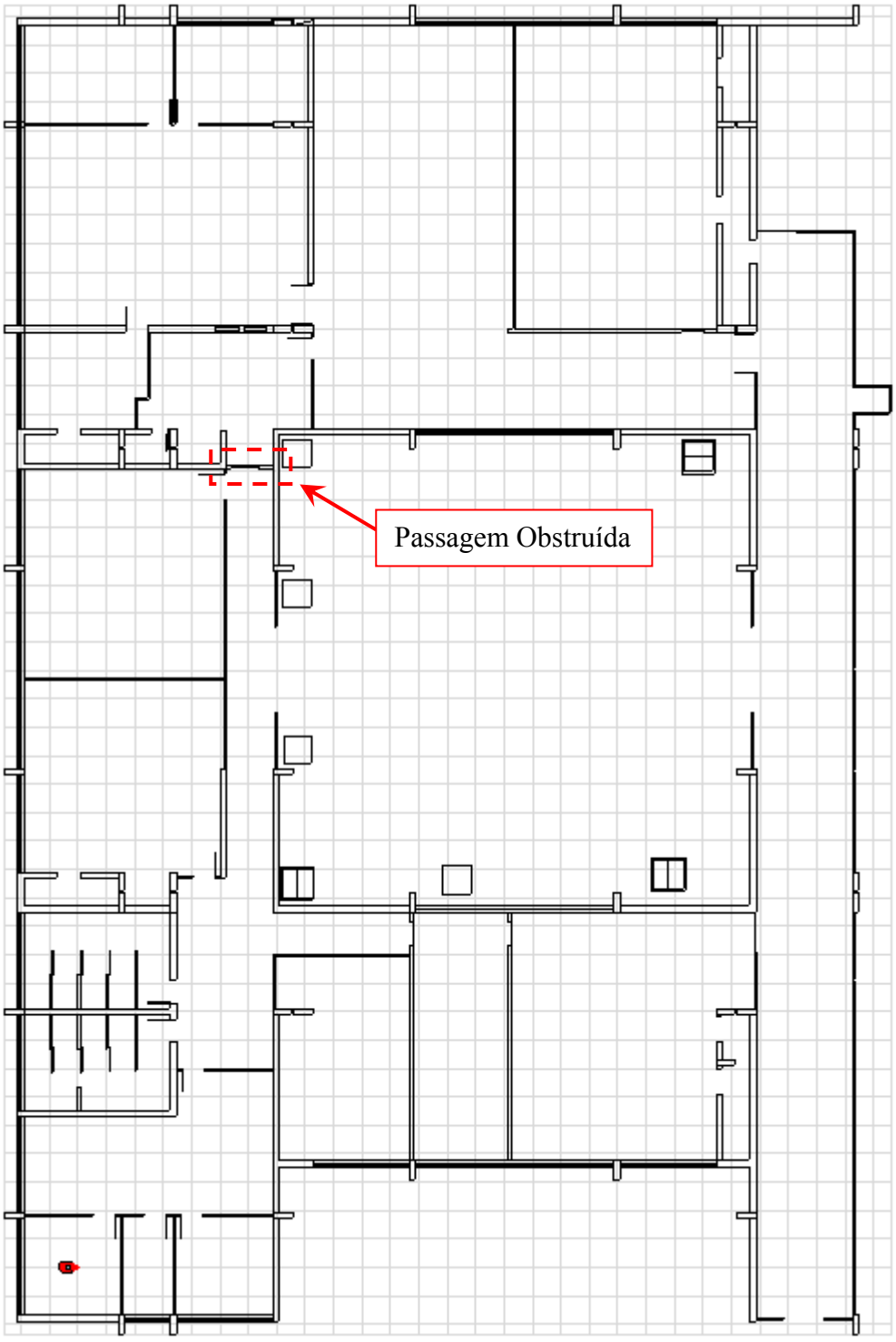

Figura 35 – Planta do ambiente de simulação com uma passagem obstruída

No ambiente existem dois agentes na simulação, um robô chamado “RoboCRTi” e um automóvel chamado “Zafira”. O “RoboCRTi” inicialmente foi colocado no lugar “SalaKleber” e a “Zafira” está estacionada no local “Garagem” que é um objeto do tipo “Garagem”. Existem dois objetos pacotes no ambiente, o “Pacote1” que está no “LEC1” e o “Pacote2” que está no veiculo “Zafira”. O diagrama da Figura 36 mostra estas relações

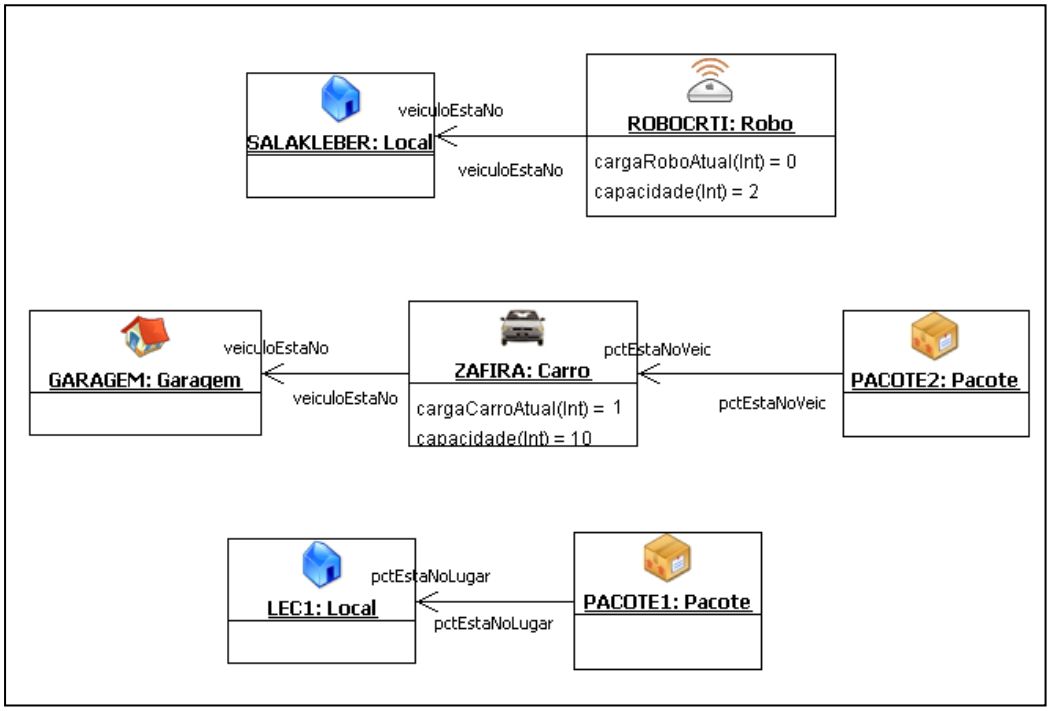

Figura 36 – Diagrama do problema de planejamento inicial

O objetivo a ser realizado é colocar os dois pacotes na “SalaLenarth”, Figura 37.

Figura 37 – Diagrama do problema de planejamento objetivo

Após estas configurações, o programa foi executado. Inicialmente ele pede para o usuário a informação da localização inicial do robô, este parâmetro é necessário para que o robô se localize no espaço, referenciando toda sua movimentação a partir daquele ponto.

Fornecido estes parâmetros, o programa tenta se conectar ao simulador através da porta TCP 8101, caso não consiga ele tentaria se conectar a um robô real. Logo após, o planejamento das ações é realizado e o robô começa a executar suas tarefas, Figura 38.

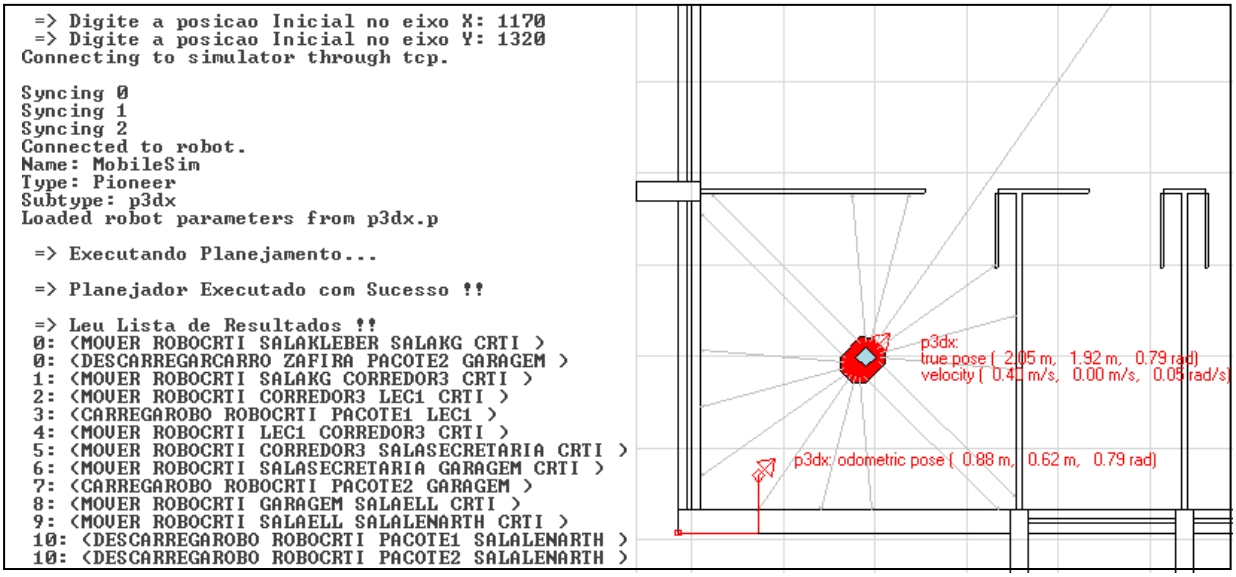

Figura 38 – Planejamento das tarefas e inicio da execução

Como pode ser visto na Figura 38, existem duas ações a serem executadas no passo 0: uma ação de “Mover” e outra de “DescarregarCarro”, isso acontece pois o planejador das ações otimiza a execução das tarefas, colocando na mesma iteração eventos que podem ser executados em paralelo. Neste caso é possível o robô se mover e uma pessoa já ir descarregando o pacote que está na “Zafira”, já que não é o robô que executa esta tarefa de descarregar um automóvel.

O robô vai executando suas tarefas de movimentação através das ações reativas (ver capítulo 5.2), que são configuradas de acordo com a tarefa que o robô está executando.

Ao chegar à tarefa 3: (“CarregaRobo”), o robô já está na entrada sala “LEC1”, mas precisa se movimentar até o ponto de carregamento desta sala que foi posicionada no centro da mesma. Após chegar a esta posição, o sistema solicita o carregamento do pacote no robô através de uma mensagem. Confirmando o carregamento o robô continua a executar suas tarefas (Figura 39)

```
 => Executando Passo:  3: (CARREGAROBO ROBOCRTI PACOTE1 LEC1 )

 => Executando Move Robo
Posicao atual: X= 2689.33mm  Y= 17768.9mm  TH= 140.978graus
Tempo= 7s  Tempo estimado= 3.40652s

 => Executando Carrega Robo

 => Robo esperando Carregamento
 => Carregamento Terminado?? (S/N): _
```

Figura 39 – Mensagem solicitando carregamento do robô

O robô conseguirá seguir o plano inicial gerado até chegar a tarefa 5:, pois a porta que ligava o "Corredor3" a "SalaSecretaria" foi propositalmente fechada. Ele tentará atravessar a porta, mas não irá conseguir, isso será identificado pelo estouro do tempo máximo permitido para a execução da tarefa. Este tempo é calculado multiplicando o tempo estimado da realização da tarefa por quatro, isso ocorre, pois o tempo estimado é calculado considerando uma velocidade constante de movimentação, o que não ocorre já que há acelerações e desacelerações durante a execução da trajetória.

O multiplicador de quatro vezes foi adotado empiricamente, já que durante a execução pode haver a necessidade do ajuste da trajetória através das ações de correção, aumentando o tempo necessário para a realização da tarefa.

Na Figura 40 pode ser visto o estouro do tempo máximo de execução e as perguntas de replanejamento.

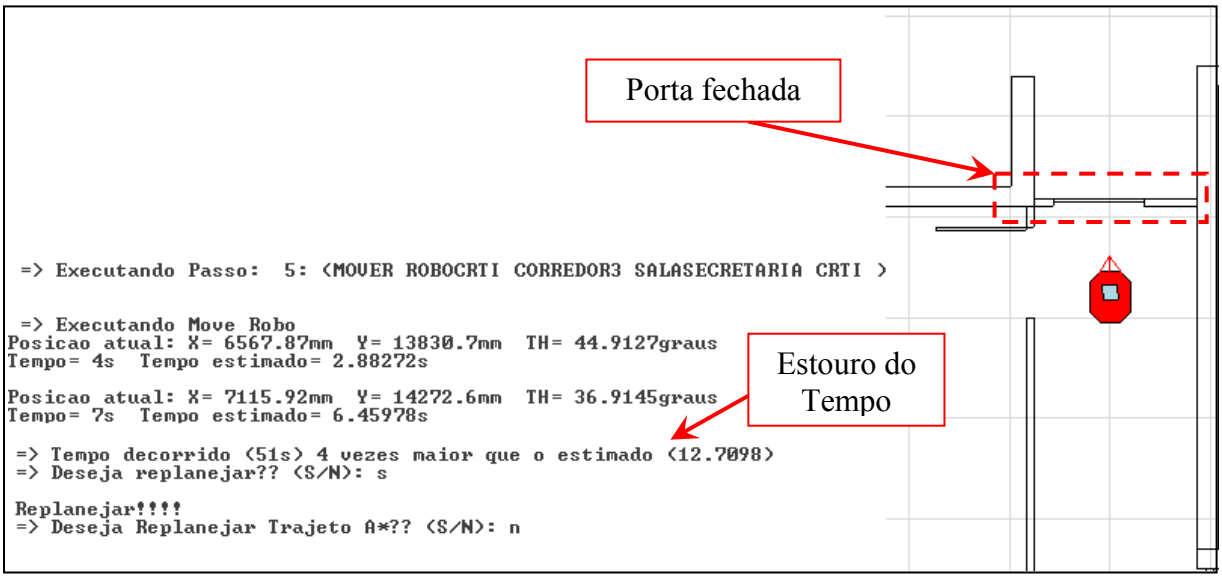

Figura 40 – Identificação de falha de execução

No programa criado, ocorrendo a identificação de uma falha, ele irá perguntar se é desejável o replanejamento das ações, esperando a resposta desta pergunta. No caso de um sistema real, não se faz necessário esta intervenção do usuário, o programa pode automaticamente decidir se será realizado o replanejamento ou não, através de uma contagem de eventos de falha. Por exemplo, pode-se tentar executar a tarefa que está falhando umas três vezes antes de replanejar, não obtendo êxito nestas tentativas o sistema se replaneja automaticamente.

Existem duas formas de planejamento, uma só replaneja a trajetória através do algoritmo A* e a outra replaneja as tarefas globais considerando todas as ações que já foram concluídas, retirando da base de dados do sistema na relação de vizinhança entre as salas da tarefa que não se conseguiu executar.

Neste exemplo, ao se optar pelo replanejamento global, o sistema gerou uma nova lista de ações e imediatamente o robô começou a executá-la. Como pode ser visto pela Figura 41.

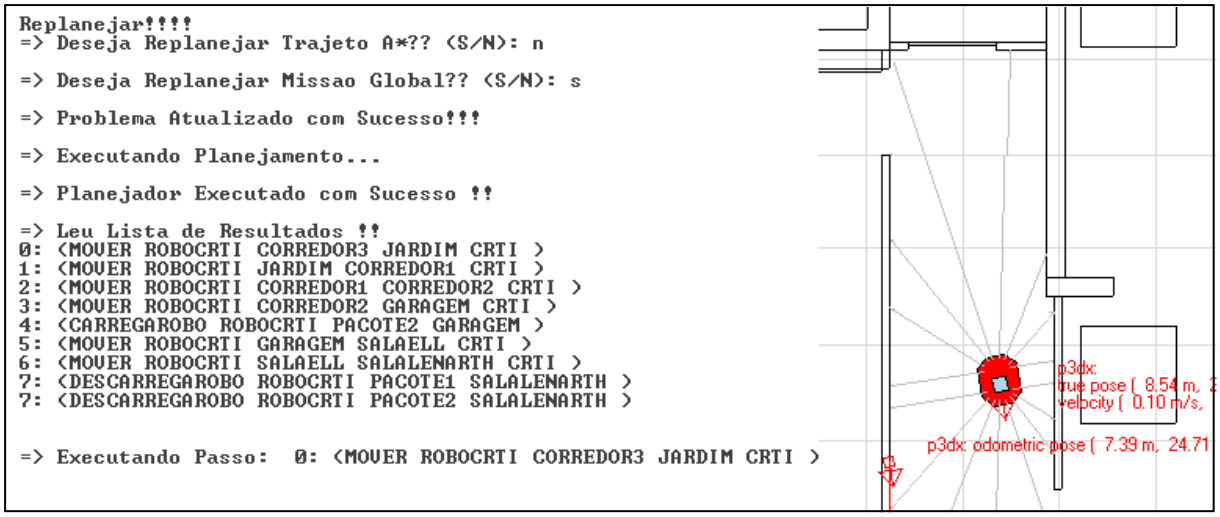

Figura 41 – Execução do replanejamento global

Nesta nova lista de ações o robô encontrou outro caminho, passando pela sala "Jardim", que possibilita a execução da missão. As tarefas serão executadas até que o objetivo final seja alcançado, finalizando o programa.

Durante a realização da missão, pode-se notar a ocorrência de alguns eventos de correção de trajetória, eventos esses comentados no final do capítulo 5.2. Quando o robô se aproxima de mais de uma parede entra em execução a ação de desvio de obstáculos afastando um pouco o robô da parede. Quando o robô se afastar o suficiente, a rotina de desvio é desabilitada e a ação que estava sendo executada volta a operar normalmente. Esta ocorrência

demonstra a robustez do sistema criado, pois ele consegue se auto-corrigir durante a execução.

Eventos para intervenção de choques com os obstáculos foram detectados no momento em que o robô tentava passar pela porta que estava fechada, sendo identificados pelos sensores *bumpers*, acionando a ação reativa de afastamento de obstáculos.

6.2 Aplicações

O sistema se mostrou robusto nas suas execuções, sendo capaz de planejar toda a logística que envolve a distribuição de pacotes em ambientes internos, realizando a distribuição destes pacotes de forma autônoma.

Sua aplicabilidade é muito ampla, pois todas as atividades produtivas humanas, de alguma forma, envolvem o transporte que algum produto ou sub-produto, seja ele um produto acabado que necessita ser armazenado para futura venda ou um que esteja se movimentando por estações de trabalho para ser terminado.

As possibilidades são enormes, pois este sistema pode se integrar com o sistema corporativo de uma empresa, planejando toda a cadeia de transporte interna e executando estas atividades de forma J*ust-in-time* (JIT).

Por exemplo, uma fábrica onde um sistema controla as execuções dos processos produtivos de forma integrada, podendo este identificar a falta de algum componente em uma estação de trabalho. O sistema poderia se auto-programar para pegar este componente e o transportar para a área requisitante, de forma flexível e otimizada, sem que fosse necessária a intervenção humana em nenhum dos processos.

7 Conclusão e Trabalhos Futuros

7.1 Conclusão

Este trabalho apresentou um sistema de navegação autônoma para robô móveis baseado na arquitetura hibrida denominada AuRA (Arkin, 1997). O sistema é capaz de planejar suas execuções de forma automática, gerando uma lista de tarefas que devem ser realizados para que seja cumprida a missão estabelecida. Essa lista é detalhada com a adição de dados cartográficos do ambiente, através da utilização de um mapa topológico que transforma as informações das rotas em coordenadas cartesianas dos pontos de transição. Além disso, o sistema através de um mapa enumerado do ambiente é capaz de prever desvios de obstáculos traçando a melhor trajetória para realização da movimentação planejada.

As execuções das tarefas planejadas e detalhadas para o veículo robô são realizadas através de um sistema de movimentação com comportamento reativo, este por sua vez é configurado pelo sistema deliberativo de planejamento, informando seus objetos para cada tarefa a ser executada. Os comportamentos reativos são capazes de executar as tarefas planejadas corrigindo possíveis erros nos planos gerados decorrentes de erros de odometria e pela precisão do mapa enumerado.

A implementação foi validada através de um software de simulação chamado MobileSim, nele é possível acompanhar o comportamento do robô no ambiente desenvolvido, sua movimentação, perícia no desvio de obstáculos e execução correta do plano gerado. Foi possível testar diversos problemas de planejamento verificando o comportamento do robô durante a execução dos planos, avaliando a qualidade e eficiência do plano gerado. Nestes testes foi possível comprovar o funcionamento do sistema, sendo ele adequado para desempenhar as tarefas para qual foi desenvolvido.

Outra funcionalidade testada e comprovada pela simulação é a capacidade do sistema de se replanejar mediante eventos que impeçam a execução do plano gerado inicialmente, sendo possível executar replanejamento de rotas e das tarefas globais dependendo da necessidade imposta pelo ambiente.

A arquitetura híbrida na qual foi baseada a implementação deste trabalho, arquitetura AuRA, mostrou-se bastante funcional, provando ser realmente flexível e modular. Sendo a integração de novos módulos de movimentação e controle de fácil implementação, sem que seja necessário fazer grandes alterações no sistema criado.

A utilização do software ItSimple para a modelagem do domínio e do problema de planejamento facilitaram a implementação dos mesmos, pois usa uma representação visual mais interativa e atraente ao usuário, apesar da existência de pequenos "*bugs*" na versão utilizada.

O planejamento independente do domínio FF (*Fast-Foward*), utilizado para gerar a lista de ações, cumpriu perfeitamente seu objetivo, gerando os planos desejados com a eficiência e a velocidade esperada.

7.2 Trabalhos Futuros

Para trabalhos futuros se propõem a implementação das tarefas relacionadas ao veículo automóvel ("Carro"), podendo este agregar uma autonomia maior para sistemas logísticos. É possível a adição de novas funcionalidades ao veículo robô, agregando atuados como garras e ventosas para a realização de novas tarefas, estas por sua vez podem ser agregadas ao domínio de planejamento.

Outro trabalho que pode ser implementado a partir desta dissertação é a integração do sistema de movimentação e planejamento desenvolvido em um ambiente de manufatura virtual, sendo possível simular a fabricação de produtos e o transporte dos mesmos.

8 Referências bibliográficas

Albus, J., Proctor, F. 1996. A reference model architecture for intelligent hybrid control systems. *Triennial World Congress.* San Francisco, 1996.

Arbib, M. A. 1992. Schema theory. *The Encyclopedia of Artificial Intelligence.* ed. 2, 1992, New York, NY: Wiley-Interscience.

Arkin, R. C, Riseman, E. R, Hanson A. R. 1987. AuRA: An architecture for vision-based robot navigation. *Proc. DARPA Image Understanding Workshop, US Dept. of Advanced Research Projects Administration, Bethesda.* 1987.

Arkin, R. C., Balch, T. 1997. AuRA: Principles and Practice in Review. *Jetai.* April, 1997, Vols. 9, 2-3, pag 175-189.

Arkin, R. C.,. 1998. *Behavior-based robotics* . s.l. : MIT Press, 1998.

Bastos, G. S. 2007. Pionner 3 DX basico. *Apostila - Universidade Federal de Itajubá.* Itajubá : s.n., 2007.

Beket, G. A. 2005. *Autonomous Robots: From Biological Inspiration to Implementation and Control.* Cambridge, USA : The MIT Press, 2005.

Bezerra, C. H. 2004. Localização de um Robô Móvel usando Odometria e Marcos Naturais. *Dissertação de Mestrado - Universidade Federal do Rio Grande do Norte.* Natal : s.n., 2004.

Brooks, R A. 1991. New approaches to robotics. *in Science.* 1991, Vols. 253, September.

Brooks, R. A. 1986. A Robust Layered Control System For A Mobile Robot. *IEEE Journal of Robotics and Automation.* 1986, Vols. RA-2.

Cardoso, L. S., Orlando, V., Ferreira, M. G. V., Biancho, A. C. 2005. Aplicação da Tecnologia de Agentes de Planejamento em Operações de Satélites. *Trabalho apresentado no VII Simpósio Brasileiro de Automação Inteligente (SBAI).* São Luis, MA : Instituto Nacional de Pesquisas Espaciais (INPE), 2005.

Choi, Y. H., Lee, T. K., Oh, S. Y. 2008. A line feature based SLAM with low grade range sensors using geometric constraints and active exploration for mobile robot. *Auton Robot.* 24, 2008, Vols. 13-27.

Choset, H., Nagatani, K. 2001. Topological Simultaneous Localization and Mapping (SLAM): Toward Exact Localization Without Explicit Localization. *IEE Transactions on Robotics an Automation.* April, 2001, Vol. 17, 2.

Connell, J. H. 1992 . A hybrid architecture applied to robot navigation. *IEEE Int. Conf. on Robotics and Automation (ICRA).* Nice, France, 1992 .

CPlusPlus. 2008. CPlusPlus. [Online] The C++ Resources Network, 2008. [Citado em: 10 de Março de 2009.] http://www.cplusplus.com/.

Davoren, J., Nerode, A. 1995. Logics for hybrid systems. (1995). *IEEE in Hybrid Systems: Theory & Applications.* 1995.

Deitel, H. M., Deitel, P. J. 2007. *C++: How to Program.* s.l. : Pearson, 2007.

Edelkamp, S., Hoffman, J. 2004. PDDL 2.2: The Language for Classical Part of the 4th International Planning Competition. *IPC-4.* 2004.

Faria, G. 2006. Uma arquitetura de controle inteligente para Múltiplos Robôs. *Tese de doutorado - Universidade de São Paulo.* São Carlos : s.n., 2006.

Fikes, R. E., Nilsson, N. J. 1970. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence Group, Tecnical note 43.* SRI Project 8259, 1970.

Firby, R. J., Prokopowicz, P. N., Swain, M. J. 1998. Artificial intelligence and mobile robots: case studies of successful robot systems. *MIT Press* . Cambridge, MA, USA, 1998.

Ghallab, M., Nau, D., Traverso, P. 2004. *Automated Planning - Theory and Practice.* s.l. : Elsrvier, 2004.

Gordon, T. W. 1988. Motion Planning for anAutonomous Vehicle. *IEEE International Conference.* 1988, Vols. 1, pag 529-533, ISBN: 0-8186-0852-8.

Grassi Jr, V. 2006. Arquitetura Híbrida para Robôs Móveis Baseada em Funções de Navegação com Iteração Humana. *Tese de Doutorado - Universidade de São Paulo.* São Paulo : s.n., 2006.

Hoffmann. 2001. FF: The Fast-Forward Planning System. *AI Magazine.* No 3, 2001, Vol. Vol 22.

Hoffmann, J. 2000. A Heuristic for Domain Independent Planning and its Use inan Enforced Hill-climbing Algorithm. *Foundations of Intelligent Systems.* 2000, Vol. Volume 1932/2009.

IPC6. 2008. International Planning Competition 6. *The Sixth International Planning Competition.* [Online] ICAPS, 2008. [Citado em: 2009 de 05 de 18.] http://eecs.oregonstate.edu/ipc-learn/.

Keyder, E., Geffner, H. 2008. The FF(ha) Planner for Planning with Action Costs. *Association for the Advancement of Artificial.* (www.aaai.org), 2008.

Laumond, J. P. 1998. *Robot Motion Planning and Control.* s.l. : Springer, 1998.

Lester, P. 2005. policyalmanac. *A* Pathfinding Beginners.* [Online] http://www.policyalmanac.org/games/aStarTutorial_port.htm, 18 de July de 2005. [Citado em: 10 de April de 2009.]

Maes, P. 1989. The dynamics of action selection. *Proceedings of the Eleventh International Joint Conference on Articial Intelligence.* Detroit, v. 2, 1989.

Meystel, A. 1991. *Autonomous mobile robots : vehicles with cognitive control* . Singapore : World Scientific, 1991. Series in automation.

MobileRobots A. 2009. ARIA. *MobileRobots.* [Online] 01 de Maio de 2009. [Citado em: 05 de Dezembro de 2009.] http://robots.mobilerobots.com/wiki/ARIA.

MobileRobots B. 2009. MobileSim. *MobileRobots.* [Online] Mobile Robts INC, 11 de Maio de 2009. [Citado em: 05 de Dezembro de 2009.] http://robots.mobilerobots.com/MobileSim/.

MobileRobots. 2006. Pioneer 3 Operations Manual. 2006. 3.

Murphy, R. 2000. *Introduction to AI Robotics.* Cambridge, Massachusetts : MIT Press, 2000.

Oliveira, L. R. Maio de 2005. Uma Abordagem para Visão Artificial em Robótica. *Dissertação de Mestrado - Universidade Federal da Bahia.* Slavador : s.n., Maio de 2005.

Pednault, E. 1987. Toward a Mathematical Theory of Plan Synthesis. *PhD Dissertation, Computer Science Department, Stanford University.* 1987.

Pieri, E. R. 2002. Curso de Robótica Móvel. *Apostila - Universidade Federal de Santa Catarina.* Florianópolis : Março, 2002.

Player. 2009. The Player Project. [Online] 20 de Outubro de 2009. [Citado em: 05 de Dezembro de 2009.] http://playerstage.sourceforge.net.

Ribeiro, C., Costa, A.,Romero, R. 2001. Robôs móveis inteligentes: Princípios e técnicas. *Anais do XXI Congresso da Sociedade Brasileira de Computação – SBC.* 2001.

Romano, V.,F. 2002. *Robótica Industrial.* São Paulo : Edgard Blucher, 2002.

Rosenschein, S., Kaelbling, L. 1986. The synthesis of digital machines with provable epistemic properties. *Proceedings Conference on Theoretical aspects of reasoning about knowledge.* San Francisco, CA: Morgan Kaufmann Publishers Inc, 1986.

Russell, S., Norvig, P. 2003. *Artificial Intelligence: A Modern Approach.* New Jersey : Prentice Hall, 2003. ISBN 0-13-790395-2.

Siegwart, R., Nourbakhsh, I. R. 2004. *Introduction to Autonomous Mobile Robots.* s.l. : Bradford Book, 2004.

UML. 2009. UML. [Online] OMG`s, 08 de Janeiro de 2009. [Citado em: 05 de Dezembro de 2009.] http://www.uml.org/.

Vaqueiro, S. T. 2007. ItSimple: Ambiente Integrado de Modelagem e Análise de Domínios de Planejamento Automático. *Dissertação de Mestrado - Universidade de São Paulo.* São Paulo : s.n., 2007.

Vidigal, M. C. A. C. F. 2007. Agentes Polimórficos Dinâmicos Inteligentes. *Dissertação de mestrado - Universidade Federal de Itajubá.* Itajubá : s.n., 2007.

Apêndice A – Código do Exemplo do Capítulo 5.1.2 em PDDL versão 2.2

```
; Arquivo Modelagem do Domínio
(define (domain Mundo_CRTi)
   (:requirements :typing :fluents :quantified-preconditions)
   (:types
     Veiculo - object
     Robo - Veiculo
     Carro - Veiculo
     Lugar - object
     Local - Lugar
     Garagem - Lugar
     Pacote - object
     Territorio - object
   )
   (:predicates
     (veiculoEstaNo ?vei - Veiculo ?lug - Lugar)
     (pctEstaNoVeic ?pac - Pacote ?vei - Veiculo)
     (pctEstaNoLugar ?pac - Pacote ?lug - Lugar)
     (noTerritorio ?lug - Lugar ?ter - Territorio)
     (vizinho ?lug - Lugar ?lug1 - Lugar)
   )
   (:functions
     (capacidade ?vei - Veiculo)
     (cargaRoboAtual ?rob - Robo)
     (cargaCarroAtual ?car - Carro)
   )
   (:action mover
    :parameters (?robo - Robo ?deOnde - Lugar ?paraOnde - Lugar ?territorio -
Territorio)
    :precondition
     (and
       (veiculoEstaNo ?robo ?deOnde)
```

```
    (noTerritorio ?deOnde ?territorio)
    (noTerritorio ?paraOnde ?territorio)
    (vizinho ?deOnde ?paraOnde)
   )
 :effect
   (and
    (veiculoEstaNo ?robo ?paraOnde)
    (not (veiculoEstaNo ?robo ?deOnde))
   )
)

(:action carregaRobo
 :parameters (?robo - Robo ?pacote - Pacote ?lugar - Lugar)
 :precondition
   (and
    (veiculoEstaNo ?robo ?lugar)
    (> (capacidade ?robo) (cargaRoboAtual ?robo))
    (pctEstaNoLugar ?pacote ?lugar)
   )
 :effect
   (and
    (increase (cargaRoboAtual ?robo) 1)
    (pctEstaNoVeic ?pacote ?robo)
    (not (pctEstaNoLugar ?pacote ?lugar))
   )
)

(:action descarregaRobo
 :parameters (?robo - Robo ?pacote - Pacote ?lugar - Lugar)
 :precondition
   (and
    (veiculoEstaNo ?robo ?lugar)
    (pctEstaNoVeic ?pacote ?robo)
   )
```

```
 :effect
  (and
    (decrease (cargaRoboAtual ?robo) 1)
    (pctEstaNoLugar ?pacote ?lugar)
    (not (pctEstaNoVeic ?pacote ?robo))
  )
)

(:action viajar
 :parameters (?carro - Carro ?deOnde - Garagem ?paraOnde - Garagem)
 :precondition
  (veiculoEstaNo ?carro ?deOnde)
 :effect
  (and
    (veiculoEstaNo ?carro ?paraOnde)
    (not (veiculoEstaNo ?carro ?deOnde))
  )
)

(:action carregarCarro
 :parameters (?carro - Carro ?pacote - Pacote ?lugar - Garagem)
 :precondition
  (and
    (veiculoEstaNo ?carro ?lugar)
    (pctEstaNoLugar ?pacote ?lugar)
    (> (capacidade ?carro) (cargaCarroAtual ?carro))
    (pctEstaNoLugar ?pacote ?lugar)
  )
 :effect
  (and
    (increase (cargaCarroAtual ?carro) 1)
    (pctEstaNoVeic ?pacote ?carro)
    (not (pctEstaNoLugar ?pacote ?lugar))
    (not (pctEstaNoLugar ?pacote ?lugar))
```

```
      )
    )

    (:action descarregarCarro
     :parameters (?carro - Carro ?pacote - Pacote ?lugar - Garagem)
     :precondition
      (and
        (veiculoEstaNo ?carro ?lugar)
        (pctEstaNoVeic ?pacote ?carro)
      )
     :effect
      (and
        (decrease (cargaCarroAtual ?carro) 1)
        (pctEstaNoLugar ?pacote ?lugar)
        (not (pctEstaNoVeic ?pacote ?carro))
      )
    )

)

; Arquivo Modelagem do Problema

(define (problem Problem1)
    (:domain Mundo_CRTi)
    (:objects
      CRTI - Territorio
      SALA2 - Local
      SALA1 - Local
      GARAGEM - Garagem
      ZAFIRA - Carro
      ROBOCRTI - Robo
      PACOTE1 - Pacote
    )
    (:init
```

```
    (noTerritorio SALA2 CRTI)
    (noTerritorio SALA1 CRTI)
    (noTerritorio GARAGEM CRTI)
    (vizinho SALA1 SALA2)
    (vizinho SALA2 SALA1)
    (vizinho GARAGEM SALA2)
    (vizinho SALA2 GARAGEM)
    (veiculoEstaNo ZAFIRA GARAGEM)
    (pctEstaNoVeic PACOTE1 ZAFIRA)
    (veiculoEstaNo ROBOCRTI SALA1)
    (= (cargaCarroAtual ZAFIRA) 0)
    (= (capacidade ZAFIRA) 10)
    (= (cargaRoboAtual ROBOCRTI) 0)
    (= (capacidade ROBOCRTI) 2)
  )
  (:goal
    (pctEstaNoLugar PACOTE1 SALA1)
  )
)
```

Apêndice B – Exemplo de Algoritmo A* baseado em (Lester, 2005)

Assumindo que nós temos alguém que quer ir do ponto A ao ponto B e que uma parede separa os dois pontos, isto está ilustrado abaixo, onde o ponto de partida A é verde, o ponto B de destino é vermelho, e os quadrados pintados azuis é uma parede entre eles.

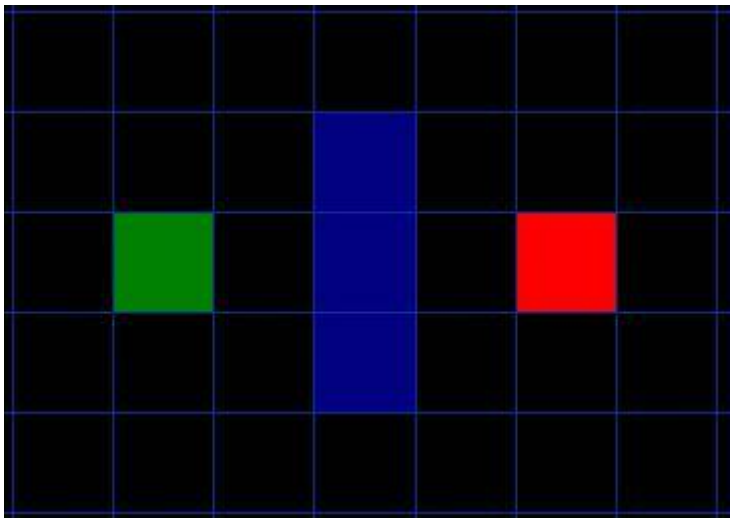

Figura 42 – Exemplo em (Lester, 2005)

Tendo uma discretização do mapa do ambiente, pode-se começar a busca pelo melhor caminho a partir do ponto de origem, chamado aqui de ponto "*A*". Acrescente o ponto "*A*" a uma "lista aberta" de células a serem consideradas. A "lista aberta" é uma lista de células que necessitam ser verificadas. Verifique as células adjacentes a "*A*", ignorando células consideradas ocupadas. Acrescente-os à "*lista aberta*". Para cada uma destas células, salve o ponto "*A*" como sua "*célula pai*". Portanto, a estrutura de cada célula precisa armazenar, além dos valores da função de custo, o endereço da célula ocupada no passo anterior da movimentação, chamada de "*célula pai*".

Remova "*A*" da "*lista aberta*" e a acrescente a uma "*lista fechada*" de células que não precisem ser mais verificadas.

Na Figura 44, pode-se observar o que foi realizado até o momento pelo algoritmo da metodologia, a célula com borda em azul indica que ela foi acrescentada à "*lista fechada*". Todas as células adjacentes estão agora na "*lista aberta*" para serem conferidas e são representadas com a borda em verde. Cada uma das células tem um ponteiro cinza que aponta para trás, a seu anterior, que é a "*célula pai*".

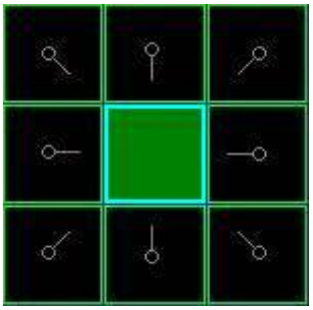

Figura 43 – Representação gráfica da metodologia A* - 1 (Lester, 2005)

Após os passos descritos, é escolhida a célula com o menor valor de função de custo para ser expandida, isto é, para se tornar a célula *"A",* sendo os passos anteriores repetidos. A metodologia de cálculo da função de custo já foi mencionada acima, mas vale ressaltar que o valor da função $g(n)$ para uma célula filha nada mais é que o valor da função da célula pai mais o valor do deslocamento entre os centros das células pai e filha.

Na Figura 44, pode ser visto as células varridas pelo algoritmo com os valores da função de custo calculados. Os quadrados em azul escuro representam células ocupadas por obstáculos e a em vermelho representam o alvo final.

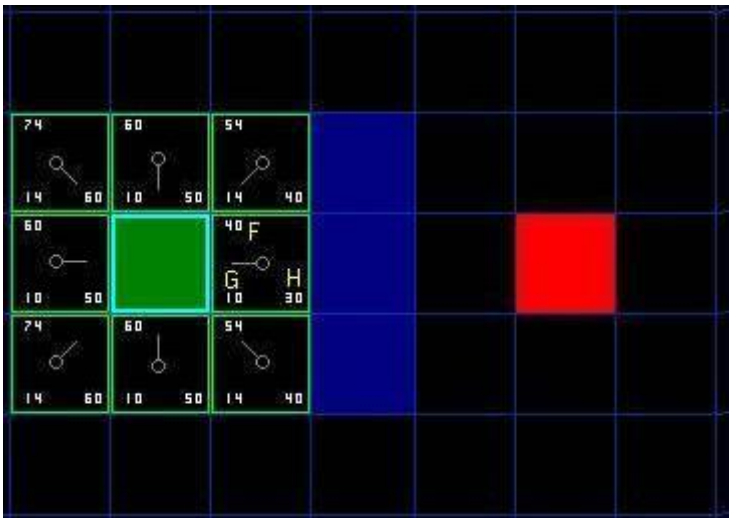

Figura 44 – Representação gráfica da metodologia A* - 2 (Lester, 2005)

Como mencionado, a célula de menor custo da "*lista aberta*" é escolhida para ser expandida, ela é retirada da "*lista aberta*" e acrescentada a "*lista fechada*". As células adjacentes a esta, desde que não estejam ocupadas, não estejam na "*lista aberta*" ou na "*lista fechada*", são acrescentadas a "*lista aberta*" e recebem a atual célula *"A"* como sendo a "*célula pai*".

Se alguma célula adjacente já estiver na "*lista aberta*", deve ser conferido se o valor da função de custo presente na célula da lista é menor que a função de custo calculada através deste novo caminho. Caso não seja menor, esta célula deverá ser modificada, recalculando os valores de $f(n)$ e modificando a informação da "*célula pai*", para a de menor caminho.

A Figura 45 exemplifica esta operação. Como se pode ver pela figura, foi escolhida para a expansão a célula com o menor valor de $f(n)$, a de custo igual a 40, ela foi colocada na "*lista fechada*" (mostrado pela borda em azul claro) e as suas células adjacentes foram examinadas. As células em azul escuro estão ocupadas e não foram acrescentas a "*lista aberta*", as demais células adjacentes já se encontram na "*lista aberta*", sendo necessária somente a comparação do valor do custo atual com o valor recalculado caso fosse necessário a passagem pela célula de $f(n) = 40$. Por exemplo, na Figura 45 a célula verificada a direita mais acima tem o valor de custo igual a 54, este valor foi calculado considerando o caminho que seria sair da célula inicial (célula em verde) e ir diagonalmente até célula em questão. Calculando o valor total do custo caso fosse necessário a passagem pela célula de valor igual a 40 para chegar a de valor 54, o custo dessa operação seria de 60, portanto maior que o caminho anterior de valor 54 sendo um caminho pior a ser seguido.

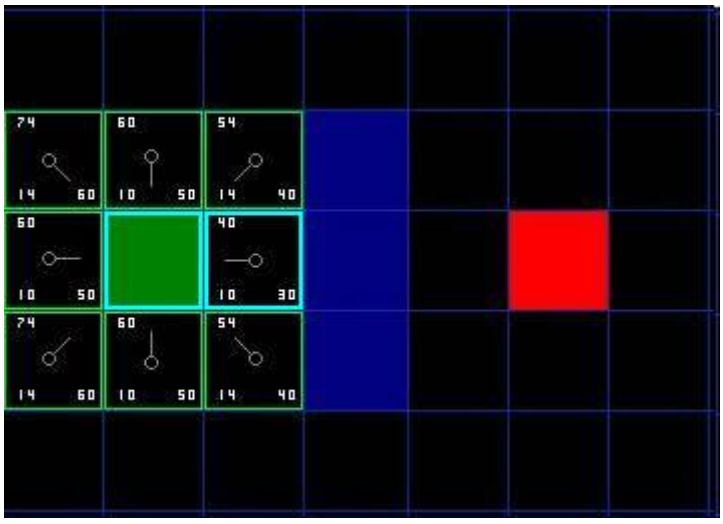

Figura 45 – Representação gráfica da metodologia A* - 3 (Lester, 2005)

Verificando-se todas as células adjacentes na Figura 45, pode-se notar que todos os caminhos para chegar até elas passando pela célula de valor igual a 40 são piores que as existentes na "*lista aberta*", não ocorrendo modificação da lista.

Com esta iteração concluída, será novamente expandida a célula de menor valor de $f(n)$, repetindo-se o procedimento comentado anteriormente.

Na Figura 46, pode-se ver a próxima iteração do exemplo da Figura 45, onde foi escolhida para expansão a célula de $f(n) = 54$, mais abaixo da expandida anteriormente.

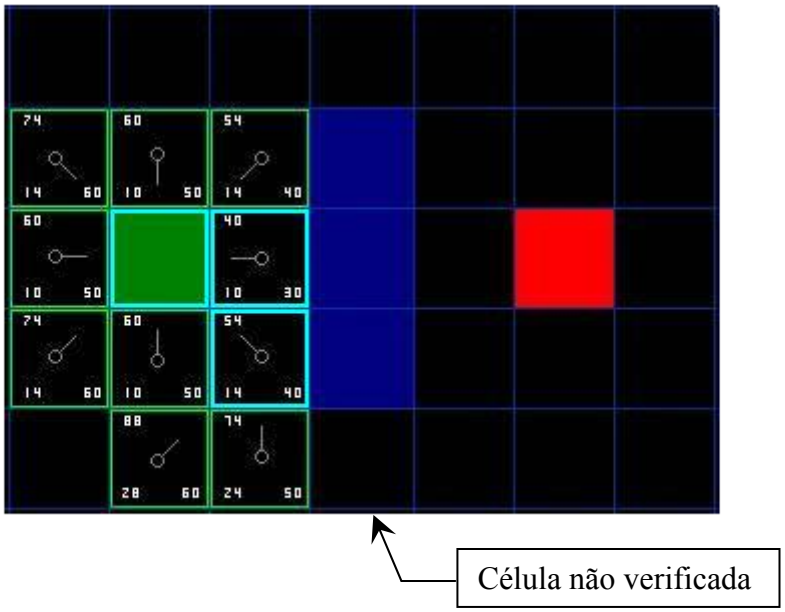

Figura 46 – Representação gráfica da metodologia A* - 4 (Lester, 2005)

Nesta figura, Figura 46, pode-se notar que nem todas as células passíveis de verificação foram verificadas. Isto acontece porque como a célula não verificada está na esquina de uma célula ocupada, o robô não conseguiria se movimentar diagonalmente sem esbarrar na ocupada. Esta verificação é feita para todas as células antes de prosseguir com a operação.

Estes procedimentos são repetidos até que se consiga colocar a célula alvo na "*lista fechada*" ou a "*lista aberta*" esteja vazia. Caso a "*lista aberta*" fique sem elementos, o problema não tem solução.

No caso de se colocar a célula alvo na "*lista fechada*", o caminho para se chegar ao objetivo foi encontrado. A Figura 47 ilustra esta ocorrência no exemplo criado por (Lester, 2005) e explanado nesta seção do trabalho. Ela mostra todas as células que foram verificadas até se chegar ao alvo (quadrado em vermelho).

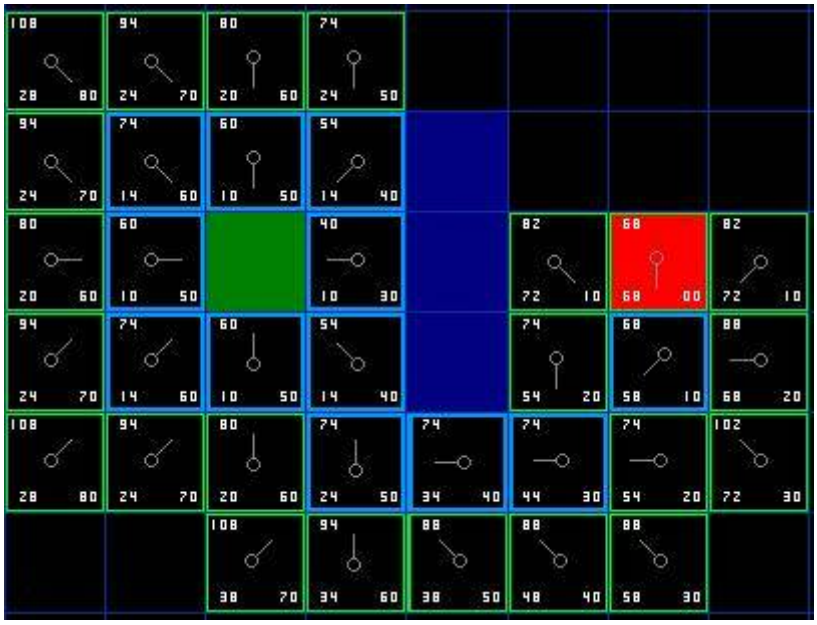

Figura 47 – Representação gráfica da metodologia A* - 5 (Lester, 2005)

A partir do momento em que a célula objetivo foi colocada na "*lista fechada*" o caminho encontrado é definido seguindo-se as informações de paternidade contida na célula alvo até se chegar à célula inicial. Utilizando-se a Figura 47 para exemplificar, para descobrir o caminho alcançado pelo algoritmo neste exemplo, basta seguir os ponteiros de cor cinza que indicam a "*célula pai*" da célula verificada, partindo do alvo (quadrado em vermelho), até se alcançar a célula inicial (quadrado em verde). Este caminho é mostrado pela Figura 48.

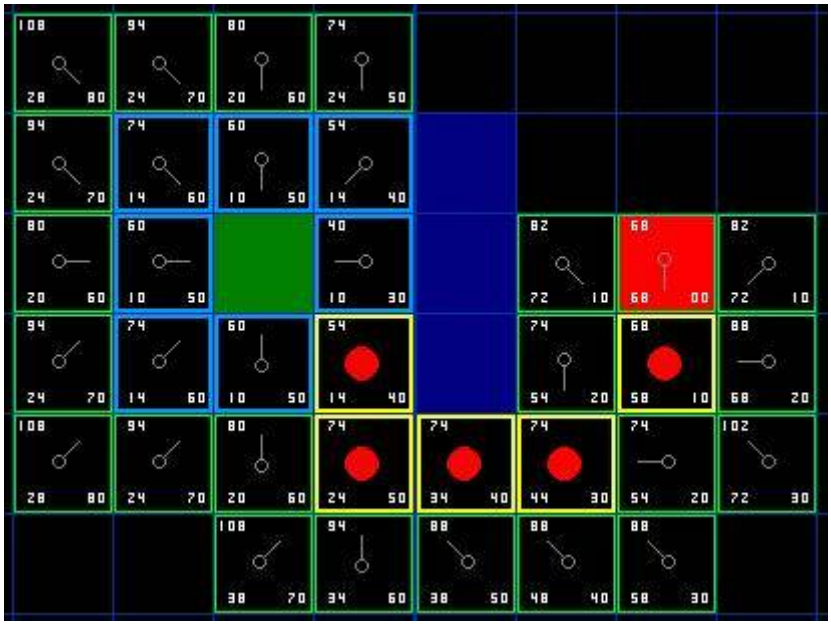

Figura 48 – Representação gráfica da metodologia A* - 6 (Lester, 2005)

Apêndice C – Família de Robôs Pioneer

A *Pioneer* é uma família de robôs, com tração em duas ou quatro rodas, incluindo os modelos *Pioneer* 1 e *Pioneer* AT, *Pioneer* 2-DX, -DXe, -DXf, -CE, -AT, o *Pioneer* 2-DX8/Dx8 Plus e –AT8/AT8 Plus, e os mais novos *Pioneer* 3-DX e –AT. Estas pequenas plataformas de pesquisa e desenvolvimento compartilham uma arquitetura comum de software com todas as outras plataformas *MobileRobots*, incluindo os robôs móveis *AmigoBot*, *PeopleBot V1*, *Performance PeopleBot* e *PowerBot*. A plataforma *MobileRobots* define os padrões para robôs móveis inteligentes por conter os componentes básicos para sensoriamento e navegação em um ambiente de mundo real(Bastos, 2007).

Figura 49 – Robôs da Plataforma MobileRobots (Bastos, 2007)

Apesar da plataforma de software ser comum a todos os membros da família, nos testes realizados no simulador foi considerado a utilização do modelo *Pioneer 3-DX.* Este modelo apresenta duas rodas com movimentação diferencial, motores DC com caixas de redução e *encoders* óticos nas duas rodas, propiciando o controle de posicionamento pela marcação da odometria, Figura 50.

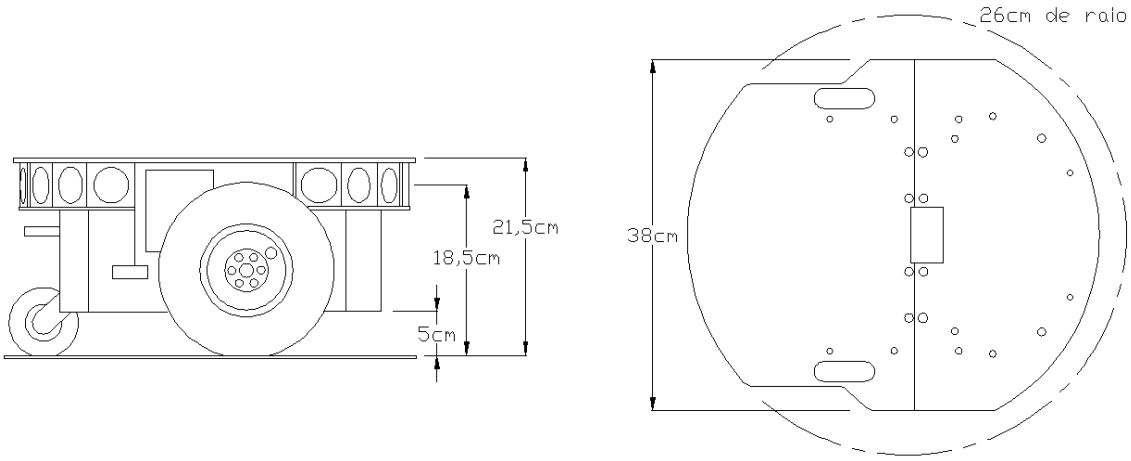

Figura 50 – Dimensional do robô Pioneer 3-DX (MobileRobots, 2006)

Este modelo de robô ainda apresenta um conjunto de sensores ultra-sônicos em configuração de semi-anel, cada um com oito transdutores que permitem a detecção de objetos e informação para se evitar colisões, reconhecimento de padrões, localização e navegação. As posições dos sonares (Figura 51) em todos os Pioneer 3 são fixas: um em cada lado, e seis à frente com intervalo de 20 graus (Bastos, 2007). No caso do modelo utilizado foram colocados dois conjunto de anéis, propiciando uma varredura do ambiente em 360 graus.

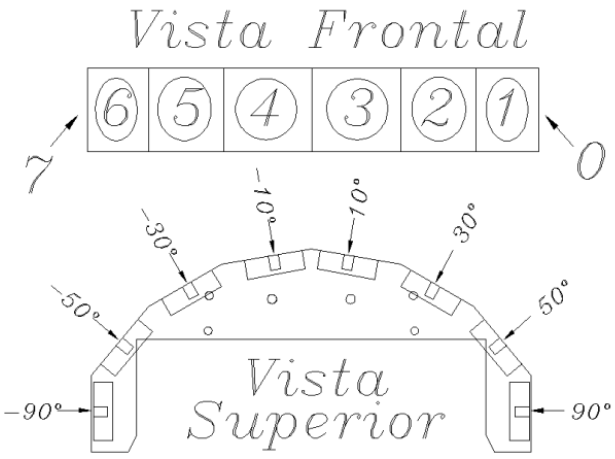

Figura 51 – Semi-anel de sonares (MobileRobots, 2006)

Para a detecção de eventos onde ocorre o contato físico do robô a algum objeto do ambiente, nas simulações efetuadas foi considerada a utilização de um anel de *Bumpers* (chaves de contato), em toda a lateral do robô, conforme se pode ver na Figura 52. Elas permitem a identificação imediata de um choque físico do robô ao ambiente.

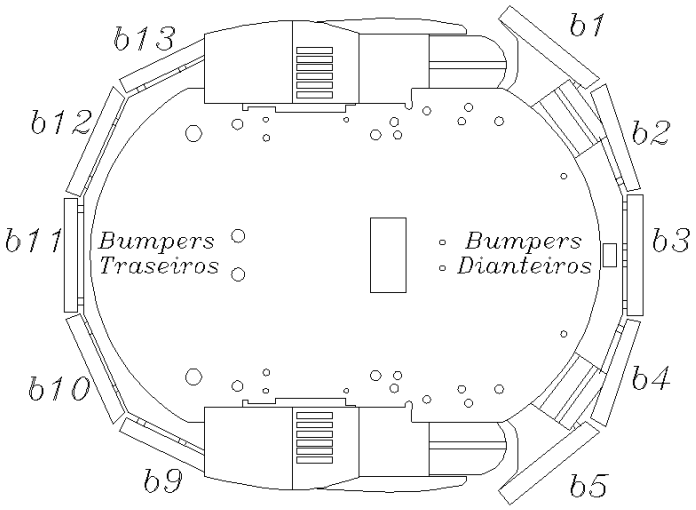

Figura 52 – Bumpers instalados em toda a lateral do robô (MobileRobots, 2006).

O controle interno do robô é realizado através de um microcontrolador rodando um sistema operacional desenvolvido para o controle de todas as funcionalidades de baixo nível do equipamento, este sistema operacional é chamado de *ARCOS*.

Todas as plataformas MobileRobots funcionam como um servidor em um ambiente cliente-servidor. Seu microcontrolador lida com as operações de baixo nível, incluindo o controle da velocidade e posição, leitura de todos os sensores, como por exemplo, as leituras dos sonares e *bumpers*. Para completar a arquitetura cliente-servidor, as plataformas MobileRobots necessitam estar conectados a um PC através de uma comunicação serial. Este PC se conecta diretamente ao microcontrolador do robô executando funções de alto nível como: planejamento das execuções, reconhecimento de características do ambiente, localização, navegação, e assim por diante (MobileRobots, 2006).

Um grande benefício da arquitetura cliente-servidor é que diferentes servidores robôs podem ser executados usando o mesmo alto nível cliente. Vários clientes também podem compartilhar a responsabilidade por controlar um único servidor robô, isso permite experiências distribuídas de controle, comunicação e planejamento.

As formas de comunicação cliente-servidor podem ser vistas na Figura 53.

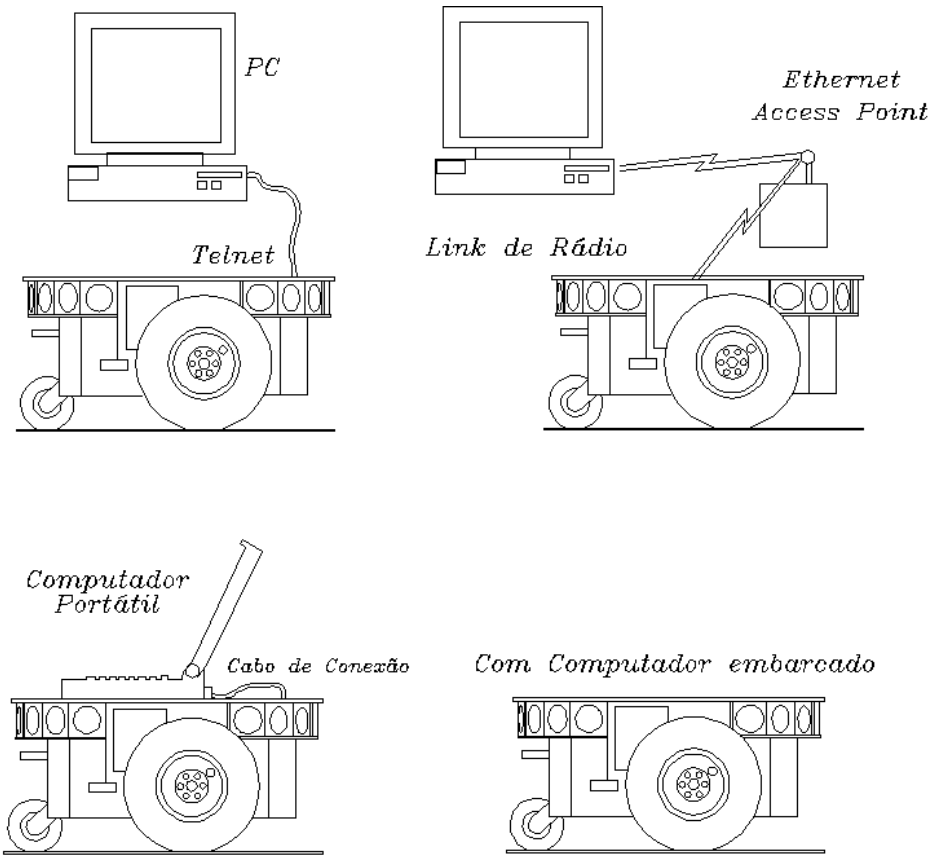

Figura 53 – Formas de comunicação entre PC e robô (MobileRobots, 2006)

Para a criação dos programas de alto nível que são carregados e utilizados no PC (*Personal Computer*) conectado ao robô, existe uma biblioteca de funções chamada ARIA (*Advanced Robotics Interface for Applications*), esta biblioteca foi utilizada para o desenvolvimento deste trabalho.