Valdir Grassi Junior

Arquitetura Híbrida para Robôs Móveis Baseada em Funções de Navegação com Interação Humana

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia.

São Paulo
2006

Valdir Grassi Junior

Arquitetura Híbrida para Robôs Móveis Baseada em Funções de Navegação com Interação Humana

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia.

Área de concentração: Engenharia Mecatrônica

Orientador: Prof. Dr. Jun Okamoto Junior

São Paulo
2006

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com anuência de seu orientador.

São Paulo, 21 de junho de 2006.

Assinatura do autor

Assinatura do orientador

Ficha Catalográfica

Grassi Junior, Valdir
Arquitetura Híbrida para Robôs Móveis Baseada em Funções de Navegação com Interação Humana/ V. Grassi Junior.– ed. rev.– São Paulo, 2006.
118 p.

Tese (Doutorado) – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Robôs (Arquitetura; Interação humana; Planejamento) I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t.

À minha família,
com amor e carinho.

Agradecimentos

Gostaria de começar agradecendo aos professores que me orientaram neste trabalho, aos quais devo muito do que foi feito. Ao Prof. Jun Okamoto Jr. pelo suporte e por discussões que foram de grande ajuda na conclusão deste trabalho e elaboração deste documento. Ao Prof. Vijay Kumar que me recebeu muito bem no GRASP Lab, me apresentou à cadeira de rodas SMARTCHAIR, e me orientou durante minha estada na Universidade da Pennsylvania.

Agradeço o apoio financeiro da FAPESP no período em que estive no Brasil, e à CAPES no período em que estive nos EUA.

Sou grato aos colegas do LPA pela paciência e ajuda em tantas coisas, e aos colegas do GRASP com os quais aprendi muito. Em especial, gostaria de agradecer à Sarangi P. Parikh por ter trabalhado comigo no projeto da cadeira de rodas, discutindo idéias, escrevendo artigos, e realizando experimentos e demonstrações. Também gostaria de agradecer ao Rahul Rao por me ajudar a me familiarizar com os detalhes de funcionamento da cadeira de rodas quando cheguei no GRASP. Ao Terry Kientz pela ajuda com a parte mecânica e baterias da cadeira. E é claro, não poderia me esquecer de agradecer ao amigo Luiz Chaimowicz pela ajuda com o software ROCI e pela companhia agradável principalmente durante as horas de almoço na UPenn.

Gostaria de agradecer aos amigos que conheci na Philadelphia por enriquecerem minha estada nos EUA compartilhando de suas diferentes culturas, países, pontos de vista, e alegria contagiante! Muitos deles puderam até experimentar a cadeira de rodas usada neste trabalho. Foi bem divertido! Também gostaria de agradecer aos amigos da IPI do Ipiranga em São Paulo. Obrigado pela amizade, encorajamento, apoio, e conselho. Eles foram minha família estendida em São Paulo. Com toda a certeza, o convívio com todas estas pessoas, tanto na Philadelphia quanto em São Paulo, adicionou muito à minha formação como pessoa.

Por fim, sou extremamente grato aos meus pais e irmãos pelo amor e apoio. Sem eles, de forma alguma teria chegado até aqui! Obrigado pela festa surpresa de retorno ao Brasil! Também sou grato à Deus, minha esperança, quem esteve sempre presente e que nos momentos difíceis alegrou meu coração me fazendo lembrar do Seu amor e cuidado por mim.

‘Would you tell me, please, which way I ought to go from here?’
‘That depends a good deal on where you want to get to,’ said the Cat.
‘I don’t much care where –’ said Alice.
‘Then it doesn’t matter which way you go,’ said the Cat.
‘– so long as I get *somewhere*,’ Alice added as an explanation.
‘Oh, you’re sure to do that,’ said the Cat, ‘if you only walk long enough.’

Lewis Carrol, Alice in Wonderland

Resumo

Existem aplicações na área da robótica móvel em que, além da navegação autônoma do robô, é necessário que um usuário humano interaja no controle de navegação do robô. Neste caso, considerado como controle semi-autônomo, o usuário humano têm a possibilidade de alterar localmente a trajetória autônoma previamente planejada para o robô. Entretanto, o sistema de controle inteligente do robô, por meio de um módulo independente do usuário, continuamente evita colisões, mesmo que para isso os comandos do usuário precisem ser modificados. Esta abordagem cria um ambiente seguro para navegação que pode ser usado em cadeiras de rodas robotizadas e veículos robóticos tripulados onde a segurança do ser humano deve ser garantida.

Um sistema de controle que possua estas características deve ser baseado numa arquitetura para robôs móveis adequada. Esta arquitetura deve integrar a entrada de comandos de um ser humano com a camada de controle autônomo do sistema que evita colisões com obstáculos estáticos e dinâmicos, e que conduz o robô em direção ao seu objetivo de navegação.

Neste trabalho é proposta uma arquitetura de controle híbrida (deliberativa/reativa) para um robô móvel com interação humana. Esta arquitetura, desenvolvida principalmente para tarefas de navegação, permite que o robô seja operado em diferentes níveis de autonomia, possibilitando que um usuário humano compartilhe o controle do robô de forma segura enquanto o sistema de controle evita colisões.

Nesta arquitetura, o plano de movimento do robô é representado por uma função de navegação. É proposto um método para combinar um comportamento deliberativo que executa o plano de movimento, com comportamentos reativos definidos no contexto de navegação, e com entradas contínuas de controle provenientes do usuário.

O sistema de controle inteligente definido por meio da arquitetura foi implementado em uma cadeira de rodas robotizada. São apresentados alguns dos resultados obtidos por meio de experimentos realizados com o sistema de controle implementado operando em diferentes modos de autonomia.

Abstract

There are some applications in mobile robotics that require human user interaction besides the autonomous navigation control of the robot. For these applications, in a semi-autonomous control mode, the human user can locally modify the autonomous pre-planned robot trajectory by sending continuous commands to the robot. In this case, independently from the user's commands, the intelligent control system must continuously avoid collisions, modifying the user's commands if necessary. This approach creates a safety navigation system that can be used in robotic wheelchairs and manned robotic vehicles where the human safety must be guaranteed.

A control system with those characteristics should be based on a suitable mobile robot architecture. This architecture must integrate the human user's commands with the autonomous control layer of the system which is responsible for avoiding static and dynamic obstacles and for driving the robot to its navigation goal.

In this work we propose a hybrid (deliberative/reactive) mobile robot architecture with human interaction. This architecture was developed mainly for navigation tasks and allows the robot to be operated on different levels of autonomy. The user can share the robot control with the system while the system ensures the user and robot's safety.

In this architecture, a navigation function is used for representing the robot's navigation plan. We propose a method for combining the deliberative behavior responsible for executing the navigation plan, with the reactive behaviors defined to be used while navigating, and with the continuous human user's inputs.

The intelligent control system defined by the proposed architecture was implemented in a robotic wheelchair, and we present some experimental results of the chair operating on different autonomy modes.

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

$\mathcal{W}$	Ambiente de trabalho.
$\mathcal{R}$	Robô no ambiente de trabalho.
$\mathcal{B}_i$	Obstáculos no ambiente de trabalho.
$\mathcal{F}_{\mathcal{R}}$	Sistemas de coordenada fixo no robô.
$\mathcal{F}_{\mathcal{W}}$	Sistemas de coordenada fixo no ambiente de trabalho.
q	Configuração do robô.
m	Número de graus de liberdade do robô, ou número de variáveis de configuração.
$\mathcal{C}$	Espaço de configuração.
$\mathcal{C}_{obs}$	Espaço de configuração ocupado por obstáculos.
$\mathcal{C}_{free}$	Espaço de configuração livre.
x	Coordenada do robô no ambiente de trabalho.
y	Coordenada do robô no ambiente de trabalho.
θ	Direção do robô.
q_{init}	Configuração inicial do robô.
q_{goal}	Configuração final de destino.
ϕ	Função potencial de navegação.
ϕ_{att}	Campo potencial de atração.
ϕ_{rep}	Campo potencial de repulsão.
β	Função que representa um circulo no ambiente.
ρ	Raio de um círculo no ambiente.
k	Parâmetro da função de navegação.
Δp	Custo para que o robô se mova de um elemento a outro na grade de ocupação
u_ϕ	Entrada de controle proveniente do plano de movimento.
u_h	Entrada de controle fornecida pelo usuário.
u_g	Entrada de controle referente ao comportamento reativo.
u_m	Entrada de controle do modificada pelo algoritmo de coordenação.
$\hat{t}_\phi$	Versor normal ao plano definido pelo gradiente da função potencial.
$\hat{t}_g$	Versor normal ao plano definido por u_g.
U_ϕ	Semi-espaço no espaço de configuração cuja normal é u_ϕ.
U_h	Semi-espaço no espaço de configuração cuja normal é u_h.

U_g	Semi-espaço no espaço de configuração cuja normal é u_g.
F	Conjunto de soluções de controle admissíveis.
f	Ângulo de uma característica no ambiente em relação ao robô.
z	Distância de uma característica no ambiente em relação ao robô.
v	Velocidade linear do robô.
w	Velocidade angular do robô.
x_c	Coordenada do ponto de controle no robô.
y_c	Coordenada do ponto de controle no robô.
J	Matrix de transformação.
φ	Ângulo entre o vetor u_h e o vetor u_ϕ.
γ	Ângulo entre o vetor $\overline{u}$ e o vetor u_g.

Sumário

1 Introdução

Existem aplicações na área da robótica móvel onde, além da navegação autônoma do robô, é necessária a interação de um usuário humano no controle de navegação do robô. Este é o caso de aplicações como, por exemplo, veículos robóticos tripulados, cadeira de rodas robóticas, exploração espacial, monitoramento de ambientes, dentre outros. No contexto destas aplicações, para tarefas de navegação, o usuário humano deve ter a possibilidade de alterar localmente a trajetória autônoma previamente planejada para o robô. Isto deve ser feito sem que o sistema de controle autônomo seja desabilitado. Assim, este sistema, independente do usuário, deve lidar com situações imprevistas e dinâmicas no ambiente, evitando colisões com obstáculos, por exemplo. Neste caso, os comandos enviados pelo usuário devem ser modificados quando comprometem a sua própria segurança e a do robô. Da mesma forma, em determinadas condições, o sistema de controle também pode modificar os comandos do usuário para que o robô não se desvie demais de um objetivo de navegação pré-estabelecido. Esta abordagem cria um ambiente seguro para navegação em que o homem efetivamente compartilha o controle do robô com o sistema.

Este trabalho procura tratar do desenvolvimento de um sistema de controle inteligente que possua as características apresentadas. Para isso é importante a definição de uma arquitetura de controle para robôs móveis adequada que possibilite que o ser humano interaja com o robô compartilhando o controle do robô com o sistema autônomo.

Um exemplo de aplicação deste sistema seria em cadeiras de rodas robóticas. Estando em um edifício inteligente, capaz de fornecer à cadeira de rodas robótica um mapa com as vias e salas acessíveis, o usuário da cadeira pode querer que ela o leve de forma completamente autônoma de um lugar a outro dentro deste edifício. Para isso, o usuário interage com o sistema da cadeira fornecendo a posição de destino desejada. Então o sistema de controle da cadeira, utilizando o mapa fornecido, decide de forma autônoma qual o melhor caminho até o destino, e conduz a cadeira ao longo deste caminho de forma segura, evitando obstáculos com auxílio dos sensores da cadeira. Nesta situação, o usuário pode chegar até seu destino sem a necessidade de interagir novamente com o sistema. Entretanto, ao longo do caminho, sem a intenção de

abandonar o seu destino final, o usuário pode querer modificar localmente a trajetória autônoma da cadeira. Isso acontece, por exemplo, quando o usuário quiser se aproximar de algo que chame sua atenção ao longo do caminho. Em situações como esta, o usuário pode assumir o controle da cadeira para mudar localmente sua trajetória e então, quando satisfeito, devolve o controle ao sistema de navegação. Neste caso, o sistema opera de forma semi-autônoma. O usuário compartilha o controle da cadeira com o sistema, que permanece ativo evitando colisões e conciliando, a medida do possível, a intenção local do usuário com seu desejo de chegar ao seu destino final. Em uma outra situação, quando a cadeira está fora de um destes edifícios inteligentes e não há um mapa por meio do qual o usuário possa especificar uma posição de destino para movimento autônomo, o usuário pode controlar a cadeira manualmente utilizando um *joystick*, ou por meio de qualquer outra interface que permita o envio de comandos de direção e velocidade à cadeira. Até mesmo em situações como esta, se for da vontade do usuário, o sistema ainda permanece ativo para garantir a sua segurança evitando colisões.

Neste tipo de aplicação, o sistema de controle inteligente da cadeira, ou seja, do robô móvel, deve permitir a operação em diferentes níveis de autonomia e possibilitar que o usuário interaja com o sistema de diferentes formas: o usuário pode especificar um objetivo desejado para navegação autônoma; o usuário acompanha a trajetória que o robô está percorrendo; e o usuário pode compartilhar o controle de movimento autônomo do robô com o sistema.

De forma geral, dependendo do tipo de aplicação e do nível de autonomia do robô, existe uma necessidade diferente de intervenção do homem nas atividades ou controle do robô, definindo a forma como a interação entre homem e robô acontece. Um dos critérios usados para classificar os sistemas de controle onde existe interação entre homem e robô é o nível de autonomia ou de intervenção humana no controle (LUMIA; ALBUS, 1988; YANCO; DURRY, 2002, 2004). De acordo com este critério pode-se dizer que um sistema é tele-operado quando o robô está 100% do tempo sendo controlado por um usuário humano. Por outro lado, um sistema é completamente autônomo quando o operador humano não intervém no controle do robô. Já as situações intermediárias, quando o robô é capaz de fazer parte da tarefa e o operador humano intervém para realizar uma outra parte, podem ser chamadas de controle semi-autônomo.

Uma outra forma de classificar a interação entre homem e robô é observando o papel que o homem exerce ao interagir com o robô (YANCO; DURRY, 2002, 2004). Este papel pode ser o de supervisor, operador, parceiro de tarefa, mecânico ou programador, e, por fim, o de espectador. O homem age como supervisor quando especifica um objetivo para que o robô alcance de forma autônoma, e monitora as ações do robô sem a necessidade de controlá-lo diretamente. O homem age como operador quando precisa interagir de forma mais intensa com o robô,

partindo em direção à tele-operação. Como parceiro de tarefa, o homem age complementando ou cooperando com o robô como colega para realizar uma tarefa. No papel de mecânico ou programador, o homem altera o *software* ou *hardware* do robô. E finalmente, no papel de espectador, o homem não controla o robô mas precisa ter certo entendimento das atividades do robô para que possa compartilhar o ambiente com ele. Um exemplo deste último seria uma pessoa que anda por uma sala onde um robô aspirador de pó está ativo. Algumas vezes a pessoa pode querer se desviar do robô quando este parece não perceber a presença dessa pessoa.

A interação entre homem e robô também pode ser classificada do ponto de vista de tempo e espaço (YANCO; DURRY, 2002, 2004). Neste caso, homem e robô podem estar no mesmo lugar, caracterizando uma interação direta, ou em lugares distintos, caracterizando uma interação remota. Além disso, a interação pode ocorrer no mesmo tempo, caracterizando uma interação síncrona, ou em períodos de tempo distintos, caracterizando uma interação assíncrona. Por exemplo, a interação do homem com robôs móveis enviados a Marte pode ser classificada como assíncrona, devido ao tempo que se leva entre o envio e o recebimento de comandos, e remota, pois homem e robô não compartilham o mesmo ambiente. Já a interação de uma pessoa sentada em uma cadeira de rodas robótica pode ser considerada síncrona e direta, pois homem e robô compartilham o mesmo espaço de trabalho e interagem sem consideráveis atrasos de tempo.

Tendo como motivação as características da cadeira de rodas mencionada no exemplo dado anteriormente, neste trabalho é proposta uma arquitetura para robôs móveis que permite a implementação de um sistema inteligente de controle voltado para tarefas de navegação, capaz de operar em diferentes níveis de autonomia, possibilitando a interação do usuário de forma direta, síncrona, no papel de supervisor e operador. Um mecanismo é proposto dentro desta arquitetura que possibilita que o usuário, no papel de operador, compartilhe o controle do robô com o sistema inteligente. Este mecanismo juntamente com a definição da arquitetura são as contribuições principais deste trabalho. A arquitetura proposta é geral suficiente para ser implementada em diferentes aplicações de robôs móveis em que o operador precisa compartilhar o controle do robô com o sistema durante a navegação. Neste trabalho, a plataforma utilizada para implementação da arquitetura proposta e obtenção de resultados é uma cadeira de rodas robótica chamada SMARTCHAIR que foi desenvolvida no laboratório GRASP (*General Robotics, Automation, Sensing and Perception*) da Universidade da Pensilvânia. Para este tipo de aplicação em específico, tolerância a falhas e confiabilidade do sistema são questões importantes que se destacam ainda mais do que em outras aplicações robóticas. Afinal um ser humano está sendo conduzido pelo robô, ou seja, pela cadeira de rodas robótica e, além disso, esta cadeira de rodas se move em um ambiente por onde outras pessoas transitam livremente. Entretanto, o enfoque principal deste trabalho se manteve em definir uma arquitetura que resolva o problema

de navegação permitindo que o operador compartilhe o controle do robô com o sistema autônomo. Questões relacionadas a tolerância a falhas e confiabilidade do sistema foram deixadas para estudos e trabalhos futuros.

Esta tese está organizada da seguinte forma. Neste capítulo foi introduzido o contexto na qual esta pesquisa está inserida e foi apresentada a motivação e objetivos deste trabalho. No capítulo 2 são apresentados alguns conceitos relacionados às arquiteturas para robôs, em particular discute-se arquiteturas deliberativas e reativas. No capítulo 3 são apresentadas algumas arquiteturas híbridas e alguns trabalhos que abordam a questão de interação entre homem e robô. No capítulo 4 são apresentados alguns métodos de planejamento de movimento para tarefas de navegação, em particular será apresentado o método e as ferramentas utilizadas na arquitetura que está sendo proposta. No capítulo 5 apresenta-se a arquitetura e o método desenvolvido para permitir que o homem compartilhe o controle do robô com o sistema inteligente em uma tarefa de navegação. No capítulo 6 são apresentados alguns resultados obtidos a partir da implementação da arquitetura em uma cadeira de rodas robótica. Finalmente, no capítulo 7 é apresentada uma breve conclusão das atividades realizadas.

2 Arquitetura para robôs móveis

Existem várias formas de se definir e entender uma arquitetura para robôs (ARKIN, 1998). Segundo James Albus, arquitetura é a descrição de como um sistema é construído a partir de componentes básicos e como estes componentes se encaixam formando o todo (KORTENKAMP; BONASSO; MURPHY, 1998). Já Dowling (1996) se refere a uma arquitetura para robôs como sendo soluções de software e hardware usadas para desenvolver o sistema de controle do robô, incluindo a forma como os componentes da arquitetura se comunicam entre si. Por outro lado, para 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. A partir da definição de arquitetura de computadores, ele define arquitetura para robôs como sendo a disciplina dedicada a desenvolver robôs altamente específicos e individuais a partir de uma coleção de blocos comuns de software. Em adição a estas idéias, para Mataric (1992), uma arquitetura fornece uma maneira principal de organizar um sistema de controle. Sendo que além de fornecer uma estrutura, a arquitetura também impõe restrições na forma como o problema de controle pode ser resolvido. Por fim, Hayes-Roth (1995) se refere a uma arquitetura como sendo os componentes estruturais em que percepção, raciocínio e ação ocorrem. A arquitetura também define a funcionalidade e interface específica de cada um destes componentes, e a intercomunicação entre estes componentes.

As definições apresentadas, embora sejam diferentes e discordem em alguns aspectos, auxiliam na compreensão geral do que é uma arquitetura para robôs. Dessa forma, para os propósitos deste trabalho, entende-se que uma arquitetura para robô móvel descreve uma maneira de se construir o software de controle inteligente do robô, apresentando quais os módulos que 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. Mas mesmo assim, a arquitetura não deve perder seu papel de guia na implementação do sistema de controle. Algumas restrições podem ser impostas na escolha de algoritmos e métodos que podem ser utilizados na implementação dos módulos do sistema.

De forma geral, os módulos ou componentes básicos encontrados em um sistema de controle e que podem ser utilizados para definir uma arquitetura para robôs podem ser classificados em três grupos principais (IYENGAR; ELFES, 1991): (1) *Percepção*, que envolve as atividades de interpretação dos sensores, integração dos sensores, modelagem do mundo real, e reconhecimento; (2) *Planejamento*, que envolve o planejamento de tarefas, a sincronização, e o monitoramento da execução de toda a atividade do robô; e (3) *Atuação*, que envolve as atividades de execução dos movimentos e ações do robô, e controle dos atuadores.

Quanto a classificação de uma arquitetura para robôs, uma das formas de se classificar está relacionada ao uso de deliberação e reatividade dentro do sistema de controle. Segundo este critério, as arquiteturas podem ser divididas em: deliberativas, reativas, e híbridas (deliberativas/reativas) (ARKIN; BALCH, 1997; ARKIN, 1998; MURPHY, 2000; RIBEIRO; COSTA; ROMERO, 2001).

No contexto deste trabalho, deliberação está associada ao processo de tomada de decisão ou planejamento das ações e movimentos do robô utilizando um modelo interno do mundo para que se possa alcançar um determinado objetivo. Em outras palavras, a deliberação envolve uma análise abrangente do modelo interno do mundo para determinar as ações do robô de forma que este alcance seus objetivos. Já a reatividade está associada a execução de ações pré-definidas em resposta a uma informação sensorial obtida localmente. Um exemplo de reatividade é quando o robô é programado para se afastar de um obstáculo assim que este obstáculo é detectado por um dos sensores do robô a uma distância muito próxima.

Este critério de classificação, baseado em deliberação e reatividade, é utilizado neste trabalho como forma principal de organizar a apresentação de exemplos de arquiteturas para robôs. O grau de deliberação e reação encontrado nas arquiteturas pode variar continuamente. Assim, pode se pensar em uma escala que vai desde arquiteturas puramente deliberativas até arquiteturas puramente reativas. Entre estes dois extremos podem existir arquiteturas com diferentes graus de hibridização. A classificação de uma arquitetura em uma das três categorias (deliberativa, reativa ou híbrida) depende muito de onde as fronteira que dividem cada um dos tipos de arquitetura são colocadas na escala que vai de deliberação até reatividade. Como estas fronteiras são muito subjetivas, algumas vezes fica difícil classificar uma arquitetura.

Outros critérios, além do utilizado neste trabalho, podem ser usados para classificar uma arquitetura para robôs. Um exemplo de critério que pode ser usado é a forma como o controle está organizado dentro da arquitetura. Segundo este critério, alguns autores classificam as arquiteturas em hierárquicas, heterárquicas, de camadas (ou subsunção), e mistas (VALAVANIS et al., 1997; FERREIRA, 2003).

Um outro critério que também pode ser usado para classificar arquiteturas está relacionado à abordagem utilizada no desenvolvimento da arquitetura, que pode ser funcional, comportamental, ou uma abordagem mista. Nas arquiteturas que têm seu desenvolvimento orientado a funções, pode-se identificar módulos ou componentes que possuem funções definidas dentro da arquitetura, tais como mapeamento, localização, planejamento, etc. Nas arquiteturas que utilizam uma abordagem comportamental, a arquitetura é formada principalmente por componentes chamados comportamentos. Cada um destes componentes é responsável por uma ação do robô, ou mesmo, responsável pela realização de uma dada tarefa pelo robô. Assim, no desenvolvimento de uma arquitetura funcional o projetista está preocupado em identificar as funções internas da arquitetura e dividi-las em módulos. No desenvolvimento de uma arquitetura comportamental, o projetista está interessado em identificar ações e tarefas que o robô deve realizar, e separar estas ações e tarefas em módulos. Uma arquitetura pode ainda ser desenvolvida procurando mesclar as duas abordagens.

Em robótica, o termo comportamento pode ter diferentes conotações dependendo do contexto em que é utilizado. Na maioria das vezes, um comportamento se refere a um componente específico, definido dentro da arquitetura. Neste sentido, no contexto de arquiteturas reativas, um comportamento pode ser definido como sendo uma função ou procedimento que mapeia entradas sensoriais diretamente a padrões de ações motoras usados para cumprir uma tarefa (MURPHY, 2000). Este tipo de comportamento pode também ser chamado de comportamento reativo.

Ainda considerando comportamento como sendo um componente da arquitetura, pode-se defini-lo de forma mais ampla que a usada em arquiteturas reativas. Mataric (1992), por exemplo, define comportamento como sendo uma lei de controle que satisfaz um conjunto de restrições para alcançar e/ou manter um objetivo em particular. Por exemplo, o comportamento de desvio de obstáculos mantém o objetivo de prevenir colisões do robô com objetos no ambiente, o comportamento de ir para casa atinge o objetivo de chegar em uma região considerada como casa. Cada comportamento pode ter como entrada as informações sensoriais do robô e/ou as informações enviadas por outros comportamentos. Já a saída de um comportamento é enviada para os atuadores do robô e/ou para outros comportamentos do sistema. Dessa forma, o sistema de controle pode ser formado por uma rede de comportamentos que interagem entre si. De acordo com Mataric (1992), comportamentos também podem armazenar informações sobre o ambiente. Assim, estes comportamentos se diferenciam de comportamentos puramente reativos que utilizam apenas informações sensoriais locais para produzir a ação. Uma outra definição que pode ser entendida como complementar a esta é dada por Ribeiro, Costa e Romero (2001) que definem comportamento como uma função que relaciona estímulos sensoriais a ações pro-

duzidas sobre os atuadores do robô, de acordo com um plano realizado a partir de um modelo interno do ambiente. De acordo com esta compreensão, dependendo da complexidade do plano e modelo interno utilizado, os comportamentos podem ser classificados em uma escala gradual que vai de comportamentos puramente reativos, que não utilizam um plano e modelo interno do mundo, até comportamentos deliberativos complexos. Esta compreensão de comportamento geralmente é utilizada no contexto de arquiteturas híbridas. Às vezes, neste contexto, comportamentos podem também ser chamados de controladores ou habilidades.

Além de se referir a um componente da arquitetura, o termo comportamento também pode ser usado em robótica para se referir ao *comportamento emergente* do robô (ARKIN, 1998). O comportamento emergente não é definido de forma explícita na arquitetura, mas surge da interação dos diversos comportamentos (componentes) da arquitetura entre si e com o ambiente quando o sistema de controle é executado. Neste caso, comportamento se refere ao resultado que pode ser observado quando o sistema de controle está em operação, ou seja, as ações que o robô realiza vistas a partir de um observador externo.

De forma bastante geral, uma arquitetura de comportamentos é uma arquitetura em que os componentes básicos são comportamentos, e estes interagem entre si e são coordenados por um mecanismo definido pela arquitetura. Uma discussão mais detalhada, com definições mais específicas a respeito de comportamentos e arquiteturas de comportamentos, é feita por Arkin (1998), Mataric (1992), Ribeiro, Costa e Romero (2001), dentre outros.

Neste trabalho, as arquiteturas de comportamento são classificadas com base no uso de deliberação e reatividade dentro da arquitetura. Dificilmente arquiteturas de comportamento são consideradas puramente deliberativas. Em geral elas são classificadas como reativas ou híbridas dependendo do tipo de comportamento utilizado na arquitetura. Arquiteturas reativas de comportamento, utilizam apenas comportamentos reativos que, por sua vez, fazem uso apenas de informações locais do ambiente percebidas por meio dos sensores do robô. As arquiteturas híbridas de comportamentos utilizam comportamentos reativos e deliberativos. Os comportamentos deliberativos utilizam uma representação global do ambiente e/ou dependem de um plano de movimento gerado a partir desta representação.

No restante deste capitulo, nas seções 2.1 e 2.2, serão apresentados exemplos de arquiteturas deliberativas e de arquiteturas reativas respectivamente. Então, na seção 2.3, apresenta-se algumas conclusões a respeito destas arquiteturas. As arquiteturas híbridas, principal foco deste trabalho, serão apresentadas no capítulo 3.

2.1 Arquiteturas deliberativas

Os primeiros trabalhos em robótica móvel utilizavam uma abordagem puramente deliberativa no sistema de controle inteligente do robô. Esta abordagem deliberativa procura, de certa forma, imitar o processo de planejamento e tomada de decisão do homem para que o robô realize uma tarefa. Para tornar isso possível, na abordagem deliberativa se utiliza o conhecimento que o robô possui sobre o mundo no qual ele está inserido. No sistema de controle do robô, este conhecimento é armazenado em um modelo interno do mundo que pode ser construído a partir do conhecimento *a priori* sobre o ambiente e de informações adquiridas pelos sensores do robô.

O modelo que o robô possui do mundo pode ser representado de formas diferentes. Este modelo pode, por exemplo, ser do tipo simbólico baseado em lógica, tradicionalmente utilizado em inteligência artificial (FIKES; NILSSON, 1971; RUSSELL; NORVIG, 1995). O modelo do mundo pode também ser do tipo geométrico, no qual o ambiente é representado de forma espacial indicando regiões livres e regiões ocupadas por obstáculos. Este tipo de representação é utilizada principalmente quando a tarefa do robô é se mover de uma posição a outra de forma autônoma pelo espaço livre no ambiente e desviar dos obstáculos conhecidos e que foram representados pelas regiões ocupadas no modelo. Além das duas formas de representação mencionadas, existem outras formas de descrever o mundo no qual o robô está inserido. Entretanto, independente de como o modelo do mundo é representado, o objetivo da tarefa a ser realizada pelo robô é definido utilizando este modelo. Por exemplo, no caso do modelo métrico, o objetivo é representado por uma posição ou configuração que o robô deve assumir em uma tarefa de navegação. De forma semelhante, no caso de modelos simbólicos, o objetivo também pode ser expresso por um estado ou configuração que o mundo, incluindo o robô, deve assumir. Entretanto, modelos simbólicos permitem uma descrição mais elaborada do objetivo, e tarefas mais complexas e abrangentes que navegação podem ser realizadas. Esta capacidade de definir e alterar os objetivos e tarefas em relação ao modelo interno confere ao sistema de controle uma certa generalidade e flexibilidade em relação às intenções e desejos do usuário do robô (ARKIN, 1989b; MATARIC, 1992).

Uma vez que o objetivo é definido, o robô deve planejar suas ações. Nesta etapa de planejamento, o modelo interno é amplamente analisado em busca de ações que permitam ao robô atingir o objetivo. Como o robô possui um conhecimento global do mundo ou ambiente, representado no modelo, por meio desta busca pode-se obter um conjunto de ações que possibilite ao robô realizar sua tarefa e atingir seus objetivos de forma eficiente ou ótima. Entretanto, como estas ações dependem do modelo do mundo, este deve ser o mais consistente, confiável, preciso e completo possível dentro do escopo da tarefa do robô. Se a informação contida no

modelo deixar de refletir a realidade do mundo, as ações determinadas pelo planejador podem não conseguir executar a tarefa com sucesso. Em arquiteturas deliberativas, estas mudanças que impedem a realização da tarefa devem ser detectadas pelo robô durante a execução das suas ações, o modelo interno do robô deve ser atualizado, e então, um novo planejamento é feito. Este ciclo, que envolve percepção, atualização do modelo interno, planejamento, e ação, é repetido sempre que necessário.

Devido a sua forte dependência de um modelo interno do mundo para gerar as ações do robô, as arquiteturas deliberativas são mais adequadas para ambientes praticamente estáticos e muito bem controlados. Em ambientes dinâmicos, onde mudanças podem ocorrer a qualquer momento, as informações sobre o ambiente adquiridas no passado, e representadas no modelo interno, podem deixar de ser válidas rapidamente. O uso de arquiteturas deliberativas nestas situações pode se tornar proibitivo quando existe a necessidade de se fazer replanejamentos freqüentes.

Um exemplo de robô móvel cujo sistema de controle foi implementado utilizando uma arquitetura deliberativa é o robô Shakey, um dos primeiros robôs móveis, construído no *Stanford Research Institute* (NILSSON, 1969, 1984). Este robô utilizava um modelo interno simbólico do mundo e um planejador chamado STRIPS (*STanford Research Institute Problem Solver*) (FIKES; NILSSON, 1971). Os robôs HILARE (GIRALT; CHATILA; VAISSET, 1984), o *Stanford Cart* e o *CMU Rover* (MORAVEC, 1977, 1983), citados dentre os trabalhos pioneiros em robótica móvel, também possuíam sistemas de controle deliberativos, mas que se baseavam em modelos internos geométricos do ambiente. Dentre outros exemplos, também pode-se mencionar a arquitetura *Nested Hierarchical Controller* (NHC) desenvolvida por Meystel (1990) e a arquitetura *NIST Realtime Control System* (RCS) desenvolvida por Albus e Proctor (1996) seguindo as mesmas idéias da arquitetura NASREM (ALBUS; MCCAIN; LUMIA, 1989).

2.1.1 *Nested Hierarchical Controller* (NHC)

Na arquitetura NHC (MEYSTEL, 1990), representada na Figura 1, o robô coleta informações sensoriais e combina estas informações em uma estrutura de dados que representa o modelo do mundo. As informações adquiridas podem ser combinadas no modelo do mundo juntamente com conhecimentos fornecidos *a priori*. O modelo do mundo pode conter, por exemplo, mapas de um prédio, regras que dizem para o robô ficar longe de corredores durante o horário de maior movimento de pessoas, etc. Esta atividade de construção e atualização deste modelo faz parte da atividade de sensoriamento, ou percepção, dentro da arquitetura.

Uma vez que o robô possui um modelo do mundo, o planejamento das suas ações pode

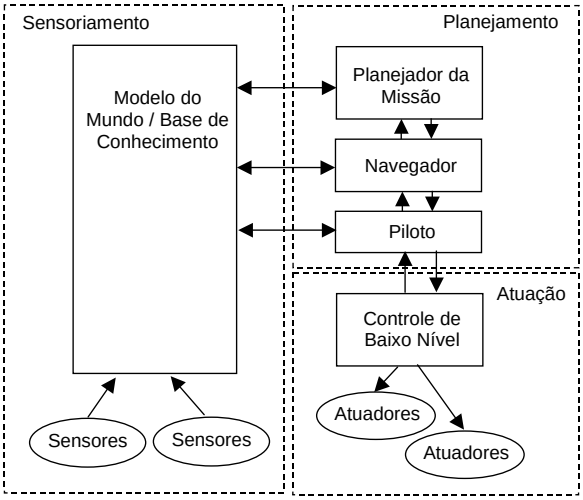

Figura 1: Diagrama da arquitetura NHC.

ser feito com base neste modelo. Na arquitetura NHC o planejamento é decomposto em três níveis hierárquicos ou níveis de abstração: *Planejador da missão, Navegador*, e *Piloto*. Estes três módulos de planejamento são executados seqüencialmente, do nível de abstração mais alto, *Planejador da missão*, até o nível de abstração mais baixo, *Piloto*. De um nível para outro, o planejamento vai se tornando mais específico, local, e detalhado. Assim, o *Planejador da missão* envia trechos da missão para o *Navegador*, que por sua vez, envia trechos de uma trajetória de navegação para o *Piloto*, que finalmente, determina ações que devem ser enviadas ao controlador de baixo nível para que o robô percorra o trecho da trajetória. Isso reflete no tipo de informação do modelo do mundo utilizada por cada um dos módulos. O *Piloto* está mais preocupado com as imediações do robô, enquanto o *Planejador da Missão* observa o mapa de forma mais global. Conforme o robô se move, novas informações são adquiridas por meio dos sensores do robô para atualização do modelo do mundo. Entretanto, quando o modelo do mundo é atualizado, o ciclo completo de planejamento não se repete. Quando necessário, o *Piloto* corrige o movimento local do robô para lidar com eventuais divergências percebidas. Se não for suficiente, então o *Navegador* gera uma outra trajetória, e somente em último caso, o *Planejador da missão* refaz o planejamento da missão.

2.1.2 *NIST Realtime Control System* (RCS)

A arquitetura RCS (ALBUS; PROCTOR, 1996) foi criada para servir como um guia para fabricantes que queiram adicionar mais inteligência a seus robôs. Ela foi baseada na arquitetura NHC e possui a mesma hierarquia no planejamento (Figura 2). No entanto foram introduzidos muitos detalhes e modificações, como, por exemplo, na parte de percepção.

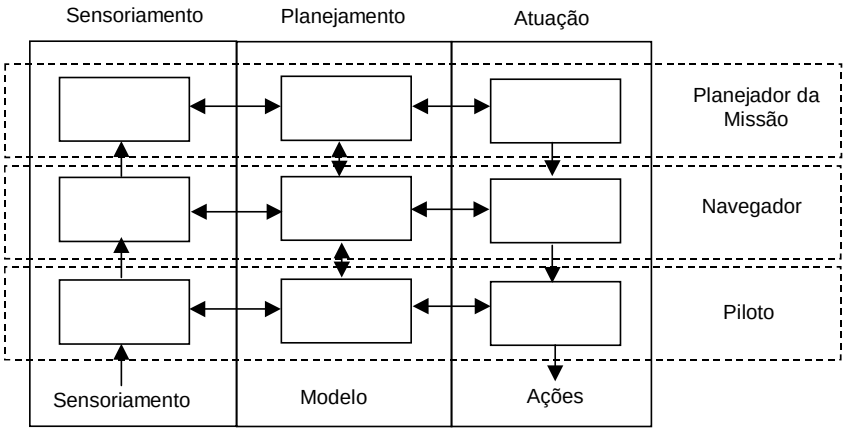

Figura 2: Diagrama da hierarquia na arquitetura RCS.

Na RCS o modelo do mundo também está organizada de forma hierárquica em vários níveis de abstração, acompanhando a hierarquia de planejamento. Módulos de percepção sensorial são responsáveis pela atualização do modelo do mundo, e também acompanham a divisão hierárquica da arquitetura. Estes módulos fazem um pré-processamento dos dados sensoriais extraindo informações em diferentes níveis de abstração para serem integradas no modelo do mundo. Outro diferencial da RCS está na simulação dos planos de ação para verificar se estes satisfazem os requisitos da tarefa. Além disso, também existem módulos que atribuem um valor às informações no modelo do mundo e aos planos de ação gerados, determinando quais informações são mais confiáveis e o que tem maior prioridade (ALBUS, 1990). A RCS e NASREM, que é uma versão da RCS desenvolvida para o JPL (*Jet Propulsion Lab*) (ALBUS; MCCAIN; LUMIA, 1989), permitem a interação de um operador humano nas diversas atividades organizadas nos vários níveis hierárquicos da arquitetura. Em qualquer nível de hierraquia, o homem pode substituir completamente as atividades do sistema com sua capacidade de percepção, tomada de decisões, e controle na execução de tarefas. Isso permite que a autonomia no sistema possa ser desenvolvida de forma incremental, enquanto as atividades dos níveis superiores da arquitetura são realizadas pelo homem. Conforme a disponibilidade de tecnologia em robótica, o robô poderia executar tarefas de maior responsabilidade e de níveis hierárquicos maiores até atingir um grau completo de autonomia. Além do homem substituir a atividade, a arquitetura também prevê a possibilidade do homem compartilhar o controle com o sistema.

2.2 Arquiteturas reativas

As arquiteturas reativas surgiram após as arquiteturas deliberativas, sendo que uma das primeiras arquiteturas reativas foi introduzida por Brooks (1986). A principal motivação das

arquiteturas reativas é permitir a implementação de sistemas de controle que possam responder de forma rápida a uma variedade de eventos ou situações no ambiente, fazendo com que robôs possam operar em ambientes extremamente dinâmicos.

As arquiteturas reativas evitam o uso de um modelo interno do mundo. Um dos lemas da abordagem puramente reativa é o de que "*o mundo é a melhor representação dele mesmo*" (BROOKS, 1991). Dessa forma, o sistema de controle depende fortemente das informações locais obtidas por meio dos sensores do robô, ou seja, de como o robô vê o mundo ao seu redor em um determinado momento. Esta abordagem, por não utilizar um modelo interno do ambiente, geralmente assume que as características do ambiente que possam ser do interesse do robô para realização da tarefa estejam sempre visíveis (ROSENBLATT, 1997a). Como isso pode ser muito restritivo dependendo da aplicação e tarefa, às vezes se faz necessário o armazenamento temporário de informações sobre o ambiente. Até mesmo porque os sensores e os algoritmos que processam os dados sensoriais não estão livres de falhas e de ruídos. Assim, o sistema pode se beneficiar de filtros que utilizam a combinação de informações obtidas a partir de leituras consecutivas dos sensores dentro de um intervalo limitado de tempo.

As ações do robô surgem de respostas pré-definidas a determinadas informações sensoriais. Em geral, a velocidade de processamento e resposta dos sistemas de controle reativos é alta devido a simplicidade no tratamento das informações sensoriais e devido a maneira direta pela qual a percepção, ou estímulo, está associada com uma ação, ou resposta.

Uma arquitetura reativa define a maneira como a informação sensorial é mapeada em uma ação ou resposta, e também define como é feita a coordenação dos diversos pares percepção-ação, ou estímulo-resposta. Estes pares geralmente são chamados de comportamentos reativos. Ao implementar um sistema de controle baseado em uma arquitetura reativa, é de responsabilidade do desenvolvedor determinar quais comportamentos são relevantes para a tarefa a ser realizada pelo robô, e determinar a maneira como eles devem ser integrados utilizando o mecanismo de coordenação definido pela arquitetura. Em geral, quanto mais complexa e elaborada a tarefa do robô, mais complexo é o sistema de controle e, conseqüentemente, mais desafiador de ser projetado.

A coordenação de comportamentos 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 determinando a ação que o robô deve realizar. Uma forma de estabelecer qual dos comportamentos possui prioridade sobre os demais é definindo explicitamente uma hierarquia entre os comportamentos, ou uma regra de arbitragem. Um exemplo é o sistema de supressão e inibição na arquitetura de subsunção (BROOKS, 1986). Já na

coordenação cooperativa, todos os comportamentos ativos contribuem para determinar a ação do robô. Um exemplo bem claro é o método utilizado nos esquemas motores (ARKIN, 1998) em que cada comportamento influencia o movimento do robô por meio de um vetor de força artificial. A ação resultante é determinada pela soma vetorial de todos os vetores de força artificiais que agem no robô em um dado momento.

Tabela 1: Mecanismo de coordenação usado em algumas arquiteturas reativas

Arquitetura Reativa	Método de coordenação	Referências
Subsunção	competitivo, supressão e inibição	(BROOKS, 1986)
Esquema Motor	cooperativo, soma vetorial	(ARKIN, 1998)
Circuito	competitivo, arbitração com abstração	(ROSENSCHEIN; KAELBLING, 1986)
Seleção e Ação	competitiva, nível de ativação	(MAES, 1989)
Colônia	competitiva, supressão	(CONNELL, 1989)

Além da arquitetura de subsunção, e dos esquemas motores, dentre muitos outros exemplos de arquiteturas reativas, pode-se mencionar a arquitetura de circuito (ROSENSCHEIN; KAELBLING, 1986), a de seleção-ação (MAES, 1989), a arquitetura de colônia (CONNELL, 1989). A Tabela 1 mostra os mecanismos de coordenação usado em cada uma destas arquiteturas.

2.2.1 Arquitetura de subsunção

A arquitetura de subsunção (*subsumption architecture*) que foi proposta por Brooks (1986) é uma das arquiteturas mais representativas dentro do paradigma puramente reativo. Nesta arquitetura, os comportamentos são módulos que mapeiam um estimulo ou uma informação sensorial em um sinal ou uma ação motora. Os comportamentos são conectados uns aos outros formando uma rede organizada em camadas de competência. Cada camada é responsável por uma atividade do robô. Nas camadas mais altas estão os comportamentos responsáveis pelo cumprimento de uma tarefa especifica que leva o robô a atingir um determinado objetivo. Nas camadas mais baixas ficam os comportamentos responsáveis por ações básicas do robô, como, por exemplo, desvio de obstáculos.

Os comportamentos em cada uma das camadas funcionam de forma concorrente e independente. Estes comportamentos são coordenados de forma competitiva, sendo que os comportamentos em camadas mais altas têm prioridade em relação aos comportamentos em camadas inferiores. A coordenação é feita por meio de dois mecanismos principais: inibição e supressão. Na supressão de um comportamento por outro, a saída produzida pelo comportamento de

prioridade mais baixa (comportamento suprimido) é substituída pela saída produzida pelo comportamento de prioridade mais alta, entretanto, ambos os comportamento permanecem ativos. Na inibição de um comportamento pelo outro, o comportamento de prioridade mais baixa é desativado pelo comportamento de prioridade superior. Na arquitetura de subsunção a hierarquia entre os comportamentos é definida de forma bastante específica. Um comportamento em uma camada superior só age inibindo ou suprimindo um determinado conjunto pré-definido de comportamentos em camadas inferiores, e não todos eles. Assim, comportamentos que preservam a integridade física do robô e que se encontram nas camadas inferiores não são necessariamente suprimidos ou inibidos por comportamentos em camadas superiores.

2.2.2 Esquema motor

Logo após o surgimento da arquitetura de subsunção, Arkin (1998) propôs um método para implementar comportamentos para robôs móveis baseado na teoria de esquemas proposta por (ARBIB, 1992).

Neste método chamado de Esquema Motor (*Motor Schema*), comportamentos são módulos que expressam a relação entre controle motor e percepção agindo de forma paralela e concorrente no sistema, cooperando uns com os outros para determinar a resposta geral do sistema. A resposta motora de cada comportamento a um dado estimulo é representada na forma de um vetor com magnitude e orientação gerado a partir de um método de campos potenciais artificiais. Os vetores de resposta de cada comportamento são somados resultando na ação que deve ser tomada pelo robô. Assim, não existe arbitragem de um comportamento em relação ao outro. Ao invés disso, cada comportamento contribui para a resposta do sistema. A influência de um comportamento nesta resposta final é determinada por seu peso relativo aos demais. Estes pesos funcionam como parâmetros que podem ser alterados para dar flexibilidade ao sistema. Além dos pesos, um comportamento também pode ter como parâmetro um nível de ativação que determina quando o comportamento deve entrar em ação. Combinando-se um conjunto de comportamentos primitivos é possível criar comportamentos motores mais complexos. No final, o comportamento emergente do robô é resultado da interação dos diversos comportamentos presentes no sistema.

Em um comportamento, um esquema motor é associado a um esquema de percepção. Cada esquema de percepção fornece em tempo hábil a informação especifica que o comportamento precisa para poder reagir. Os esquemas de percepção também podem ser definidos recursivamente, ou seja, cada esquema de percepção pode fornecer um pedaço de informação que juntos são processados por um outro esquema de percepção, resultando em uma informação que pode

ser mais relevante para um dado comportamento.

Quando um comportamento se encontra ativo ele produz uma resposta reativa na forma de um vetor. Este vetor pode ser considerado como sendo uma força artificial que age no robô para movê-lo em determinada direção, indicada pela orientação do vetor, e com uma dada velocidade, indicada pela magnitude do vetor. A magnitude e direção do vetor podem ser funções da informação dada pelo esquema de percepção do comportamento. Comportamentos definidos desta forma se baseiam no método de campos potenciais introduzido por Khatib (1985). Uma variedade de comportamentos pode ser definido, tais como, desvio de obstáculos, mover-se em direção a uma posição específica do ambiente, seguir um líder, mover-se em uma determinada direção, manter-se no centro de um corredor, etc.

É interessante notar que os vetores de resposta são gerados do ponto de vista do robô a partir das informações sensoriais que o robô possui naquela posição. Estes vetores não são computados previamente para todo o ambiente, ou seja, para todas as possíveis posições ou configurações que o robô pode assumir. No caso do desvio de obstáculos, por exemplo, quando o robô percebe um obstáculo a uma dada distância e direção, um vetor de repulsão é gerado pelo comportamento. Este vetor possui direção oposta ao obstáculo, e magnitude em função da distância entre o robô e o obstáculo. Ao mesmo tempo, um outro comportamento pode estar gerando um vetor, que é função da posição atual do robô e de uma posição de destino no ambiente, para atrair o robô para um objetivo de navegação. O vetor resultante vai ser a soma de cada um dos vetores multiplicados pelo peso dado aos seus respectivos comportamentos.

Não existe hierarquia pré-definida para a coordenação. Ao invés disso, os comportamentos são configurados em tempo de execução baseado na intenção e capacidade do robô, e nas restrições do ambiente. Esquemas podem ser instanciados ou desinstanciados a qualquer momento baseado em eventos perceptíveis. Portanto a estrutura é mais parecida com uma rede dinamicamente variável do que com uma arquitetura de camadas.

2.2.3 Arquitetura de circuito

A arquitetura de circuito desenvolvida por Rosenschein e Kaelbling (1986) combina os princípios de reatividade, uso de formalismo lógico (JOHNSON, 1983), e níveis de abstração, como por exemplo, os usados na arquitetura RCS (BARBERA et al., 1984) e no robô Shakey (NILSSON, 1969, 1984).

Nesta arquitetura, os comportamentos podem ser agrupados em composições, que por sua vez podem ser novamente agrupadas, permitindo a obtenção de diferentes níveis de abstração

de comportamentos. A arbitração pode ocorrer dentro de cada nível de abstração. Este tipo de coordenação de comportamentos é chamada de mediação hierárquica, ou arbitração com abstração.

Os comportamentos são expressos utilizando um modelo que usa lógica formal (KAELBLING; ROSENSCHEIN, 1991). Este modelo permite o desenvolvimento de um circuito que representa os objetivos e funcionalidade do robô. O uso de lógica permite estabelecer propriedades para o sistema de controle que podem ser provadas (ROSENSCHEIN; KAELBLING, 1986). A linguagem utilizada nesta arquitetura para programação dos comportamentos e definição dos circuitos é chamada de REX/GAPPS (KAELBLING, 1987, 1988).

2.2.4 Seleção de ação

A arquitetura de seleção de ação (*Action-Selection*) desenvolvida por Maes (1989) utiliza um mecanismo dinâmico para selecionar os comportamentos que devem ser ativados. Ao invés de utilizar uma estratégia pré-definida como na arquitetura de subsunção, a arquitetura de seleção de ação utiliza um nível de ativação para determinar em tempo de execução qual comportamento deve ser selecionado. Este nível de ativação é afetado pela situação atual em que o robô se encontra, pelos objetivos de alto nível, ou pela inibição causada por comportamentos conflitantes. Níveis de ativação também podem ser afetados pelo tempo, podendo decair conforme o tempo passa. O comportamento com nível de ativação maior é escolhido dentre um conjunto de todos os comportamentos cujas pré-condições também estejam satisfeitas. Este processo de seleção é repetido tão rápido quanto possível, e o estado dos níveis de ativação e conseqüentemente o comportamento emergente do robô se altera conforme as circunstâncias do ambiente ao redor do robô mudam. Como nesta arquitetura, diferentemente da arquitetura de subsunção, não existe uma organização clara dos comportamentos em camadas pré-definidas, é difícil predizer qual o comportamento global emergente que o robô apresentará em um ambiente dinâmico.

2.2.5 Arquitetura de colônia

A arquitetura de colônia (CONNELL, 1989) é descendente direto da arquitetura de subsunção. No entanto, ela usa apenas 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.

2.3 Discussão comparativa

Arquiteturas deliberativas e arquiteturas reativas possuem características próprias distintas que fazem com que cada uma se torne adequada para um certo tipo de aplicação.

Nas arquiteturas deliberativas existe uma forte dependência de um modelo interno do mundo, ou do ambiente, no qual o robô opera. Com base neste modelo interno, as tarefas e objetivos do robô podem ser explicitamente definidas pelo usuário. Então, as ações que o robô deve realizar para atingir seus objetivos e cumprir suas tarefas são determinadas por uma etapa de planejamento baseado no modelo do mundo. O planejamento geralmente utiliza um método de busca que considera as informações globais sobre o mundo. Isso permite encontrar um plano de ação para que o robô alcance os objetivos e realize suas tarefas de forma eficiente. Além disso, a capacidade de planejamento e de representação interna do mundo confere ao sistema de controle deliberativo uma certa flexibilidade, generalidade, e adaptação às intenções, tarefas e objetivos que o usuário pode querer definir (ARKIN, 1989b; MATARIC, 1992). Em particular, quando existe uma representação simbólica e um planejamento a nível de missão e tarefa. Entretanto, dependendo do método de planejamento e modelo do ambiente, um alto custo computacional pode ser necessário para o planejamento e atualização do modelo interno do mundo. A atualização do modelo interno deve ser feita quando este deixa de refletir a realidade devido a mudanças no ambiente. Principalmente quando estas mudanças impedem a execução do plano de ação do robô. Neste caso, após atualizado o modelo, um novo plano de ação deve ser formulado. Em ambientes dinâmicos, freqüentes replanejamentos podem ser necessários tornando inviável o uso de uma arquitetura deliberativa. Assim, devido às suas características, as arquiteturas deliberativas são mais adequadas a ambientes controlados onde não ocorrem mudanças freqüentes.

Nas arquiteturas reativas as ações do robô são determinadas a partir de informações sensoriais locais. Modelos internos globais e planejamento baseado nestes modelos não são utilizados. O sistema é construído por pares pré-programados de percepção-ação que se tornam ativos em condições pré-definidas. Cada par pode ser chamado de comportamento reativo, e mapeia por meio de uma certa função um estimulo sensorial a uma resposta. Os diferentes comportamentos reativos e a maneira como cada um deles é combinado dentro do sistema faz com que o robô opere de forma coerente com o propósito para o qual o sistema foi desenvolvido. A identificação dos comportamentos necessários e a integração apropriada dos mesmos deve ser feita na etapa de desenvolvimento do sistema. Dependendo da complexidade da tarefa e variedade de situações nas quais o robô deve operar, o desenvolvimento de um sistema reativo pode ser algo trabalhoso. Principalmente por não armazenarem informações em um modelo interno do

mundo, as arquiteturas reativas podem ser menos gerais e flexíveis que as arquiteturas deliberativas em relação a definição de tarefas e missões (ARKIN, 1989b; MATARIC, 1992). Isso é observado principalmente em sistemas reativos mais reflexivos, que após serem construídos para um determinado propósito, são extremamente rígidos quanto a tarefa que executam. Entretanto, os comportamentos pré-definidos não requerem uma computação pesada e permitem que o sistema quando executado responda rapidamente aos estímulos no meio ambiente. Devido a suas características, as arquiteturas reativas são adequadas a ambientes extremamente dinâmicos.

Tabela 2: Características das arquiteturas deliberativas e reativas

Arquiteturas deliberativas	Arquiteturas reativas
– Modelos internos globais;	– Informações sensoriais locais;
– Modelo interno completo e preciso;	– Sensores robustos;
– Planejamento;	– Ações pré-definidas às informações sensoriais;
– Maior generalidade e flexibilidade na definição de tarefas e objetivos;	– Sistemas mais dedicados às tarefas e problemas específicos;
– Resposta mais lenta às mudanças no ambiente;	– Resposta mais rápida às mudanças no ambiente;
– Adequadas aos ambientes controlados, quase estáticos.	– Adequadas aos ambientes dinâmicos.

A Tabela 2 mostra um contraste entre as principais características das arquiteturas deliberativas e das arquiteturas reativas. De forma geral, o enfoque da abordagem deliberativa está na formulação de um plano de ação que permita atingir os objetivos especificados, e o enfoque da abordagem reativa está na execução das ações do robô em tempo real em um ambiente dinâmico. As arquiteturas híbridas procuram combinar estas duas características principais das abordagens deliberativa e reativa, procurando, ao mesmo tempo, diminuir a restrição em relação ao domínio de aplicação de cada uma destas abordagens. A questão principal é como integrar as abordagens deliberativa e reativa de forma coerente dentro de uma arquitetura híbrida.

3 Arquiteturas híbridas (deliberativa/reativa)

As arquiteturas híbridas, que combinam a abordagem deliberativa e reativa, predominam nos dias atuais. Este tipo de arquitetura utiliza deliberação para planejar as ações do robô a partir de uma representação interna global do conhecimento do mundo, de forma que os objetivos do robô possam ser atingidos eficientemente. Uma vez que as ações são planejadas, a execução do plano de ação é feita utilizando reatividade, pois por meio dela é possível responder rapidamente a mudanças dinâmicas no ambiente. Assim, a arquitetura híbrida procura ser adequada para solução de problemas complexos atingindo objetivos de maneira ótima e eficiente (utilizando deliberação) em ambientes dinâmicos que exigem rapidez na resposta (utilizando reatividade).

A arquitetura híbrida define o papel 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. Algumas estratégias principais de como deliberação, ou planejamento, interage com a reação, ou execução do plano de ação, são identificadas por Arkin (1998):

- *Seleção*: *O planejamento age configurando o sistema de execução.* Os componentes de planejamento determinam um conjunto de comportamentos e seus parâmetros para serem usados durante a execução. O planejamento pode reconfigurar este conjunto conforme necessário, por exemplo, quando ocorre uma falha de execução no sistema.

- *Conselho*: *O planejamento fornece conselhos para o sistema de execução.* Os componentes de planejamento sugerem mudanças que o sistema de execução pode ou não usar. Assim, o planejamento oferece opções de ação, mas os componentes de controle reativo determinam se estas opções são apropriadas.

- *Adaptação*: *O planejamento faz adaptações contínuas no sistema de execução.* A partir de mudanças nas condições do mundo e nos requisitos da tarefa, o planejador continuamente altera o componente reativo que está sendo usado.

- *Adiamento*: *O planejamento é feito somente quando necessário.* O planejador espera para tomar decisões sobre ações até o último momento possível, ou seja, até que estas decisões sejam absolutamente necessárias. Isto permite que informações recentes de sensores possam ser usadas para que se possa gerar um plano de ações mais eficiente do que aquele que teria sido gerado se o planejador não esperasse.

Além dos aspectos apresentados que podem diferenciar uma arquitetura híbrida, percebe-se que algumas arquiteturas possuem uma divisão bem clara da funcionalidade do sistema em módulos ou componentes (MURPHY, 2000). Alguns destes componentes funcionais são: seqüenciador, gerenciador de recursos, cartógrafo, planejador da missão, e monitor de desempenho e solucionador de problemas.

- *Seqüenciador*: Gera um conjunto de comportamentos para ser usado na execução de um plano de ação. Ele também determina a seqüência 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. Ele poderá verificar se o sensor de infravermelho é suficiente para detectar objetos a uma determinada distância, ou se o sensor estéreo pode atualizar a uma velocidade adequada para a velocidade do robô, ou se existe potência suficiente disponível para o sonar produzir leituras confiáveis.

- *Cartógrafo*: Responsável por criar, armazenar, e manter atualizado os mapas e informações espaciais. Ele 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 fazendo progresso no cumprimento de sua tarefa.

Algumas arquiteturas utilizam estes componentes em camadas da arquitetura mais voltada às atividades deliberativas e de interface com a camada reativa de execução das tarefas que utiliza comportamentos. Entretanto, podem 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.

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

3.1 AuRA

A arquitetura AuRA (*Autonomous Robot Architecture*) proposta por Arkin (1986, 1987b) foi uma das primeiras arquiteturas híbridas (deliberativa/reativa). A parte deliberativa da arquitetura é composta por um planejador hierárquico baseado em técnicas tradicionais de inteligência artificial. Este planejador utiliza o conhecimento sobre o ambiente e o conhecimento a respeito dos comportamentos disponíveis no sistema para configurar a parte reativa responsável pela execução da missão ou tarefa do robô. A parte reativa da arquitetura é composta por um controlador de comportamentos baseados na teoria de esquemas (ARBIB, 1992) (Figura 3).

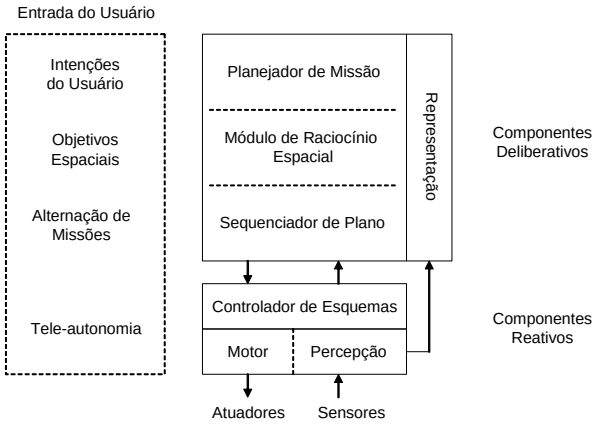

Figura 3: Arquitetura AuRA.

O planejador hierárquico é composto de um *planejador de missão*, de um *módulo de raciocínio espacial*, e um *seqüenciador de plano de execução*. À semelhança dos sistemas de planejamento tradicionais (ALBUS; MCCAIN; LUMIA, 1989; MEYSTEL, 1986; SARIDIS; VALVANIS, 1987), o nível mais alto na hierarquia de planejamento é ocupado pelo *planejador de missão*, responsável por estabelecer objetivos de alto-nível para o robô e restrições dentro das quais o robô deve operar. No nível intermediário da hierarquia está o *módulo de raciocínio espacial*, originalmente chamado de navegador (ARKIN, 1987b). Ele utiliza o conhecimento cartográfico

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. Por fim, o nível mais baixo na hierarquia de planejamento é ocupado pelo *seqüenciador de plano*, também chamado de piloto em trabalhos anteriores. Para cada trecho de navegação gerado pelo módulo de raciocínio espacial, o seqüenciador de plano especifica um conjunto de comportamentos motores que devem ser enviados para execução. Na implementação original, o *seqüenciador de plano* era um sistema rudimentar baseado em regras. Mais recentemente ele foi implementado como um seqüenciador de estados finitos (MACKENZIE; CAMERON; ARKIN, 1995). Finalmente, a coleção de comportamentos (esquemas) especificada pelo seqüenciador de plano é enviada para execução no robô. Neste ponto, a deliberação acaba e se inicia a execução reativa.

O *controlador de esquemas* é responsável pelo controle e monitoramento dos comportamentos reativos em tempo de execução. Cada comportamento motor (ou esquema) é associado a um esquema perceptivo capaz de prover o estimulo requerido para um determinado comportamento. Esta percepção orientada à ação é a base para esta forma de navegação baseada em comportamento (ARKIN, 1990). Cada comportamento gera um vetor de resposta de modo análogo ao método de campos potenciais. Os esquemas operam de forma assíncrona transmitindo seus resultados para um processo (mover-robô) que soma e normaliza estas entradas e transmite o resultado para um sistema de controle de baixo nível responsável pela execução.

Uma vez que a parte reativa começa a ser executada, a parte deliberativa não é reativada a menos que seja detectado uma falha na parte reativa, que tipicamente é indicada por uma falta de progresso na execução da tarefa. Neste ponto, o planejador hierárquico é reativado uma camada por vez, começando da camada inferior e partindo para a camada superior, até que o problema seja resolvido. Primeiramente o seqüenciador do plano é chamado 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 dentro do contexto local, o módulo de raciocínio espacial é chamado e tenta estabelecer um novo caminho que desvie da região onde está ocorrendo o problema. Se ainda assim isso não se mostrar satisfatório, o planejador de missão é chamado e o usuário é informado da dificuldade, podendo desistir da missão ou reformulá-la.

A arquitetura AuRA é uma arquitetura bastante modular, flexível e geral. Como os componentes da arquitetura são modulares, eles podem ser substituídos de acordo com a aplicação e evolução das tecnologias utilizadas para implementar cada um desses módulos. Além das características mencionadas, a arquitetura AuRA também permite que o sistema possa se adap-

tar utilizando métodos de aprendizado. A adaptação é feita alterando-se a importância ou peso dado a cada um dos esquemas motores utilizados na realização da tarefa.

3.2 SFX

A arquitetura SFX (*Sensor Fusion Effects*) (MURPHY; ARKIN, 1992; MURPHY; MALI, 1997; MURPHY, 2000) começou como sendo uma extensão da arquitetura AuRA para introduzir módulos que pudessem tratar de forma mais robusta a informação sensorial. Para isso foram introduzidos módulos de fusão sensorial e módulos para lidar com eventual falha nos sensores, que permitem a recuperação do robô quando estas falhas acontecem. Com o tempo, surgiu a arquitetura SFX (Figura 4) a partir de uma reorganização dos componentes reativos e deliberativos da arquitetura AuRA.

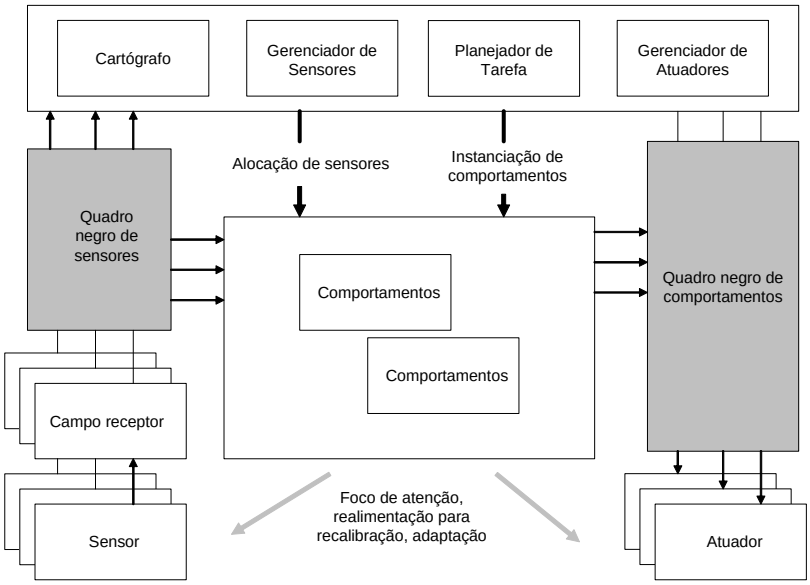

Figura 4: Arquitetura SFX.

O parte deliberativa da arquitetura SFX é dividida em módulos ou agentes especializados em uma área de competência. Em uma camada deliberativa superior está o *planejador de missão* que é responsável em interagir com o homem e especificar as restrições da missão. Em uma camada deliberativa inferior, subordinados ao planejador de missão, estão os módulos que gerenciam os recursos do sistema, ou seja, *planejador de tarefa*, *gerente de sensores*, e *gerente de atuadores*. Cada um destes 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. Estes gerenciadores utilizam técnicas de Inteligência Artificial para planejamento, seqüenciamento, e resolução de problemas, para de-

terminar qual a melhor forma de alocar os recursos em termos de sensores e atuadores dado um conjunto de esquemas motores e perceptivos, de forma que o robô possa apresentar um determinado comportamento. Nesta camada deliberativa inferior também se encontra o *cartógrafo* e *agentes para monitorar o desempenho* do sistema. O cartógrafo é responsável pelo mapeamento usado no planejamento da tarefa. Os agentes que monitoram o desempenho procuram observar o progresso do robô em direção ao seus objetivos, e são capazes de perceber quando o robô não está obtendo sucesso em alcançar estes objetivos.

A parte reativa da arquitetura SFX é dividida em duas camadas: uma camada composta de comportamentos estratégicos e outra de comportamentos táticos. Os comportamentos estratégicos ditam qual deve ser o comportamento geral do robô, como, por exemplo, qual a direção estratégica 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. Os comportamentos táticos agem como filtros garantindo que o robô opere de maneira segura ao atender a situação imediata do robô sem se desviar do objetivo da tarefa indicado pelos comportamentos estratégicos. De certa forma isto é similar ao método de subsunção (BROOKS, 1986), ou seja, alguns comportamentos fundamentais sobrescrevem outros. No entanto, na arquitetura SFX, ao contrário da arquitetura de subsunção, são os comportamentos táticos de nível mais baixo que sobrescrevem os comportamentos estratégicos de nível mais alto (MURPHY, 2000).

3.3 DAMN

A arquitetura DAMN (*Distributed Architecture for Mobile Navigation*) foi desenvolvida por Rosenblatt (1995, 1997a, 1997b) (Figura 5). É uma arquitetura comportamental, distribuída, flexível, que se diferencia das demais pelo seu mecanismo de coordenação.

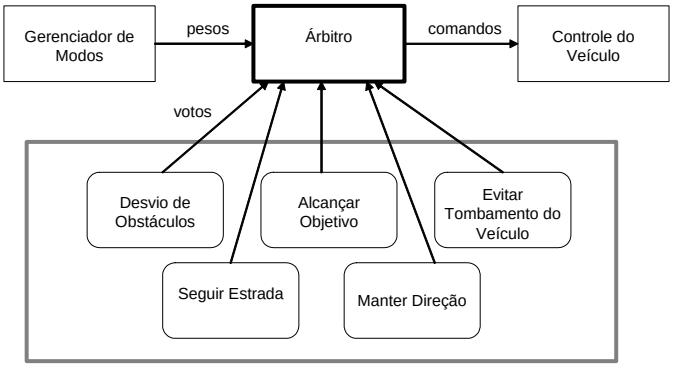

Figura 5: Arquitetura DAMN.

Nesta arquitetura, cada comportamento é um módulo responsável por completar uma deter-

minada tarefa, ou cuidar de um aspecto de controle influenciando as ações do robô. Estes comportamentos funcionam de forma assíncrona e em paralelo, sendo que cada um gera sua saída em intervalos de tempo próprios dependendo do processamento interno do comportamento. Comportamentos reflexivos geram uma saída a 10Hz, outros comportamentos que armazenam informações locais podem gerar uma saída a 1Hz, enquanto planejadores que utilizam informações globais armazenadas em uma representação interna do ambiente podem gerar uma saída a 0.1Hz. Para controle do robô, define-se nesta arquitetura um conjunto de ações e respostas que o robô pode ter. A saída de cada comportamento é feita na forma de um conjunto de votos a favor e contra um conjunto destas ações e respostas. Um árbitro faz a fusão desses votos e seleciona as ações vencedoras. Na arquitetura podem existir múltiplos árbitros, cada um para um tipo de ação diferente, como, por exemplo, controle de velocidade e controle de direção. A arbitragem de cada ação ocorre de forma independente. Cada árbitro toma o cuidado de fazer uma interpolação dos valores de controle para evitar efeitos de discretização na atuação do robô.

Vale ressaltar que apesar dos votos terem que passar pelo árbitro antes de resultar em um comando para o robô, a função do árbitro é bastante simples não resultando em um gargalo central no sistema.

Na arquitetura DAMN, comportamentos deliberativos e reativos compartilham concorrentemente o controle do robô, permitindo que o robô opere de forma estratégica para atingir um objetivo, ao mesmo tempo que preserva a capacidade tática de responder rapidamente a situações imprevistas e de emergência. Cada comportamentos dentro da arquitetura pode ter um peso que reflete na sua prioridade em controlar o veículo. Um módulo de raciocínio, que gerencia os modos de operação do robô, pode ser usado para variar estes pesos durante o curso da missão, de acordo com o conhecimento que o sistema tem sobre quais comportamentos são mais relevantes em determinadas situações. Mesmo que alguns comportamentos possam ter pesos maiores em determinadas circunstâncias, comportamentos com menor prioridade não são totalmente ignorados e continuam influenciando as ações do robô. Por esse motivo a arquitetura não pode ser considerada hierárquica na maneira em que deliberação é combinada com reação.

Os comportamentos podem manter uma representação interna complexa do mundo. Isso permite que o robô possa ser guiado por um plano de movimento criado a partir de uma representação do ambiente, e conseqüentemente, que o planejador participe do controle do robô. Uma das formas de planejamento utilizada pela arquitetura é o método de programação dinâmica que permite estabelecer qual a melhor forma de se chegar a uma posição de destino no ambiente a partir de qualquer posição inicial do robô (ROSENBLATT; PAYTON, 1989).

Conceitualmente, os comportamentos na arquitetura DAMN podem ser divididos em três

níveis de competência: o nível de segurança, de movimento, e de objetivos. O nível de segurança possui comportamentos que garantem a integridade física do robô, tais como, desvio de obstáculo e comportamentos para garantir que restrições dinâmicas do robô possam ser atendidas. O nível de movimento possui comportamentos que adicionam movimentos básicos úteis ao robô, tais como, seguir estrada e comportamentos para permitir tele-operação. O nível de objetivos possui comportamentos que buscam atingir algum objetivo de nível maior e geralmente utilizam informação contida na representação do ambiente. É importante observar que estes níveis são conceituais e não implicam necessariamente em uma hierarquia dentro da arquitetura. Estes níveis são convenientes para descrever a maneira incremental como o sistema pode ser desenvolvido, começando pelo nível de segurança e evoluindo em direção ao nível de objetivo.

3.4 Arquitetura de agente

A arquitetura de agente (*Animated Agent Architecture*) proposta por Firby, Prokopowicz e Swain (1998) é uma arquitetura desenvolvida com o intuito de ser usada em robôs que trabalham em um ambiente junto a seres humanos (Figura 6).

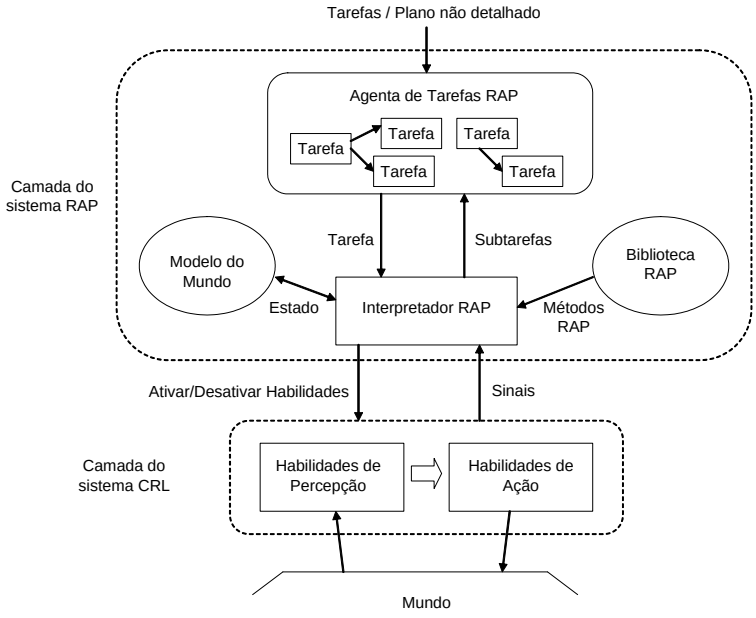

Figura 6: Arquitetura de Agente

A arquitetura possui duas camadas principais. A camada de nível mais baixo é composta por habilidades de percepção e atuação. Estas habilidades correspondem a processos contínuos, ou seja, controladores de malha fechada implementados de forma modular para controlar os sensores e atuadores do robô. Cada habilidade pode ser habilitada ou desabilitada de forma

independente e assíncrona permitindo que a camada inferior da arquitetura seja reconfigurada em tempo de execução. As habilidades podem se comunicar entre si e podem sinalizar à camada superior da arquitetura quando algum estado específico é atingido pelo robô, ou quando algo de interesse para a tarefa é encontrado no ambiente.

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 responsável por selecionar uma seqüência de ações e programar o nível inferior em tempo de execução. Este nível de seqüenciamento para execução de tarefa foi implementado utilizando o sistema RAP (*Reactive Action Packages*).

O sistema RAP usado na camada de nível superior da arquitetura parte de um plano não muito detalhado composto de tarefas em um alto nível de abstração. Em tempo de execução o sistema vai detalhando cada vez mais estas tarefas em níveis de abstração cada vez menores, até chegar em tarefas primitivas que podem ser executadas por habilidades no nível inferior da arquitetura. Para detalhar tarefas o sistema usa uma biblioteca hierárquica de métodos que definem como este refinamento de tarefas pode ser feito, e também utiliza o estado atual do robô e informações do ambiente representado por um modelo do mundo.

3.5 Arquiteturas de três camadas

Nas arquiteturas de três camadas, deliberação e reação estão geralmente divididas em duas camadas distintas. Entre estas duas camadas existe uma camada intermediária responsável por coordenar deliberação e reação. A camada reativa é a camada de nível inferior responsável pelo controle do robô com o uso de comportamentos, a camada deliberativa é a camada de nível superior responsável pelo planejamento, e a camada intermediária faz a conexão entre as duas camadas por meio de um mecanismo de seqüenciamento que determina quais comportamentos devem estar habilitados em cada situação de forma que o robô execute o plano determinado pelo planejamento. As três camadas geralmente operam de forma paralela.

Alguns exemplos de arquiteturas de três camadas são: a arquitetura SSS (CONNELL, 1992), a arquitetura Atlantis (GAT, 1991), a arquitetura genérica do LAAS-CNRS (NOREILS; CHATILA, 1995; ALAMI et al., 1998), e a arquitetura 3T (FIRBY, 1989; BONASSO, 1991; BONASSO; DEAN, 1996).

É interessante notar que embora possuam três camadas, cada uma destas arquiteturas se distingue uma das outras pela forma como cada uma das camadas funciona, e também pela forma como acontece a comunicação entre as camadas da arquitetura. Por exemplo, a arquite-

tura SSS se baseia no mecanismo de subsunção para realizar o seqüenciamento. A arquitetura 3T inicialmente utilizava um sistema de seqüenciamento chamado REX/GAPPS desenvolvido por Kaelbling (1987, 1988) e posteriormente passou a utilizar um sistema chamado RAP (*Reactive Action Packages*) desenvolvido por Firby (1989). Já a arquitetura Atlantis utilizava RAP para fazer o seqüenciamento, e então partiu para um novo sistema ou linguagem chamado ESL (GAT, 1997).

3.5.1 Atlantis e 3T

A arquitetura Atlantis é uma arquitetura híbrida de três camadas proposta por Gat (1991). Os componente responsáveis por cada uma das camada da arquitetura são chamados de *Controlador*, *Deliberador*, e *Seqüenciador* (Figura 7).

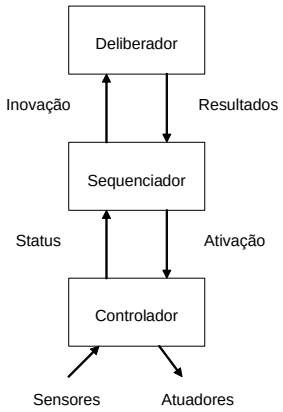

Figura 7: Arquitetura Atlantis.

O *Controlador* é formado por um conjunto de habilidades ou comportamentos primitivos. Estes comportamentos são controladores de malha fechada que associam sensores a atuadores de forma intima por meio de uma função de transferência. Pode-se citar como exemplo de comportamento: seguir uma parede, mover para uma posição de destino enquanto se evita colisões, e mover através de uma porta aberta. Cada uma destas habilidades é habilitada ou desabilitada por uma entrada externa ao componente *Controlador*. A arquitetura impões algumas restrições ao *Controlador*. Por exemplo, o tempo necessário para calcular uma iteração do algoritmo usado no comportamento deve ser pequeno o suficiente para que o robô seja operado de forma estável. Além disso, deve-se evitar o armazenamento interno de informações sobre o ambiente no *Controlador*. Quando for necessário, a informação armazenada internamente deve ser efêmera, ou seja, guardada somente por um intervalo conhecido de tempo e descartada em seguida. Dessa forma, se, por algum motivo, essa informação deixar de representar a realidade,

o erro ficará restrito ao intervalo conhecido e pequeno de tempo durante o qual essa informação esteve armazenada. Por fim, cada comportamento no *Controlador* deve reconhecer quando acontece uma falha nele mesmo, e avisar outros componentes da arquitetura para que alguma providência possa ser tomada a respeito.

O *Seqüenciador* seleciona o comportamento primitivo que deve ser usado no *Controlador* em um dado momento, e fornece parâmetros para a execução desse comportamento. O robô é conduzido a realizar tarefas complexas quando se alterna de um comportamento ao outro em momentos estratégicos. Portanto, o *Seqüenciador* deve ser capaz de responder de forma condicional à situação atual do robô, qualquer que seja ela. Para isso utiliza-se um método de seqüenciamento condicional. Neste método, de certa forma, a história de execução do robô é levada em consideração. Um plano de execução complexo motivado pela maneira como instruções humanas são seguidas é usado. Em geral, estes planos de execução são descritos utilizando linguagens desenvolvidas especialmente para este propósito, tais como RAP (FIRBY, 1989), PRS (GEORGEFF; LANSKY, 1987), Linguagem de Comportamento (BROOKS, 1989), REX/GAPPS (KAELBLING, 1987, 1988; BONASSO, 1992), e ESL (GAT, 1997). Na arquitetura Atlantis é utilizada a linguagem ESL.

O *Deliberador* é responsável pelo planejamento e pelo modelo do mundo. Ele se comunica com o restante do sistema respondendo a pedidos específicos provenientes do *Seqüenciador*. Em termos de restrição temporal, os algoritmos usados no *Deliberador* são os que levam maior tempo dentro do sistema para serem executados. Várias transições entre comportamentos primitivos podem ocorrer entre o instante de tempo em que o algoritmo é invocado e o momento em que ele gera o resultado.

A arquitetura 3T desenvolvida por Bonasso et al. (1997) é muito semelhante à arquitetura Atlantis pois ambas surgiram do mesmo trabalho. Entretanto, a arquitetura 3T utiliza uma representação diferente na camada de seqüenciamento. A 3T utiliza RAP e a Atlantis utiliza ESL. Além disso, na Atlantis a camada do seqüenciador é responsável por grande parte do controle, e é esta camada que controla a operação do planejador (camada deliberativa). A camada de seqüenciamento da Atlantis pede por trechos de planejamento conforme necessário. Na 3T, a iniciativa parte do deliberador que produz planos geralmente completos que são executados pelo seqüenciador (BONASSO et al., 1997). Entretanto, nada impede que na Atlantis o *Seqüenciador* peça ao *Deliberador* um plano completo de ação, e que na arquitetura baseada em RAP algoritmos deliberativos sejam invocados em tempo de execução para responder pedidos específicos do seqüenciador.

3.5.2 SSS

A arquitetura SSS (*servo*, *subsumption*, e *symbolic*) foi desenvolvida no Centro de Pesquisa T.J. Watson da IBM, e como o próprio nome sugere, ela pode ser dividida em três camadas (CONNELL, 1992) (Figura 8). Cada camada utiliza tecnologias diferentes para lidar com parte do problema.

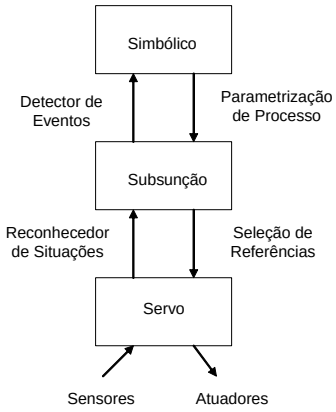

Figura 8: Arquitetura SSS.

A camada de nível mais baixo da arquitetura é formada por controladores servo convencionais de malha fechada e processadores de sinais. Este nível é composto principalmente por dois controladores de velocidade, um para translação e outro para rotação, que operam a 256Hz. Todos os cálculos para controle PID e geração de perfis de aceleração são feitos nesta camada.

No nível intermediário estão comportamentos, ou controladores, reativos multi-agentes. Esta camada se baseia na arquitetura de subsunção e juntamente com o nível de controle servo é responsável pelo controle tático do robô. Os comportamentos reativos agem no nível de controle servo ajustando a referência para as malhas de realimentação, por exemplo, definindo a velocidade das rodas. A interface sensorial entre o nível servo e a camada de comportamentos ocorre por meio de filtros que reconhecem situações relevantes em que o robô se encontra a partir da leitura de sensores. Os comportamentos neste nível intermediário operam na ordem de 7.5Hz.

Finalmente, no nível superior, estão sistemas simbólicos de inteligência artificial baseados em representações internas. O sistema simbólico utiliza uma tabela de contingência que possui alguns planos de ações pré-compilados para responder a determinados eventos. Nesta camada também podem ser usados mapas geométricos com pouca precisão que armazenam a direção e distância entre intersecções relevantes no ambiente. Um plano de navegação é determinado a partir desta representação, e os comportamentos no nível de subsunção (camada intermediá-

ria) são configurados para cada segmento da rota de navegação. Dessa forma, a interface de comando entre o nível simbólico e o nível de subsunção é feita pela habilitação e definição dos parâmetros para cada comportamento. A interface sensorial entre a camada de comportamento e a camada simbólica acontece por meio de um mecanismo que verifica eventos. Um evento acontece quando várias situações se tornam válidas. Por exemplo, quando o sistema reconhece que o robô não está no objetivo e também não tem feito progresso. Neste caso, o evento "caminho bloqueado"é transmitido à camada simbólica.

Os três níveis da arquitetura se formam a partir de uma discretização progressiva do espaço e do tempo. Na camada de baixo nível, os controladores servo e sensores operam praticamente no espaço e tempo contínuo. Na camada intermediária, baseada em comportamentos, existe a necessidade de identificar algumas situações com o auxílio dos sensores. Dessa forma, esta camada trabalha com uma discretização dos possíveis estados do mundo em um pequeno número de categorias dependendo da tarefa. Já o sistema simbólico na camada de alto nível da arquitetura vai mais além e discretiza o tempo em termos de eventos significativos. Nesta camada utiliza-se termos como "faça X depois de Y"ou "faça A até que B aconteça".

Ao comparar a arquitetura SSS com a arquitetura Atlantis, pode-se notar que na arquitetura Atlantis cada comportamento é responsável em reportar seu progresso e indicar a ocorrência de falha. Na SSS, são os eventos externos e não os estados internos que indicam se ocorreu falha no sistema e este deve ser reconfigurado. Além disso, na Atlantis, uma vez que um plano é gerado pelo sistema deliberativo, a camada de seqüenciamento fica responsável em executar este plano havendo apenas a possibilidade de ajustes simples no plano. Na arquitetura SSS, a camada simbólica fica de certa forma dentro da malha de controle durante a execução da tarefa. A tabela de contingência na camada simbólica da SSS permite que decisões possam ser tomadas de forma rápida enquanto o sistema simbólico se preocupa em replanejar a estratégia e monitorar a execução de cada etapa.

É importante notar também que apesar da arquitetura SSS ser considerada uma arquitetura de três camadas, as suas camadas não correspondem diretamente às camadas de outras arquiteturas. Nas outras arquiteturas, a camada de nível inferior corresponde à camada de nível intermediário da arquitetura SSS. A camada de controle servo da SSS aparece nas outras arquiteturas como parte da camada de comportamento ou como funções implícitas do robô, e portanto não consideradas na arquitetura. Já a camada deliberativa da SSS possui os elementos presentes nas duas camadas de nível mais alto das demais arquiteturas de três camadas.

3.5.3 Arquitetura genérica do LAAS-CNRS

Desenvolvida pelo LAAS-CNRS (NOREILS; CHATILA, 1995; ALAMI et al., 1998), esta arquitetura híbrida genérica possui três camadas principais: nível funcional, nível de execução e nível de decisão.

O nível funcional é composto por uma coleção de módulos responsáveis pelas funções de percepção e ação do robô. Estes módulos funcionais podem trocar informações entre si e são independentes do hardware do robô. Dentre os módulos da camada funcional pode-se citar planejadores baseados em mapa, módulos para desvio de obstáculo, módulos de mapeamento e localização, etc.

O nível intermediário da arquitetura controla e coordena a execução das funções distribuídas nos módulos de acordo com os requisitos da tarefa. Este controle e coordenação é feito ativando ou desativando os módulos do nível funcional.

O nível de decisão é o mais simbólico, e é responsável por produzir um plano de tarefa e supervisionar sua execução ao mesmo tempo que responde a eventos gerados pelo nível intermediário. O nível de decisão pode ter várias camadas de acordo com a aplicação do robô. A estrutura básica de cada uma destas camadas é um par planejador/supervisor que permite integrar deliberação e reação. Dependendo da aplicação, cada camada do nível de decisão pode utilizar uma abstração diferente de representação, algoritmos diferentes, e ter requisitos de tempo diferentes. Por exemplo, um nível de decisão pode ser decomposto em duas camadas. Uma camada responsável por planejar e supervisionar a missão dividindo-a em tarefas, e outra camada responsável por planejar uma tarefa, refinando-a ainda mais, e supervisionar a execução desta tarefa.

A arquitetura define uma série de ferramentas de representação, programação e métodos de processamento usados para atender os requisitos de cada um dos níveis da arquitetura. Utilizando estas ferramentas é possível fazer uma validação temporal e lógica das propriedades da parte reativa do sistema.

De forma geral, esta arquitetura se assemelha à arquitetura 3T conforme descrito por Bonasso et al. (1997). No entanto a 3T é baseada principalmente na hierarquia existente entre o planejador e o seqüenciador, ou seja, um plano é definido e então executado pela parte reativa. Na arquitetura desenvolvida pelo LAAS-CNRS a estrutura planejador/supervisor permite que a reação seja feita durante o planejamento e não após ele. Além dessa diferença estrutural, a arquitetura 3T utiliza RAP na camada de seqüenciamento, e esta arquitetura usa o sistema PRS.

3.6 Saphira

A arquitetura Saphira (KONOLIGE; MYERS, 1998; SAFFIOTTI; RUSPINI; KONOLIGE, 1993; SAFFIOTTI; KONOLIGE; RUSPINI, 1995; CONGDON et al., 1993), representada na Figura 9, tem como componente central uma representação do ambiente ao redor do robô chamada de LPS (*Local Perceptual Space*) que auxilia tanto na reatividade quanto na deliberação do sistema. A maior parte das ações são planejadas e executadas tendo como base este conhecimento. A representação LPS permite que sejam utilizados vários níveis de interpretação de informação sensorial, como também, informação fornecida *a priori* na forma de mapas. Em um dos níveis de representação do LPS, uma grade semelhante a uma grade de ocupação (MORAVEC; ELFES, 1985) pode ser construída a partir da fusão de leituras sensoriais. Representações mais analíticas que interpretam elementos desta grade podem ser adicionadas em outro nível de representação do LPS, como por exemplo, pode-se usar uma reta para representar uma superfície linear no ambiente. Além disso, em um outro nível, podem ser utilizados descrições semânticas tais como portas e corredores. Estes elementos que carregam uma descrição semântica do mundo são chamados de artefatos, e podem ser resultados de uma interpretação feita a partir de leituras de sensores, ou podem até mesmo resultar de uma interpretação de um mapa fornecido *a priori*.

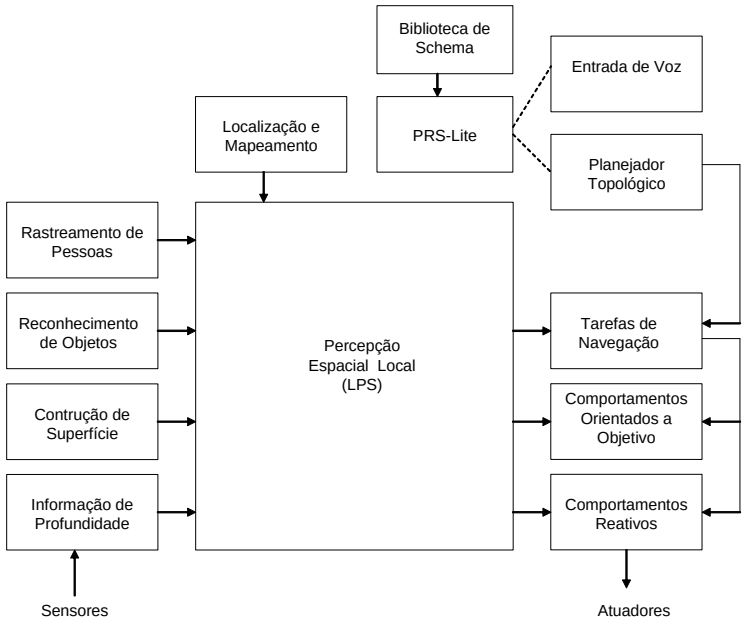

Figura 9: Arquitetura Saphira.

Saphira é uma arquitetura baseada em comportamentos, ou seja, o controle do robô é decomposto em pequenas unidades de comportamentos básicos, tais como, desvio de obstáculos e se mover ao longo de um corredor. O que diferencia a arquitetura Saphira das demais é que

os comportamentos são escritos e combinados utilizando técnicas baseadas em lógica nebulosa (SAFFIOTTI; RUSPINI; KONOLIGE, 1993; SAFFIOTTI, 1997).

Cada comportamento, por meio de uma função de desejo, indica quanto ele deseja que uma ou mais variáveis de controle tenham um determinado valor. Estas variáveis pode ser, por exemplo, velocidade de translação, velocidade de rotação, estado de uma garra para manipulação, etc. O valor final da variável de controle é calculado a partir de uma média que considera uma função de peso baseada na prioridade do comportamento dentro do contexto atual da aplicação. Assim o robô pode, por exemplo, levar em consideração o seu objetivo de navegação ao desviar de um obstáculo de forma reativa. Este tipo de coordenação que atribui prioridades aos comportamentos dependendo do contexto é de certa forma semelhante ao método de esquema motor utilizado por Arkin (ARKIN, 1987a, 1989a) na arquitetura AuRA onde cada comportamento é definido por um campo potencial artificial (KHATIB, 1985).

Na arquitetura Saphira, os comportamentos reativos de nível mais baixo e que são sensíveis ao tempo de resposta, como por exemplo, desvio de obstáculos, dependem de um processamento simples dos sensores. Por isso utilizam como entrada as leituras diretas dos sensores ou então estas leituras após passarem por algum tipo de transformação ou filtro. Informações disponíveis na representação LPS também podem ser usadas, tais como obstáculos no mapa de grades.

Comportamentos mais complexos que buscam cumprir algum tipo de objetivo servem de guia para os comportamentos reativos e geralmente utilizam informação do LPS contidas em um nível de representação mais elevado, tais como superfícies e artefatos. Um exemplo disso é o comportamento de seguir corredor que utiliza retas identificadas no mapa para se manter no corredor mesmo na presença de portas ao longo do mesmo. Posições de controle para navegação são adicionadas no LPS como artefatos e usadas para guiar comportamentos mais elaborados de navegação.

Cada comportamento é responsável pelo controle do robô em uma determinada situação. Mas para que o robô possa cumprir uma tarefa, os comportamentos são seqüenciados e o progresso monitorado por um controlador baseado em PRS (*Procedural Reasoning System*) chamado de PRS-Lite.

O PRS-Lite fornece uma estrutura por meio da qual é possível especificar e gerenciar as tarefas de um robô. Assim como em outros controladores de tarefa, o PRS-Lite é capaz de integrar de forma suave atividades que buscam cumprir objetivos, e atividades acionadas por eventos. Ele é capaz de responder em tempo hábil a mudanças inesperadas no ambiente, e decompor tarefas de forma hierárquica. Além destas características, o PRS-Lite possui algumas que são

próprias, tais como, gerenciar processos contínuos, possuir um conjunto rico de mecanismos de controle declarativo, e possuir uma semântica de objetivos que substitui conceitos de sucesso e falha por níveis de satisfação em relação ao objetivo.

Geralmente, linguagens de seqüenciamento modelam uma tarefa utilizando eventos para ativar a transição entre um comportamento e outro. Muitas vezes mesmo quando cada comportamento é um processo contínuo, no momento da transição entre comportamentos ocorre uma descontinuidade na execução da tarefa. Por meio do PRS-Lite é possível especificar uma transição suave entre um comportamento e outro.

A arquitetura Saphira assume que o robô é uma plataforma móvel capaz de oferecer serviços básicos utilizando um protocolo específico. Então os componentes do Saphira se comunicam com o robô por meio de uma interface cliente/servidor. Dessa forma é possível utilizar esta arquitetura em diferentes robôs desde que estes utilizem o protocolo usado pelo Saphira.

3.7 Planejador-Reator

A arquitetura Planejador-Reator foi proposta por Lyons e Hendriks (1992, 1995) para integrar planejamento e reatividade (Figura 10). A arquitetura possui dois componentes principais que funcionam de forma paralela: o sistema reativo, ou *Reator*; e o sistema de planejamento, ou *Planejador*.

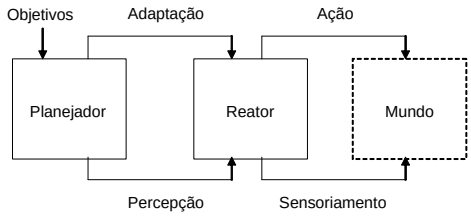

Figura 10: Arquitetura Planejador-Reator.

O *Reator* é formado por uma rede de comportamentos reativos, ou seja, composições entre percepções sensoriais e ações motoras. O *Reator* é um sistema de tempo real que continuamente inspeciona o ambiente ao redor do robô e produz uma ação sempre que qualquer um de seus comportamentos reativos estiver ativado. Diferentemente de um executor de planos de ação, o *Reator* age independentemente do *Planejador*, podendo produzir um comportamento útil mesmo sem planejamento.

O *Planejador* não é encarado como um sistema de nível mais elevado que gera um plano de ação que deve ser executado pelos níveis inferiores. Nesta arquitetura, o planejador é visto como

um sistema que está no mesmo nível hierárquico do *Reator*. Ele ajusta o *Reator* continuamente, de forma incremental, para que este produza um comportamento geral apropriado, capaz de alcançar os objetivos da aplicação. Este ajuste é feito por meio de pequenas adaptações na configuração do sistema reativo. Estas adaptações são feitas para lidar com eventuais mudanças no objetivo ou no ambiente. Para fazer estes ajustes, o *Planejador* utiliza como entrada: um modelo do ambiente no qual o robô está operando, uma descrição da estrutura do reator, e também informações do usuário sobre os objetivos que devem ser alcançados.

O *Reator* é descrito por um modelo formal chamado RS (*Robot Schema*) que é usado para representar planos flexíveis de ação para o robô. Com o auxílio deste modelo formal, o comportamento do *Reator* e as regras por meio das quais o *Reator* é modificado podem ser analisadas matematicamente. Com isso determina-se as restrições que o *Planejador* deve obedecer para que adaptações seguras no *Reator* possam ser feitas garantindo a convergência em direção ao *Reator* desejado.

O conhecimento utilizado pelo *Planejador* é diferente do tipo de conhecimento utilizado pelo *Reator*. A informação sensorial usada pelo *Reator* serve para determinar se um comportamento reativo deve ser ativado ou não. Já o *Planejador* utiliza informação sensorial para prever o futuro progresso do ambiente e do *status* do *Reator*.

Ao comparar esta arquitetura com a SSS, percebe-se que nas duas arquiteturas os componentes deliberativos e reativos se interagem de forma assíncrona permitindo que planejamento e reação ocorram simultaneamente, ambos agindo de forma mais intensa no controle do robô. A diferença, no entanto, está na forma com que o componente deliberativo atua sobre o componente reativo. Na arquitetura SSS, o componente deliberativo atua apenas habilitando e desabilitando comportamentos, enquanto que na arquitetura Planejador-Reator, comportamentos completamente novos podem ser gerados pois o planejador modifica de forma incremental a estrutura do reator.

Assim como na arquitetura AuRA, a arquitetura Planejador-Reator também se baseia em esquemas, e ocorre interação assíncrona entre deliberação e reação. No entanto, na AuRA um plano é gerado e então executado, podendo haver adaptação nos parâmetros do sistema reativo. Na arquitetura Planejador-Reator, o planejador faz adaptações incrementais no sistema reativo, principalmente na sua estrutura ao invés de adaptar apenas os parâmetros. A arquitetura Planejador-Reator se beneficia de aplicações onde a repetitividade da tarefa pode ser explorada para que o sistema melhore de forma iterativa e incremental seu desempenho.

3.8 DD&P

DD&P é uma arquitetura híbrida de duas camadas (HERTZBERG; SCHöNHERR, 2001). A camada reativa é definida utilizando uma estrutura chamada DD (*Dual Dynamics*). Os comportamentos na camada reativa são especificados como sendo sistemas dinâmicos contínuos expressos na forma de equações diferenciais ordinárias. Além disso, cada comportamento especifica, de forma separada, a sua ação física resultante, e um valor escalar subjetivo que representa o grau de confiança que o comportamento tem de que ele é apropriado. Os comportamentos DD são organizados em níveis hierárquicos. Cada nível influencia o outro, e o nível 0 age diretamente nos atuadores do robô.

A camada deliberativa da arquitetura DD&P é responsável pelo planejamento que age influenciando qualquer nível da camada DD. O planejamento de ação é feito de forma semelhante aos métodos clássicos de Inteligência Artificial. Entretanto, o método utilizado é capaz de gerar rapidamente um plano bastante curto e abstrato. Este plano de ação é colocado em uma área de acesso compartilhado entre planejador e um componente chamado monitor, responsável pela execução do plano de ação. Ao iniciar a execução, o monitor copia o plano de ação disponível nesta área compartilhada para si, permitindo que o plano na área de acesso compartilhado possa ser sobrescrito. Assim que o planejador acaba de gerar um plano, ele atualiza o plano na área de acesso compartilhada, e imediatamente começa a preparar um novo plano ou uma atualização do plano atual. A idéia é que o planejamento seja feito continuamente, de modo que quando o monitor acabe de executar um plano curto de ação, ele tenha acesso somente ao plano de ação que foi gerado recentemente, que por sua vez, se baseia nas informações mais atuais disponíveis ao sistema.

Na arquitetura DD&P, executar o plano de ação significa interferir no nível de ativação dos comportamentos de modo que os comportamentos que contribuem para a execução do plano se sobressaiam, e que os comportamentos que agem no sentido contrário ao plano de execução sejam abafados.

Além do planejador e monitor, na camada deliberativa também existe uma base de conhecimento do sistema que contém informações sobre fatos atuais e objetivos do robô. Esta base de conhecimento recebe entradas provenientes do usuário e provenientes de um componente responsável por manter atualizado o modelo de mundo.

Todos os componentes da arquitetura DD&P trabalham de forma paralela. Esta característica da arquitetura é explorada mediante uma implementação fisicamente paralela da arquitetura.

3.9 Discussão comparativa

Com relação a maneira como planejamento interage com a execução do sistema, quatro estratégias principais podem ser identificadas nas arquiteturas híbridas: seleção, conselho, adaptação, adiamento (ARKIN, 1998). A Tabela 3 mostra as principais características das arquiteturas híbridas apresentadas, bem como a estratégia utilizada por cada arquitetura na interface entre planejamento e execução.

Tabela 3: Principais características das arquiteturas híbridas

Arquitetura	Principais Características	Estratégia de Interface	Referências
AuRA	planejamento hierárquico; planeja depois executa; seleção e configuração de comportamentos; esquema motor;	seleção	(ARKIN, 1986) (ARKIN, 1987b)
SFX	componentes deliberativos agem em paralelo; gerenciador de recursos; fusão sensorial; comportamentos indicam falhas neles mesmos; comportamentos em camadas estratégicas e táticas; seleção e configuração de comportamentos; coordenação por meio de filtros;	seleção	(MURPHY; ARKIN, 1992) (MURPHY; MALI, 1997) (MURPHY, 2000)
DAMN	comportamentos deliberativos e reativos em paralelo; não existe hierarquia entre comportamentos; gerenciador de pesos; peso de comportamentos depende da situação; comportamentos votam em ações;	seleção	(ROSENBLATT, 1997a) (ROSENBLATT, 1997b) (ROSENBLATT, 1995) (ROSENBLATT; PAYTON, 1989)

Tabela 3: Principais características das arquiteturas híbridas (continuação)

Arquitetura	Principais Características	Estratégia de Interface	Referências
Agente	duas camadas (comportamento e seqüenciamento); camada de comportamento capaz de identificar eventos; seqüenciamento em tempo de execução utiliza RAP; modelo do ambiente usado para identificar eventos, sucesso e falha na execução de tarefas;	seleção, adiamento	(FIRBY; PROKOPOWICZ; SWAIN, 1998)
Atlantis	três camadas em paralelo; comportamentos indicam falhas neles mesmos; seqüenciador pede ao deliberador informações específicas sobre o plano; seleção e configuração de comportamentos; planeja depois executa; seqüenciador usa ESL	conselho, seleção	(GAT, 1991)
SSS	três camadas; camada inferior possui controladores servo; métodos simbólicos de deliberação; seleção e configuração de comportamentos; baseada em subsunção; camada superior simbólica é estratégica; camada intermediária é tática; planejamento e reação formam uma "malha de controle", mas possuem escopo espacial e temporal distintos; falhas e progresso indicados por "eventos"percebidos no ambiente;	seleção, adaptação	(CONNELL, 1992)

Tabela 3: Principais características das arquiteturas híbridas (continuação)

Arquitetura	Principais Características	Estratégia de Interface	Referências
Arquitetura Genérica	três camadas; planejamento hierárquico; planejamento acoplado com supervisor para cada nível hierárquico; planejamento antes e durante execução (auxiliado pelo supervisor); seqüenciamento de módulos funcionais (comportamentos); seqüenciador utiliza PRS;	seleção	(NOREILS; CHATILA, 1995) (ALAMI et al., 1998)
Saphira	representação do ambiente em diferentes níveis de interpretação; comportamentos deliberativos (estratégicos) guiam comportamentos reativos (táticos); seqüenciador utiliza PRS-Lite; coordenação por meio de lógica nebulosa;	seleção, adiamento	(KONOLIGE; MYERS, 1998) (SAFFIOTTI; RUSPINI; KONOLIGE, 1993) (SAFFIOTTI; KONOLIGE; RUSPINI, 1995) (CONGDON et al., 1993)
Planejador-Reator	planejamento e reação acoplados formando uma "malha de controle"; configuração de parâmetros e adaptação da estrutura de comportamentos; novos comportamentos são gerados;	adaptação	(LYONS; HENDRIKS, 1992) (LYONS; HENDRIKS, 1995)
DD&P	duas camadas; baseada em comportamentos paralelos; planejamento contínuo e incremental; configuração de parâmetros dos comportamentos; planeja e depois executa;	seleção, adaptação	(HERTZBERG; SCHöNHERR, 2001)

Ainda relacionado com a forma como planejamento e execução interagem dentro da arquitetura, em algumas arquiteturas como a AuRA, o planejamento ocorre em uma etapa anterior a execução e só é refeito se necessário. Na arquitetura Atlantis, o seqüenciador toma a iniciativa de pedir ao planejador por planos de ação conforme necessário. Em outras arquiteturas, planejamento ocorre em paralelo a execução fazendo parte de uma espécie de malha de realimentação, como no caso da arquitetura SSS, DD&P, e Planejador-Reator. Nestas arquiteturas, o planeja-

mento é feito de forma incremental, constantemente alterando o sistema reativo de execução ou fornecendo um trecho de plano de ação para que possa ser executado. O intervalo de cada interação entre os componentes de planejamento e os componentes de execução pode variar de arquitetura para arquitetura.

Em relação a forma como planejamento está organizado na arquitetura, em algumas das arquiteturas híbridas, como a AuRA e a Arquitetura Genérica, o planejamento é hierárquico. Nestas arquiteturas o planejamento vai sendo detalhado de um nível alto e geral (planejamento de missões) até um nível mais baixo e específico (planejamento de movimento). Em contraste, em algumas arquiteturas, como no caso da DAMN, o planejamento está mais interligado com os comportamentos da arquitetura, sugerindo uma organização mais heterárquica do controle.

Na maioria das arquiteturas pode-se identificar camadas que agrupam componentes que desempenham papeis semelhantes. Algumas possuem duas camadas, como a DD&P, Planejador-Reator, e Arquitetura de Agente. Outras possuem três camadas, como a Atlantis, SSS, e Arquitetura Genérica. Para algumas arquiteturas esta divisão pode não ficar tão clara como, por exemplo, na arquitetura Saphira que é organizada ao redor da representação que o robô tem do mundo.

Em relação a maneira como a parte deliberativa interage com os comportamentos, os comportamentos podem ser seqüenciador de acordo com um plano, como na arquitetura Saphira, Atlantis, e Arquitetura Genérica. Além disso, os parâmetros que influenciam na coordenação dos comportamentos podem ser configurados, como ocorre na AuRA e na DAMN. Também os comportamentos podem ser organizados em camadas, como na arquitetura DD&P e SFX, onde comportamentos mais deliberativos, considerados como estratégicos, podem ser usados como guias de comportamentos mais reativos, considerados como táticos. Os mecanismos utilizados para coordenar os comportamentos podem ser, por exemplo, subsunção (SSS), campos potenciais (AuRA), votação (DAMN), lógica nebulosa (Saphira), filtros (SFX).

3.10 Interação homem-robô

A interação homem-robô pode ocorrer de diversas formas dependendo da aplicação. Alguns critérios para classificar a interação homem-robô foram definidos por Yanco e Durry (2002, 2004). Um destes critérios está relacionado ao papel que o homem desempenha na interação com o robô, que pode ser o de supervisor, operador, parceiro de tarefa, mecânico ou programador, e espectador.

No papel de supervisor, o homem define objetivos de navegação, tarefas ou missões que

o robô deve realizar de forma autônoma, e supervisiona a execução destas tarefas. Em alguns sistemas de controle, como nos trabalhos de Scholtz, Antonishek e Young (2004) e Murphy e Rogers (1996), o usuário interage de forma remota com o robô. Estes sistemas de controle, em particular, foram desenvolvidos para auxiliar o usuário na tarefa de supervisão do robô e na interpretação das informações sensoriais adquiridas pelo robô enquanto ele se move de forma autônoma no ambiente. O sistema de controle, após interpretar as informações sensoriais, avisa o usuário quando o robô encontra algo no ambiente, ou quando o robô está em uma determinada situação de particular interesse. Assim, o sistema ajuda a reduzir o nível de atenção e fatiga do usuário na tarefa de supervisão e monitoramento das atividades do robô.

Em muitos outros sistemas de controle o homem desempenha o papel de supervisor ao especificar um objetivo de navegação. Thrun et al. (1999), por exemplo, desenvolveram um robô para servir de guia em um museu. O visitante escolhia as exibições que gostaria de ver e o robô o conduzia por estas exibições. Durante o horário de abertura do museu, os visitantes podiam interagir diretamente com o robô. Neste caso, além do papel de supervisor, o usuário também interagia no papel de espectador já que dividia o mesmo espaço físico que o robô autônomo. Interagindo dessa forma, o usuário acabava aprendendo como se comportar e como dividir o mesmo espaço com o robô. Durante alguns horários pré-definidos, quando o museu estava fechado, o usuário voltava a interagir com o robô no papel de supervisor, pois podia escolher remotamente, pela Internet, posições para as quais gostaria que o robô se deslocasse de forma autônoma.

Além destes exemplos, muitas das arquiteturas apresentadas, como é o caso da arquitetura híbrida AuRA (ALI; ARKIN, 2000) e do sistema deliberativo hierárquico chamado NASREM (LUMIA; ALBUS, 1988), permitem ao usuário interagir no papel de supervisor ao especificar um objetivo de navegação, uma missão, ou uma tarefa ao robô. Na arquitetura AuRA, também no papel de supervisor, o homem pode alterar o comportamento geral do sistema ajustando os pesos atribuídos a cada comportamento.

No papel de operador o homem interage com robô de forma mais intensa do que no papel de supervisor. Afinal, quando o homem está operando o robô ele está interferindo de forma mais específica e contínua nas ações do robô. Este tipo de interação é observada na tele-operação e no controle semi-autônomo do robô. Na arquitetura hierárquica deliberativa NASREM (LUMIA; ALBUS, 1988), por exemplo, o homem pode controlar diretamente os movimentos de baixo nível do robô (tele-operação). Na verdade, a arquitetura NASREM prevê a interação do homem com o sistema em níveis distintos do controle do robô. Já na arquitetura AuRA, o usuário pode compartilhar o controle com o robô ao fornecer em tempo de execução direções de navegação. Estes

comandos de direção são tidos pelo sistema como se fossem entradas de controle provenientes de um comportamento. Esta entrada é combinada com os demais comportamentos do sistema (desvio de obstáculo, por exemplo) para gerar uma resposta que satisfaça ambos. No caso da arquitetura AuRA, as entradas de controle são consideradas como vetores de força artificial e a combinação destas entradas é feita por meio da soma destes vetores. Outras arquiteturas também permitem ao usuário compartilhar o controle com o sistema no papel de operador de forma semelhante à AuRA, como é o caso da DAMN e da SFX.

Diferentemente do papel de supervisor e operador, no papel de parceiro de tarefa o homem não especifica diretamente as ações que o robô deve desempenhar. Ao invés disso, ele ajuda o robô a desempenhar sua tarefa como um colega de trabalho. Esta forma de interação pode ser vista no trabalho de Nicolescu e Mataric (2003). Quando o robô não consegue realizar uma tarefa, ele procura o homem e pede para que ele o ajude no cumprimento da tarefa. Observando a ação realizada pelo homem, o robô aprende a lidar com a situação com a qual teve dificuldade. Em alguns casos, o robô apenas se beneficia da ação do homem no ambiente, como, por exemplo, quando o homem remove um obstáculo que impede que o robô alcance seu objetivo. Um outro trabalho onde o homem é visto como parceiro de tarefa do robô é apresentado por Marble et al. (2004).

Por fim, existem ocasiões em que o software ou o hardware do robô precisa ser alterado. Então o homem interage com o robô no papel de mecânico ou programador. Os robôs enviados a Marte, por exemplo, podiam receber atualizações de software enviadas a partir da base na Terra. Estas atualizações eram feitas para que o robô pudesse lidar de forma melhor com as condições do planeta, melhorando seu desempenho. Os desenvolvedores determinavam as atualizações necessárias estudando as informações enviadas pelo robô. Algumas vezes, estas atualizações poderiam corrigir erros, ou algum problema no sistema, que fosse detectado durante a missão do robô em Marte (MARS…, 2004).

Neste trabalho em particular, a arquitetura híbrida desenvolvida para o robô móvel deve permitir a interação do homem no papel de supervisor e operador. No papel de operador do robô, o usuário humano deve poder compartilhar o controle do robô com o sistema de navegação autônoma. Esta arquitetura e o mecanismo desenvolvido para coordenar as entradas de controle do operador humano com as entradas de controle do sistema de navegação autônoma são mostrados no capítulo 5.

4 Planejamento de movimento

Planejamento baseado em um modelo do mundo, ou do ambiente, é uma das atividades deliberativas que compõe uma arquitetura híbrida para robôs móveis. Em geral, o planejamento pode ser feito em diferentes níveis de abstração. Estes níveis podem variar desde um nível mais alto, como no caso do planejamento de missões e de tarefas, até um nível mais baixo e específico, como é o caso do planejamento de movimento, que procura resolver principalmente o problema de navegação autônoma (LAVALLE, 2006).

O planejamento em um nível alto de abstração procura encontrar formas realizar missões ou tarefas complexas e abrangentes dividindo-as em atividades, ou subtarefas, mais simples e específicas. O modelo do mundo utilizado neste tipo de planejamento pode envolver uma descrição mais simbólica do ambiente, como a utilizada pelo método de planejamento STRIPS (FIKES; NILSSON, 1971), ou mesmo a representação de entidades do ambiente como é feito na arquitetura Saphira. O planejador também pode levar em conta o conhecimento a respeito do próprio robô, como, por exemplo, os comportamentos que o robô é capaz de desempenhar e as situações onde cada um destes comportamento pode ser aplicado, informações sobre os recursos do robô (sensores, atuadores), etc. Algumas arquiteturas híbridas utilizam linguagens específicas para descrever comportamentos, habilidades, ou até mesmo, subtarefas, auxiliando no planejamento das ações do robô. Algumas destas linguagens são: RAP (FIRBY, 1989), PRS (GEORGEFF; LANSKY, 1987), ESL (GAT, 1997), dentre outras. O resultado do planejamento pode ser um plano de ação descrito por uma seqüência de subtarefas que podem ser executadas por comportamentos básicos disponíveis no sistema.

A navegação autônoma de um robô móvel é uma das tarefas ou atividades que pode fazer parte de uma missão mais complexa do robô. Utilizando-se métodos de planejamento de movimento é possível determinar quais movimentos o robô deve realizar de forma que alcance uma posição ou configuração desejada no ambiente sem que ocorram colisões com obstáculos (LATOMBE, 1991), cumprindo assim a tarefa de navegação autônoma. Os métodos de planejamento de movimento podem envolver outras questões além de determinar trajetórias livres de colisões dentro de um ambiente conhecido pelo robô. Uma destas questões é gerar trajetórias possíveis

de serem executadas pelo robô considerando aspectos da dinâmica do robô e suas restrições de movimento. Além disso, os métodos de planejamento de movimento podem considerar formas de se lidar com obstáculos dinâmicos percebidos durante a execução do movimento, ou com incertezas no movimento.

A arquitetura que está sendo proposta neste trabalho procura resolver principalmente tarefas de navegação enquanto permite a interação com o usuário humano. Dessa forma, o enfoque deste trabalho será em métodos de planejamento de movimento para navegação. Estes métodos são apresentados de forma detalhada por Latombe (1991), Choset et al. (2005), e LaValle (2006), dentre outros. Algumas das abordagens utilizadas para planejamento de movimento são: *roadmaps*, decomposição em células, campos potenciais, e planejamento baseado em amostragem. Estas abordagens geralmente utilizam o espaço de configurações para representar o ambiente ou espaço de trabalho do robô.

4.1 Espaço de configurações

Antes de definir configuração e espaço de configurações de um robô, é interessante falar sobre o espaço de trabalho de um robô. O espaço de trabalho, $\mathcal{W}$, é o espaço físico por onde o robô se move, que pode ser definido no espaço Euclidiano bidimensional ou tridimensional representado por $\mathbb{R}^2$ e $\mathbb{R}^3$ respectivamente. No caso de robôs móveis que se deslocam apenas no plano, o espaço de trabalho é bidimensional, ou seja, definido em $\mathbb{R}^2$. Já para robôs manipuladores e robôs móveis aéreos ou submarinos, o espaço de trabalho é tridimensional, ou seja, definido em $\mathbb{R}^3$. Algumas vezes o espaço de trabalho pode ter um significado especial. No caso de robôs manipuladores, geralmente o espaço de trabalho é considerado apenas como sendo o conjunto de pontos do espaço Euclidiano tridimensional que podem ser alcançados pelo efetuador do robô.

Um robô, $\mathcal{R}$, pode ser representado como sendo um subconjunto compacto, ou seja, um subconjunto fechado e limitado, do espaço de trabalho $\mathcal{W}$. Um conjunto é considerado fechado se ele inclui todos os seus pontos limites. Por exemplo, o intervalo $[0, 1) \subset \mathbb{R}$ é semi-aberto e limitado, e portanto, não é compacto. No entanto, o intervalo $[0, 1]$ é fechado e limitado, e conseqüentemente compacto. Já o espaço $\mathbb{R}^n$ não é limitado e, portanto, não é compacto.

Os obstáculos no ambiente, $\mathcal{B}_i, ..., \mathcal{B}_q$, são definidos como subconjuntos fechados de $\mathcal{W}$. Define-se também $\mathcal{F}_\mathcal{R}$ e $\mathcal{F}_\mathcal{W}$ como sendo os sistemas de coordenadas cartesianas em $\mathcal{R}$ e $\mathcal{W}$ respectivamente.

A configuração, q, de um robô, $\mathcal{R}$, pode ser definida como sendo a especificação completa

da posição de todos os pontos do robô relativa ao sistema de coordenadas fixo do ambiente, $\mathcal{F}_{\mathcal{W}}$. O número mínimo de parâmetros necessários para especificar de forma completa a configuração de um robô é o número de graus de liberdade, m, deste robô. Para uma dada configuração q, o subconjunto do espaço de trabalho, $\mathcal{W}$, ocupado pelo robô, $\mathcal{R}$, é representado por $\mathcal{R}(q)$.

Finalmente, o espaço de configurações, $\mathcal{C}$, de um robô é definido como sendo o conjunto de todas as configurações possíveis para este robô. Neste espaço de configurações, o robô é considerado como sendo um ponto, q. Este espaço é uma ferramenta muito utilizada na formulação de problemas de planejamento de movimento, sendo que toda a geometria da tarefa pode ser mapeada neste espaço. Esta idéia foi primeiramente introduzida por Udupa (1977), e popularizada na área de planejamento de movimento por Lozano-Perez (1983) que explorou a idéia de forma sistemática.

Dentro do contexto de espaço de configurações, pode-se definir dois conjuntos que auxiliam a formulação de problemas de planejamento de movimento: o espaço de configurações ocupadas por obstáculos, $\mathcal{C}_{obs}$, e espaço de configurações livres, $\mathcal{C}_{free}$.

O espaço de configurações ocupado por obstáculos, $\mathcal{C}_{obs}$, é definido como sendo o conjunto de configurações para as quais existe intersecção entre o robô e os obstáculos:

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} / \mathcal{R}(q) \cap \mathcal{B} \neq \emptyset\} \tag{4.1}$$

O espaço de configurações livres, $\mathcal{C}_{free}$, é definido como sendo o espaço em que não ocorre intersecção do robô com os obstáculos:

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} / \mathcal{R}(q) \cap \mathcal{B} = \emptyset\} \tag{4.2}$$

A partir de um mapa do ambiente de trabalho que represente os obstáculos, e a partir da geometria do robô, é possível determinar $\mathcal{C}_{obs}$ e conseqüentemente $\mathcal{C}_{free}$.

Como exemplo, considere um robô circular que efetua somente movimentos de translação em um plano. A configuração do robô é dada pelas coordenadas do ponto de referência do robô localizado no seu centro, $q = \{x, y\}$. O espaço de configurações para este robô é bidimensional e pode ser representado em $\mathbb{R}^2$. Além disso, para deste robô o ambiente de trabalho também é bidimensional representado pelo espaço Euclidiano $\mathbb{R}^2$. Entretanto, é importante notar que mesmo tendo a mesma dimensão, o espaço de configurações, $\mathcal{C}$, é diferente do espaço de trabalho, $\mathcal{W}$. Para determinar o espaço de configurações ocupado pelos obstáculos, desloca-se o robô ao redor de cada obstáculo no espaço de trabalho conforme mostra a Figura 11. A trajetória descrita pelo ponto de referência do robô, $\{x, y\}$, define a fronteira do obstáculo em $\mathcal{C}$. Dessa

forma determina-se a restrição que o obstáculo impõe sobre a configuração do robô. Para este caso, os obstáculos no espaço de configurações equivalem aos obstáculos no ambiente de trabalho expandidos pela dimensão radial do robô circular. Na Figura 11, $\mathcal{C}_{obs}$ é representado pelas áreas em cinza e preto. O espaço de configurações livres, $\mathcal{C}_{free}$, corresponde ao complemento de $\mathcal{C}_{obs}$ em $\mathcal{C}$.

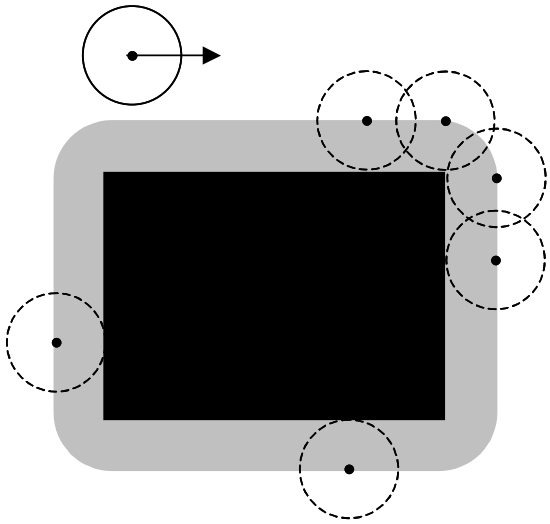

Figura 11: Espaço de configurações de um obstáculo quadrado para um robô circular. A área preta representa o obstáculo no ambiente de trabalho, e a área cinza o obstáculo no espaço de configurações.

No caso de um robô móvel com dimensão retangular alongada, capaz de se transladar e rotacionar livremente em um plano, ou seja, no espaço de trabalho $\mathcal{W} = \mathbb{R}^2$, a variável de configuração é dada por $q = \{x, y, \theta\}$, e o espaço de configurações é tridimensional, $\mathcal{C} = \mathbb{R}^3$. Para este robô, a construção do espaço de configurações ocupado pelos obstáculos é feita da seguinte forma. Para cada valor de orientação do robô, deslizar-se o robô em torno dos obstáculos no ambiente de trabalho, conforme ilustrado na Figura 12. As posições que o ponto de referência do robô descreve enquanto o robô desliza ao redor dos obstáculos define a fronteira do obstáculo no espaço de configurações. Assim, de maneira semelhante ao caso exemplificado anteriormente, pode-se determinar $\mathcal{C}_{obs}$ e $\mathcal{C}_{free}$. Nota-se entretanto que, neste caso, dependendo da orientação do robô, o obstáculo assume uma geometria diferente no espaço de configurações (Figura 13). Isso reflete o fato de que regiões do espaço de trabalho só podem ser alcançadas quando o robô está em determinada orientação. Por exemplo, no espaço de configurações, corredores estreitos, cuja largura é pouco maior que a largura do robô mas menor que o seu comprimento, aparecerão obstruídos dependendo da orientação do robô.

Uma vez definido o espaço de configurações livres, planejar um caminho livre de colisões é encontrar um caminho em $\mathcal{C}_{free}$ que conecta a configuração inicial q_{init} à configuração de-

sejada de destino q_{goal}, se estas configurações estiverem no mesmo componente conectado de $\mathcal{C}_{free}$. Caso contrário, o planejador deve reportar que não existe caminho sem colisão entre as configurações inicial e final.

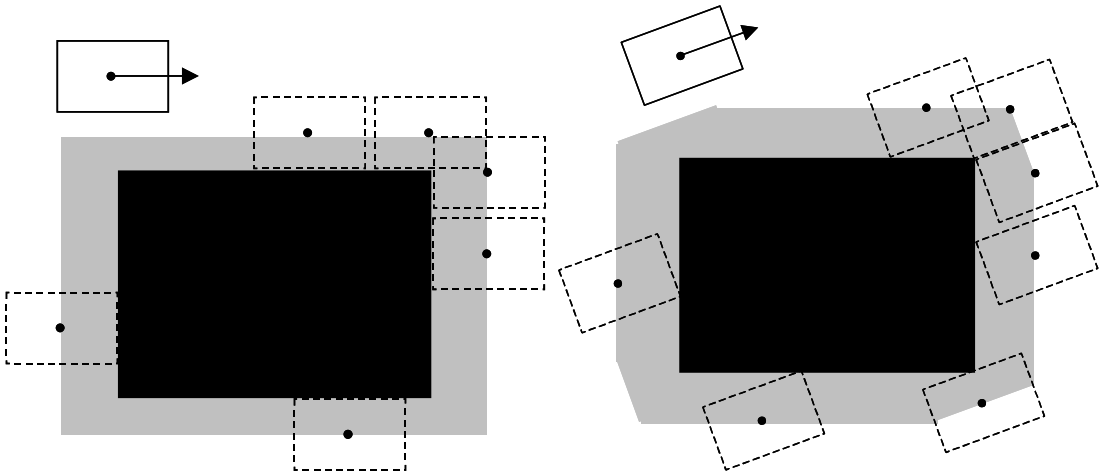

Figura 12: Espaço de configurações de um obstáculo quadrado para um robô retangular em duas direções diferentes. A área preta representa o obstáculo no ambiente de trabalho, e a área cinza o obstáculo no espaço de configurações.

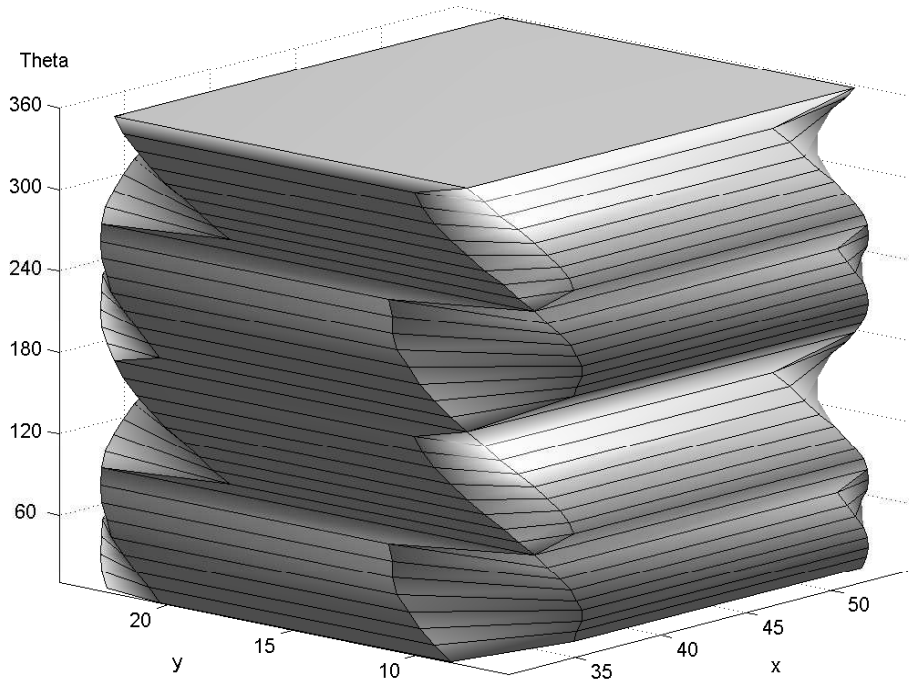

Figura 13: Espaço de configurações de um obstáculo quadrado para um robô retangular em função da orientação do robô.

4.2 Planejamento utilizando *roadmaps*

Uma *roadmap* é uma rede de curvas unidimensionais que captura a conectividade do espaço de configurações livres. Uma vez que esta rede é construída, ela pode ser usada para fornecer soluções de caminhos por onde o robô pode se mover. Esta rede pode ser comparada a uma malha rodoviária principal. Para viajar de uma cidade a outra, um indivíduo primeiramente encontra um caminho que o leve de seu ponto de origem até esta malha rodoviária principal. Então dirige nesta malha até um ponto próximo à cidade de destino. Neste ponto, deixa a rodovia principal para chegar na cidade de destino desejada. Da mesma forma, o planejamento de trajetória utilizando uma *roadmap* envolve duas etapas. A primeira é encontrar um caminho que conecte a configuração inicial, q_{init}, e a configuração final do robô, q_{goal}, a *roadmap*. Seja q'_{init} e q'_{goal}, respectivamente, o ponto de conexão de q_{init} e q_{goal} a *roadmap*. Então, a segunda etapa é procurar um caminho na *roadmap* que ligue os dois pontos de conexão q'_{init} e q'_{goal}.

Vários métodos que utilizam *roadmaps* são apresentados por Latombe (1991) e Choset et al. (2005). Destes, pode-se citar o método do grafo de visibilidade, método de retração (diagrama de Voronoi generalizado), e o método da silhueta. Cada um destes métodos propõe uma maneira diferente de se construir a *roadmap*.

O método do grafo de visibilidade foi um dos métodos mais antigos de planejamento de trajetória (NILSSON, 1969) e pode ser aplicado para espaços de configurações bidimensionais com obstáculos ($\mathcal{C}_{obs}$) poligonais. Em um grafo de visibilidade, um nó do grafo é conectado a outro se estes nós estão no campo de visão um do outro. Além disso, cada ponto do espaço de configurações livres deve estar no campo de visão de pelo menos um nó do grafo de visibilidade. Esta última condição garante que é possível acessar a *roadmap* de qualquer ponto de $\mathcal{C}_{free}$. A criação de um grafo de visibilidade é feita quando se conecta com uma linha reta dois pares de vértices na fronteira de $\mathcal{C}_{free}$ (vértices dos obstáculos em $\mathcal{C}_{obs}$) desde que esta linha não atravesse o interior de nenhum obstáculo. A Figura 14 mostra um exemplo de grafo de visibilidade indicando uma possível solução que conecta dos pontos no espaço de configurações.

O método de retração (Ó'DUNLAING; SHARIR; YAP, 1983) consiste em construir uma *roadmap* a partir de um mapeamento contínuo de $\mathcal{C}_{free}$ na *roadmap*. Um exemplo deste método é a construção de uma *roadmap* pela retração de $\mathcal{C}_{free}$ no seu diagrama de Voronoi generalizado (CHOSET et al., 2005). Este diagrama é definido como sendo um subconjunto unidimensional de $\mathcal{C}_{free}$ que maximiza a distância entre o robô e os obstáculos (Figura 15).

O método da silhueta proposto por Canny (CANNY, 1988) é um método geral de *roadmap* que resolve o planejamento de movimento em tempo exponencial à dimensão do espaço de con-

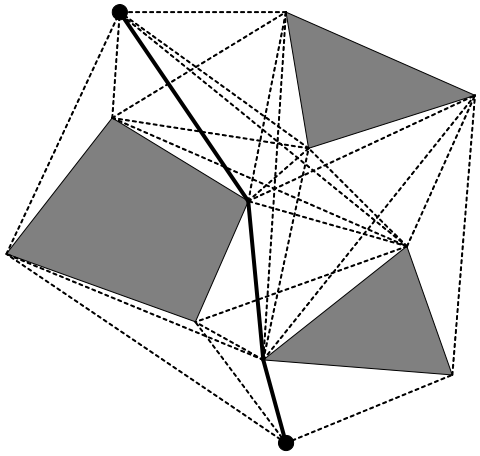

Figura 14: Exemplo de um grafo de visibilidade.

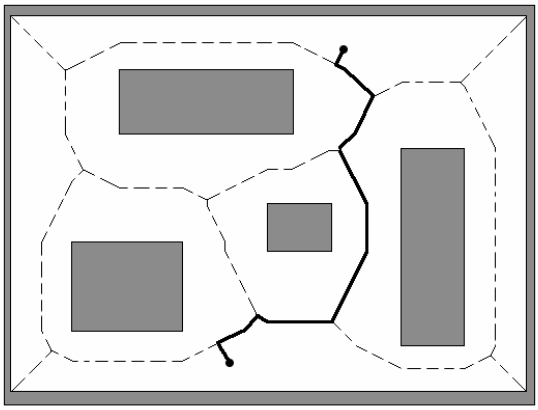

Figura 15: Exemplo de um diagrama generalizado de Voronoi.

figurações. Este método constrói uma *roadmap* a partir da silhueta do espaço de configurações livres. No entanto, esta silhueta é formada por segmentos de curva desconexos. O método então adiciona alguns segmentos na *roadmap* para conectar estes segmentos de curva desconexos da silhueta. Esta conexão ocorre em pontos críticos da silhueta, definidos de modo geral como sendo pontos onde ocorre uma mudança na conectividade da *roadmap*. Uma definição mais detalhada deste método é apresentada por Latombe (1991) e Choset et al. (2005).

4.3 Decomposição em células

A decomposição do espaço de configurações livres em regiões menores chamadas células é uma estratégia que pode ser utilizada para simplificar o planejamento de trajetória. A idéia é que dentro de cada célula, a trajetória entre dois pontos possa ser facilmente gerada.

Quando duas células possuem uma fronteira em comum, elas são células adjacentes. A relação de adjacência ou conectividade entre as células pode ser então representada por um grafo não direcional. Cada nó neste grafo representa uma célula, e cada aresta significa que duas células são adjacentes. Uma vez que o espaço de configurações livres é dividido em células, e este grafo de adjacência é construído, o planejamento de movimento é feito em duas etapas. Na primeira etapa, determina-se quais células contém a configuração inicial e a final do robô. Então uma busca é feita no grafo de adjacência para encontrar uma seqüência de células que vá da célula que contém a configuração inicial até a célula que contém a configuração final. A segunda etapa do método de planejamento é determinar, dentro de cada célula da seqüência, uma curva que ligue pontos na fronteira dessas células, permitindo que o robô se mova de uma célula a outra, partindo da configuração inicial e chegando na configuração final desejada. A Figura 16 mostra um exemplo de um espaço de configurações dividido em células, e uma trajetória que liga um ponto a outro no espaço.

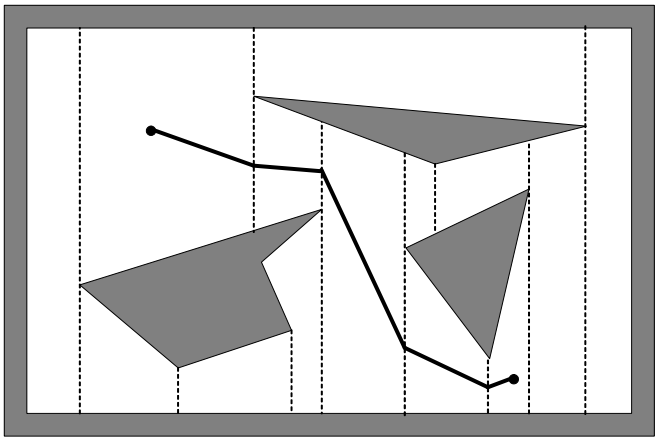

Figura 16: Exemplo de decomposição do ambiente em células.

Os métodos de decomposição em células podem ser divididos em métodos de decomposição exata e métodos de decomposição aproximada.

Na decomposição exata, a união das células corresponde exatamente ao espaço de configurações livres. A divisão de células é feita em pontos críticos do espaço, ou seja, pontos em que ocorre a divisão do espaço livre por um obstáculo.

Na decomposição aproximada, o espaço de configurações livres é decomposto em células de um formato específico, como, por exemplo, retângulos de diversos tamanhos. A união destas células não corresponde exatamente ao espaço livre, mas está incluída nele. Neste tipo de decomposição, a divisão das células não ocorre necessariamente em um ponto crítico do espaço. Um exemplo de divisão aproximada em células é a divisão do ambiente em *quadtrees* quando

o espaço de configurações é bidimensional, e *octrees* quando o espaço é tridimensional.

Os métodos de decomposição exata são considerados métodos de planejamento completo, ou seja, o método é capaz de encontrar uma solução para o problema de planejamento se ela existir, e de informar da não existência de soluções caso contrário. Já o método de decomposição aproximada é considerado um método de resolução-completa, ou seja, se uma solução para o problema existir, o método é capaz de encontrá-la dependendo da resolução máxima usada para dividir o ambiente. Quanto maior a resolução, mais a decomposição se aproxima da decomposição exata, e mais provável de se encontrar uma solução se esta existir. Dependendo da configuração inicial e final do robô e dos obstáculos no ambiente, existe um determinado valor de resolução para o qual a solução pode ser encontrada. Entretanto, quanto maior a resolução, maior também é o espaço computacional necessário para representar a decomposição em células.

4.4 Campo potencial

Nos métodos de campo potencial, um campo escalar $\phi(q)$, que também pode ser chamado de função potencial, é definido sobre o espaço de configurações $\mathcal{C}$. O robô representado por um ponto q neste espaço de configurações age como se fosse uma partícula que se move sob a influência de uma força artificial dada pelo gradiente negativo da função potencial, $-\nabla\phi(q)$. A cada intervalo de tempo, o gradiente negativo da função é calculado para a configuração na qual o robô se encontra. Este gradiente é considerado como sendo a direção de movimento mais promissora para que o robô alcance seu objetivo.

O método de campos potenciais foi desenvolvido inicialmente por Khatib (1985) como um método para navegação e desvio de obstáculos quando não se tem conhecimento *a priori* do ambiente, e portanto os obstáculos são percebidos a medida que o robô navega.

Dentro deste contexto, um potencial de atração, ϕ_{att}, é associado à configuração de destino desejada, q_{goal}. Um potencial de repulsão, ϕ_{rep}, é associado aos obstáculos no espaço de configurações, $\mathcal{C}_{obs}$, que vão sendo percebidos a medida que o robô navega. O potencial ϕ_{att} independe dos obstáculos, e o potencial ϕ_{rep} independe da configuração de destino. A soma destes dois campos potenciais resulta no campo potencial ϕ que age sob o robô.

Esta abordagem é considerada como um planejamento de movimento local pois não se considera os obstáculos que se encontram além de uma determinada região ao redor do robô, ou seja, fora do alcance de seus sensores. Dessa forma, esta abordagem é muito utilizada para definição de comportamentos reativos em diversas arquiteturas reativas e híbridas.

Apesar de ser um método de tempo real eficiente se comparado com os outros métodos de planejamento de movimento, não se pode garantir que o robô alcance a configuração desejada q_{goal}. Na maioria dos métodos baseados em campos potenciais, o robô pode ficar preso em mínimos locais. Apesar disso, existem algumas técnicas *ad hoc* para lidar com estas situações e tentar fazer com que o robô saia destes mínimos locais (BARRAQUAND; LATOMBE, 1991).

4.4.1 Função de navegação

Rimon e Koditschek (1992) introduziram um tipo especial de função potencial que não possui mínimos locais. Para o cálculo desta função, chamada função de navegação, é necessário o conhecimento global do ambiente de trabalho, incluindo os obstáculos, ou seja, o conhecimento prévio do espaço de configurações livres do robô. Dessa forma, este método é considerado um método de planejamento global, assim como o método de decomposição em células e o método de *roadmaps*.

Uma função de navegação é definida da seguinte forma (RIMON; KODITSCHEK, 1992):

Seja $\mathcal{C}_{free}$ o espaço de configurações livres do robô, e q_{goal} a configuração final desejada para o robô dentro de $\mathcal{C}_{free}$. Um mapeamento $\phi : \mathcal{C}_{free} \rightarrow [0, 1]$ é uma função de navegação se ela é:

1. suave em $\mathcal{C}_{free}$, ou seja, ela possui derivadas de segunda ordem contínuas;
2. polar em q_{goal}, ou seja, possui um único mínimo em q_{goal} no componente conectado de $\mathcal{C}_{free}$ que contém q_{goal};
3. admissível em $\mathcal{C}_{free}$, ou seja, uniformemente máxima na fronteira de $\mathcal{C}_{free}$;
4. uma função do tipo Morse.

Uma função do tipo Morse é uma função cujos pontos críticos, ou seja, os pontos em que a derivada ou gradiente da função é zero, não são degenerativos. Isso significa que os pontos críticos são isolados, e que quando se usa um método de descida pelo gradiente de uma função do tipo Morse, qualquer perturbação aleatória é capaz de fazer com que o método saia de um ponto de sela ou de máximo.

Rimon e Koditschek (1992) utilizam os requisitos 2, 3 e 4 da definição para provar a convergência da função de navegação para praticamente qualquer configuração inicial do robô, q_{init}, desde que q_{init} pertença ao domínio de atração da função de navegação, ou seja, q_{init} e q_{goal} pertençam ao mesmo subconjunto conectado de $\mathcal{C}_{free}$. O requisito 1, que estabelece que a função

de navegação deve ser suave, é imposto pois geralmente deseja-se que o robô siga o gradiente negativo da função. Já o requisito 3 faz com que o robô não colida com os obstáculos representados por $\mathcal{C}_{obs}$ ao navegar pelo gradiente negativo da função. A função de navegação deve ser do tipo Morse, requisito 4, pois se q_{init} estiver em um ponto de sela onde $\nabla\phi = 0$, qualquer perturbação faz com que o robô se liberte deste ponto e inicie seu movimento em direção a q_{goal}.

Rimon e Koditschek (1992) mostram como calcular uma função de navegação para um *ambiente de esferas*. A fronteira deste ambiente é definida por um disco n-dimensional. Os obstáculos contidos neste ambiente também possuem a forma de disco, mas não deve haver intersecção entre os obstáculo, e nem entre um obstáculo e a fronteira do *ambiente de esferas*.

A fronteira externa de um *ambiente de esferas* é representada por β_0, e é dada pela seguinte expressão em função de seu raio ρ_0 e seu centro q_0:

$$\beta_0(q) = -\|q - q_0\|^2 + {\rho_0}^2, \tag{4.3}$$

Neste ambiente, um obstáculo circular β_j, para $j = 1, 2, ..., n$, é dado pela seguinte expressão em função de seu raio ρ_j e a coordenada do seu centro q_j:

$$\beta_j(q) = \|q - q_j\|^2 - {\rho_j}^2. \tag{4.4}$$

A função de navegação neste ambiente é dada por:

$$\phi_k(q) = \frac{\|q - q_{goal}\|^2}{[\|q - q_{goal}\|^2 + \beta(q)]^{1/k}} \tag{4.5}$$

onde,

$$\beta \triangleq \prod_{j=0}^{m} \beta_i \tag{4.6}$$

O parâmetro k controla a suavidade da função de navegação. Para um dado ambiente de trabalho este parâmetro deve ser definido de forma que a função não apresente mínimos locais. Conforme k aumenta, o número de mínimos locais diminui. A Figura 17 ilustra uma função de navegação criada para um determinado ambiente de esferas. Para calculo desta função foi usado $k = 2.9$ que, para este ambiente, fez com que a função não possuísse nenhum mínimo local.

Dificilmente um *ambiente de esferas* representa um ambiente real. Por esta razão, Rimon e Koditschek (1992) mostram que se um *ambiente de esferas* puder ser mapeado de modo difeomórfico em um ambiente de geometria mais complexa, existe uma função de navegação

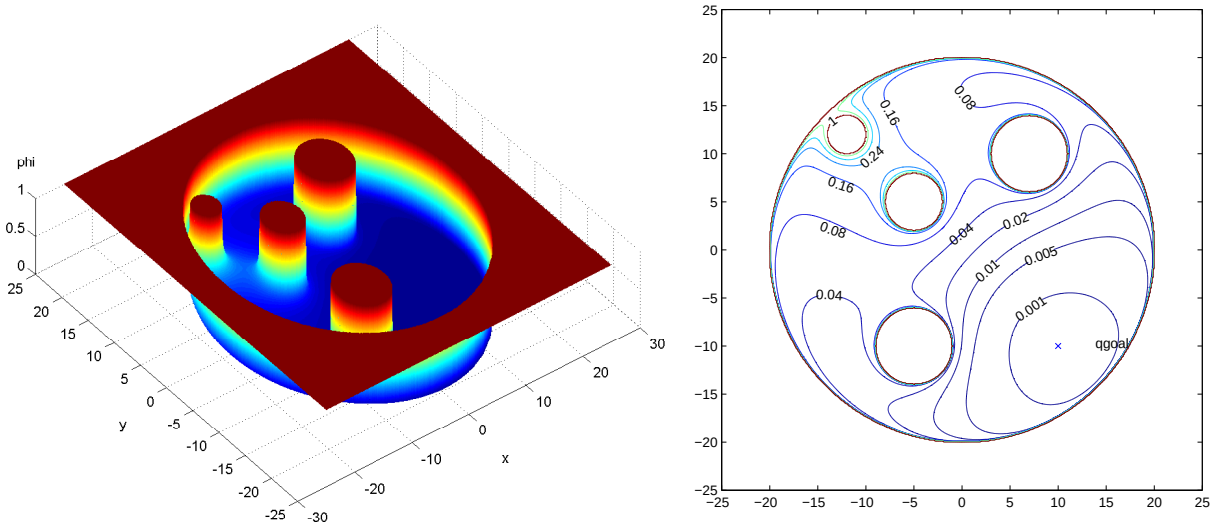

Figura 17: Função de navegação em um mundo de esferas.

definida neste *ambiente de esferas* que também pode ser mapeada para este outro ambiente preservando as suas propriedades de função de navegação. Um mapeamento difeomórfico é um mapeamento suave, bijetor, e que possui inversa também suave.

4.4.2 Seqüência de funções de navegação

Muitas vezes é difícil definir uma única função de navegação para todo o espaço de configurações livres do robô, mesmo utilizando ferramentas de difeomorfismo. Um artifício que pode ser usado para contornar esta dificuldade é utilizar uma seqüência de várias funções de navegação. Neste caso, o domínio de convergência de cada uma destas funções de navegação é definido como sendo um subconjunto do espaço de configurações livres, $\mathcal{C}_{free}$, e a união de todos os domínios de convergência abrange todo $\mathcal{C}_{free}$. Estas funções são definidas de forma que uma função é responsável por levar o robô até o domínio de convergência de outra função, e assim sucessivamente, até que o robô alcance o domínio de convergência da função de navegação responsável por levá-lo à configuração final global desejada.

Uma forma de entender este método é imaginar cada função de navegação como sendo um funil. A parte mais estreita do funil equivale à configuração de destino da função, ou seja, ao objetivo da função, ou região de mínimo global da função. Para qualquer configuração inicial que o robô possua, desde que dentro da região para a qual a função é definida, o robô é conduzido até o objetivo da função de navegação como se estivesse *escorregando* ao longo do funil. Pode-se então definir uma seqüência de funis, sendo que um funil conduz o robô para dentro de outro funil (Figura 18), até que o robô chegue ao seu destino final.

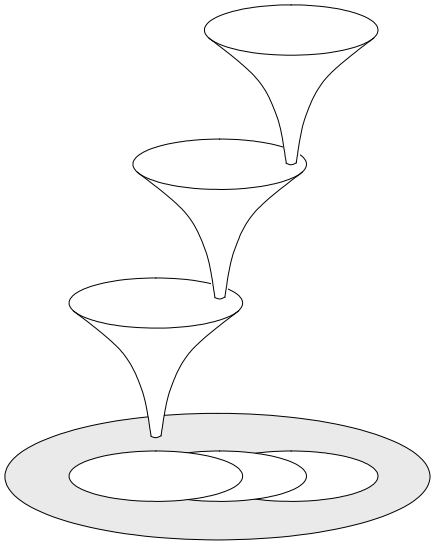

Figura 18: Seqüenciamento de controladores.

Burridge, Rizzi e Koditschek (1999) apresentam esta idéia de composição seqüencial de controladores para resolver problemas complexos de controle de robôs manipuladores. No trabalho apresentado por eles, os controladores não são necessariamente funções de navegação, mas sim controladores projetados para desempenhar um comportamento dentro da tarefa de manipulação. Cada um destes controladores possui um domínio de convergência próprio, sendo que a união destes domínios de convergência abrange todo o espaço de configurações livres da tarefa.

Conner, Rizzi e Choset (2003) utilizam o seqüenciamento de funções de navegação combinado com o método de decomposição de células para resolver problemas de navegação com robôs móveis. No método apresentados por eles, o espaço de configurações é primeiramente dividido em células de geometria simples, para as quais pode-se definir funções de navegação sem muita dificuldade. Então, uma vez que a configuração final para o robô é especificada, determina-se possíveis seqüências de células pelas quais o robô deve caminhar para que chegue a célula que contém a configuração final. Até aqui o método se assemelha ao método de planejamento baseado em decomposição de células. Finalmente, uma função de navegação é definida dentro de cada célula do ambiente. Isto é feito de forma que o robô possa ser conduzido ao longo de uma seqüência de células até a configuração final desejada. Para isso, cada função dentro de uma célula conduz o robô até a fronteira da célula com a próxima célula da seqüência. Uma vez que o robô atravessa a fronteira da célula em que está, ele cai no domínio de convergência de outra função de navegação. Isso é feito sucessivamente, até que a função de navegação na última célula conduz o robô até a configuração de destino desejada.

4.4.3 Funções potenciais numéricas

Funções de navegação analíticas, conforme descrito na seção 4.4.1, são difíceis de serem construídas para todo um espaço de configurações com geometria complexa. Na seção anterior, foi apresentada uma solução de planejamento de movimento que lida com esta dificuldade utilizando uma seqüência de funções de navegação analíticas. Uma outra solução é fazer o cálculo numérico de uma função de navegação para todo o espaço de configurações.

Barraquand, Langlois e Latombe (1992) apresentam algoritmos para calcular numericamente uma aproximação de uma função de navegação. Uma das formas de calcular este tipo de função numérica aproximada é usando programação dinâmica em um espaço de configurações representado na forma de uma grade. Para isso pode-se utilizar um algoritmo de planejamento chamado de *wave-front*.

Este algoritmo funciona da seguinte forma. Considere, por exemplo, um espaço de configurações bidimensional. Inicialmente é dado ao planejador uma grade binária que representa o espaço de configurações. Cada elemento, ou pixel desta grade que corresponde ao espaço livre possui valor zero, e cada pixel que corresponde a um obstáculo possui um valor infinito (na prática, um valor muito alto que é usado para identificar que o pixel está ocupado por um obstáculo). Uma vez fornecida a configuração de destino desejada, q_{goal}, o planejador atribui o valor 1 ao pixel que equivale a esta configuração. A partir daí o algoritmo de *wave-front* tem início, e calcula valores de potencial para os pixels do espaço de configurações livres (os pixels que contém obstáculos já foram marcados com valor de potencial infinito). Define-se Δp como sendo o custo que o robô tem de se mover de um pixel para o seu pixel vizinho. No primeiro passo, cada um dos pixels vizinhos de q_{goal} que estão no espaço livre recebem o valor de q_{goal} acrescentado de Δp. Por exemplo, se apenas os vizinhos 4-conectados forem considerados, estes poderiam receber o valor 2. Se, ao invés disso, os vizinhos 8-conectados forem considerados, pode-se, por exemplo, atribuir o valor 2 aos vizinhos que se encontram ao norte, sul, leste e oeste de q_{goal}, e 2.5 aos vizinhos que estão conectados por uma diagonal com q_{goal}. Isto assume que o custo de se mover na diagonal é maior que o custo de se mover lateralmente, já que a distância percorrida entre o centro dos pixels é maior quando se move na diagonal. No segundo passo, seja q_i um vizinho de q_{goal} que acabou de ter seu valor atualizado, verifica-se entre os vizinhos de q_i (não ocupados por um obstáculo) se existe um pixel q_j com valor zero ou com valor superior ao valor de q_i somado a Δp. Se houver, o pixel q_j terá seu valor atualizado para que seja igual ao valor q_i somado a Δp. O algoritmo procede até que não existam mais pixels que precisem ter seus valores atualizados.

Este tipo de algoritmo resulta em uma função com um único mínimo local. Além do mais,

o gradiente negativo desta função conduzirá o robô pelo caminho de menor distância, ou de menor custo total, entre a configuração atual do robô e a configuração final de destino. Entretanto, a função potencial numérica resultante deste método não pode ser considerada uma função de navegação pois não possui todas as propriedades de tal função. Por exemplo, existe uma descontinuidade da função na fronteira de $\mathcal{C}_{free}$ e $\mathcal{C}_{obs}$. Nos pixels de $\mathcal{C}_{free}$ em torno dos obstáculos a função não é uniformemente máxima. Dessa forma, o caminho percorrido pelo robô, ao seguir o gradiente negativo da função, passará bem próximo dos obstáculos, na borda de $\mathcal{C}_{obs}$. Por fim, em relação a aplicação do algoritmo apresentado, ele geralmente é adequado quando a dimensão do espaço de configurações é pequena, ou seja, $m = 2$ e 3, já que o algoritmo percorre todo o espaço de configurações e sua complexidade é exponencial à dimensão do espaço de configurações.

Uma variação do método de *wave-front* descrito anteriormente procura fazer com que a função potencial conduza o robô o mais longe possível dos obstáculos, ou seja, conduza o robô ao longo do diagrama de Voronoi generalizado do espaço de configurações livres (Figura 15). Entretanto, utilizando a função potencial resultante deste método, o caminho descrito pelo robô não será mais o caminho de menor distância até a configuração final desejada. Este algoritmo é descrito em detalhes por Latombe (1991) e LaValle (2006).

A função de navegação também pode ser calculada utilizando um método de elementos finitos (PIMENTA et al., 2005). Este método permite a discretização de um espaço de configurações de dimensão n em elementos de tamanho variável. Em regiões de grande variação do gradiente da função de navegação, o tamanho dos elementos utilizados para discretizar o ambiente é menor, e em regiões de menor variação do gradiente da função de navegação, o tamanho dos elementos pode ser maior. No calculo da função de navegação utilizando este método, podem ser impostas condições para que a função possua um valor uniforme na fronteira com os obstáculos, fazendo com que o robô ao navegar não se aproxime demais deles.

4.5 Planejamento de movimento baseado em amostragem

Os métodos de planejamento de movimento baseados em amostragem, diferentemente dos demais métodos apresentados, não utilizam uma construção explícita do espaço de configurações dos obstáculos, $\mathcal{C}_{obs}$. Ao invés disso estes métodos realizam uma busca por caminhos livres no espaço de configurações utilizando métodos de amostragem. Por isso, estes métodos são eficientes no planejamento de movimento de robôs com grande número de graus de liberdade, ou seja, em um espaço de configurações de dimensão maior que 3. Entretanto, por trabalharem

com uma amostragem do espaço de configurações, estes métodos não podem ser considerados métodos completos de planejamento. Ao invés disso, utiliza-se um conceito mais fraco que este, e considera-se que os métodos baseados em amostragem são probabilisticamente completos. Ou seja, conforme o tempo gasto na busca pela solução tende a infinito, a probabilidade do método retornar uma solução, se ela existir, tende a 1.

Os métodos baseados em amostragem podem ser classificados em métodos de *busca única* (*single query*), ou métodos de *busca múltipla* (*multiple queries*). Os métodos de *busca única* retornam uma única solução para uma configuração inicial e final fornecidas. O método também reporta falha caso não consiga encontrar uma solução dentro de um limite de tempo pré-estabelecido. Os métodos de *busca múltipla* investem um esforço significativo no pré-processamento do espaço de configurações para construir uma estrutura de dados (uma *roadmap*, por exemplo) que é usada para tornar eficiente a busca por soluções. Nestes métodos, uma vez que esta estrutura é criada, considerando que não ocorrem mudanças no espaço de configurações, podem-se encontrar soluções para múltiplos pares de configurações iniciais e finais.

Como exemplo de métodos de *busca única* pode-se citar as Árvores Aleatórias de Rápida Exploração (*Rapidly-exploring Random Trees*, *ou RRTs*) e as Árvores Densas de Rápida Exploração (*Rapidly-exploring Dense Trees*, ou RDTs)(LAVALLE, 2006). Como exemplo de método de *busca múltipla*, pode-se mencionar *roadmaps* probabilísticas (*Probabilistic Roadmaps*, ou PRMs) (KAVRAKI et al., 1996).

4.6 Considerações finais

Com exceção de um dos métodos de campo potencial, o que se baseia na soma de potenciais independentes de repulsão (atribuídos a obstáculos) e de atração (atribuído ao objetivo de navegação), os métodos de planejamento de movimento apresentados utilizam um mapa do ambiente. Mais precisamente, estes métodos utilizam o espaço de configurações livre do robô determinado a partir do mapa do ambiente. Assim, o planejamento é feito assumindo-se que o ambiente é completamente conhecido e estático.

Nestas condições, os métodos de *roadmap*, de decomposição em células, funções de navegação e funções potenciais globais calculadas numericamente, podem ser considerados métodos completos de planejamento. Ou seja, garantem encontrar uma solução para o problema se tal solução existir, ou dizer se nenhuma solução existe. Quando o espaço de configurações é aproximado por uma grade, *quadtree*, *octree*, ou qualquer forma de decomposição não exata do

ambiente, diz-se que o método é completo na resolução utilizada para aproximação do espaço de configurações. Já o método de planejamento baseado em amostragem é probabilisticamente completo. Neste caso, encontrar uma solução para o problema, se esta solução existir, depende do tempo gasto na busca, ou melhor, gasto para amostragem do espaço de configurações.

No entanto, muitas vezes o ambiente não pode ser considerado estático. Obstáculos podem mudar de posição ou serem adicionados no ambiente, obstruindo a trajetória descrita pelo plano de movimento do robô. Ou ainda, obstáculos dinâmicos podem transitar pelo ambiente enquanto o robô navega. Embora o robô possa parar e replanejar a trajetória levando em conta as mudanças percebidas no ambiente, esta alternativa nem sempre é a mais adequada. Muitas vezes o robô poderia ser capaz de se desviar de um obstáculo imprevisto encontrado ao longo do caminho, e então continuar seguindo a trajetória planejada inicialmente sem a necessidade de um replanejamento. Este replanejamento somente seria feito quando o desvio reativo de obstáculos não fosse suficiente para lidar com a mudança inesperada no ambiente, ou seja, o robô não conseguisse progredir na sua trajetória em direção ao objetivo de navegação. Entretanto, dependendo do método de planejamento escolhido, pode ser difícil implementar de forma satisfatória o desvio reativo de obstáculos de forma coerente com o plano de movimento, e identificar situações em que o replanejamento se faz necessário.

Esposito e Kumar (2002) apresentam uma metodologia que permite combinar comportamentos reativos (desvio de obstáculos e controle de formação, por exemplo) com um plano de movimento descrito por uma função de navegação. Eles mostram que o controlador que segue o gradiente negativo da função não é o único que garante a convergência do robô para o objetivo de navegação. Na verdade, existe um conjunto de soluções admissíveis, sendo que o gradiente negativo é a solução de controle ótimo. Este conjunto é determinado a cada instante ao longo do movimento do robô. Os comportamentos reativos são definidos na forma de restrições dinâmicas que quando ativadas podem limitar a escolha de um controlador dentro do conjunto de soluções admissíveis. A cada instante, o sistema deve escolher um controlador apropriado que faça com que o robô atinja o objetivo de navegação ao mesmo tempo que satisfaz as restrições impostas pelos comportamentos reativos. Se em um dado momento não existir um controlador que atenda ao mesmo tempo o plano de movimento e os comportamentos reativos, duas alternativas podem ser analisadas. A primeira é desconsiderar, se possível, um dos comportamentos reativos. Isto é feito quando o comportamento não é vital na realização da tarefa. É claro que o comportamento de desvio de obstáculos não deve ser desconsiderado para que não ocorram colisões, mas um comportamento de controle de formação pode ser descartado momentaneamente dependendo da aplicação. Finalmente, se mesmo assim ainda não houver soluções possíveis, deve-se fazer um replanejamento global da função de navegação levando-se em conta as mu-

danças no ambiente que impedem o robô de progredir rumo ao seu objetivo. Neste caso, se devido às mudanças no ambiente, o robô estiver em uma região do espaço de configurações que não está conectada ao objetivo de navegação, o algoritmo de planejamento indicará que não existe solução para o problema de navegação.

Outra forma de abordar o problema de planejamento em ambientes dinâmicos é intercalar planejamento, movimento e sensoriamento de forma apropriada (*interleaved planning*). Um exemplo disso é o método de planejamento proposto por Choset e Burdick (2000) baseado em *roadmaps*. Eles propõe o planejamento de movimento por meio de um Diagrama de Voronoi Generalizado Hierárquico utilizando apenas a informação sensorial no campo de visão do robô. O método proposto é utilizado em situações em que o mapa do ambiente não está disponível, e pode ser usado para espaços de configurações com dimensões altas. Neste tipo de planejamento baseado em sensores, o plano de movimento é construído de forma incremental enquanto o robô está se movimentando.

Um outro aspecto não discutido até aqui e que pode ser desejável em um método de planejamento de movimento é a possibilidade deste método lidar com a dinâmica e eventuais restrições de movimento do robô. Os métodos apresentados anteriormente geralmente se preocupam em encontrar um caminho no espaço de configurações livres para um robô holonômico, que não possui restrições cinemáticas de movimento. No entanto, muitos robôs móveis possuem restrições não-holômicas de movimento. Estes robôs precisam fazer manobras para atingirem determinadas configurações.

Alguns dos métodos apresentados podem ser modificados para que incluam a restrição de movimento não-holonômica no planejamento. Um exemplo disso são os métodos de planejamento baseado em amostragem. No método de amostragem de busca única, tal como a RRT, a partir da configuração inicial do robô, constrói-se uma árvore de possíveis movimentos que explora o espaço de configurações até que um de seus ramos chegue na configuração final desejada para o robô. Esta árvore pode ser construída de forma que se leve em consideração as restrições de movimento e a dinâmica do robô. Assim, para uma dada configuração, a amostragem das próximas configurações (ramificações da árvore) é feita somente no espaço de configurações possível de ser alcançado ao considerar as restrições de movimento e dinâmica do robô.

Campos potenciais e funções de navegação também podem ser utilizadas neste contexto. No trabalho de Esposito e Kumar (2002), um campo potencial dipolar é utilizado pois o gradiente negativo deste campo descreve curvas que respeitam a restrição não-holonômica de um robô móvel do tipo carro. Uma vez que o robô se encontra alinhado ao gradiente deste campo dipolar, o gradiente conduzirá o robô por uma trajetória possível de ser seguida. Tanner, Loizou

e Kyriakopoulos (2001) também utilizam um campo potencial dipolar para definir de forma adequada uma função de navegação para robôs não-holonômicos. Restrições dinâmicas no movimento também podem ser tratadas de forma especial. Conner, Rizzi e Choset (2003) definem a função de navegação dentro de cada uma das células de forma a considerar as restrições dinâmicas do robô. A velocidade do robô é controlada pela função de navegação dentro de cada célula de forma a se evitar descontinuidades no movimento na transição de uma célula a outra.

Para a arquitetura híbrida proposta neste trabalho é importante que seja possível integrar comportamentos reativos com o plano de movimento para navegação do robô. Além disso, a possibilidade de interação do usuário com o sistema de controle autônomo do robô deve ser considerada. Dentre os algoritmos de planejamento apresentados, o método de planejamento utilizando uma função global de navegação foi escolhido para a arquitetura proposta. Este método apresenta uma solução completa de planejamento. A função de navegação é definida para todo o espaço de configurações livres, e não depende da configuração inicial do robô, somente da configuração final. Assim, este método permite que o robô possa ser controlado de forma contínua e em malha fechada de qualquer posição inicial do ambiente até o seu objetivo de navegação. A integração do plano de movimento com comportamentos reativos pode ser feita de forma adequada (ESPOSITO; KUMAR, 2002). Além disso, a abordagem de campos potenciais possibilita a integração contínua da entrada de controle do usuário com o plano de movimento. Um exemplo disso é a coordenação de entradas que existe na arquitetura AuRA por meio da soma de vetores. Por fim, embora o enfoque deste trabalho não seja resolver o problema de planejamento de movimento considerando a dinâmica e restrições não-holonômicas do robô, na literatura podem ser encontrados alguns trabalhos que procuram resolver estas questões utilizando funções de navegação (TANNER; LOIZOU; KYRIAKOPOULOS, 2001; ESPOSITO; KUMAR, 2002; CONNER; RIZZI; CHOSET, 2003). Portanto, estes aspectos podem ser considerados em trabalhos futuros.

5 Arquitetura para robô móvel desenvolvida

Uma arquitetura híbrida (deliberativa/reativa) foi desenvolvida com o objetivo principal de lidar com tarefas de navegação e permitir que um operador humano interaja com o robô em diferentes níveis de controle por meio das interfaces disponíveis. Esta arquitetura utiliza funções de navegação para descrever o plano de movimento do robô. Dentro deste contexto, foi desenvolvido um método para coordenar um comportamento deliberativo, que executa o plano de movimento, com comportamentos reativos, que impõe restrições durante a navegação em tempo de execução, e com as entradas de controle contínuas provenientes de um usuário humano.

A arquitetura híbrida desenvolvida, representada na Figura 19, é composta dos seguintes componentes principais: módulos de percepção e atuação, módulos de mapeamento e localização, planejador de movimento, comportamentos reativos e comportamento deliberativo, coordenador de entradas de controle para navegação, e um módulo para seleção de comportamentos.

Neste capítulo serão apresentados os principais componentes da arquitetura. Em particular, na seção 5.4, será apresentada a metodologia utilizada para combinar o comportamento deliberativo com comportamento reativo e com entradas de controle fornecidas pelo usuário humano em uma tarefa de navegação. Este método juntamente com a definição da arquitetura são as contribuições principais deste trabalho.

5.1 Percepção e atuação

Os módulos de percepção são responsáveis pela leitura dos sensores e processamento dos dados obtidos para extração de informações que possam ser relevantes a outros componentes do sistema, tais como, comportamentos, localização e mapeamento.

Já os módulos de atuação são responsáveis por traduzir a entrada de controle recebida em comandos que devem ser enviados aos atuadores. Em algumas aplicações, sensores e atuado-

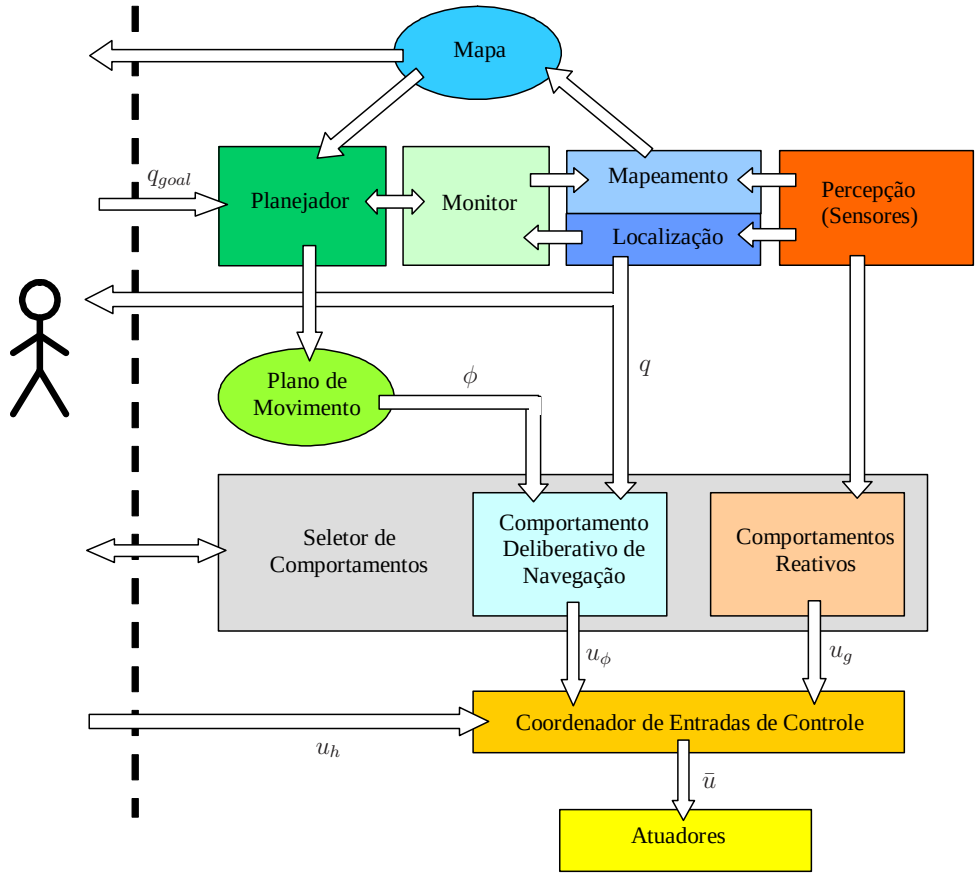

Figura 19: Arquitetura desenvolvida para robôs móveis

res, além de estarem presentes no robô a ser controlado, também podem estar distribuídos no ambiente.

De forma geral, por meio dos módulos de percepção e atuação acontece a interface do sistema de controle com o hardware necessário para obter informações a respeito do ambiente e atuar neste ambiente de trabalho. A implementação de cada módulo depende dos sensores e atuadores disponíveis na aplicação.

5.2 Mapeamento e localização

Mapeamento e localização são atividades que auxiliam as tarefas de navegação dentro de uma arquitetura.

Por meio do mapeamento, um modelo interno do ambiente no qual o robô está inserido, representado na forma de um mapa, é construído e/ou atualizado. Este mapa é utilizado pelo planejador para gerar um plano de movimento para que o robô alcance seu objetivo de navega-

ção. Um mapa do ambiente pode ser totalmente gerado pelo robô, ou então fornecido ao robô *a priori*. Neste último caso, após ser fornecido ao robô, o mapa passa a ser atualizado a medida que o robô navega pelo ambiente. Também existem situações em que o robô começa a mapear o ambiente, e em um dado momento, parte ou todo o mapa é fornecido a ele. Nestas situações, as informações obtidas pelo robô até o momento devem ser combinadas com as novas informações fornecidas.

Já a localização é responsável por determinar uma estimativa da posição do robô em relação ao mapa. Localização e mapeamento podem são tratadas como atividades interdependentes. Afinal, para que um mapa fiel possa ser gerado, o robô precisa saber sua localização. Por outro lado, o robô pode contar com o auxílio de um mapa para se localizar no ambiente. Alguns métodos procuram solucionar o problema de localização e mapeamento de forma simultânea, e são conhecidos como métodos de *SLAM* (*Simultaneous Localization And Mapping*) (DISSANAYAKE et al., 2001).

Na literatura, dependendo de como é feita a navegação do robô, diversos métodos para localização e mapeamento podem ser utilizados. Em geral estes métodos podem ser divididos em métodos métricos (ELFES, 1986; SMITH; SELF; CHEESEMAN, 1990) e topológicos (KUIPERS; BYUN, 1987), sendo que um uso integrado de ambos também é possível (TOMATIS; NOURBAKHSH; SIEGWART, 2003).

Na arquitetura proposta, o módulo de mapeamento deve criar e/ou atualizar um mapa do tipo métrico que indique o espaço ocupado pelos obstáculos conhecidos no ambiente. Este mapa fica disponível ao usuário para que este possa selecionar uma posição para a qual deseja que o robô navegue de forma autônoma. A posição selecionada deve ser uma posição não ocupada por obstáculos e que pertença a uma área que já foi mapeada. Então, selecionada esta posição, o mapa é utilizado pelo planejador de movimentos que determina um plano de movimento para o robô, ou seja, uma forma de fazer com que o robô se mova até a posição selecionada. Nesta arquitetura, o mapa utilizado deve ser do tipo métrico pois a partir deste tipo de mapa o planejador pode calcular a função de navegação utilizada para descrever o plano de movimento autônomo do robô.

Como a localização deve ser feita de forma coerente ao mapa utilizado, e nesta arquitetura se utiliza um mapa métrico, a posição do robô é dada em coordenadas métricas globais em relação a um sistema de coordenadas fixo no mapa. Este tipo de localização é necessária para que o robô possa executar o tipo de plano de movimento gerado nesta arquitetura.

5.3 Planejamento e comportamentos para navegação

O planejamento e execução da tarefa de navegação é realizado pelo módulo de planejamento de movimento, comportamento deliberativo de navegação e comportamentos reativos que possam ser relevantes na tarefa de navegação.

O módulo planejador de movimento gera um plano de movimento a partir do mapa de ocupação e da posição de destino indicada neste mapa pelo usuário. O plano de movimento representa a maneira como o robô deve se mover para chegar de sua posição atual até a posição de destino. Nesta arquitetura, o plano de movimento é representado por uma função global de navegação (LATOMBE, 1991; RIMON; KODITSCHEK, 1992), conforme apresentado no capítulo 4. Esta função de navegação é definida para o espaço livre do mapa. Independente da posição em que o robô se encontre no mapa, esta função guiará o robô até o objetivo, isso se existir um caminho livre até este objetivo.

Um comportamento deliberativo de navegação é o principal responsável pela execução do plano de movimento gerado pelo planejador. Este comportamento utiliza o plano de movimento juntamente com a informação sobre a posição global do robô, resultante do módulo de localização, para gerar continuamente uma entrada de controle que indica a direção e velocidade com a qual o robô deve se mover naquele instante para poder atingir de forma ótima a posição de destino indicada pelo usuário (objetivo de navegação). Este comportamento é considerado deliberativo justamente por usar o plano de movimento calculado a partir de um modelo interno global do ambiente (mapa).

Durante a navegação, são usados comportamentos reativos para lidar em tempo de execução com mudanças dinâmicas no ambiente e eventuais restrições do problema que devem ser observadas, e que não estão embutidas no plano de movimento. Estes comportamentos associam a informação dos sensores com os atuadores do sistema de forma mais direta. Em tarefas de navegação, um comportamento importante é o de desvio reativo de obstáculos que não estão modelados no mapa do ambiente. Entretanto, comportamentos reativos também podem ser definidos para que o robô atenda algum critério durante a tarefa de navegação. Alguns exemplos seriam: comportamento para manter uma determinada formação de robôs, comportamento que mantenha o robô em regiões onde ele possa se localizar mais facilmente, ou ainda, regiões que apresentem sinal para comunicação, etc (SPLETZER et al., 2001; PEREIRA et al., 2002; ESPOSITO; KUMAR, 2002).

5.4 Coordenação de entradas de controle para navegação

O comportamento deliberativo e os comportamentos reativos para navegação devem ser coordenados de maneira adequada. Além disso, esta arquitetura deve possibilitar que o homem coopere com o sistema autônomo do robô em tarefas de navegação. Assim, a qualquer momento durante a navegação o usuário humano pode interagir com o sistema de controle ao enviar comandos contínuos de velocidade e direção desejadas. Neste contexto, estes comandos devem ser coordenados juntamente com o comportamento deliberativo de navegação e com os comportamentos reativos. Isto é feito por um módulo de coordenação de entradas de controle. Este módulo foi desenvolvido em colaboração com Sarangi P. Parikh e Prof. Dr. Vijay Kumar no Laboratório GRASP (PARIKH et al., 2004, 2005), e tem como base o método de coordenação proposto por Esposito e Kumar (2002).

Seja $\phi(q)$ a função de navegação que representa o plano de movimento, onde q é a configuração do robô. Para se chegar à configuração final desejada (posição escolhida no mapa pelo usuário), o comportamento deliberativo de navegação deve conduzir o robô de forma a satisfazer a restrição $\dot{\phi}(q) < 0$. Esta restrição é satisfeita quando o robô se move ao longo do gradiente negativo da função de navegação, $-\nabla\phi(q)$. Sendo assim, o controlador usado para representar o comportamento deliberativo de navegação é dado por:

$$\dot{q} = u_\phi = -\nabla\phi(q). \tag{5.1}$$

Entretanto, $-\nabla\phi(q)$ não é a única entrada de controle que faz com que o robô se mova para a posição de destino. Qualquer entrada de controle no mesmo semi-espaço definido por $-\nabla\phi(q)$ é um controlador admissível (ESPOSITO; KUMAR, 2002). Então, conforme mostra a Figura 20, U_ϕ é o semi-espaço que contém todos os vetores de velocidade que o robô pode seguir e que satisfazem a restrição de navegação $\dot{\phi}(q) < 0$. Isso é usado para combinar o comportamento deliberativo de navegação com os comportamentos reativos e entrada de controle do usuário humano.

Dentro desse contexto de navegação, os comportamentos reativos são expressos como restrições dinâmicas que devem ser satisfeitas enquanto se navega em direção ao objetivo. Cada uma destas restrições é representada por uma inequação $g_i(q,t) \leq 0$. Uma vez que um dado comportamento reativo é ativado, a restrição dada pela inequação é imposta ao módulo de coordenação de entradas de controle, e uma entrada de controle que satisfaça esta restrição deve ser escolhida. O espaço de soluções que satisfazem a restrição é dado por U_g. E o espaço de soluções que fazem com que o robô navegue em direção ao alvo ao mesmo tempo que satisfaz

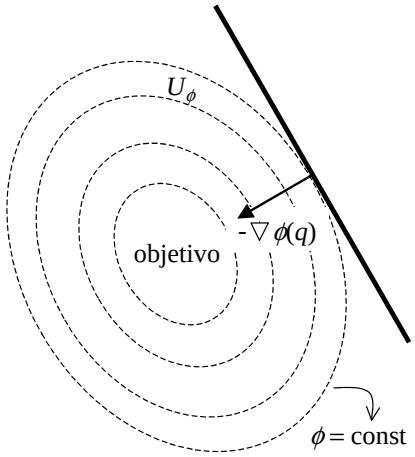

Figura 20: Semi-espaço U_ϕ definido pelo plano de movimento.

a restrição dinâmica imposta pelo comportamento reativo é $F = U_\phi \cap U_g$.

No caso do desvio reativo de obstáculos não modelados no mapa, a detecção dos obstáculos é feita em tempo de execução pelos sensores disponíveis. Seja z_i a distância entre o robô e um objeto i medida através de um sensor, f_i o ângulo do obstáculo em relação ao robô, e δ a distância mínima requerida entre o robô e um obstáculo, pode-se definir uma função $g_i = g_i(z_i, f_i, q)$ de tal forma que $g_i \leq 0$ quando $|z_i| \geq \delta$, e $\dot{g}_i < 0$ quando a distância z_i aumenta. Esta função representa o comportamento reativo de desvio de obstáculos, e nela pode-se levar em conta a dinâmica do robô e do obstáculo, conforme mostra Esposito e Kumar (2002). Uma maneira simplificada de determinar o momento em que o comportamento de desvio reativo de obstáculo deve ser ativado é utilizar a condição $|z_i| \leq \delta$. Assim que esta condição se torna verdadeira, ativa-se o comportamento de desvio de obstáculo. A partir de então, uma entrada de controle deve ser escolhida de tal forma que faça com que a restrição dinâmica $g_i \leq 0$ seja satisfeita. A condição de ativação apresentada considera que o obstáculo está parado e não leva em conta a velocidade do robô. Além disso, esta condição de ativação também assume que o robô se move de tal forma que se o desvio de obstáculo se iniciar a uma distância δ do obstáculo, o robô consegue se desviar. Entretanto, uma condição de ativação mais geral pode ser usada para determinar o instante de ativação do comportamento reativo em função da velocidade do robô e do obstáculo, conforme apresentado por Esposito e Kumar (2002).

Uma entrada de controle que garante que $\dot{g}_i < 0$ (robô se afasta do obstáculo) é $u_g = -\nabla g(q)$. Ou seja, o robô se move na direção oposta ao obstáculo. Entretanto, esta não é a única entrada de controle que faz com que a restrição imposta pelo comportamento de desvio

de obstáculos seja satisfeita. Existe um semi-espaço, na forma de um cone, ou complemento de um cone, referenciado por U_g, que satisfaz a restrição imposta.

Portanto, para coordenar o comportamento deliberativo de navegação com o comportamento reativo de desvio de obstáculos, deve-se selecionar uma entrada de controle que satisfaça $\dot{\phi} \leq 0$ (para que a distância até o objetivo de navegação diminua) e $\dot{g}_i \leq 0$ (para que a distância entre o robô e o obstáculo aumente). Para isso, escolhe-se qualquer entrada u que pertença ao espaço de soluções admissíveis dado por $F = U_\phi \cap U_g$. Como deseja-se que o robô navegue de forma ótima até o objetivo, escolhe-se o vetor de entrada no espaço F que mais se aproxima da solução ótima, $-\nabla\phi(q)$. A Figura 21 procura mostrar os semi-espaços definidos pelo comportamento deliberativo de navegação e pelo comportamento reativo de desvio de obstáculos.

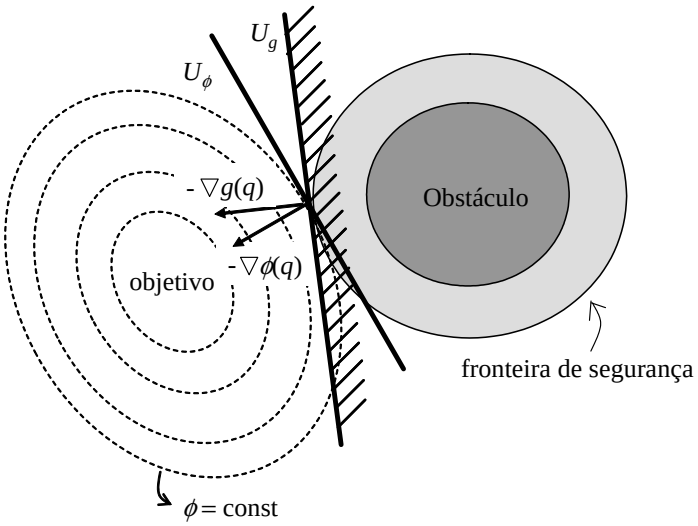

Figura 21: Comportamento deliberativo de navegação combinado com o comportamento de desvio de obstáculos. A região de soluções admissíveis é dada por $F = U_\phi \cap U_g$.

Na arquitetura desenvolvida, o usuário humano pode fornecer uma entrada de controle a qualquer momento, indicando a direção e velocidade com a qual ele deseja que o robô se mova. Esta entrada de controle também deve ser coordenada com o comportamento deliberativo de navegação e o comportamento reativo de desvio de obstáculos. Para que isso pudesse ser feito, foram estabelecidos graus de prioridade para cada uma das entradas de controle. Dessa maneira, o desvio de obstáculos possui o grau de prioridade maior em tarefas de navegação. A restrição imposta por este comportamento deve ser sempre mantida garantindo a segurança ao evitar colisões. Com grau de prioridade abaixo do comportamento de desvio de obstáculos está o usuário humano. Assim, optou-se por sempre que existir uma entrada de controle do usuário humano, a ele é dado o controle do robô enquanto se mantém a prioridade de segurança. Finalmente, quando não existe entrada do usuário, e o comportamento deliberativo de navegação

está ativo, o robô se move de forma totalmente autônoma.

Portanto, de forma geral, a seguinte hierarquia de prioridades é usada na coordenação de comportamentos: em primeiro lugar estão os comportamentos que garantem a integridade física do usuário e do robô como, por exemplo, desvio de obstáculos; logo em seguida estão as entradas de controle fornecidas pelo usuário; então, finalmente, estão os outros comportamentos relacionados com a navegação autônoma do robô, que se preocupam em atingir os objetivos de navegação definidos e/ou manter objetivos secundários durante a navegação como, por exemplo, o comportamento deliberativo de navegação, ou um comportamento para manter uma determinada formação de um time de robôs.

Em situações específicas, apesar dos comportamentos que procuram atingir um objetivo de navegação possuírem prioridade inferior a dos comandos do usuário, a entrada de comando do usuário pode ser modificada para que estes comportamentos de navegação possam ser atendidos. Fica a critério do desenvolvedor do sistema de controle definir estas situações, que geralmente estão relacionadas a casos em que a partir da entrada de controle do usuário humano não é possível ter certeza de que o usuário deseja interromper os comportamentos de navegação autônoma abandonando os objetivos associados a estes comportamentos.

A Figura 22 mostra como a entrada do usuário humano, u_h, é combinada com o comportamento deliberativo de navegação. Neste caso, o usuário já havia selecionado uma posição no mapa para a qual ele deseja que o robô navegue. Utilizando o comportamento deliberativo de navegação, o robô se dirige rumo ao objetivo. Em um dado momento, o usuário pode resolver intervir no movimento do robô modificando localmente a trajetória que o robô está descrevendo de forma autônoma, para isso ele utiliza um *joystick*, por exemplo, para enviar a entrada de controle u_h ao coordenador de entradas. Se a entrada do usuário é consistente com o objetivo inicial de navegação, ou seja, $u_h \in U_\phi$, então é permitido que o usuário tenha o controle completo da cadeira de rodas, $u = u_h$, conforme mostra a Figura 22 (esquerda). Se a entrada do usuário não é consistente com o objetivo de navegação, então, existem duas opções, ilustradas na Figura 22 (direita). Uma delas ocorre quando a entrada do usuário estiver em um cone definido no semi-espaço de velocidade oposto a U_ϕ. Neste caso, admite-se que o usuário deseja realmente ir naquela direção abandonando temporariamente o objetivo inicial. Por isso, o objetivo de navegação é ignorado, e novamente o usuário tem o controle da cadeira, ou seja, $u = u_h$. No entanto, se a entrada do usuário não estiver consistente com o objetivo de navegação, mas estiver em uma região do espaço oposto a U_ϕ próximo à sua fronteira, modifica-se a direção da entrada do usuário para que fique alinhada à fronteira de U_ϕ. Neste caso, assume-se que, embora o usuário queira modificar localmente a trajetória do robô, a intenção do usuário não

é se afastar do objetivo inicial. O vetor de controle usado nesta situação é $u = u_m$, sendo que u_m possui o mesmo módulo de u_h, ou seja, $||u_m|| = ||u_h||$, ele pertence ao plano cuja normal é $-\nabla\phi(q)$, e possui direção dada por:

$$\hat{u}_m = \frac{u_h - (u_h \cdot \hat{t}_\phi)\hat{t}_\phi}{||u_h - (u_h \cdot \hat{t}_\phi)\hat{t}_\phi||} \tag{5.2}$$

onde

$$\hat{t}_\phi = \frac{\nabla\phi(q)}{||\nabla\phi(q)||} \tag{5.3}$$

é o versor normal ao plano.

Para todos os casos mencionados em que o usuário interfere no movimento autônomo do robô, assim que o usuário deixa de fornecer uma entrada de controle, o robô retorna a se mover em direção ao objetivo inicial de navegação utilizando o vetor de controle $u = -\nabla\phi(q)$.

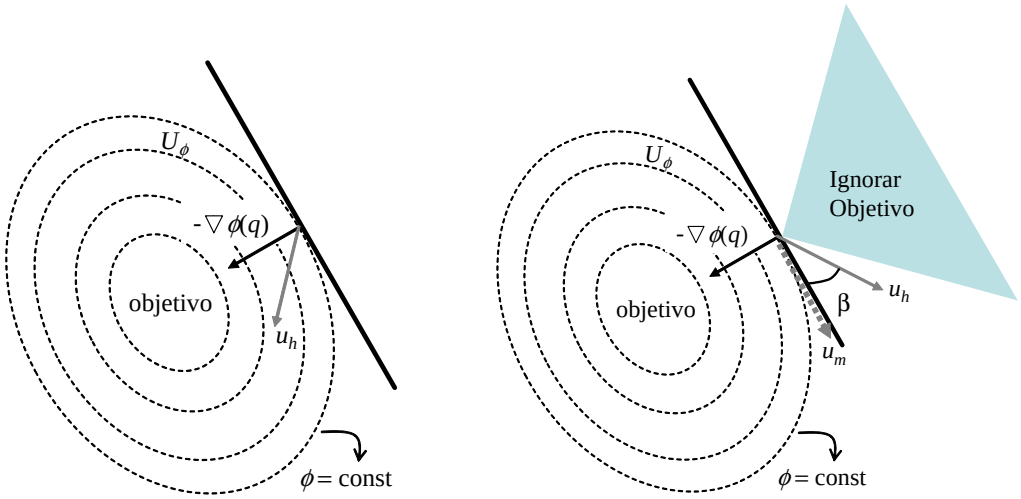

Figura 22: Entrada do usuário combinada com o comportamento deliberativo de navegação. Na situação à esquerda, a região de soluções admissíveis é o semi-espaço contendo $-\nabla\phi(q)$. À direita, a entrada humana ocorre fora da região admissível e é modificada para que seja consistente com o objetivo.

Mesmo quando não existe um objetivo de navegação definido, ou seja, o comportamento deliberativo de navegação está desabilitado, o usuário ainda pode querer controlar manualmente o robô. Nesta situação, o controle é definido simplesmente como sendo $u = u_h$, onde u_h é o vetor de entrada humana.

No entanto, se o robô ao ser dirigido manualmente se próxima de obstáculos, é importante combinar as entradas de controle do usuário com o comportamento de desvio de obstáculos para se evitar colisões. Se a entrada de controle do usuário, u_h, estiver no semi-espaço de velocidade

admissível, U_g, ou seja, a entrada de controle não resultará em colisão com o obstáculo, então esta entrada é utilizada para controlar o robô, e $u = u_h$, conforme mostra a Figura 23 (esquerda). No entanto, se a entrada do usuário está localizada na região do espaço de velocidade restrita pelo comportamento de desvio de obstáculos, então é preciso parar o movimento ou modificar a entrada do usuário para que o robô desvie do obstáculo. A Figura 23 (direita) ilustra esta situação. Define-se um cone no espaço restrito chamado de região de parada. Se a entrada do usuário estiver dentro desta região, optou-se por parar o robô. Se a entrada do usuário estiver fora desta região de parada, mas ainda dentro do espaço restrito pelo comportamento reativo de desvio de obstáculos, a direção da entrada do usuário é modificada para que fique na fronteira do espaço admissível, U_g. Isto permite que o robô se mova aproximadamente na direção especificada pelo usuário, preservando a intenção de movimento do usuário ao mesmo tempo que o obstáculo é evitado. O vetor de controle usado nesta situação é $u = u_m$, sendo que u_m possui o mesmo módulo de u_h, ou seja, $||u_m|| = ||u_h||$, ele pertence ao plano cuja normal é $-\nabla g_i(q)$, e possui direção dada por:

$$\hat{u}_m = \frac{u_h - (u_h \cdot \hat{t}_g)\hat{t}_g}{||u_h - (u_h \cdot \hat{t}_g)\hat{t}_g||} \tag{5.4}$$

onde

$$\hat{t}_g = \frac{\nabla g_i(q)}{||\nabla g_i(q)||} \tag{5.5}$$

é o versor normal ao plano.

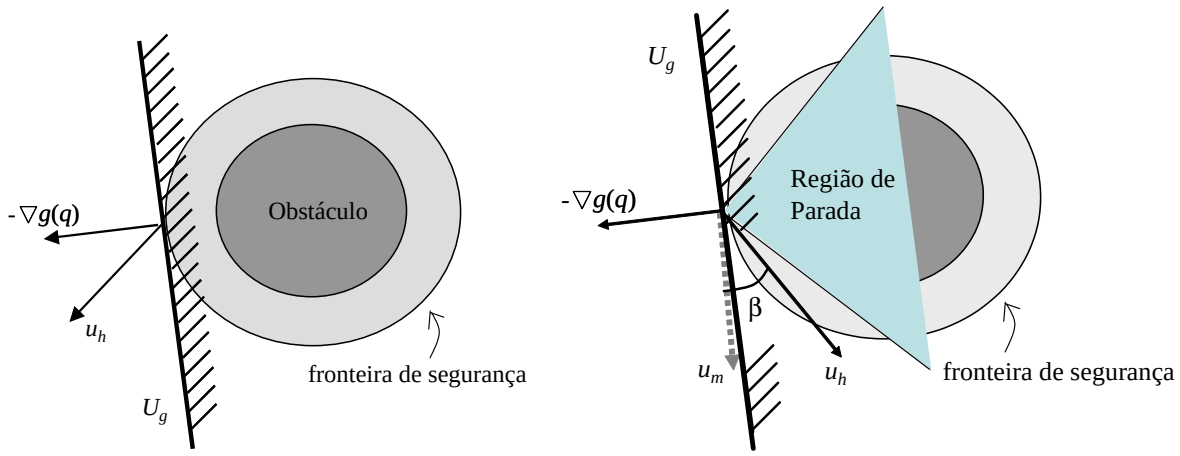

Figura 23: Entrada do usuário combinada com o desvio reativo de obstáculo. Na situação à esquerda, a entrada do usuário é consistente com o semi-espaço determinado pelo desvio reativo de obstáculo. À direita, a entrada do usuário está dentro da região restrita, então a direção da entrada do usuário é modificada permitindo que o usuário mantenha um controle limitado sobre o robô sem que ocorra colisões.

Finalmente, existem situações durante a navegação em que as três fontes de entrada de

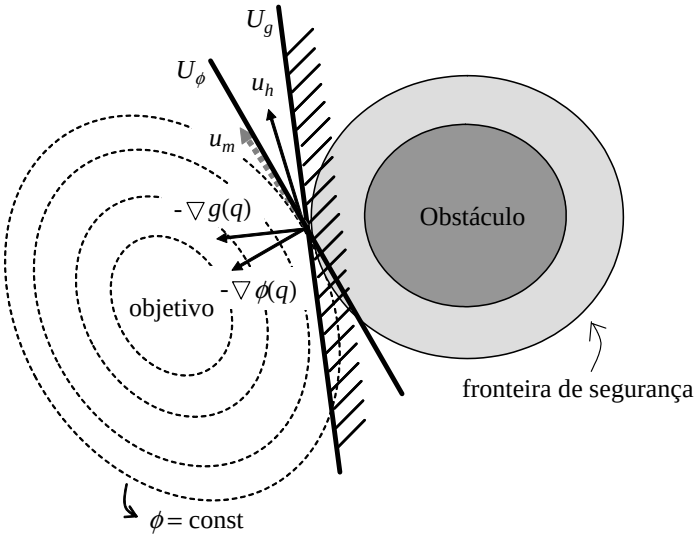

Figura 24: Entrada do usuário combinada com o plano de movimento e com o desvio reativo de obstáculo. Embora a entrada do usuário não esteja no espaço de soluções admissíveis, o controlador modificado, u_m, dá ao usuário um controle limitado enquanto previne colisões.

controle (comportamento deliberativo de navegação, comportamento reativo de desvio de obstáculo, e entrada do usuário) devem ser combinadas simultaneamente. Nestas situações, como regra geral, a entrada de controle do usuário sempre é utilizada quando esta não está em conflito com o objetivo de navegação ou com o comportamento de desvio de obstáculo. Em outras palavras, $u = u_h$ se $u_h \in U_\phi \cap U_g$. Do contrário, aplica-se as regras descritas anteriormente. Primeiro a entrada do usuário é coordenada com o comportamento deliberativo de navegação. A entrada de controle resultante será a própria entrada do usuário, caso o objetivo de navegação for ignorado, ou a entrada do usuário modificada para que o robô não se afaste do objetivo. Depois de feita esta coordenação inicial, a entrada de controle resultante é coordenada com o comportamento reativo de desvio de obstáculos. Então, a entrada de controle pode ser mantida se estiver na região que não resultará em colisão, ou a entrada de controle é modificada para que o robô possa desviar do obstáculo. Isto permite que o usuário possa compartilhar o controle com o sistema enquanto colisões são evitadas. A Figura 24 procura mostrar um exemplo da coordenação destas três entradas de controle.

É interessante observar que apesar das entradas de controle passarem pelo módulo coordenador de entradas antes que a entrada de controle resultante seja enviada para os atuadores, este módulo possui um algoritmo relativamente simples que não consome muito tempo de processamento. Por esta razão o coordenador de entradas não é considerado um “gargalo” no sistema.

5.5 Monitor de progresso e replanejamento

Existem situações em que o comportamento reativo de desvio de obstáculos não obtém sucesso ao lidar com certos obstáculos não modelados no mapa. Algumas destas situações se encontram ilustradas na Figura 25, e abrangem basicamente situações em que o caminho que o robô está percorrendo em direção ao objetivo está completamente bloqueado, ou quando a geometria local do obstáculo não favorece o método de desvio reativo.

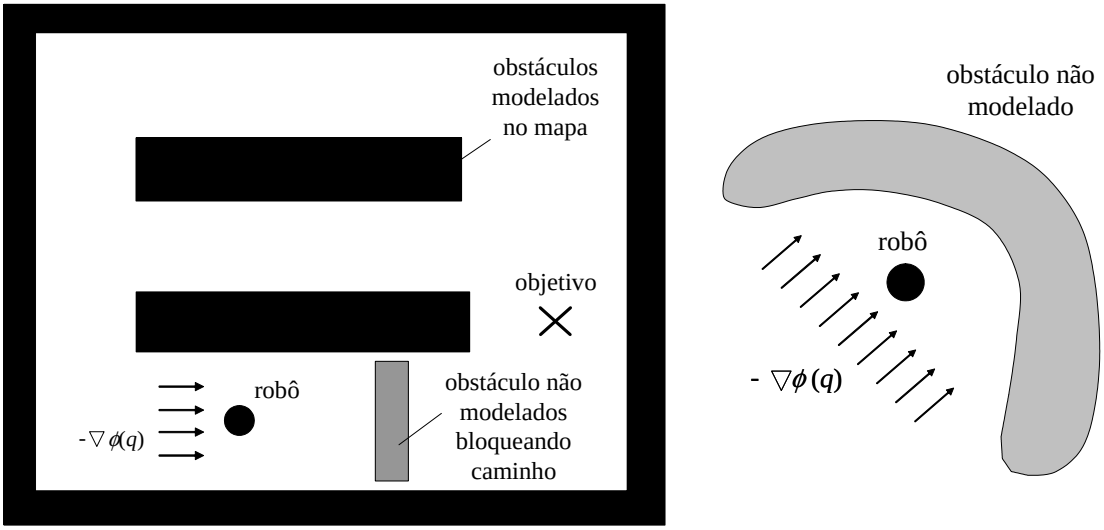

Figura 25: Situações onde replanejamento é necessário. À esquerda, quando um obstáculo não modelado no mapa bloqueia completamente o caminho do robô. À direita, quando um obstáculo não modelado no mapa possui uma geometria que não favorece o desvio reativo do robô.

Nestas situações, durante a navegação completamente autônoma, o robô pode chegar a um ponto em que não consegue progredir no seu movimento em direção ao objetivo devido a um obstáculo não modelado no mapa. Um monitor de progresso pode ser usado para detectar em tempo de execução quando isso acontece. O que pode ser evidenciado quando o espaço de soluções de controle admissível, que atende ao mesmo tempo o desvio reativo de obstáculos e o plano de movimento, é nulo. Além de observar o espaço de soluções admissíveis, procurando identificar quando este é nulo, o monitor de progresso também deve utilizar um histórico da posição global do robô juntamente com o objetivo da tarefa do robô para identificar se o robô está progredindo ou não.

Caso o monitor de progresso indique a necessidade de um replanejamento, o obstáculo detectado que impede o progresso do robô deve ser incluído no mapa pelo módulo de mapeamento, e um novo plano de movimento deve ser calculado utilizando este mapa recém-atualizado. Muitas vezes, o robô não consegue detectar em tempo de execução toda a geometria do obstáculo

que impede o progresso do robô. Mesmo assim, em uma primeira tentativa, atualiza-se o mapa incluindo apenas a parte que foi detectada do obstáculo. O novo plano de movimento gerado a partir dete mapa atualizado fará com que o robô tente se desviar desta parte detectada. Se, por ventura, o robô ficar preso novamente em outra parte do obstáculo que não foi mapeada na etapa anterior, o mapa é atualizado mais uma vez para incluir esta parte agora detectada, e um outro plano é calculado. Isso se repete até que o robô consiga fazer progresso em direção ao objetivo utilizando o plano de movimento atual e os comportamentos de desvio reativo de obstáculos. Ou então, até que seja detectado por meio do mapa que não existe caminho possível até o objetivo. Neste caso o usuário do robô deve ser avisado da impossibilidade de se atingir o objetivo.

Apesar de prever a necessidade do monitor de progresso na arquitetura proposta, este trabalho não se preocupou com o desenvolvimento de algoritmos específicos que poderiam ser utilizados para implementá-lo.

5.6 Outros comportamentos

Além dos comportamentos utilizados para navegação, outros comportamentos também podem ser definidos para que o robô realize pequenas tarefas específicas como, por exemplo, atravessar portas abertas, se mover ao longo de um corredor evitando obstáculos, seguir pessoas ou outros robôs, se aproximar de um objeto (mesa, parede, estação de recarga, etc) (PATEL et al., 2002). Cada um destes comportamentos pode ser definido e implementado por controladores específicos.

Dependendo da tarefa ou modo de operação do robô, certos comportamentos devem ser habilitados e outros desabilitados. Este papel é desempenhado por um módulo de seleção de comportamentos.

No momento, na arquitetura desenvolvida e representada pela Figura 19, a seleção dos comportamentos habilitados é feita pelo operador humano. Mas vale ressaltar que quando o usuário deseja realizar uma tarefa de navegação, por exemplo, ele deve habilitar no início da tarefa o comportamento deliberativo de navegação juntamente com o comportamento reativo de desvio de obstáculos e, possivelmente, algum outro comportamento reativo desejado referente a navegação. Embora o comportamento reativo de desvio de obstáculos permaneça habilitado durante toda a tarefa, ele só é ativado quando o robô detecta automaticamente um obstáculo que precisa ser desviado. O módulo coordenador de entradas de controle permite a interação do usuário com o sistema nas tarefas de navegação.

Em trabalhos futuros, a arquitetura pode ser modificada permitindo uma seleção automática de comportamentos, a semelhança de outras arquiteturas híbridas, como Atlantis e Saphira. Isso permitiria que missões mais complexas e abrangentes do que navegação pudessem ser realizadas de forma autônoma. Entretanto, para que isso possa ser realizado, a exemplo da arquitetura Saphira, dentre outras, o robô deve possuir outras informações no seu modelo interno de mundo além da posição dos obstáculos. Da mesma forma, além do planejamento de movimento realizado para resolver tarefas de navegação, também seria necessário um planejamento de missão mais abrangente. Este planejamento resultaria em um plano formado por uma seqüência de comportamentos capaz de realizar uma missão complexa. Então, o selecionador de comportamentos, a exemplo da camada intermediária de uma arquitetura de três camadas, seria responsável pelo seqüenciamento adequado dos comportamentos, ou controladores, conforme o plano da missão.

5.7 Análise comparativa

A arquitetura desenvolvida se assemelha à arquitetura AuRA na estrutura utilizada para integrar planejamento e reação. Em ambas as arquiteturas, planejamento é realizado antes da execução. O replanejamento também acontece de forma semelhante, iniciado apenas quando a parte inferior da arquitetura, responsável pela execução, não consegue progredir no cumprimento da tarefa, caracterizando uma falha de execução. Na arquitetura AuRA, o planejador é do tipo hierárquico composto de várias camadas: planejador de missão, planejador espacial, e seqüenciador. A arquitetura desenvolvida possui uma versão simplificada desta estrutura feita com o propósito de resolver principalmente o problema de navegação. Entretanto, futuramente um módulo planejador de missão e de seqüenciamento automático de comportamentos poderiam ser integrados de forma hierárquica na arquitetura.

Outra semelhança da AuRA com a arquitetura desenvolvida está em ambas usarem campos potenciais para navegação. Entretanto, existe uma diferença na forma como o campo potencial é usado nas duas arquiteturas, e também em como são coordenados os comportamentos durante a navegação. A arquitetura desenvolvida utiliza uma função de navegação global que define um potencial para todo espaço de configurações livres. Isto se assemelha mais a maneira como o campo potencial para navegação é definido na arquitetura DAMN, conforme descrito por Payton, Rosenblatt e Keirsey (1990). Tanto na arquitetura DAMN como na arquitetura desenvolvida, o campo potencial global de navegação age como uma referência para os demais comportamentos. Na arquitetura AuRA, os campos potenciais gerados pelos esquemas motores utilizam uma abordagem mais local. Além disso, o mecanismo de coordenação de comporta-

mentos para navegação utilizado na arquitetura desenvolvida é diferente do utilizado nas demais arquiteturas. Isto também inclui a forma como a entrada do usuário é tratada na arquitetura.

Neste trabalho, a entrada do usuário possui uma prioridade alta, definida de forma explícita logo abaixo do comportamento de desvio de obstáculo. Então a entrada do usuário sobrescreve a maioria dos comportamentos, com exceção de algumas situações particulares. Um exemplo, é quando não fica claro para o sistema se o usuário pretende ou não abandonar o objetivo de navegação representado pela função global de navegação. Nesta situação particular, a entrada do usuário é levemente modificada para conciliar com o objetivo de navegação. Entretanto, em geral, o sistema dá ao usuário total controle do robô quando é seguro fazê-lo. Interessante observar que este controle ainda é diferente de um controle totalmente manual pois os comportamentos de desvio de obstáculo permanecem ativos a menos que sejam explicitamente desabilitados pelo usuário.

No caso das arquiteturas AuRA, SFX ou DAMN, o coordenador de entrada, por meio da soma vetorial, de filtros, ou soma de votos, considera todos os comportamentos ativos juntamente com a entrada humana. Evidentemente cada arquitetura possui uma forma de priorizar um determinado comportamento atribuindo um peso a ele. Mas dependendo deste peso, os outros comportamentos podem exercer uma certa influência no resultado, e teoricamente, a entrada do usuário é sempre modificada por estes comportamentos.

6 Implementação e resultados

A arquitetura desenvolvida foi implementada em uma cadeira de rodas robótica para possibilitar a realização de tarefas de navegação, permitindo a interação do usuário com a cadeira de rodas em diferentes níveis de controle.

No nível mais alto de controle, o sistema implementado permite ao usuário especificar uma posição de destino para a navegação em um mapa do ambiente. Um plano de movimento é gerado a partir desta entrada do usuário e do mapa do ambiente fornecido *a priori*. Este plano de movimento descreve como a cadeira deve se mover para chegar até a posição de destino selecionada. Este plano de movimento é executado por um comportamento deliberativo que conduz a cadeira de forma autônoma até o seu destino. Enquanto isso, um comportamento reativo de desvio de obstáculos possibilita que a cadeira desvie de obstáculos, que não foram representados no mapa, encontrados ao longo do caminho. Conforme a cadeira se move de forma autônoma, o usuário pode interagir a qualquer momento no nível mais baixo de controle utilizando um *joystick* para alterar o movimento autônomo da cadeira. Assim, o usuário compartilha o controle do sistema autônomo da cadeira.

Neste contexto de aplicação, é fácil imaginar situações onde a cadeira tenha que responder, ao mesmo tempo, aos três tipos de entrada de controle implementadas. Isto é, às entradas de controle provenientes do comportamento deliberativo para mover a cadeira até o objetivo, às entradas de controle provenientes do comportamento reativo de desvio de obstáculos, e às entradas do usuário por meio do *joystick*. Por exemplo, considere a cadeira navegando de forma autônoma em um museu na direção de uma sala de exibição especificada pelo usuário. Para isso ela utiliza o comportamento deliberativo e o plano de movimento gerado automaticamente. Enquanto isso, a cadeira tem que se desviar de outros visitantes no museu utilizando o comportamento reativo de desvio de obstáculos implementado. Ao mesmo tempo que tudo isso acontece, o usuário que vai em rumo à exibição desejada pode querer se aproximar de quadros que chamem sua atenção ao longo do caminho, e para isso modifica a trajetória autônoma do robô em tempo de execução utilizando o *joystick*.

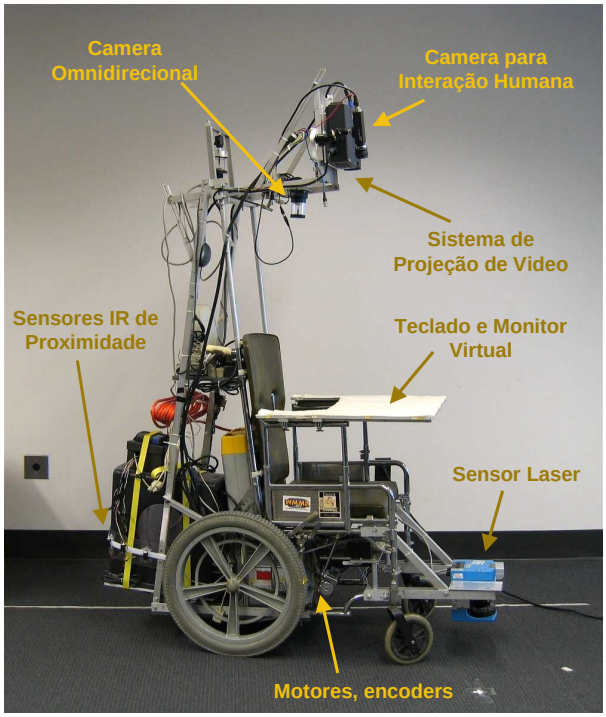

Figura 26: A SMARTCHAIR do laboratório GRASP

No restante do capítulo serão apresentados: na seção 6.1, a cadeira de rodas utilizada como plataforma de pesquisa e o modelo da cadeira utilizado no controle; na seção 6.2, apresenta-se aspectos da implementação da arquitetura desenvolvida, incluindo a plataforma de software utilizada na implementação do sistema de controle; e finalmente na seção 6.3, os resultados obtidos com o sistema implementado.

6.1 A cadeira de rodas SMARTCHAIR

A cadeira de rodas robótica utilizada como plataforma de pesquisa é denominada SMARTCHAIR, e foi desenvolvida no laboratório GRASP da Universidade da Pennsylvania. Esta cadeira de rodas é equipada com um computador embarcado e uma variedade de sensores como pode ser visto na Figura 26.

A câmera omnidirecional montada sobre a cabeça do usuário permite que ele tenha uma visão de 360^o ao redor da cadeira. O sistema de projeção exibe imagens sobre a mesa na frente do usuário, permitindo que o usuário envie comandos à cadeira por meio de uma interface visual. Uma câmera apontada para esta mesa forma juntamente com o projetor um sistema de realimentação visual. A interação do usuário é então efetuada pela oclusão de regiões da imagem projetada na mesa.

Além deste sistema de visão para interface com o usuário, a cadeira de rodas possui um sensor laser montado na frente da cadeira e abaixo dos pés do usuário. Este sensor laser varre uma região de 180^o graus e retorna valores de distância entre o sensor e os obstáculos a cada meio grau. De modo similar, sensores infra-vermelho (IR) de proximidade foram colocados na traseira da cadeira de rodas para detectar qualquer obstáculo atrás. A cadeira de rodas também possui *encoders* nos seus motores permitindo uma estimativa da velocidade e posição da cadeira. A cadeira de rodas SMARTCHAIR é apresentada em maiores detalhes em (RAO et al., 2002).

6.1.1 Modelo do sistema

A SMARTCHAIR pode ser modelada como um robô não-holonômico de duas rodas do tipo *cart*. As equações que governam o seu movimento são bem conhecidas (MA; KOSECKA; SASTRY, 1999):

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{6.1}$$

onde o vetor de entrada $\nu = [v, \omega]^T$, consiste da velocidade linear da cadeira, v, e da velocidade angular da cadeira, ω, enquanto que (x, y) são as coordenadas do centro do eixo das rodas em um sistema de coordenadas inercial. θ é o ângulo que o sistema de coordenadas da cadeira de rodas forma com o sistema de coordenadas inercial. Conforme pode ser visto na Figura 27, características do ambiente (obstáculos, alvos, etc) são descritos por (f_i, z_i) em um sistema fixo ao corpo da cadeira, onde f_i é o ângulo relativo e z_i é a distância relativa à característica i do ambiente.

Neste trabalho, para controle da cadeira de rodas, utilizou-se um espaço de configurações simplificado dado por $q = [x_c, y_c]^T$, que são coordenadas de um ponto de referência na cadeira de rodas localizado a uma distância d à frente do eixo das rodas. Da Equação (6.1), pode-se escrever:

$$\dot{q} = \begin{pmatrix} \dot{x_c} \\ \dot{y_c} \end{pmatrix} = \begin{pmatrix} \cos\theta & -d\sin\theta \\ \sin\theta & d\cos\theta \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{6.2}$$

ou

$$\dot{q} = J\nu. \tag{6.3}$$

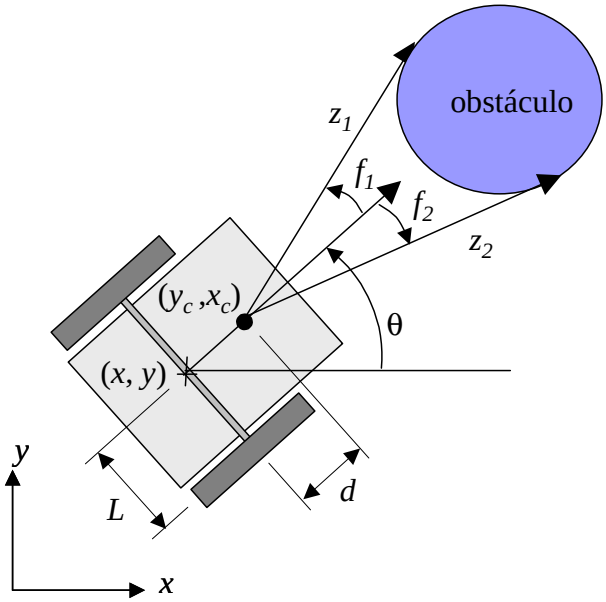

Figura 27: Modelo do sistema. (x, y) é a coordenada do centro do eixo das rodas enquanto que (x_c, y_c) é o ponto que está sendo controlado.

Como J é sempre inversível se d não for zero, pode-se considerar o seguinte modelo linear desacoplado:

$$\dot{q} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} u. \tag{6.4}$$

Assim, qualquer vetor de entrada u pode ser mapeado a um vetor de entrada ν.

No espaço de configurações simplificado que foi utilizado, somente a posição da cadeira de rodas é controlada de forma explícita. Não se tem controle explícito sobre qual orientação a cadeira deve assumir em um dado momento. Portanto, para que se possa planejar trajetórias seguras nesse espaço de configurações, aproximou-se a forma da cadeira por um círculo de raio ρ e, quando necessário, expande-se em ρ os obstáculos no ambiente. Assim, o espaço livre no ambiente representa as posições que o ponto de controle pode assumir sem que ocorra colisão de nenhuma parte da cadeira com os obstáculos. O raio ρ é escolhido levando-se em consideração a dimensão da cadeira mais um fator de segurança devido a restrições não-holonômicas não consideradas no planejamento do movimento da cadeira.

6.2 Aspectos de implementação da arquitetura

A arquitetura proposta foi implementada de maneira simplificada tendo como objetivo principal tornar possível a realização de testes de navegação da cadeira de rodas em que o usuário

possa compartilhar o controle da cadeira com o sistema autônomo.

6.2.1 Mapeamento e localização

Na implementação da arquitetura, utilizou-se um mapa do ambiente fornecido *a priori*. Este mapa é uma grade de ocupação que representa a posição de obstáculos conhecidos no ambiente, tais como mesas e paredes. Os obstáculos neste mapa de ocupação foram expandidos conforme discutido na seção anterior para que a cadeira de rodas pudesse ser considerada como um ponto neste mapa. A Figura 28 mostra o mapa do ambiente em que foram realizados os experimentos. O mapa expandido visto na Figura 28 é usado para calcular a função potencial para a navegação.

Não foram implementados métodos de mapeamento e de atualização do mapa. No entanto, um método de mapeamento e atualização de mapas baseado em grades de probabilidade de ocupação seria compatível com esta arquitetura.

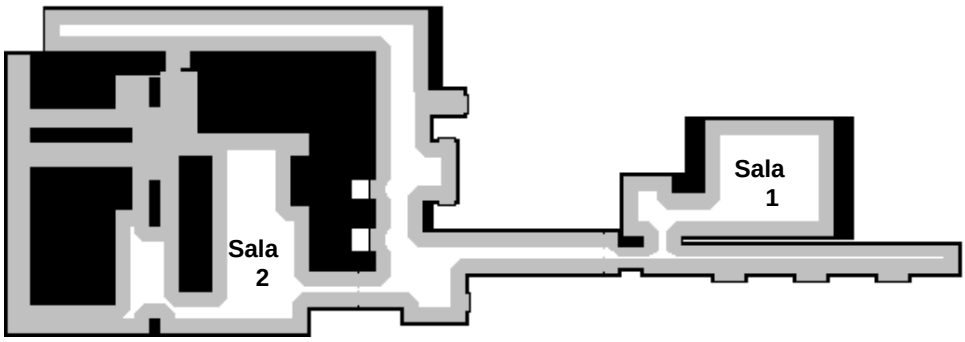

Figura 28: Mapa do ambiente. As regiões escuras são mesas e outros objetos conhecidos no ambiente. A região mais clara é a expansão do mapa, que se torna uma fronteira de segurança que leva em consideração o tamanho da cadeira de rodas. A região branca é o espaço de configurações livres no ambiente.

A localização global da cadeira de rodas durante os experimentos realizados foi feita utilizando apenas a odometria da cadeira de rodas. De forma geral, a odometria de um robô é insuficiente para produzir uma boa estimativa de sua posição global devido ao acumulo de erro quando o robô percorre longos percursos. Por isso, geralmente utilizam-se métodos que combinam a odometria com informações externas do ambiente para reduzir estes erros e obter uma melhor estimativa da posição. Geralmente são utilizados filtros, tais como filtro de partículas (FOX et al., 2001) ou filtro de Kalman (SMITH; SELF; CHEESEMAN, 1990). Neste trabalho, nenhum destes métodos robustos de localização para longos percursos foi implementado apesar de previstos na arquitetura. Nos experimentos realizados, a distância máxima percorrida foi de aproximadamente 10 metros. Para esta distância observou-se que os erros de localização na odometria causados pelo deslizamento das rodas e das correias não foram tão significativos

(PARIKH et al., 2003).

6.2.2 Planejamento de movimento

O planejador de movimento na arquitetura proposta é responsável por gerar uma função global de navegação usada para conduzir o robô até a posição de destino selecionada pelo usuário (objetivo de navegação)(LATOMBE, 1991; RIMON; KODITSCHEK, 1992). Entretanto, é difícil computar esta função instantaneamente. Então, ao invés, calculou-se uma função de navegação aproximada utilizando programação dinâmica no mapa de grade de ocupação, conforme método de *wave-front* apresentado na seção 4.4.3. Como esta função aproximada não satisfaz todas as propriedades de uma função de navegação, ela é chamada aqui de função potencial, representada por $\phi(q)$. No entanto, ela possui um mínimo global único localizado no objetivo de navegação. Em contraste com outros trabalhos onde controladores baseados em campos potenciais são considerados reativos, neste trabalho considera-se o controlador resultante da utilização da função potencial como sendo um comportamento deliberativo pois a função potencial é construída utilizando o conhecimento global do ambiente na forma de um mapa do ambiente.

A Figura 29 mostra um exemplo de uma função de potencial de navegação gerada automaticamente para uma posição de destino especificada na sala 2, nas coordenadas $x = 15.0m$ e $y = 2.9m$ do mapa. O mínimo global desta função potencial localiza-se na posição de destino especificada.

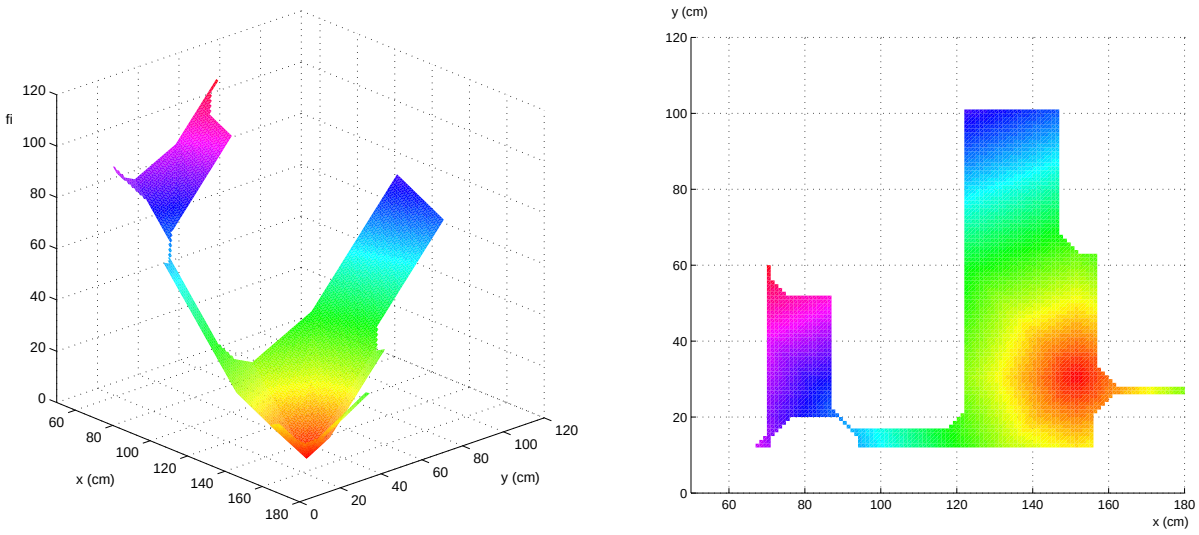

Figura 29: Função de navegação para o objetivo especificado pelo usuário na Sala 2.

A função potencial apresentada na Figura 29 foi definida no espaço livre do mapa de ocupação. Para o espaço ocupado do mapa, incluindo o espaço ocupado após expansão dos obs-

táculos, definiu-se um campo vetorial de repulsão. Este campo vetorial garante que, se o robô começar a se aproximar da área ocupada, ele será conduzido de volta à area livre.

6.2.3 Coordenação de comportamentos e entrada do usuário

Foram implementados dois comportamentos que quando coordenados são responsáveis pela navegação autônoma do robô: um comportamento deliberativo de navegação e um comportamento reativo de desvio de obstáculos.

O comportamento deliberativo de navegação move o robô em direção ao objetivo de navegação especificado pelo usuário. Este comportamento pode ser descrito pelo controlador de malha fechada $u_\phi = -\nabla\phi(q)$. Portanto, a cada intervalo de tempo, calcula-se o gradiente negativo da função potencial de navegação para a posição atual do robô. O gradiente negativo indica a direção que o robô deve seguir para que alcance o objetivo de navegação. Esta direção na verdade define um semi-espaço de soluções possíveis que podem ser usadas para aproximar o robô do seu objetivo.

O comportamento reativo de desvio de obstáculo foi implementado utilizando o sensor de varredura laser para obter a distância entre o robô e os obstáculos no ambiente. A cada varredura do sensor, determina-se qual a direção e a distância do obstáculo mais próximo. Quando um obstáculo se encontra a uma distância menor ou igual a um valor pré-definido, que foi $0,7m$ nesta implementação, o comportamento reativo é ativado. O comportamento de desvio de obstáculos é definido como sendo uma restrição às entradas de controle que podem ser enviadas ao robô. Assim, somente as entradas que não infringem esta restrição, ou seja, que não conduzem o robô para mais perto do obstáculo, podem ser escolhidas. Esta restrição é representada pelo semi-espaço definido pela direção do obstáculo que deve ser evitado. Portanto, quando o comportamento está ativo, esta direção é enviada ao coordenador de entradas.

Existem situações em que um obstáculo aparece abruptamente na frente da cadeira de rodas a uma distância tão próxima que a única alternativa para evitar uma colisão é parar a cadeira. Pensando nestas situações, definiu-se uma condição de parada de emergência da cadeira quando um obstáculo é detectado a uma distância menor que um valor pré-definido. Este valor deve ser menor que a distância de ativação do comportamento reativo, e foi definido como sendo $0,4m$ nesta implementação.

Além destes comportamentos utilizados para navegação autônoma, o sistema implementado também permite que o usuário utilize um *joystick* para entrada de comandos. Por meio deste *joystick*, o usuário indica a direção e velocidade para a qual deseja que a cadeira se mova em

relação a sua posição atual.

As entradas de controle geradas pelos comportamentos implementados e a entrada de controle fornecida pelo usuário por meio do *joystick* são coordenadas utilizando o método apresentado na seção 5.4 do capítulo 5. Este método foi implementado de acordo com o algoritmo apresentado na Figura 30.

```
ū := 0
if posição de destino foi especificada no mapa then
  u_φ := −∇φ(q)
  if existe uma entrada do usuário, u_h then
    φ := ângulo entre o vetor u_h e o vetor u_φ
    if φ > π/2 e φ < φ_limit then
      ū := u_m
    else
      ū := u_h
    end if
  else
    ū := u_φ
  end if
else if existe uma entrada do usuário, u_h then
  ū := u_h
end if
if obstáculo está próximo do robô e ū ≠ 0 then
  u_g := −∇g(q)
  γ := ângulo entre o vetor ū e o vetor u_g
  if γ > π/2 e γ < γ_limit then
    ū := u_m
  else if γ > γ_limit then
    ū := 0
  end if
end if
Usar ū como entrada de controle para o robô
```

Figura 30: Algoritmo usado para combinar o comportamento deliberativo de navegação, comportamento reativo de desvio de obstáculos, e entradas do operador humano.

6.2.4 Plataforma de software utilizada na implementação

A arquitetura foi implementada para a SMARTCHAIR utilizando uma plataforma de software chamada ROCI (Remote Objects Control Interface) (CHAIMOWICZ et al., 2003). Esta plataforma foi desenvolvida no laboratório GRASP, possui licença de código aberto, e pode ser obtida gratuitamente na Internet (ROCI, 2006).

ROCI oferece ao desenvolvedor ferramentas que permitem a criação e execução de um sistema modular, distribuído, cujos componentes possam ser reutilizados em diferentes aplicações. Um sistema desenvolvido utilizando ROCI é composto de módulos que possuem funções específicas programadas pelo desenvolvedor. Estes módulos possuem entradas e saídas por meio das quais os módulos são conectados uns aos outros formando uma rede distribuída de proces-

sos. Assim, cada módulo recebe dados como entrada, processa estes dados de acordo com a função programada e gera dados de saída. Quando um módulo é criado, o programador deve especificar o tipo de dado ou informação que cada entrada e saída utilizará. Assim, a saída de um módulo só pode ser conectada a entrada de outro se o tipo de dado desta entrada e saída forem o mesmo.

De certa forma, cada módulo ROCI pode ser comparado a um circuito integrado (CI) responsável por algum tipo de processamento. Conectando-se estes CIs, ou módulos ROCI, entre si de forma adequada, pode-se realizar processamentos complexos. Além do mais, assim como um CI pode ser reutilizado para diferentes aplicações, um módulo ROCI também pode ser reutilizado sem que haja necessidade de programação adicional. Cada módulo após desenvolvido e compilado, fica disponível na forma de uma DLL para o sistema operacional Windows da Microsoft.

Na plataforma ROCI, uma tarefa é especificada por uma coleção de módulos e pela maneira como estes módulos são conectados entre si. Utiliza-se um ou mais arquivos no formato XML para definir uma tarefa. Assim, a construção de aplicações quando já se possui os módulos desenvolvidos se resume apenas em escrever arquivos especificando tarefas em XML, que podem ser facilmente criadas e modificadas.

A plataforma ROCI possui um kernel responsável pelo controle principal de uma aplicação ROCI. O kernel do ROCI deve ser executado em cada entidade na rede (computadores, robôs, sensores remotos, etc) que fará parte da aplicação. Estas entidade são consideradas como nós da rede ROCI. O kernel gerencia a rede e mantém uma base de dados atualizada de todos os nós e serviços disponíveis na rede ROCI. Ele é responsável também por carregar e remover tarefas (descritas em arquivos XML) e módulos em qualquer nó da rede, fazendo e desfazendo conexões entre módulos. Assim, aplicações podem ser especificadas e executadas dinamicamente.

ROCI usa recursos disponíveis na plataforma de desenvolvimento Microsoft .NET. Módulos ROCI podem ser desenvolvidos utilizando as linguagens de programação C# e C++. O kernel e os módulos ROCI podem ser executadas em dispositivos que possuam versões do sistema operacional Windows (incluindo dispositivos portáteis, tais como Pocket PCs).

Como conseqüência de se utilizar a plataforma ROCI na implementação da arquitetura para a SMARTCHAIR, obteve-se um software extremamente modular de controle. Módulos deste software podem ser reutilizados em outras aplicações e até mesmo para controlar outros robôs. Além disso estes módulos podem ser executados de forma distribuída, ou seja, em computadores diferentes conectados na mesma rede.

6.3 Resultados

O sistema de controle desenvolvido permite que a cadeira de rodas seja operada em três modos distintos, cada um correspondente a um nível de autonomia do robô:

- *Controle autônomo*: o usuário seleciona no mapa uma posição de destino e a cadeira dirige de forma autônoma até a posição selecionada;
- *Controle manual*: o usuário controla manualmente a cadeira a e rodas utilizando um *joystick*;
- *Controle semi-autônomo*: o usuário compartilha o controle de movimento da cadeira enquanto a cadeira se move de forma autônoma.

Foram feitos testes com a cadeira de rodas operando nestes três modos em duas salas diferentes. Ao total, foram 43 pessoas que testaram a cadeira de rodas. Cada uma testou a cadeira nos três modos, nas duas salas. A distância média percorrida pela cadeira de rodas em cada experimento, incluindo modo manual, autônomo, e semi-autônomo, foi 9,77 metros. Ou seja, no total foram realizados 258 experimentos, sendo que a cadeira percorreu aproximadamente 2,5 km ao final de todos estes experimentos. Entretanto, no início de cada experimento a localização global da cadeira era corrigida manualmente, sendo que os erros de localização acumulados em um experimento não eram transferidos para outro. Durante cada teste foram coletados os dados de odometria da cadeira, o número de interações do usuário por meio do *joystick*, e informações referentes às entradas de controle (deliberativa, reativa, e usuário humano) ativas a cada instante. Nesta seção são mostrados alguns dos resultados mais significativos.

6.3.1 Modo de operação autônoma

No modo de operação autônoma, o usuário da cadeira seleciona em um mapa uma posição de destino para a qual a cadeira deve se mover. Para cada uma das salas, uma posição de destino foi escolhida e, após a seleção, um plano deliberativo de movimento na forma de uma função potencial de navegação é criado utilizando o mapa. Um comportamento deliberativo de navegação faz com que a cadeira siga o gradiente negativo da função potencial de navegação para se mover até o destino selecionado. Entretanto, quando um obstáculo não modelado no mapa é encontrado no caminho, o comportamento reativo de desvio de obstáculo é ativado. Então uma entrada de controle que combina o comportamento deliberativo de navegação com o comportamento reativo de desvio de obstáculo é produzida para enviar aos atuadores da cadeira. A

coordenação é feita conforme método apresentado no capítulo 5 e implementado pelo algoritmo apresentado na Figura 30 da seção 6.2. Vale ressaltar que, no modo de operação autônoma, o usuário não interage com a cadeira enviando comandos contínuos por meio do *joystick*.

A Figura 31 mostra uma trajetória realizada pela cadeira no modo de operação autônoma. O obstáculo indicado na figura representa um obstáculo que não havia sido modelado no mapa utilizado para gerar o plano de movimento. Este obstáculo foi encontrado pela cadeira ao longo do caminho, e o desvio foi feito de forma reativa. A trajetória em linha tracejada azul é o caminho que a cadeira realiza quando este obstáculo não está presente. No lado direito da Figura 31 encontra-se um *zoom* de um trecho da trajetória. As flechas representam a direção do gradiente negativo da função potencial. Como pode ser visto, a cadeira é guiada pelo gradiente negativo enquanto o comportamento reativo de desvio de obstáculo não é ativado (trecho em preto). Quando a cadeira se encontra próxima ao obstáculo não representado no mapa, o comportamento de desvio de obstáculo é ativado (trecho em verde). Então, é escolhida uma direção de movimento que faça com que o robô desvie do obstáculo ao mesmo tempo que se aproxima do objetivo de navegação.

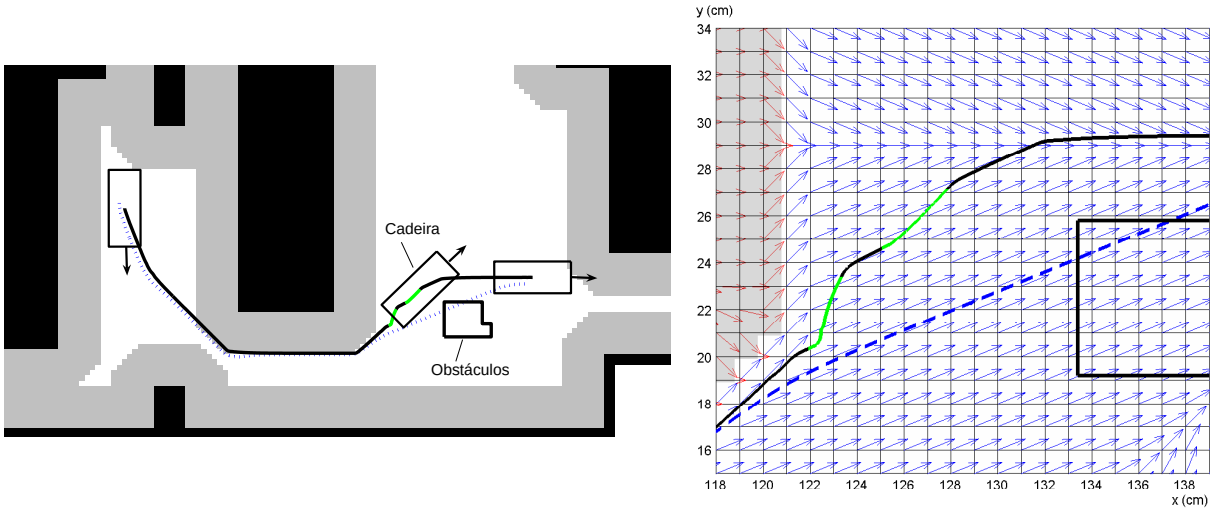

Figura 31: Exemplo de trajetória descrita pela cadeira de rodas na Sala 2 para o modo autônomo. A trajetória é resultado da combinação do comportamento deliberativo de navegação com o comportamento reativo de desvio de obstáculos. A linha sólida (preta) mostra quando apenas o comportamento deliberativo está ativo. A linha mais clara (verde) mostra quando o comportamento de desvio de obstáculos foi composto com o comportamento deliberativo. A linha tracejada (azul) é a trajetória que a cadeira descreve quando não existem obstáculos.

A Figura 32 mostra, sobrepostas, 10 das 43 trajetórias realizadas pela cadeira de rodas no modo autônomo. Estas 10 trajetórias ilustram o resultado obtido nos demais experimentos no modo autônomo para cada uma das salas. É possível perceber que o sistema autônomo possui repetibilidade movendo a cadeira da mesma forma cada uma das vezes ao longo do caminho

mais curto até o objetivo.

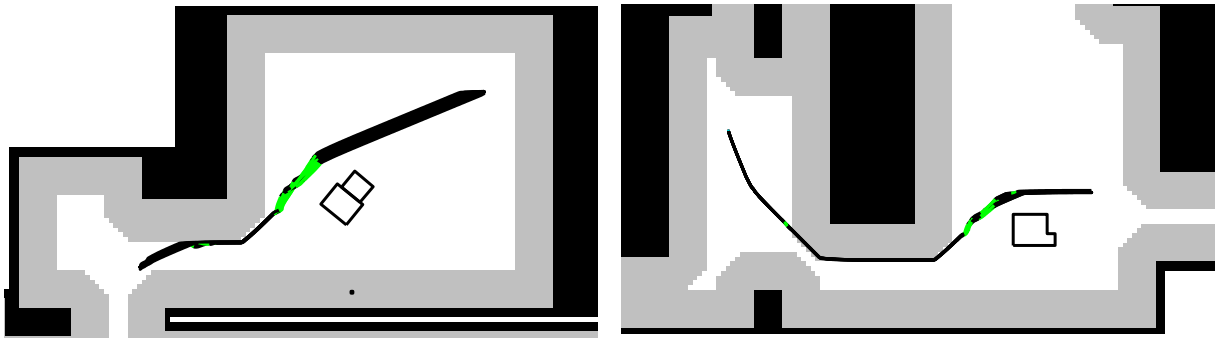

Figura 32: Dez exemplos de trajetórias descritas pela cadeira de rodas durante o modo autônomo. Dois ambientes diferentes são mostrados. Os trechos representados com linhas sólidas (preto) mostram quando apenas o comportamento deliberativo de navegação está ativo, e as linhas mais claras (verdes) mostram quando ocorreu ativação do comportamento de desvio de obstáculos.

6.3.2 Modo de operação manual

No modo de operação manual, o usuário dirige a cadeira apenas usando o *joystick*. Mesmo neste modo, o comportamento reativo de desvio de obstáculos é ativado quando a cadeira se aproxima de obstáculos no ambiente. Então as entradas do usuário são combinadas com este comportamento para evitar colisões ao mesmo tempo que a cadeira se move na direção sugerida pelo usuário por meio do *joystick*. A Figura 33 ilustra uma trajetória realizada por um usuário na Sala 1.

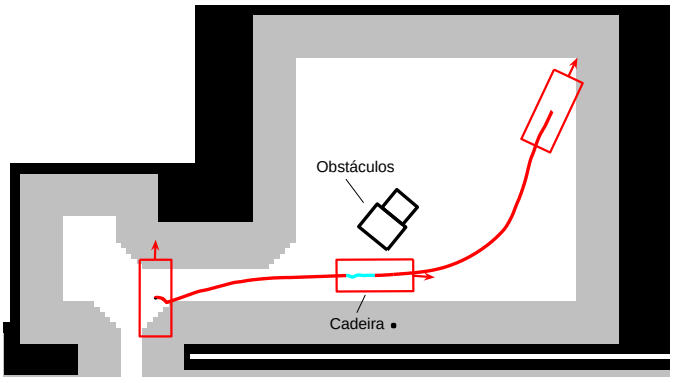

Figura 33: Exemplo de trajetória (vermelha) descrita pelo usuário na Sala 1 utilizando o modo manual. Os segmentos de linha mais claros (azul) representam quando o comportamento de desvio de obstáculos esteve ativo e a entrada humana teve de ser modificada.

6.3.3 Modo de operação semi-autônoma

O modo de operação semi-autônoma funciona da mesma forma que o modo de operação autônoma com a exceção de que o usuário pode, a qualquer momento que desejar, entrar com comandos contínuos por meio de um *joystick*. Dessa forma, o usuário compartilha o controle da cadeira de rodas com o sistema de navegação autônoma. Ou seja, neste modo podem ser coordenados ao mesmo tempo: o comportamento deliberativo de navegação, o comportamento reativo de desvio de obstáculos, e entradas de controle contínuas provenientes do usuário.

A Figura 34 mostra uma trajetória realizada por um usuário em uma das salas no modo semi-autônomo. Nesta figura, é indicado quando os diferentes comportamentos foram usados e quando o usuário compartilhou o controle com o sistema. É interessante observar que a posição de destino selecionada para movimento autônomo foi a mesma utilizada no modo de operação autônoma. O obstáculo não modelado no mapa também permaneceu para cada uma das salas na mesma posição dos demais experimentos. Por essa razão, caso o usuário não interferisse no movimento autônomo da cadeira, a trajetória seria semelhante às trajetórias apresentadas na Figura 32. Mas observa-se que devido a interferência do usuário, a trajetória da cadeira foi diferente. A cadeira passa pelo obstáculo não modelado por um lado diferente ao da trajetória apresentada na Figura 31. À direita da Figura 34, mostra-se o detalhe de um trecho da trajetória realizada com o gradiente negativo da função potencial de navegação sobreposto.

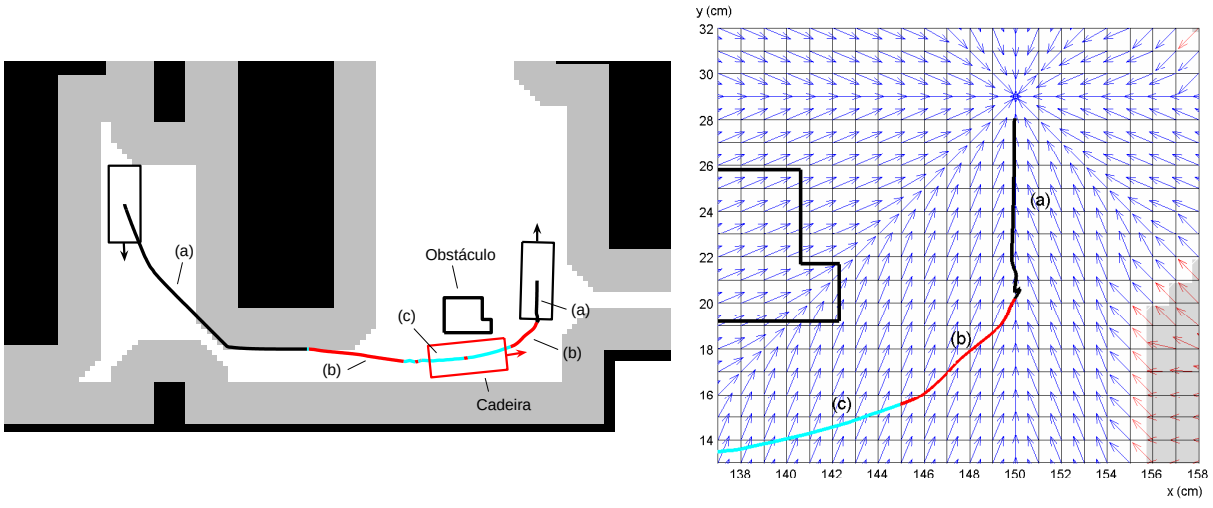

Figura 34: Exemplo de trajetória no modo semi-autônomo. Parte(a) representa a trajetória descrita quando apenas o comportamento deliberativo de navegação esteve ativo, parte(b) representa quando ocorreu entrada do usuário consistente com o plano de movimento, parte(c) representa quando o comportamento de desvio de obstáculos foi ativado.

Pode-se notar que no modo de operação semi-autônoma ocorre uma integração coerente do comportamento deliberativo de navegação com o comportamento reativo de desvio de obstáculo

e como as entradas humanas.

Nos testes realizados observou-se que o número de interações entre o homem e o robô, medido pela quantidade de vezes que o usuário movia o *joystick*, foi significativamente menor que o número de interações no modo manual.

Também foi observado que em alguns testes ocorreram algumas colisões da cadeira com o obstáculo devido à falta de sensores na lateral da cadeira de rodas. Estas colisões ocorreram quando o usuário deixou de usar o *joystick* no momento em que a cadeira estava próxima ao obstáculo mas este não estava sendo "visto" pelo sensor laser colocado na frente da cadeira. Nesta situação, o comportamento de desvio de obstáculos não estava ativo. Então o comportamento deliberativo de navegação levou a cadeira a fazer uma virada rápida para que esta se alinhasse ao gradiente negativo da função de navegação. Isso fez com que a cadeira batesse no obstáculo não modelado no mapa que naquele instante se encontrava no ponto cego dos sensores.

Uma forma de resolver este problema seria equipar a cadeira com sensores laterais e traseiros. Dessa forma o comportamento reativo de desvio de obstáculos poderia lidar melhor com objetos próximos à cadeira, pois obstáculos ao lado da cadeira não passariam desapercebidos pelos sensores. Outra forma de se evitar colisões laterais seria usar uma memória de curto prazo que guardasse a última posição onde o obstáculo foi visto. Usando esta memória de curto prazo sobre o obstáculo, a cadeira poderia continuar evitando a colisão caso o obstáculo permanecesse próximo a cadeira, mesmo que o obstáculo não esteja mais sendo visto pelo sensor laser frontal.

Apesar das colisões ocorridas devido à falta de sensores nas laterais da cadeira, de modo geral obteve-se bons resultados com o sistema de controle desenvolvido. O usuário pode efetivamente alterar parte da trajetória da cadeira de rodas utilizando o *joystick*.

6.3.4 Desempenho do sistema

O sistema de controle implementado, que utiliza a plataforma de programação ROCI, foi executado inteiramente em um computador embarcado na SMARTCHAIR. Este computador embarcado possui processador Pentium 4 de 2.4GHz e sistema operacional Windows 2000. Durante os experimentos realizados, não foi observada nenhuma falha do sistema operacional ou da plataforma ROCI.

O sensor de varredura laser de marca SICK utilizado para detecção de obstáculos é lido por meio de canal serial. Cada leitura do sensor transmite 360 valores correspondentes à distância entre o obstáculo e o sensor a cada 0.5 grau, totalizando 180 graus de varredura. Durante a execução do sistema de controle, observou-se que a freqüência de leitura do sensor laser foi de

aproximadamente 5Hz.

A posição da cadeira no ambiente foi determinada pela odometria baseada na leitura de *encoders* instalados nas duas rodas motoras da cadeira. Na execução do sistema, observou-se que a posição da cadeira pode ser lida a uma freqüência máxima de aproximadamente 9Hz. Este mesmo valor foi observado para a freqüência máxima com que comandos de referência de velocidade podem ser enviados para a cadeira.

A velocidade da cadeira de rodas deve ser limitada em um valor que possibilite ao sistema responder a tempo à detecção de um obstáculo, podendo parar ou desviar o movimento da cadeira antes que ocorra a colisão. Este valor de velocidade está relacionado com a freqüência com que o sistema consegue ler o sensor usado para percepção de obstáculos, com o tempo de processamento necessário para se calcular uma resposta a este estímulo, e com a freqüência com que o sistema envia comandos de velocidade para o robô. Assim, ao limitar a velocidade da cadeira de rodas, pode-se evitar que, estando a cadeira em movimento, ocorra uma colisão entre o instante em que acontece a detecção de um obstáculo e o instante em que um comando é enviado em resposta.

Portanto, por medida de segurança, durante os experimentos realizados, limitou-se a velocidade máxima de translação da cadeira de rodas em $0.15m/s$. Este valor foi obtido a partir de testes preliminares com o sistema. Durante os experimentos realizados, para o modo autônomo em uma das salas, por exemplo, a cadeira percorreu em média um percurso de $8.64m$ em $68.5s$. Ou seja, uma velocidade média de $0.13m/s$. Vale observar que quando o comportamento de desvio de obstáculo era ativado automaticamente quando um obstáculo se encontrava próximo ao robô, a velocidade da cadeira era diminuída para metade da velocidade máxima. Experimentalmente verificou-se que para os critérios utilizados para ativação do comportamento de desvio de obstáculos usados neste trabalho, esta redução na velocidade foi necessária para que a cadeira de rodas pudesse desviar dos obstáculos de forma segura, evitando colisões com os mesmos.

7 Conclusão

Neste trabalho apresentou-se uma arquitetura híbrida (deliberativa/reativa) para robô móvel, desenvolvida para que um usuário humano pudesse interagir com o robô em diferentes níveis da arquitetura. Em um nível superior da arquitetura, o usuário interage no papel de supervisor e define objetivos de navegação para o robô. No nível inferior, o usuário pode compartilhar o controle do robô com o seu sistema de controle autônomo em tarefas de navegação.

A arquitetura utiliza uma função de navegação em sua parte deliberativa para descrever o plano de movimento que o robô deve executar para atingir o objetivo de navegação definido pelo usuário humano. Esta função de navegação é calculada a partir de um mapa global do ambiente fornecido *a priori* e atualizado por um módulo de mapeamento. A função de navegação conduz o robô até o objetivo ao mesmo tempo que desvia de obstáculos que estavam modelados no mapa no momento do planejamento. Para lidar com situações imprevistas e dinâmicas no ambiente, são utilizados comportamentos reativos de navegação. Em particular, o comportamento de desvio reativo de obstáculo evita colisões do robô com obstáculos que não estavam modelados no mapa no momento do cálculo da função de navegação. Além disso, entradas de controle fornecidas pelo usuário humano são combinadas com os comportamentos reativos e o plano de movimento. Isso é feito dentro da arquitetura por um módulo de coordenação de entradas de controle. Este módulo utiliza um conjunto de regras desenvolvido que prioriza a entrada do usuário em relação ao plano de movimento autônomo do robô enquanto preserva a integridade física do sistema. Em situações particulares, a entrada de controle fornecida pelo usuário é modificada para que ela possa permanecer de acordo com o objetivo de navegação. O sistema de desvio reativo de obstáculos permanece ativo durante a navegação. Assim, mesmo quando a entrada do usuário tenta levar o robô em direção a um obstáculo, o sistema evita colisões.

A arquitetura desenvolvida e o mecanismo de coordenação de entradas de controle são as principais contribuições deste trabalho.

Esta arquitetura foi implementada em uma cadeira de rodas robótica chamada SMART-CHAIR que foi desenvolvida no laboratório GRASP da Universidade da Pensilvânia. Para sim-

plificar a implementação da arquitetura, utilizou-se a odometria do robô para localização e um mapa do ambiente fornecido *a priori*, que não é atualizado durante a navegação. Apesar disso, a arquitetura prevê a utilização de módulos de mapeamento e localização mais elaborados. Na arquitetura desenvolvida, o mapa deve ser representado de forma geométrica para que a função de navegação possa ser definida sobre este mapa. Além disso, a localização do robô deve ser feita de forma métrica de acordo com a resolução do mapa. O módulo monitor de progresso e o replanejamento apesar de previstos na arquitetura também não foram implementados.

Neste trabalho, além do comportamento deliberativo que executa o plano de movimento, foi implementado o comportamento reativo de desvio de obstáculo. Apesar deste ter sido o único comportamento reativo usado durante a navegação, outros comportamentos reativos definidos como restrições na forma de inequações poderiam ser utilizados.

Utilizando o sistema de controle implementado foram feitos experimentos com a cadeira de rodas operando em três modos de autonomia diferente: autônomo, semi-autônomo e manual. Os resultados mostram que qualitativamente o usuário pode compartilhar de forma efetiva o controle do robô.

Futuramente módulos podem ser adicionados na arquitetura para que esta se torne mais adequada a tarefas mais abrangentes que navegação. Neste sentido, podem ser utilizados representações mais complexas do ambiente e do mundo, planejamento de missões além do planejamento de movimento, seqüenciador automático de comportamentos, e comportamentos reativos para tarefas específicas que o robô precise desempenhar.

Além disso, como trabalhos futuros podem também ser estudadas maneiras de melhorar a navegação, por exemplo, permitindo o controle explícito da orientação do robô durante a navegação, e explorar outras formas de interação do usuário com o robô, por exemplo, interação remota com o robô. Conforme previstos na arquitetura, métodos de mapeamento, localização e replanejamento também podem ser futuramente implementados.

Referências

ALAMI, R.; CHATILA, R.; FLEURY, S.; GHALLAB, M.; INGRAND, F. An architecture for autonomy. *International Journal of Robotics Research*, v. 17, n. 4, p. 315–337, April 1998. Disponível em: <ftp://ftp.laas.fr/pub/ria/malik/articles/ijrr97.pdf>. Acesso em: 10/02/2006.

ALBUS, J. Theory of intelligent systems. In: *Fifth IEEE Symposium on Intelligent Control*. Philadelphia, PA: [s.n.], 1990. p. 866–875.

ALBUS, J.; MCCAIN, H.; LUMIA, R. *NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*. Gaithersburg, MD, 1989.

ALBUS, J.; PROCTOR, F. A reference model architecture for intelligent hybrid control systems. In: *Proceedings of the 1996 Triennial World Congress, International Federation of Automatic Control (IFAC)*. San Francisco, CA: [s.n.], 1996. Disponível em: <http://www.isd.cme.nist.gov/documents/albus/ifac13.pdf>. Acesso em: 30/03/2006.

ALI, K. S.; ARKIN, R. C. Multiagent teleautonomous behavioral control. *Machine Intelligence and Robotic Control*, v. 1, n. 2, p. 3–10, 2000.

ARBIB, M. A. Schema theory. In: SAPHIRO, S. (Ed.). *The Encyclopedia of Artificial Intelligence*. 2. ed. New York, NY: Wiley-Interscience, 1992. p. 1427–1443.

ARKIN, R. C. Path planning for a vision-based autonomous robot. In: *Proceedings of the SPIE Conference on Mobile Robots*. Cambridge, MA: [s.n.], 1986. p. 240–250.

ARKIN, R. C. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In: *Proc. IEEE Int. Conf. Robotics and Automation*. [S.l.: s.n.], 1987. v. 4, p. 264 – 271.

ARKIN, R. C. *Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments*. Tese — University of Massachusetts, Department of Computer and Information Science, 1987.

ARKIN, R. C. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, v. 8, n. 4, p. 92–112, 1989.

ARKIN, R. C. Towards the unification of navigational planning and reactive control. In: *Working notes of the AAAI 1989 Spring Symposium on Robot Navigation*. Stanford University: [s.n.], 1989. Disponível em: <http://www.cc.gatech.edu/ai/robot-lab/online-publications/stanford.pdf>. Acesso em: 30/03/2006.

ARKIN, R. C. The impact of cybernetics on the design of a mobile robot system: A case study. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 20, n. 6, p. 1245–1257, November/December 1990.

ARKIN, R. C. *Behavior-based robotics*. Cambridge,Massachusets: MIT Press, 1998.

ARKIN, R. C.; BALCH, T. AuRA: principles and practice in review. In: *Journal of Experimental and Theoretical Artificial Intelligence*. Great Britain: Taylor & Francis LTD, 1997. v. 9, n. 2-3, p. 175–189. Disponível em: <http://www.cc.gatech.edu/ai/robot-lab/online-publications/jetai-final.pdf>. Acesso em: 30/03/2006.

BARBERA, A.; FITZGERALD, M.; ALBUS, J.; HAYNES, L. RCS: The NBS realtime control system. In: *Proceedings of the Robots 8 Conference*. Detroit, MI: [s.n.], 1984. p. 19.1–19.38. Disponível em: <http://www.isd.mel.nist.gov/documents/barbera/Loc_26.pdf>. Acesso em: 30/03/2006.

BARRAQUAND, J.; LANGLOIS, B.; LATOMBE, J. Numerical potential field techniques for robot path planning. *IEEE Transactions on Man and Cybernetics*, v. 22, n. 2, p. 224–241, 1992.

BARRAQUAND, J.; LATOMBE, J. C. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, v. 10, n. 6, p. 628–649, December 1991.

BONASSO, R. P. Integrating reaction plans and layered competences through synchronous control. In: *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*. San Francisco: Morgan Kaufmann Publishers, 1991. p. 1225–1231.

BONASSO, R. P. Using parallel program specifications for reactive control of underwater vehicles. *Journal of Applied Intelligence*, v. 2, n. 2, p. 201–223, 1992.

BONASSO, R. P.; DEAN, T. L. Robots with AI: A retrospective on the AAAI robot competitions and exhibitions. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press, 1996.

BONASSO, R. P.; FIRBY, R. J.; GAT, E.; KORTENKAMP, D.; MILLER, D.; SLACK, M. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, v. 9, n. 2, p. 237–256, 1997. Disponível em: <http://www.traclabs.com/~korten/publications/jetai.pdf>. Acesso em: 31/03/2006.

BROOKS, R. A. A robust layered control system for a mobile robot. In: *IEEE Journal of Robotics and Automation*. [S.l.: s.n.], 1986. v. 2, n. 1, p. 14–23.

BROOKS, R. A. *The Behavior Language User's Guide*. Cambridge, MA, 1989. Disponível em: <http://www.ece.pdx.edu/~mperkows/ML_LAB/Giant_Hexapod/brooks1227.pdf>. Acesso em: 31/03/2006.

BROOKS, R. A. Intelligence without representation. *Artificial Intelligence*, v. 47, n. 1-3, p. 139–159, January 1991. Disponível em: <http://people.csail.mit.edu/brooks/papers/representation.pdf>. Acesso em: 31/03/2006.

BURRIDGE, R. R.; RIZZI, A. A.; KODITSCHEK, D. E. Sequential composition of dynamically dexterous robot behaviors. In: *The International Journal of Robotics Research*. Cambridge, MA: MIT Press, 1999. v. 18, n. 6, p. 534–555. Disponível em: <http://www.robots.ox.ac.uk/~jennet/ijrr/multimedia-demo/article.pdf>. Acesso em: 31/03/2006.

CANNY, J. F. *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

CHAIMOWICZ, L.; COWLEY, A.; SABELLA, V.; TAYLOR, C. J. ROCI: A distributed framework for multi-robot perception and control. In: *Proceedings of the 2003 IEEE/RJS International Conference on Intelligent Robots and Systems*. Las Vegas, NV, USA: [s.n.], 2003. p. 266–271. Disponível em: <http://repository.upenn.edu/cis_papers/42/>. Acesso em: 31/03/2006.

CHOSET, H.; BURDICK, J. Sensor based exploration: the hierarchical generalized voronoi graph. In: *International Journal of Robotics Research*. Cambridge, MA: MIT Press, 2000. v. 19, n. 2, p. 96–125. Disponível em: <http://www.ri.cmu.edu/pubs/pub_4437.html>. Acesso em: 31/03/2006.

CHOSET, H.; LYNCH, K. M.; HUTCHINSON, S.; KANTOR, G.; BURGARD, W.; KAVRAKI, L. E.; THRUN, S. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Boston: MIT Press, 2005.

CONGDON, C. B.; HUBER, M.; KORTENKAMP, D.; BIDLACK, C.; COHEN, C.; HUFFMAN, S.; KOSS, F.; RASCHKE, U.; WEYMOUTH, T. CARMEL vs. Flakey: A comparison of two robots. *AI Magazine*, v. 14, n. 1, p. 43–55, 1993. Disponível em: <http://citeseer.ist.psu.edu/82836.html>. Acesso em: 10/02/2006.

CONNELL, J. H. *A Colony Architecture for an Artificial Creature*. MIT AI Laboratory, Cambridge, MA, August 1989.

CONNELL, J. H. SSS: A hybrid architecture applied to robot navigation. In: *Proceedings of the 1992 IEEE Int. Conf. on Robotics and Automation (ICRA)*. Nice, France: [s.n.], 1992. p. 2719–2724. Disponível em: <http://www.research.ibm.com/people/j/jhc/pubs/jhc-sss.pdf>. Acesso em: 31/03/2006.

CONNER, D. C.; RIZZI, A. A.; CHOSET, H. Composition of local potential functions for global robot control and navigation. In: *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, Nevada: [s.n.], 2003. p. 3546–3551. Disponível em: <http://www.ri.cmu.edu/pubs/pub_4556.html>. Acesso em: 31/03/2006.

DISSANAYAKE, M. W. M. G.; NEWMAN, P.; CLARK, S.; DURRANT-WHYTE, H. F.; CSORBA, M. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, v. 17, n. 3, p. 229–241, 2001.

DOWLING, K. *Robotics: comp.robotics Frequently Asked Questions*. 1996. Disponível em: <http://www.faqs.org/faqs/robotics-faq/>. Acesso em: 10/02/2006.

Ó'DUNLAING, C.; SHARIR, M.; YAP, C. K. Retraction: A new approach to motion planning. In: *Proceedings of the 15th ACM Symposium on the Theory of Computing*. Boston, MA: [s.n.], 1983. p. 207–220.

ELFES, A. A sonar-based mapping and navigation system. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. San Francisco, CA: [s.n.], 1986. p. 1151–1156.

ESPOSITO, J.; KUMAR, V. A method for modifying closed loop motion plans to satisfy unpredictable dynamics constraints at run time. In: *IEEE Int. Conf. on Robotics and Automation*. [s.n.], 2002. p. 1691–1696. Disponível em: <http://repository.upenn.edu/meam_papers/13/>. Acesso em: 31/03/2006.

FERREIRA, G. A. N. *Desenvolvimento de uma arquitetura de controle baseada em objetos para um robô móvel aquático*. Dissertação (Dissertação (Mestrado)) — Escola Politécnica da USP, São Paulo, SP, 2003. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3132/tde-31072003-153011/>. Acesso em: 31/03/2006.

FIKES, R. E.; NILSSON, N. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, v. 5, n. 2, p. 189–208, 1971.

FIRBY, R. J. *Adaptive Execution in Complex Dynamic Worlds*. [S.l.], 1989. Disponível em: <http://citeseer.ist.psu.edu/firby89adaptive.html>. Acesso em: 31/03/2006.

FIRBY, R. J.; PROKOPOWICZ, P. N.; SWAIN, M. J. The animate agent architecture. In: KORTENKAMP, D.; BONASSO, R. P.; MURPHY, R. (Ed.). *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. Cambridge, MA, USA: MIT Press, 1998. p. 243–275. ISBN 0-262-61137-6.

FOX, D.; BURGARD, W.; DELLAERT, F.; THRUN, S. Particle filters for mobile robot localization. In: DOUCET, A.; FREITAS, N. de; GORDON, N. (Ed.). *Sequential Monte Carlo Methods in Practice*. New York: Springer Verlag, 2001. Disponível em: <http://www.cs.washington.edu/ai/Mobile_Robotics/mcl/postscripts/particle-chapter-00.ps.gz>. Acesso em: 31/03/2006.

GAT, E. *Reliable Goal-directed Reactive Control for Real-World Autonomous Mobile Robots*. Tese — Dept. of Computer Science, Virginia potytechnic Institute and State University, 1991.

GAT, E. ESL: A language for supporting robust plan execution in embedded autonomous agents. In: *Proceedings of the IEEE Aerospace Conference*. Los Alamitos, CA: IEEE Computer Society Press, 1997. Disponível em: <http://www.flownet.com/gat/papers/esl.pdf>. Acesso em: 31/03/2006.

GEORGEFF, M. P.; LANSKY, A. L. Reactive reasoning and planning. In: *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*. Seatle, USA: [s.n.], 1987.

GIRALT, G.; CHATILA, R.; VAISSET, M. An integrated navigation and motion control system for autonomous multisensory mobile robots. In: BRADY, M.; PAUL, R. (Ed.). *First International Symposium on Robotics Research*. Cambridge MA: MIT Press, 1984. p. 191–214.

HAYES-ROTH, B. An architecture for adaptive intelligent systems. *Artificial Intelligence*, v. 72, n. 1-2, p. 329–365, January 1995. Disponível em: <http://citeseer.ist.psu.edu/hayes-roth95architecture.html>. Acesso em: 31/03/2006.

HERTZBERG, J.; SCHöNHERR, F. Concurrency in the dd&p robot control architecture. In: *Proceedings of the International NAISO Congress on Information Science Innovations (ISI)*. Dubai: ICSC Academic Press, 2001. p. 1079–1085. Disponível em: <http://www.agentec.de/pdf/hesc2001.pdf>. Acesso em: 31/03/2006.

IYENGAR, S. S.; ELFES, A. *Autonomous Mobile Robots: Control, Planning, and Architecture*. Los Alamitos, California: IEEE Computer Society Press, 1991.

JOHNSON, S. D. *Synthesis of Digital Designs from Recursion Equations*. Cambridge, MA: MIT Press, 1983.

KAELBLING, L. P. REX: A symbolic language for the design and parallel implementation of embedded systems. In: *Proceedings of the AIAA Conference on Computers in Aerospace*. Wakefield, MA: [s.n.], 1987. p. 255–260.

KAELBLING, L. P. Goals as parallel program specifications. In: *Proceedings of the Sixth National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press, 1988.

KAELBLING, L. P.; ROSENSCHEIN, S. Action and planning in embedded agents. In: MAES, P. (Ed.). *Designing Autonomous Agents*. Cambridge, MA: MIT Press, 1991. p. 35–48.

KAVRAKI, L. E.; SVESTKA, P.; LATOMBE, J.-C.; OVERMARS, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In: *IEEE Transaction on Robotics and Automation*. [s.n.], 1996. v. 12, n. 4, p. 566–580. Disponível em: <http://citeseer.ist.psu.edu/6614.html>. Acesso em: 31/03/2006.

KHATIB, O. Real-time obstacle avoidance for manipulators and mobile robots. In: *Proc. IEEE Int. Conf. Robotics and Automation*. St. Louis, MO: [s.n.], 1985. p. 500–505.

KONOLIGE, K.; MYERS, K. The saphira architecture for autonomous mobile robots. In: KORTENKAMP, D.; BONASSO, R. P.; MURPHY, R. (Ed.). *Artificial intelligence and mobile robots: case studies of successful robot systems*. Cambridge, MA, USA: MIT Press, 1998. p. 211–242. ISBN 0-262-61137-6.

KORTENKAMP, D.; BONASSO, R. P.; MURPHY, R. *Artificial intelligence and mobile robots: case studies of successful robot systems*. Menlo Park, California: The AAAI Press/The MIT Press, 1998.

KUIPERS, B. J.; BYUN, Y. T. A qualitative approach to robot exploration and map-learning. In: *Proceedings of the IEEE Workshop on Spatial Reasoning and Multi-Sensor Fusion*. Los Altos, CA: [s.n.], 1987. p. 390–404.

LATOMBE, J.-C. *Robot Motion Planning*. Boston: Kluwer, 1991.

LAVALLE, S. M. *Planning Algorithms*. Cambridge University Press, 2006. Disponível em: <http://msl.cs.uiuc.edu/planning/>. Acesso em: 10/02/2006.

LOZANO-PEREZ, T. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32, n. 2, p. 108–120, 1983.

LUMIA, R.; ALBUS, J. S. Teleoperation and autonomy for space robotics. *Robotics*, v. 4, p. 27–33, 1988.

LYONS, D. M.; HENDRIKS, A. Planning as incremental adaptation of a reactive system. *Robotics and Automation Systems*, v. 14, n. 4, p. 255–288, 1995. Disponível em: <http://trill.cis.fordham.edu/~lyons/Papers/jkit.ps>. Acesso em: 10/02/2006.

LYONS, D. M.; HENDRIKS, A. J. Planning for reactive robot behavior. In: *Proceedings of the 1992 IEEE Int. Conf. on Robotics and Automation (ICRA)*. Nice, France: [s.n.], 1992. p. 2675–2680. Disponível em: <http://trill.cis.fordham.edu/~lyons/Papers/ra92.ps>. Acesso em: 10/02/2006.

MA, Y.; KOSECKA, J.; SASTRY, S. Vision guided navigation for a nonholonomic mobile robot. *Transactions on Robotics and Automation*, v. 15, n. 3, p. 521–536, June 1999.

MACKENZIE, D.; CAMERON, J.; ARKIN, R. Specification and execution of multiagent missions. In: *Proceedings of the International Conference on Intelligent Robotics and Systems (IROS)*. Pittsburgh, PA: IEEE Computer Society, 1995. p. 51–58.

MAES, P. The dynamics of action selection. In: *Proceedings of the Eleventh International Joint Conference on Articial Intelligence (IJCAI89)*. Detroit, MI: [s.n.], 1989. v. 2, p. 991–998.

MARBLE, J. L.; BRUEMMER, D. J.; FEW, D. A.; DUDENHOEFFER, D. D. Evaluation of supervisory vs. peer-peer interaction with human-robot teams. In: *37th Hawaii International Conference on System Sciences*. Hawaii: [s.n.], 2004.

MARS Rovers Get Software Updates. 2004. Disponível em: <http://www.space.com/news/rover_software_040413.html>. Acesso em: 31/03/2006.

MATARIC, M. Behavior-based control: Main properties and implications. In: *Proceedings of Workshop on Intelligent Control Systems, International Conference on Robotics and Automation*. Nice, France: [s.n.], 1992. Disponível em: <http://www-robotics.usc.edu/˜maja/publications/icraws92.pdf>. Acesso em: 31/03/2006.

MEYSTEL, A. Planning in hierarchical nested controller for autonomous robots. In: *Proceedings of the Twenty-fifth Conference on Decision and Control*. Athens, Greece: [s.n.], 1986. p. 1237–1249.

MEYSTEL, A. Knowledge based nested hierarchical control. In: SARIDIS, G. (Ed.). *Advances in Automation and Robotics*. [S.l.]: JAI Press, 1990. v. 2, p. 63–152.

MORAVEC, H. P. Towards automatic visual obstacle avoidance. In: *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. Cambridge, MA: [s.n.], 1977. p. 584.

MORAVEC, H. P. The stanford cart and the CMU rover. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1983. v. 71, p. 872–884.

MORAVEC, H. P.; ELFES, A. E. High resolution map from wide-angle sonar. In: *Proceedings of IEEE International Conference on Robotics and Automation*. Washington, DC: [s.n.], 1985. p. 116–121.

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

MURPHY, R.; ARKIN, R. C. SFX: An architecture for action-oriented sensor fusion. In: *Proceedings of the International Conference on Intelligent Robotics and Systems (IROS)*. Raleigh, NC: [s.n.], 1992. p. 1079–1086.

MURPHY, R.; MALI, A. Lessons learned in integrating sensing into autonomous mobile robot architectures. In: *Journal of Experimental and Teoretical Artificial Intelligence special issue on Software Architectures for Hardware Agents*. Great Britain: Taylor & Francis, 1997. v. 9, n. 2-3, p. 191–209.

MURPHY, R.; ROGERS, E. *Cooperative Assistance for Remote Robot Supervision*. 1996. Disponível em: <http://citeseer.ist.psu.edu/murphy96cooperative.html>. Acesso em: 10/02/2006.

NICOLESCU, M.; MATARIC, M. Linking perception and action in a control architecture for human-robot interaction. In: *Hawaii Int. Conf. on System Sciences, (HICSS-36)*. Hawaii, USA: [s.n.], 2003.

NILSSON, N. J. A mobile automaton: An application of artificial intelligence techniques. In: *Proceedings of the 1st International Joint Conference on Artificial Intelligence*. Washington, D.C.: [s.n.], 1969. p. 509–520. Disponível em: <http://www.ai.sri.com/pubs/files/tn040-nilsson69.pdf>. Acesso em: 31/03/2006.

NILSSON, N. J. Shakey the robot. In: *Technical Note No. 323*. Artificial Intelligence Center, SRI International, Menlo Park, CA: [s.n.], 1984. Disponível em: <http://www.ai.sri.com/pubs/files/629.pdf>. Acesso em: 31/03/2006.

NOREILS, F.; CHATILA, R. Plan execution monitoring and control architecture for mobile robots. *IEEE Transactions on Robotics and Automation*, v. 11, n. 2, p. 255–266, April 1995.

PARIKH, S. P.; GRASSI, V.; KUMAR, V.; OKAMOTO, J. Incorporating user inputs in motion planning for a smart wheelchair. In: *Proc. IEEE Int. Conf. Robotics and Automation*. New Orleans, LA: [s.n.], 2004. p. 2043–2048. Disponível em: <http://repository.upenn.edu/meam_papers/30/>. Acesso em: 31/03/2006.

PARIKH, S. P.; GRASSI, V.; KUMAR, V.; OKAMOTO, J. Usability study of a control framework for an intelligent wheelchair. In: *Proc. IEEE Int. Conf. Robotics and Automation*. Barcelona, Spain: [s.n.], 2005.

PARIKH, S. P.; RAO, R.; JUNG, S.-H.; KUMAR, V.; OSTROWSKI, J. P.; TAYLOR, C. J. Human robot interaction and usability studies for a smart wheelchair. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [s.n.], 2003. v. 4, p. 3206–3211. Disponível em: <http://repository.upenn.edu/meam_papers/19/>. Acesso em: 31/03/2006.

PATEL, S.; JUNG, S.; OSTROWSKI, J.; RAO, R.; TAYLOR, C. Sensor based doorway navigation for a nonholonomic vehicle. In: *International Conference on Robotics and Automation*. Washington, DC: [s.n.], 2002.

PAYTON, D. W.; ROSENBLATT, J. K.; KEIRSEY, D. M. Plan guided reaction. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 20, n. 6, p. 1370–1382, 1990.

PEREIRA, G. A. S.; KUMAR, V.; SPLETZER, J.; TAYLOR, C. J.; CAMPOS, M. F. M. Cooperative transport of planar objects by multiple mobile robots using object closure. In: *Proceedings of the 8th International Symposium on Experimental Robotics (ISER'02)*. Sant'Angelo d'Ischia, Italy: [s.n.], 2002. p. 275–284.

PIMENTA, L. C. A.; FONSECA, A. R.; PEREIRA, G. A. S.; MESQUITA, R. C.; SILVA, E. J.; CAMINHAS, W. M.; CAMPOS, M. F. M. On computing complex navigation functions. In: *Proc. IEEE Int. Conf. Robotics and Automation*. Barcelona, Spain: [s.n.], 2005. p. 3463–3468.

RAO, R.; CONN, K.; JUNG, S.; KATUPITIYA, J.; KIENTZ, T.; KUMAR, V.; OSTROWSKI, J.; PATEL, S.; TAYLOR, C. Human robot interaction: Applications to smart wheelchairs. In: *International Conference on Robotics and Automation*. Washington, DC: [s.n.], 2002. Disponível em: <http://repository.upenn.edu/meam_papers/29/>. Acesso em: 31/03/2006.

RIBEIRO, C. H. C.; COSTA, A. H. R.; ROMERO, R. A. F. Robôs móveis inteligentes: Princípios e técnicas. In: MARTINS, A. T.; BORGES, D. L. (Ed.). *Anais do XXI Congresso da Sociedade Brasileira de Computação*. Fortaleza: SBC, 2001. v. 3, p. 257–306.

RIMON, E.; KODITSCHEK, D. E. Exact robot navigation using artificial potential functions. *IEEE Trans. on Robotics and Automation*, v. 8, n. 5, p. 501–518, 1992.

ROCI. *The Remote Operations Control Interface*. 2006. Disponível em: <http://sourceforge.net/projects/rociproject/>. Acesso em: 13/02/2006.

ROSENBLATT, J. DAMN: A distributed architecture for mobile navigation. In: *Proc. of the AAAI Spring Symp. on Lessons Learned from Implemented Software Architectures for Physical Agents*. Palo Alto, CA: [s.n.], 1995. p. 167–178. Disponível em: <http://www.ri.cmu.edu/pubs/pub_3209.html>. Acesso em: 31/03/2006.

ROSENBLATT, J. DAMN: A distributed architecture for mobile navigation. *Journal of Experimental and Theoretical Artificial Intelligence*, v. 9, n. 2 / 3, p. 339 – 360, 1997. Disponível em: <http://www.ri.cmu.edu/pubs/pub_3134.html>. Acesso em: 31/03/2006.

ROSENBLATT, J. *DAMN: A Distributed Architecture for Mobile Navigation*. Tese (Doutorado) — Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, January 1997. Disponível em: <http://www.ri.cmu.edu/pubs/pub_1439.html>. Acesso em: 31/03/2006.

ROSENBLATT, J.; PAYTON, D. A fine-grained alternative to the subsumption architecture for mobile robot control. In: *Proceedings of the International Joint Conference on Neural Networks*. [s.n.], 1989. p. 317–23. Disponível em: <http://www.umiacs.umd.edu/users/julio/papers/Fine_Grained_Alternative.ps.gz>. Acesso em: 31/03/2006.

ROSENSCHEIN, S.; KAELBLING, L. The synthesis of digital machines with provable epistemic properties. In: *Proceedings of the 1986 conference on Theoretical aspects of reasoning about knowledge*. San Francisco, CA: Morgan Kaufmann Publishers Inc, 1986. p. 83 – 98. Disponível em: <http://www.tark.org/proceedings/tark_mar19_86/p83-rosenschein.pdf>. Acesso em: 31/03/2006.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. Upper Saddle River, NJ: Prentice Hall, 1995.

SAFFIOTTI, A. The uses of fuzzy logic in autonomous robot navigation: a catalogue raisonn'e. *Soft Computing Research Journal*, v. 1, n. 4, p. 180 – 197, December 1997. Disponível em: <http://www.aass.oru.se/˜asaffio/Papers/scj97.html>. Acesso em: 31/03/2006.

SAFFIOTTI, A.; KONOLIGE, K.; RUSPINI, E. H. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, v. 76, n. 1-2, p. 481–526, 1995. Disponível em: <http://www.aass.oru.se/˜asaffio/Papers/aij95.html>. Acesso em: 31/03/2006.

SAFFIOTTI, A.; RUSPINI, E. H.; KONOLIGE, K. Blending reactivity and goal-directedness in a fuzzy controller. In: *Proceedings of the IEEE Int. Conf. on Fuzzy Systems*. San Francisco, California: IEEE Press, 1993. p. 134–139. Disponível em: <http://aass.oru.se/ asaffio/Papers/fuzzieee93.html>. Acesso em: 31/03/2006.

SARIDIS, G.; VALVANIS, K. On the theory of intelligent controls. In: *Proceedings of the SPIE Conference on Advances in Intelligent Robotic Systems*. Cambridge, MA: [s.n.], 1987. p. 488–495.

SCHOLTZ, J.; ANTONISHEK, B.; YOUNG, J. Evaluation of a human-robot interface: Development of a situational awareness methodology. In: *Proc. of the 37th Hawaii Int. Conf. on System Sciences*. Big Island, Hawaii: [s.n.], 2004.

SMITH, R.; SELF, M.; CHEESEMAN, P. Estimating uncertain spatial relationships in robotics. In: COX, I. J.; WILFONG, G. T. (Ed.). *Autonomous Robot Vehicles*. [S.l.]: Springer-Verlag, 1990. p. 167– 193.

SPLETZER, J.; DAS, A.; FIERRO, R.; TAYLOR, C.; HUMAR, V.; OSTROWSKI, J. Cooperative localization and control for multi-robot manipulation. In: *Proceedings of the Conference on Intelligent Robots and Systems (IROS 2001)*. [S.l.: s.n.], 2001.

TANNER, H. G.; LOIZOU, S.; KYRIAKOPOULOS, K. J. Nonholonomic stabilization with collision avoidance for mobile robots. In: *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Maui, Hawaii: [s.n.], 2001. p. 1220–1225. Disponível em: <http://www.unm.edu/~tanner/Papers/IROS01.pdf>. Acesso em: 31/03/2006.

THRUN, S.; BENNEWITZ, M.; BURGARD, W.; CREMERS, A.; DELLERT, F.; FOX, D.; HAHNEL, D.; ROSENBERG, C.; ROY, N.; SCHULTE, J.; SCHULZ, D. MINERVA: a second generation museum tour-guide robot. In: *Proc. of IEEE Int. Conf. on Robotics and Automation*. Detroit, MI: [s.n.], 1999.

TOMATIS, N.; NOURBAKHSH, I.; SIEGWART, R. Hybrid simultaneous localization and map building: a natural integration of topological and metric. *Robotics and Autonomous Systems*, v. 4, n. 1, p. 3–14, July 2003. Disponível em: <http://www.cs.cmu.edu/~illah/PAPERS/ras02.pdf>. Acesso em: 13/03/2006.

UDUPA, S. *Collision Detection and Avoidance in Computer Controlled Manipulators*. Tese — Department of Electrical Engineering, California Institute of Technology, 1977.

VALAVANIS, K. P.; GRACANIN, D.; MATIJASEVIC, M.; KOLLURU, R.; DEMETRIOU, G. A. Control architectures for autonomous underwater vehicles valavanis. *IEEE Control System Magazine*, v. 17, n. 6, p. 48–64, 1997.

YANCO, H.; DURRY, J. L. A taxonomy for human-robot interaction. In: *AAAI Fall Symposium on Human-Robot Interaction, AAAI Technical Report FS-02-03*. Falmouth, Massachusetts: [s.n.], 2002. p. 111–119. Disponível em: <http://www.cs.uml.edu/~holly/papers/yanco-drury-taxonomy-fss02.pdf>. Acesso em: 31/03/2006.

YANCO, H.; DURRY, J. L. Classifying human-robot interaction: An updated taxonomy. In: *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*. [s.n.], 2004. p. 2841–2846. Disponível em: <http://www.cs.uml.edu/~holly/papers/yanco-drury-tax-smc04.pdf>. Acesso em: 31/03/2006.