UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ANDRÉ FERREIRA

DESVIO TANGENCIAL DE OBSTÁCULOS PARA UM ROBÔ MÓVEL NAVEGANDO EM AMBIENTES SEMI-ESTRUTURADOS

VITÓRIA
2004

ANDRÉ FERREIRA

DESVIO TANGENCIAL DE OBSTÁCULOS PARA UM ROBÔ MÓVEL NAVEGANDO EM AMBIENTES SEMI-ESTRUTURADOS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Engenharia Elétrica, na área de concentração em Automação.
Orientador: Prof. Dr. Mário Sarcinelli Filho.

VITÓRIA
2004

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

Ferreira, André, 1974-

F383d Desvio tangencial de obstáculos para um robô móvel navegando em ambientes semi-estruturados / André Ferreira. – 2004.

76 f. : il.

Orientador: Mário Sarcinelli Filho.

Co-Orientador: Teodiano Freire Bastos Filho.

Dissertação (mestrado) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Robôs móveis. 2. Robôs - Sistemas de controle. I. Sarcinelli Filho, Mário. II. Bastos Filho, Teodiano Freire. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 621.3

ANDRÉ FERREIRA

DESVIO TANGENCIAL DE OBSTÁCULOS PARA UM ROBÔ MÓVEL NAVEGANDO EM AMBIENTES SEMI-ESTRUTURADOS

Dissertação submetida ao programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisição parcial para a obtenção do Grau de Mestre em Engenharia Elétrica - Automação.

Aprovada em 28 de junho de 2004.

COMISSÃO EXAMINADORA

Prof. Dr. Mário Sarcinelli Filho
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Teodiano Freire Bastos Filho
Universidade Federal do Espírito Santo
Co-orientador

Prof. Dr. Eduardo Oliveira Freire
Universidade Federal de Sergipe

Profa. Dra. Eliete Maria de Oliveira Caldeira
Faculdade Novo Milênio

Dedico esta Dissertação à minha família,
cujo apoio, alegria e união foram de grande importância
para a conclusão de mais esta etapa da minha vida, e para minha esposa,
pela paciência e compreensão nos momentos de ausência.

Agradecimentos

Dedico meus sinceros agradecimentos

– ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio dado através da bolsa de estudos durante o período de realização desta Dissertação;

– aos professores do PPGEE/UFES, principalmente os professores Dr. Mário Sarcinelli Filho (orientador), Dr. Teodiano Freire Bastos Filho (co-orientador) e Dr. Hansjörg Andreas Schneebeli, sempre prestativos e dispostos a ajudar;

– a todos os amigos do doutorado, mestrado e graduação.

"Dias de calmaria, noites de ardentia, dedos no leme e olhos no horizonte, descobri a alegria de transformar distâncias em tempo. Um tempo em que aprendi a entender as coisas do mar, a conversar com as grandes ondas e não discutir com o mau tempo. A transformar o medo em respeito, o respeito em confiança. Descobri como é bom chegar quando se tem paciência. E para se chegar, onde quer que seja, aprendi que não é preciso dominar a força, mas a razão. É preciso antes de mais nada, querer."

Amyr Klink

Resumo

Neste trabalho é proposta uma nova estratégia para desvio de obstáculos em robótica móvel. A arquitetura de controle é baseada numa abordagem reativa, portanto com baixa demanda computacional, e é adequada a ambientes semi-estruturados. A estratégia proposta permite que o robô navegue em segurança até um ponto de destino, desviando de obstáculos que possam surgir em seu trajeto. O ângulo de desvio é calculado de forma que o robô siga o contorno do obstáculo, propiciando trajetórias mais eficientes quando navegando em presença de obstáculos, no sentido de que o ponto de destino é atingido em menor tempo, com menor desgaste dos motores do robô e menor consumo de energia. Outra característica desta estratégia é que os comportamentos *Seguir Paredes*, *Seguir Corredores* e *Evitar Obstáculos* podem ser implementados pelo mesmo controlador. A metodologia proposta é validada através de exemplos simulados e também de experimentos reais, usando o robô móvel *PIONEER 2-DX*.

Abstract

This thesis proposes a new strategy for a mobile robot to deviate from obstacles while navigating in a semi-structured environment. The proposed control architecture is based on a reactive approach, thus demanding low computational effort. It allows the robot to navigate from a starting point to a destination point without colliding to any obstacle in its path. The deviation from an obstacle is performed according to an escape angle calculated so that the new robot orientation is tangent to the obstacle. It is also shown that such strategy generates more efficient trajectories, in the sense that the destination point is reached in less time while saving energy and reducing the demand on the robot motors. Another interesting aspect of the proposed strategy is that it also allows to implement the behaviors *Wall Following* and *Corridor Following*. The proposed methodology is validated through simulated examples and practical experiments as well, which were run in a *PIONEER 2-DX* mobile robot.

Sumário

Lista de Figuras

1 Introdução

Os robôs, que antes figuravam apenas em filmes de ficção científica, hoje fazem parte do nosso cotidiano. As tarefas realizadas por essas máquinas vão desde a montagem de equipamentos em indústrias por robôs manipuladores até a manutenção de oleodutos submersos a grandes profundidades por robôs submarinos. Recentemente, também se incluiu entre tais tarefas a exploração de outros planetas por robôs móveis.

O primeiro robô móvel construído e reconhecido na bibliografia é o *Shakey*, desenvolvido pelo *Stanford Research Institute*, em 1968. Desde então, várias aplicações vêm sendo desenvolvidas, podendo ser classificadas em diferentes áreas, tais como robôs para uso doméstico (aspirador de pó robotizado), uso hospitalar (monitoração e distribuição de medicamentos), exploração de ambientes de risco e regiões inóspitas (vulcões e outros planetas, bem como regiões marinhas a grandes profundidades), uso militar (desarmamento de bombas), uso na agricultura (grandes colheitadeiras robotizadas), etc.

Um fato inquestionável, na maioria das aplicações em robótica móvel, é a necessidade de um método eficiente para desvio de obstáculos, quando o robô está navegando. Os algoritmos que propiciam implementar tal ação variam bastante em complexidade. Os mais simples requerem que o robô permaneça estático enquanto realiza uma varredura panorâmica do ambiente, e só depois de concluí-la ele é novamente movimentado. Outros, mais complexos, além de detectarem os obstáculos ainda obtêm informações quantitativas e qualitativas sobre os mesmos, tais como distância, forma e dimensões, permitindo que o robô não apenas desvie do anteparo, mas que o faça de forma inteligente.

Entre os métodos tradicionalmente utilizados para desvio de obstáculos encontram-se o Método de Detecção de Bordas, o Método da Grade de Certeza (*Certainty Grid*), o Método dos Campos Potenciais, o Método do Campo de Forças Virtuais (VFF) e o Método do Histograma de Campo Vetorial (VFH).

Uma forma de classificação desses métodos é feita em relação à abordagem adotada por cada um, que pode ser caracterizada como *reativa* ou *deliberativa*. Sob esse ponto

de vista, os métodos VFH, VFF e Grade de Certeza se enquadram na segunda opção, utilizando os dados provenientes dos sensores do robô para elaboração de um modelo do ambiente e, assim, planejar as ações do veículo. Em geral, essa abordagem pode ser dividida nas seguintes tarefas:

- aquisição de dados referentes aos objetos/anteparos presentes no ambiente;
- construção de um modelo/mapa apropriado do "mundo"no qual o robô realizará as tarefas;
- planejamento da trajetória desejada;
- desvio de obstáculos enquanto percorre esta trajetória.

Uma vantagem desta abordagem, quando comparada a um comportamento puramente reativo, descrito adiante, é permitir que se encontre um caminho mais "limpo"até o ponto destino, ou seja, livre de obstáculos e "armadilhas", desde que esse caminho exista. Entretanto, esses métodos geralmente requerem grande capacidade computacional, e perdem muita eficácia em caso de alterações no ambiente.

A abordagem reativa elimina completamente a parte do conhecimento e planejamento, ou seja, não há modelos do ambiente. Ela é baseada em comportamentos que acoplam informação sensorial (percepções) diretamente a ações dos atuadores, sem o uso de representações simbólicas (mapas) com o objetivo de modelar todo ou parte do ambiente de operação do robô [1]. Como resultado, obtêm-se simplicidade e menor demanda computacional, além de permitir que haja variações no ambiente. Os métodos de Detecção de Bordas e Campos Potenciais, dentre os anteriormente citados, se enquadram na abordagem reativa. Em [2] são apresentadas as características gerais do controle reativo:

- tipicamente caracterizado pela decomposição em comportamentos primitivos;
- representações globais são proibitivas/evitadas;
- uso independente dos sensores é preferido à fusão sensorial;
- é mais adequado para ambientes com variações dinâmicas.

Portanto, tal estratégia de controle se mostra mais adequada a ambientes semi-estruturados[1], enquanto a abordagem deliberativa é mais adequada a ambientes estruturados.

[1]Ambiente semi-estruturado é aquele cujas informações não são totalmente conhecidas a priori, não sendo possível, portanto, um planejamento prévio de trajetórias.

Existe ainda uma forma *híbrida*, na qual a abordagem deliberativa é utilizada para navegação global (tarefa de mais alto nível), enquanto a abordagem reativa é empregada em tarefas de baixo nível (desvio de obstáculos, por exemplo).

Uma tendência atual em robótica móvel é o uso, cada vez maior, de robôs de baixo custo, trabalhando em cooperação para desempenhar tarefas que um só não poderia realizar ou gastaria muito tempo para fazê-lo. Neste contexto, é bastante comum o uso de sensores acústicos, os quais são um meio barato para se determinar proximidade de obstáculos, e muito útil na implementação de sistemas para navegação. Dessa forma, o uso de uma abordagem reativa, aliado a um sensoriamento via ultra-som e sistemas operacionais de código aberto, pode ser uma boa opção para essa nova tendência. Esta é, genericamente falando, a linha de ação adotada neste trabalho, a qual será detalhada ao longo desta Dissertação.

1.1 Definição do Problema

O problema principal a ser tratado nesta Dissertação é a navegação segura e eficiente de um robô móvel em um ambiente semi-estruturado[2], utilizando uma arquitetura de controle reativa e sensores ultra-sônicos. O enfoque principal é guiar o robô enquanto ele navega entre dois pontos do ambiente, evitando quaisquer obstáculos. Tais obstáculos podem, inclusive, ser dinâmicos, ou seja, podem surgir de forma inesperada no caminho do robô.

1.2 Objetivo

O principal objetivo é desenvolver uma arquitetura de controle reativa, utilizando algoritmos de controle confiáveis (propriedade dos algoritmos de controle clássicos) e adaptados para garantir a um robô móvel manobras evasivas mais suaves e eficientes (menor gasto de energia, menor desgaste dos motores e menor tempo de execução das tarefas), quando em presença de obstáculos.

Dentro deste trabalho, também é realizada a união dos comportamentos *Desvio de Obstáculos*, *Seguir Paredes* e *Seguir Corredores*, utilizando a mesma arquitetura de controle, o que a abordagem aqui proposta permite, como será visto adiante.

Para consecução do objetivo proposto, torna-se necessário o cumprimento das seguin-

[2]O ambiente semi-estruturado considerado é plano e fechado.

tes tarefas:

- revisão da literatura e análise dos diferentes métodos relacionados ao tema de desvio de obstáculos para robôs móveis, atentando-se para suas vantagens e desvantagens, bem como para motivos de falha e sucesso;
- definição de algoritmos a serem adotados dentro da abordagem proposta para a navegação do robô móvel, sua implementação computacional e simulações;
- realização de experimentos diversos, cujos resultados serão analisados e comparados com outros similares disponíveis na literatura, destacando-se vantagens do método e/ou aplicação desenvolvida.

1.3 Trabalhos Relacionados

Durante o desenvolvimento deste trabalho, vários métodos inerentes a esta área de pesquisa foram consultados. A seguir, serão discutidos e comentados os de maior relevância.

1.3.1 Método de Detecção de Bordas

Neste método, a idéia principal é a utilização de um algoritmo para detecção e determinação da posição das bordas verticais dos obstáculos, e, em seguida, calcular uma trajetória que permita ao robô contornar essas bordas. Calculando linhas que unem duas destas bordas detectadas é possível determinar limites do obstáculo [3][4].

Algumas desvantagens dessa abordagem se devem, principalmente, aos problemas inerentes ao uso dos sonares, dentre os quais estão:

- a *baixa diretividade* implica baixa resolução angular, ou seja, limita a precisão na determinação da posição espacial das bordas, dependendo da distância ao obstáculo e do ângulo entre a superfície do obstáculo e o eixo acústico;
- *erros de leitura* podem ser gerados por fontes externas de ruído ultra-sônico e também por reflexões oriundas de outros transdutores vizinhos (*crosstalk*). Nem sempre esses dados errôneos podem ser filtrados, fazendo com que o algoritmo detecte objetos a distâncias diferentes das reais;

- *reflexões especulares* ocorrem quando o ângulo entre a frente de onda acústica e a normal à superfície refletora é muito grande. Nesta situação, os obstáculos não são detectados ou apenas fracamente identificados, já que toda (ou quase toda) onda é refletida para outra direção.

1.3.2 Grade de Certeza (*Certainty Grid*)

Esta estratégia foi introduzida inicialmente por pesquisadores da Universidade *Carnegie Mellon* (CMU), tendo grande aplicabilidade em sistemas que utilizam sensores imprecisos.

A idéia deste método é criar um mapa reticulado referente ao ambiente de operação do robô, com valores indicando a possibilidade de existência de objetos nestas pequenas regiões, ou seja, uma representação probabilística dos obstáculos [5][6][7]. Este reticulado é representado através de uma matriz bidimensional.

À medida que o robô navega pelo ambiente, as medidas dos sensores (sonares, por exemplo) são utilizadas para atualizar constantemente o mapa de probabilidades. Quanto maior o número de vezes que um anteparo é detectado numa certa região, maior o grau de certeza que é atribuído à célula correspondente no mapa. Medidas espúrias geram regiões com baixa probabilidade, sendo, praticamente, desconsideradas.

A construção do mapa reticulado pode ser feita combinando-se as informações provenientes de diferentes sensores, de modo que cada sonar (ou outro sensor) fornece uma contribuição para a avaliação das células. Um exemplo disso é a situação em que um ou mais sensores indicam um provável obstáculo em um dado ponto do mapa. Desse modo, a probabilidade da região estar ocupada aumenta. A Figura 1 representa um exemplo deste mapa de probabilidades. Áreas vazias com um alto fator de certeza são representadas por espaços em branco e as áreas com baixo fator de certeza, utilizam o símbolo "+" com espessura variável. Áreas ocupadas são representadas pelo símbolo "x" e áreas desconhecidas, pelo símbolo ".". Os círculos representam posições nas quais o robô realizou varreduras do ambiente e, por último, as linhas sólidas são uma representação bidimensional do ambiente no qual o experimento foi realizado.

Nas aplicações realizadas na CMU, o robô permanecia estacionário enquanto fazia uma varredura panorâmica com seus 24 sensores ultra-sônicos. Posteriormente, uma função de probabilidade era aplicada às medidas e o mapa era atualizado. Finalmente, o robô movia-se para nova posição, parava e repetia o procedimento. Após percorrer toda a sala,

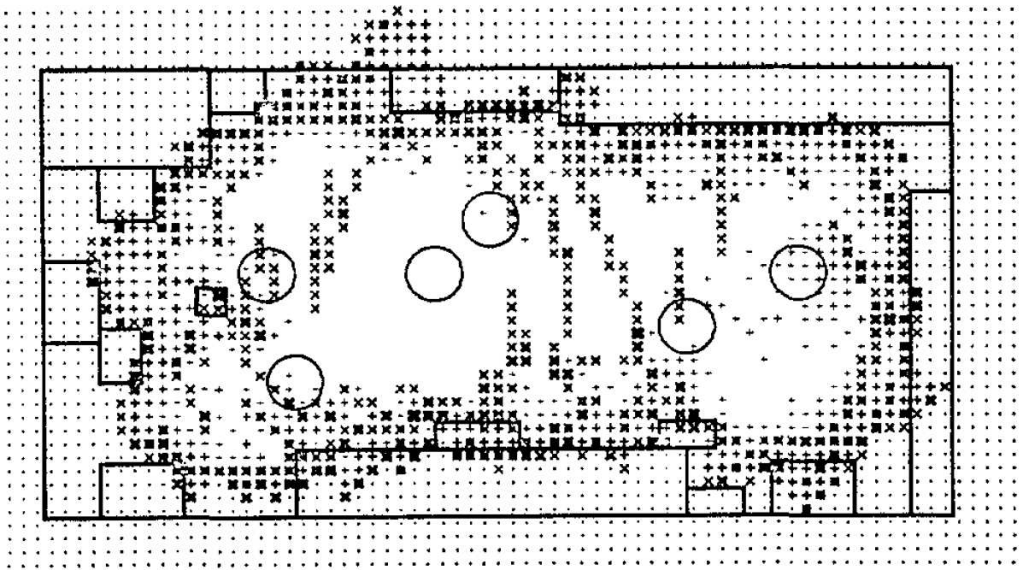

Figura 1: Mapa 2D gerado com o método *Certainty Grid* [5].

repetindo-se o procedimento, é obtido um mapa de probabilidades muito fiel à realidade. Um método para traçar uma trajetória livre de colisões pode, então, ser aplicado, para que o robô realize movimentos em segurança, dentro do ambiente previamente explorado.

Dentre as vantagens do método, pode-se destacar que ele permite a integração de informação proveniente de vários sensores, inclusive de sensores diferentes, representa uma boa opção para geração de mapas precisos de um ambiente, e, por fim, permite diminuir consideravelmente os problemas causados pelo uso dos sonares.

1.3.3 Método dos Campos Potenciais

Em 1986 foram publicados os primeiros artigos tratando de forças imaginárias atuando sobre um robô [8]. A idéia central destes trabalhos é que os obstáculos presentes no ambiente em que o robô atua exercem forças repulsivas sobre o veículo, enquanto o alvo (ponto de destino do robô) exerce uma força de atração. A resultante dessas forças pode ser calculada e, a partir dela, pode-se determinar o novo vetor velocidade do robô, dando continuidade ao seu trajeto pelo ambiente.

Thorpe utilizou este método para o planejamento de trajetórias *off-line* [9]. Mais tarde, Krogh e Thorpe sugeriram um método híbrido para planejamento de trajetórias locais e globais, utilizando uma abordagem de *Campo Potencial Generalizado* [10]. Newman e Hogan introduziram a construção de funções potenciais através da combinação de funções de obstáculos individuais com operações lógicas [11].

Um ponto comum entre esses métodos é a suposição de um modelo de mundo conhecido, no qual formas geométricas simples e pré-definidas representam obstáculos e que a trajetória do robô é gerada em modo *off-line*.

Brooks e Arkin utilizaram um robô equipado com sensores de ultra-som em experimentos envolvendo a teoria de campos potenciais [12][13]. Na implementação de Brooks, os dados oriundos dos sensores eram tratados como um vetor de forças repulsivas. Se a magnitude da soma destas forças exceder um dado limiar, o robô pára, orienta-se de acordo com a resultante do vetor de forças e executa o movimento. Nesta implementação, apenas um conjunto de leituras é utilizado, enquanto que os valores anteriores são descartados. Em seus experimentos, Arkin obteve valores de velocidade linear em torno de 120 mm/s para transpor obstáculos, utilizando estratégia semelhante à de Brooks.

Como exemplo de uso deste método pode ser citada a arquitetura de controle baseada em impedância, na qual a relação entre forças fictícias geradas por obstáculos e o movimento do robô é regulada [14]. Esta arquitetura, bem como as suas características, serão tratadas no capítulo seguinte.

A teoria de Campos Potenciais é particularmente atrativa por sua elegância e simplicidade. Entretanto, há alguns problemas inerentes ao método. Entre as principais desvantagens em se utilizar o Método dos Campos Potenciais encontram-se as oscilações em presença de obstáculos e em passagens estreitas, além de dificuldades para transpor obstáculos próximos entre si (os marcos de uma porta, por exemplo) [15].

1.3.4 Método do Campo de Forças Virtuais (VFF)

Como visto anteriormente, a necessidade de parar diante de um provável obstáculo, para realizar uma varredura panorâmica, e os problemas causados pela forte dependência das medidas provenientes dos sensores ultra-sônicos (nem sempre precisas), apresentam-se como principais desvantagens do Método de Detecção de Bordas.

O *Virtual Force Field* (VFF), ou Campo de Forças Virtuais, é um arranjo híbrido, utilizando o método de *Certainty Grid* e a teoria de Campos Potenciais, ambos vistos nas subseções anteriores [16]. A idéia principal é utilizar as informações geradas pelo primeiro método, para um mapeamento local do ambiente de trabalho do robô, e pelo segundo, para desvio de eventuais obstáculos, bem como a guiagem do veículo até o seu destino final (alvo). Em [16] também foram discutidos o problema de mínimos locais e foram realizados experimentos com um robô se movendo a uma velocidade máxima de

780 mm/s.

O *Certainty Grid* propicia um melhor tratamento das medidas provenientes dos sonares, geralmente pouco precisas, além da possibilidade de fácil integração entre múltiplos sensores. Por essa razão, este método é bastante utilizado para mapeamento do ambiente, bem como dos obstáculos.

Neste método, representado na Figura 2, o robô navega enquanto os dados oriundos dos sonares servem para atualizar constantemente o mapa de certezas. Além disso, um algoritmo varre uma região restrita do mapa global (esta região corresponde ao entorno do veículo), com a finalidade de gerar uma trajetória local, desvencilhando o robô de obstáculos inesperados.

Neste algoritmo, células com valores de certeza (CV - *Certainty Values*) acima de um dado limiar geram forças repulsivas atuantes sobre o robô. Essas forças de repulsão são inversamente proporcionais ao quadrado da distância do centro do robô à célula considerada, e diretamente proporcionais ao grau de certeza vinculado à mesma.

A força repulsiva resultante, $\mathbf{F_r}$, é dada pela soma vetorial das forças geradas por todas as células, e, para que o robô alcance seu destino final, uma força de atração $\mathbf{F_t}$ é gerada pelo alvo. A resultante $\mathbf{R}$, entre forças atrativas e repulsivas, é dada por $\mathbf{R} = \mathbf{F_t} + \mathbf{F_r}$. Depois de calculada a resultante $\mathbf{R}$, é possível calcular a nova orientação do robô (ver Figura 2). Vale lembrar que medidas errôneas geram células com baixos valores de CV, sendo, portanto, insignificantes no cálculo das forças.

Entre as vantagens desse método estão:

- maior imunidade a erros de medição dos sensores ultra-sônicos;
- o uso do Método *Certainty Grid* propicia maior facilidade quanto à integração sensorial, mesmo que os sensores sejam de tipos diferentes;
- não há necessidade de o robô parar enquanto realiza a varredura do ambiente;
- a atualização do mapa de certezas, com os dados provenientes dos sensores, e a utilização deste mapa para navegação evitando obstáculos são independentes e funcionam de forma assíncrona, podendo ser executadas da maneira mais otimizada possível. Isto não ocorre no Método de Detecção de Bordas, por exemplo, no qual o sistema de navegação executa sempre a seguinte seqüência de atividades: detectar obstáculos, parar o robô, fazer medições e encontrar contornos do obstáculo, recalcular a trajetória e realizar o novo movimento.

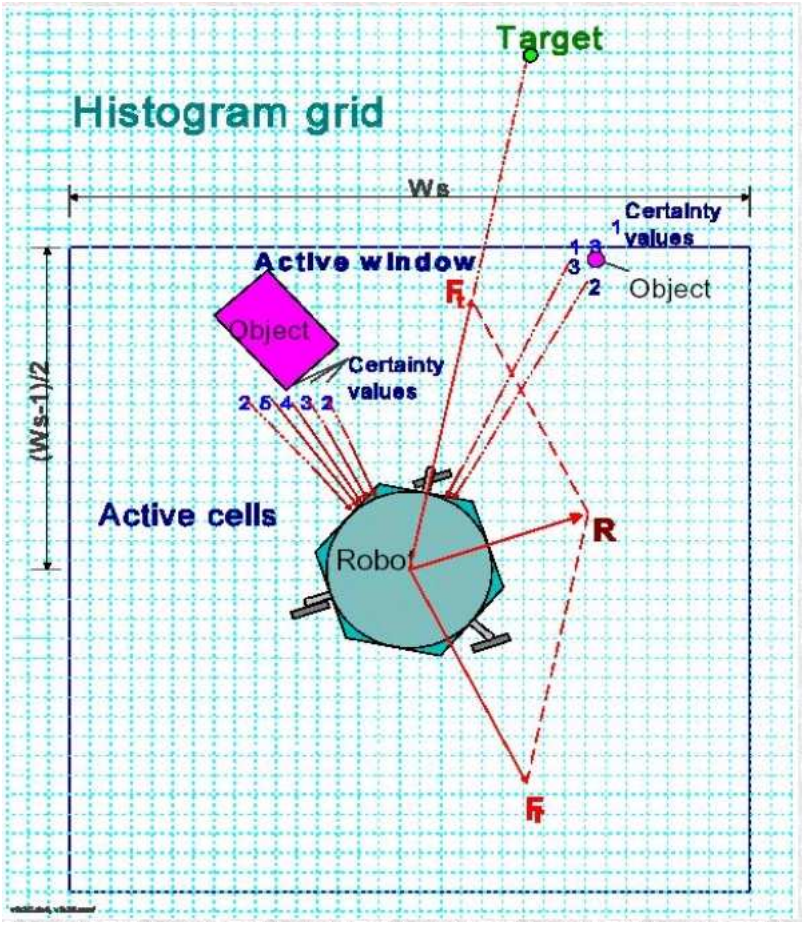

Figura 2: Método do Campo de Forças Virtuais [16].

Como desvantagens deste método, podem ser citadas:

- grandes dificuldades em passar por obstáculos razoavelmente próximos (valores menores de $1,70\ m$, como os marcos de uma porta, por exemplo), em virtude das forças repulsivas geradas por ambos os lados da passagem (inerente ao uso de Campos Potenciais Artificiais);
- movimento oscilatório e até mesmo instabilidade quando o robô passa pela transição de um corredor mais largo para um mais estreito (ver Figura 3);
- flutuações consideráveis no controle de orientação do robô, geradas quando sua posição mapeada no reticulado muda de uma célula para outra, causando mudanças abruptas (de até 40%) nos valores da resultante **R**.

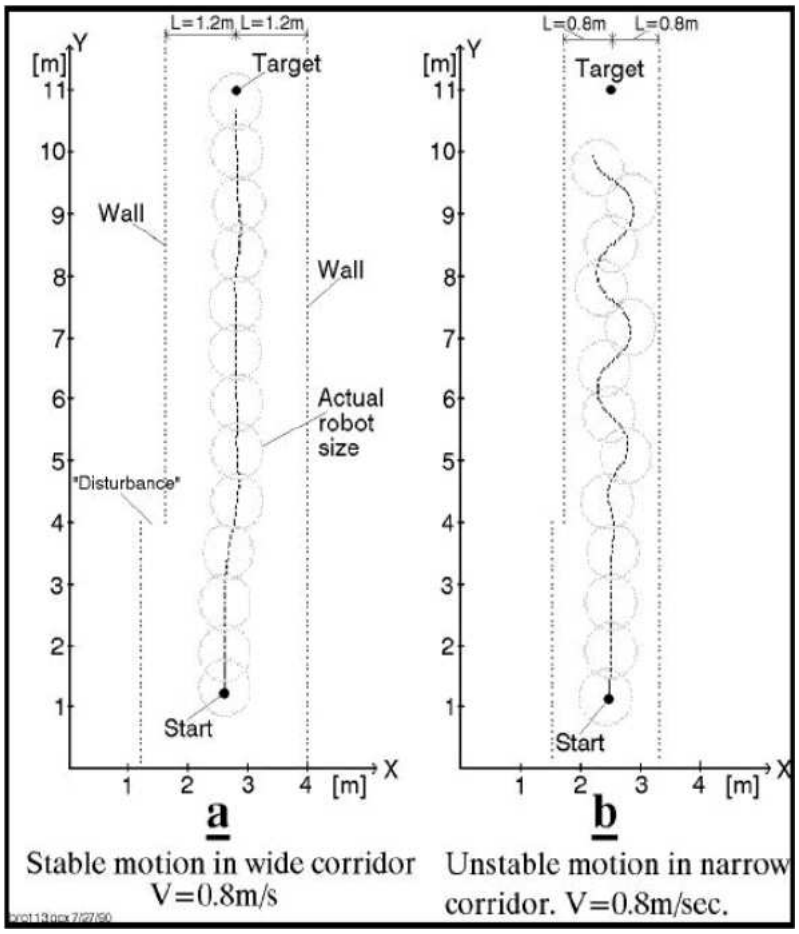

Figura 3: Problema do VFF com corredores estreitos [15].

1.3.5 Método do Histograma de Campo Vetorial (VFH)

Para suprir as principais deficiências do método anterior (VFF), Borenstein e Koren desenvolveram um novo método, o *Vector Field Histogram* (VFH), ou Histograma de Campo Vetorial [17]. O ponto chave na elaboração desse método foi perceber que no VFF perde-se muita informação acerca do ambiente e seus obstáculos, quando a força resultante $\mathbf{F_r}$ é calculada. Quando a grande quantidade de dados provenientes do mapa é processada, é gerado apenas um vetor, o $\mathbf{F_r}$.

Neste método é acrescentado um estágio intermediário de processamento. Depois que o reticulado com as probabilidades está pronto e disponível, esse mapa bidimensional é projetado num histograma polar unidimensional. O mapa é, então, dividido em setores angulares, como indicado na Figura 4, e cada um deles é mapeado no histograma, representando a densidade de obstáculos, na forma polar, daquele setor (ver Figura 5). Por último, é escolhida a nova direção a ser tomada pelo robô. Para tal decisão, varre-se o histograma obtido à procura de regiões com vales, que indicam ausência de obstáculos,

enquanto os picos expressam grande probabilidade de existência de obstáculos. No caso de uma região com muitos vales, é tomado aquele que mais se aproxima da direção do destino final (alvo). Em experimentos, esse método permitiu que o robô utilizado (CARMEL) atingisse velocidades em torno de 780 mm/s

Um ponto muito importante deste método é a não utilização da teoria de campos potenciais, ou seja, não há forças repulsivas ou atrativas influenciando o movimento do robô. É feita apenas a varredura do ambiente para obtenção da nova orientação a ser seguida. O ambiente não exerce qualquer tipo de força virtual sobre o robô, que apenas "*enxerga*" a presença dos obstáculos e calcula qual a melhor rota (local) a ser tomada.

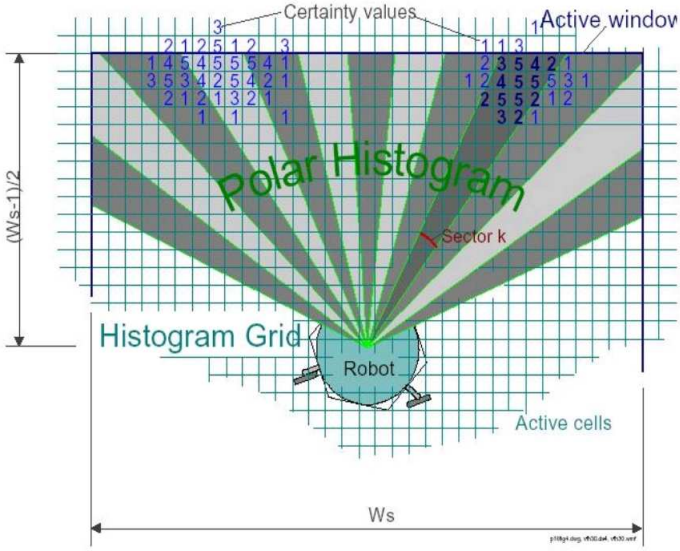

Figura 4: Mapeamento dos setores para o histograma polar [17].

Atravessar uma porta torna-se razoavelmente simples com a utilização deste método. Para realizar esta tarefa, é necessário apenas que o sistema de controle do robô identifique o vale, que representa a passagem através da porta, e oriente o veículo naquela direção. Métodos que utilizam a teoria de campos potenciais teriam grande dificuldade na realização desta atividade em razão das forças de repulsão exercidas pelas laterais da porta. Além disso, robôs controlados via VFH praticamente não oscilam quando navegando em corredores estreitos, pelo mesmo motivo já descrito.

1.4 Contextualização do Trabalho

De acordo com o exposto até o momento, percebe-se claramente uma evolução dos métodos para desvio de obstáculos, desde o mais simples (Detecção de Bordas) até o

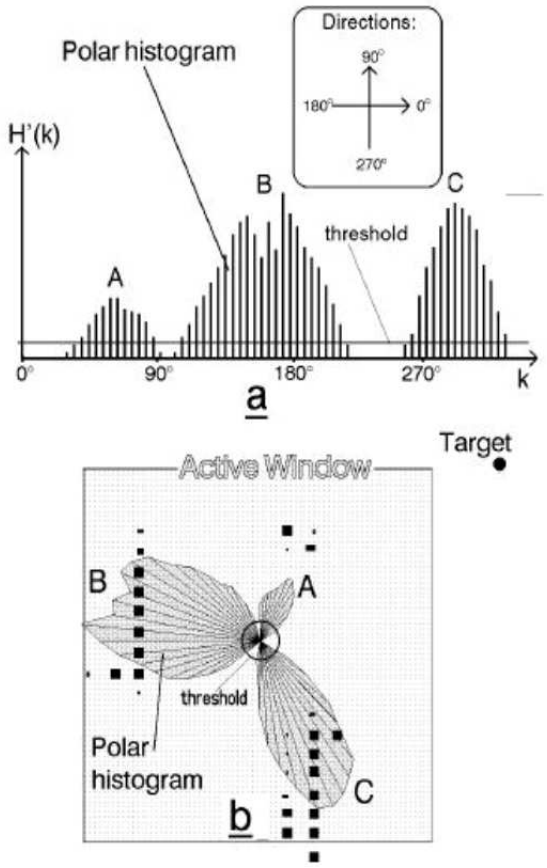

Figura 5: Histogramas polares de densidade de obstáculos [17].

considerado mais complexo e completo, representado pelo Histograma de Campo Vetorial (VFH).

O método de detecção de bordas é considerado o mais simples, dentre os mencionados, apresentando grandes problemas em função do uso dos sonares. Não é viável a utilização deste método como único sistema destinado à tarefa de evitar obstáculos em aplicações que exijam precisão.

A grade de certezas (*Certainty Grid*) é bastante útil para geração de mapas precisos do ambiente, e quando usada em outros métodos, tais como o VFF e o VFH, torna o sistema muito mais seguro e confiável.

O uso de Campos Potenciais implica problemas com obstáculos próximos entre si, representando um ponto fraco do VFF. Esses problemas não são encontrados no VFH, justamente por não utilizar essa teoria de campos potenciais artificiais. No Histograma de Campo Vetorial (VFH), o sistema apenas "*enxerga*" o ambiente e descobre qual o melhor caminho, sem que forças atuem sobre o robô. Em relação à velocidade de percurso, os

métodos VFF e VFH mostram-se mais adequados a altas velocidades do que os outros métodos.

Em navegação autônoma de robôs, os ambientes geralmente são conhecidos, mas com possibilidade do aparecimento de obstáculos inesperados. Com o conhecimento prévio do ambiente, é interessante que se utilize um planejador global de trajetórias, propiciando caminhos mais eficientes. Para tarefas isoladas, como o desvio de obstáculos, e em situações em que informações acerca do ambiente não estão disponíveis, é utilizado um planejador local de trajetórias.

"Armadilhas" geradas em decorrência de pontos de mínimo são consideradas como um problema comum a todos os métodos apresentados. Quando isto ocorre, o robô geralmente apresenta um comportamento cíclico, descrevendo trajetórias fechadas, das quais, muitas vezes, não é possível encontrar um caminho de fuga. Este problema se deve ao fato da utilização de um planejador local de trajetória, ao invés de um planejador global. Os algoritmos mais sofisticados utilizam artifícios para superar tais deficiências. Um exemplo disso é empregado no VFH, através de um *monitor de trajetória*, cuja função é detectar esses pontos de mínimo. Em caso positivo, o sistema de navegação é alterado para um navegador global (utiliza apenas informação disponível no mapa construído) e só retorna ao modo local quando não se encontrar mais na situação de armadilha.

Nesta Dissertação é assumido que os obstáculos, bem como o ambiente de trabalho do robô, não são conhecidos *a priori*, sendo analisados, principalmente, sob o aspecto da navegação local para desvio de obstáculos.

A arquitetura de controle baseada em impedância [14], introduzida sucintamente na subseção 1.3.3 e explanada de forma minuciosa no capítulo seguinte, servirá de base para a arquitetura de controle de navegação e desvio de obstáculos proposta nesta Dissertação, porém, sem a utilização da teoria de campos potenciais artificiais.

1.5 Estrutura da Dissertação

Essa Dissertação de Mestrado está organizada da seguinte forma:

- **Capítulo 1: Introdução**

  É descrito o tema desta Dissertação, em linhas gerais. São apresentados o problema a ser resolvido e também os objetivos aos quais se propõe este trabalho. São expostos alguns dos métodos mais relevantes, em se tratando de desvio de obstáculos em

robótica móvel, bem como suas vantagens e desvantagens. Por fim, é apresentado um resumo sucinto acerca do que é tratado em cada capítulo subseqüente.

- **Capítulo 2: Arquitetura de Controle Baseada em Impedância**

  Neste capítulo, é apresentada a arquitetura de controle baseada em impedância, além do modelamento matemático do robô móvel utilizado, a cinemática do veículo, os sistemas de coordenadas, a estrutura do Controle Baseado em Impedância, a estabilidade do sistema, os fatores inerentes ao método, além de simulações e breve comentário sobre seus resultados.

- **Capítulo 3: O Sistema de Controle Proposto**

  Neste capítulo é descrita a estrutura do sistema de controle proposto, a estabilidade do sistema, fatores inerentes ao método, além de simulações e comentários sobre seus resultados.

- **Capítulo 4: Resultados e Discussões**

  Neste capítulo é apresentado, inicialmente, o simulador utilizado. Em seguida, os resultados das simulações realizadas nos Capítulos 2 e 3 são discutidos e comparados. São descritas, também, as configurações de *hardware* e *software* encontradas no robô que foi utilizado nos experimentos. Também são discutidos neste capítulo a implementação do *software* e o sensoriamento do robô, além das plataformas ARIA e Saphira. Por fim, é discutida a implementação da arquitetura proposta e feita uma análise dos resultados experimentais obtidos, resultando na validação do método desenvolvido.

- **Capítulo 5: Conclusões**

  Neste capítulo é feita uma análise conclusiva do trabalho realizado, e são propostos trabalhos futuros.

2 *Controle Baseado em Impedância*

Esta arquitetura de controle, na qual está baseada a arquitetura realizada neste trabalho, utiliza a abordagem reativa [14][18]. Uma desvantagem da arquitetura original se deve à utilização da teoria de campos potenciais. Na arquitetura proposta nesta Dissertação, para evitar as deficiências comentadas anteriormente, a teoria de campos potenciais não será utilizada.

A arquitetura de controle baseada em impedância faz uso de dois laços de controle: um interno, referente a um controlador de posição final, responsável por levar o robô de uma dada posição até um destino final (alvo), e um laço externo, referente ao controlador de impedância, com a função de evitar obstáculos durante o trajeto do robô através do ambiente. A estrutura detalhada desta arquitetura de controle será descrita a seguir.

2.1 Modelo Cinemático do Robô

Na Figura 6 é apresentado um sistema de coordenadas inercial $< g >$ e o veículo é posicionado inicialmente num ponto diferente da origem deste sistema [19]. O movimento do robô é regido por uma ação conjunta da velocidade angular ω e da velocidade linear u, sempre sobre um dos eixos de seu sistema de coordenadas $< a >$.

O conjunto de equações cinemáticas que descreve a posição cartesiana (x, y) do veículo, e sua orientação ϕ, é dado por

$$\begin{aligned} \dot{x} &= u\cos\phi, \\ \dot{y} &= u\ sen\phi, \\ \dot{\phi} &= \omega, \end{aligned} \tag{2.1}$$

onde x, y e ϕ são medidos em relação ao referencial inercial $< g >$.

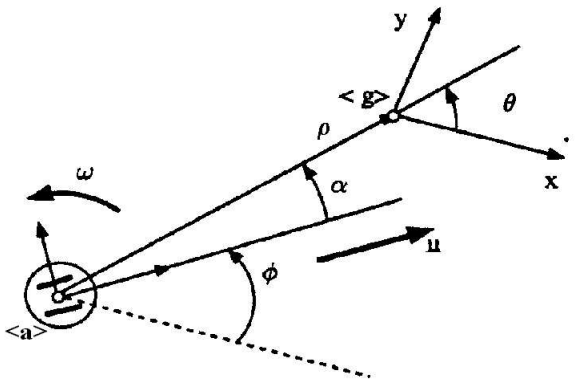

Figura 6: Diagrama para obtenção do modelo cinemático do robô.

Uma representação polar da posição do veículo utilizando o erro de posição $\rho > 0$ e sua orientação θ em relação ao referencial inercial $< g >$ é dada por

$$\begin{aligned} \dot{\rho} &= -u\cos(\theta - \phi), \\ \dot{\theta} &= u\frac{sen\alpha}{\rho}, \\ \dot{\phi} &= \omega. \end{aligned} \tag{2.2}$$

Utilizando a igualdade $\alpha = \theta - \phi$, que expressa o ângulo entre o eixo principal do veículo (orientação do robô) e o vetor erro ρ, o conjunto de equações (2.2) se transforma em

$$\begin{aligned} \dot{\rho} &= -u\cos\alpha, \\ \dot{\alpha} &= -\omega + u\frac{sen\alpha}{\rho}, \\ \dot{\theta} &= u\frac{sen\alpha}{\rho}, \end{aligned} \tag{2.3}$$

lembrando que o valor de ρ deve ser diferente de zero, visto que as equações cinemáticas estão expressas em coordenadas polares e os valores de α ou θ não têm sentido para tais valores de ρ.

2.2 Controlador de Posição Final

A função deste controlador é fazer com que o robô alcance um dado ponto destino, e lá permaneça. Para desempenhar tal tarefa, o controlador recebe informações de posição e orientação do robô e, em seguida, efetua o cálculo dos erros de posição angular e distância

até um alvo, previamente estabelecido. As ações de controle (velocidade angular ω e velocidade linear u) são calculadas e enviadas ao robô. Dessa forma, ele é capaz de alcançar o ponto destino.

Para gerar as ações de controle, os erros de orientação e distância são calculados, tendo como base o diagrama apresentado na Figura 7. O símbolo $< g >$ representa o sistema de coordenadas do ponto destino e o símbolo $< a >$ representa o sistema de coordenadas do robô. O vetor ρ representa o erro de posição entre a posição do robô e o ponto destino. O ângulo α expressa o desvio angular entre a orientação corrente do robô e a direção do alvo. Além disso, o ângulo ϕ representa o desvio angular entre a orientação corrente e a orientação final desejada (no caso da Figura 7, a orientação final é representada pelo eixo x_g).

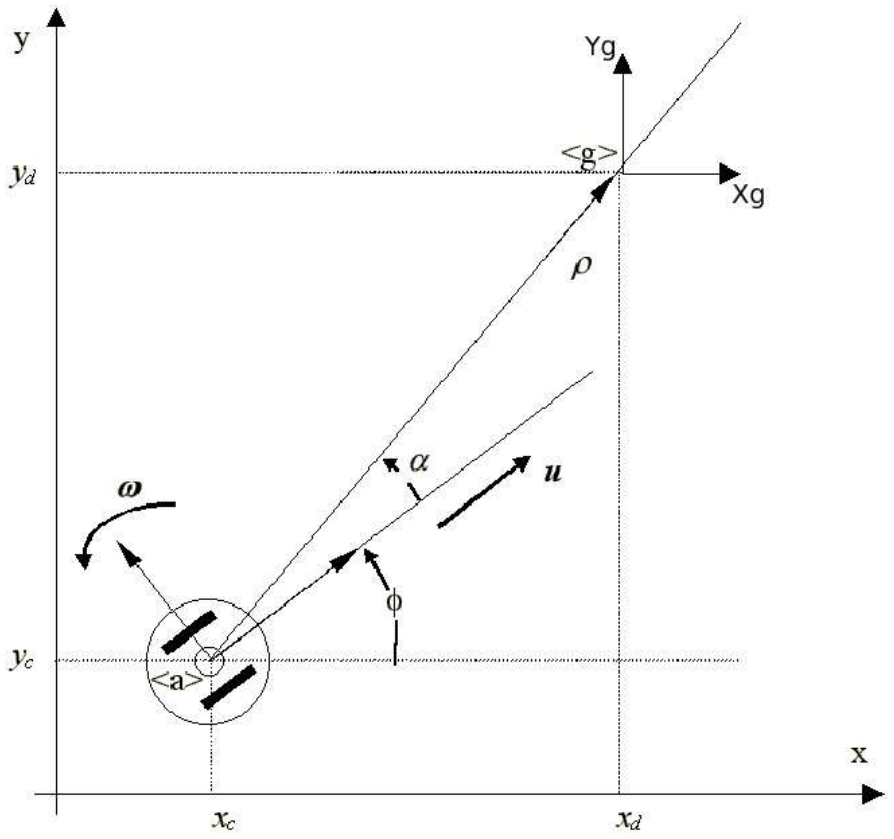

Figura 7: Erros de posição e orientação do robô em relação ao alvo.

O modelo cinemático do robô, relativo ao sistema de coordenadas do ponto destino e às variáveis de estado ρ e α, pode ser descrito através das equações (2.3).

Considerando a seguinte função candidata de Lyapunov, para análise de estabilidade,

$$V(\rho,\alpha) = V_1 + V_2 = \frac{1}{2}h_\rho\rho^2 + \frac{1}{2}\alpha^2 \quad para\ h_\rho > 0, \tag{2.4}$$

e assumindo

$$\begin{aligned} u &= k_u \tanh\rho\cos\alpha \quad para\ h_u > 0, \\ \omega &= k_\omega\alpha + k_u\frac{\tanh\rho}{\rho}sen\alpha\cos\alpha, \end{aligned} \tag{2.5}$$

como ações de controle, é possível provar que a derivada temporal da função candidata de Lyapunov é definida negativa. Isto implica convergência assintótica global das variáveis de estado ρ e α a zero, assegurando, portanto, que o objetivo de controle é alcançado. A prova completa está em [14][20]. Em ambas as referências também está incluída a análise quanto à orientação final do robô, θ, de modo a garantir que o veículo não fique indefinidamente em um movimento de rotação ao chegar no ponto destino.

É importante observar que este controlador utiliza restrição das ações de controle, ou seja, o problema da saturação do sinal de controle é levado em consideração. No cálculo da velocidade linear u é possível restrigir seu módulo, dado que $|u_{max}| = k_u$. Para a velocidade angular ω, o seu valor máximo é dado por $|\omega_{max}| = k_\omega\ \pi + 0,5\ k_u$. Esses valores são obtidos por análise direta das Equações 2.5. Além disso, a fim de propiciar maior independência à variação dos ganhos k_u e k_ω é utilizada a estratégia de aproximação do alvo real, posicionando-o sempre 1 m de distância à frente do robô, e, em seguida, realizando a rotação necessária para a orientação tangencial do robô em relação ao obstáculo. Para fins ilustrativos, pode-se considerar uma dada aplicação na qual seja necessário manter a relação entre os ganhos k_u e k_ω, resultando num valor de k_u, por exemplo, que levará o controlador à saturação, ou seja, com um sinal de controle u acima do máximo permitido pelo sistema. Neste caso, com valores de ρ (distância robô-obstáculo) abaixo de 5 m, é possível trazer o sinal de controle de volta à faixa aceitável, ou seja, evitar a saturação.

A partir das ações de controle, é possível escrever as equações do sistema em malha fechada, as quais são

$$\begin{aligned} \dot{\rho} &= -k_u \tanh\rho\cos^2\alpha, \\ \dot{\alpha} &= -k_\omega\alpha, \\ \dot{\theta} &= k_u\frac{\tanh\rho}{\rho}sen\alpha\cos\alpha. \end{aligned} \tag{2.6}$$

Este sistema de controle possui sua estrutura de acordo com o diagrama da Figura 8.

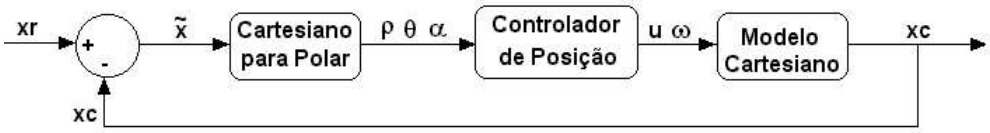

Figura 8: Diagrama de blocos do controlador de posição final.

2.3 O Controle Baseado em Impedância Implementado

As idéias acerca de impedância e admitância são familiares aos profissionais da área de sistemas elétricos, representando formas generalizadas da resistência e condutância elétricas, dependentes da freqüência.

Nesta Dissertação, o conceito de impedância expressa uma relação dinâmica (impedância mecânica) entre a posição do robô e uma força fictícia (campos potenciais artificiais) atuando sobre o mesmo, de modo a representar, de forma mais fiel à realidade, a interação entre o robô e o meio no qual as tarefas são realizadas.

A teoria sobre o controle de impedância foi inicialmente sugerida por Hogan, sendo utilizada principalmente em aplicações para robôs manipuladores e em próteses para portadores de deficiência [21]. Mais tarde, essa teoria foi adaptada para navegação de robôs móveis [14][18][20].

A impedância Z é definida como

$$Z(s) = Bs + K, \tag{2.7}$$

onde B e K são constantes positivas que representam, respectivamente, o amortecimento e a elasticidade da interação robô-obstáculo, quando na zona de repulsão. Esse conceito é utilizado na implementação de um laço externo de controle, funcionando em conjunto com um controlador de posição final, compondo, assim, um sistema completo de navegação ponto-a-ponto, com desvio de obstáculos.

2.4 O Sistema Completo

A partir dos controladores de posição final e desvio de obstáculos, é montado um sistema de controle para navegação de robôs móveis, cujo diagrama de blocos está representado na Figura 9.

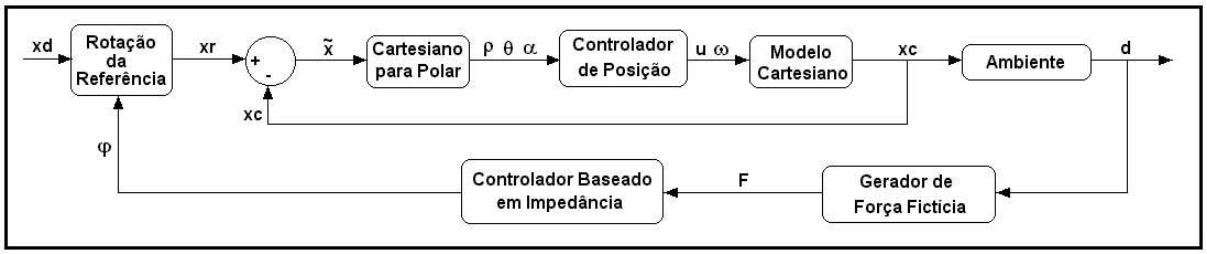

Figura 9: Diagrama de blocos para o sistema de controle baseado em impedância.

A Figura 10 mostra a situação em que um obstáculo é detectado pelo robô. Uma força **F** de repulsão é gerada, fazendo com que o ponto destino x_d seja rotacionado temporariamente, permitindo, assim, que o robô desvie do obstáculo. As componentes da força **F** — $\mathbf{F_t}$ (componente na direção da orientação do robô) e $\mathbf{F_r}$ (componente perpendicular à orientação do robô) — também estão representadas. Um fato importante a ser observado

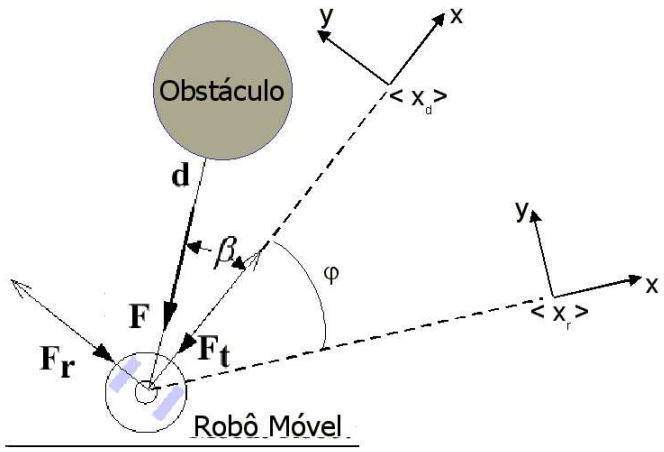

Figura 10: Detecção do obstáculo e a força fictícia gerada.

é que esta força de interação entre o robô e o obstáculo é dada em função da distância entre os dois, já que o contato entre eles significaria uma colisão.

O módulo da força de repulsão **F**, que o obstáculo exerce sobre o robô, é calculado como

$$F = a - b[d - d_{min}]^2, \tag{2.8}$$

onde a e b são constantes positivas, tais que $a = b.[d_{max} - d_{min}]^2$, sendo que d_{min} é a menor distância que o sistema sensorial é capaz de medir, d_{max} é a máxima distância desejada, capaz de gerar uma força fictícia de repulsão, e d é a distância medida até o obstáculo ($d_{min} < d < d_{max}$).

O erro de impedância x_a é calculado, utilizando a componente $\mathbf{F_t}$ (Figura 10), como

$$x_a = Z(s)^{-1} F_t, \tag{2.9}$$

enquanto o ângulo φ de rotação do alvo pode ser obtido através de

$$\varphi = x_a.sign(\mathbf{F_r}). \tag{2.10}$$

Aplicando uma transformação adequada, a posição real de destino, x_d, é rotacionada para a posição temporária x_r, a qual é dada por

$$x_r = \begin{bmatrix} \cos\varphi & sen\varphi & 0 \\ -sen\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} x_d. \tag{2.11}$$

O novo alvo (x_r) é, então, passado como valor de referência para o *controlador de posição final* (laço interno responsável por levar o robô até um ponto destino), conforme mostra a Figura 9.

Sempre que obstáculos forem detectados e estiverem na zona de repulsão, uma força fictícia $\mathbf{F}$ será gerada, e produzirá um ângulo de giro φ diferente de zero, fazendo com que o robô desvie do obstáculo (laço externo, referente ao controlador de impedância [21]). O ângulo de desvio é inversamente proporcional à distância robô-obstáculo e ao ângulo β. Nesta estratégia, o caminho de fuga não é levado em consideração, salvo que o veículo deve realizar a manobra evasiva buscando aproximar-se do alvo real sempre que possível. Em caso de ausência de obstáculos, o valor de φ é nulo, e a matriz de rotação torna-se uma matriz identidade, não alterando o valor de x_d. Desse modo, este sistema de controle permite que o robô seja capaz de chegar até o ponto destino, desvencilhando-se de obstáculos que possam surgir durante o trajeto. Este algoritmo de controle é estável no sentido de Lyapunov, conforme mostrado em [14] e [18]. Isto significa que o robô sempre atinge o ponto de destino, independentemente dos obstáculos em seu caminho (sempre que o ponto destino for alcançável).

2.5 Simulações Utilizando a Arquitetura de Controle Baseada em Impedância

Uma simulação utilizando esta arquitetura de controle é analisada nesta seção. O objetivo, nesta simulação, é alcançar o ponto de coordenadas (8000,4000) (valores expressos em milímetros), evitando os obstáculos presentes na trajetória. A Figura 11 mostra a trajetória descrita pelo robô desde o ponto inicial (0,0), seguindo por uma seqüência de corredores, até chegar ao ponto destino. Quando obstáculos são detectados, uma força fictícia é gerada, fazendo com que o robô realize um desvio. O valor escolhido para d_{max} foi de 70 cm e os ganhos utilizados no controlador de posição final foram $k_u = 0,3$ e $k_\omega = 0,9$. A Figura 12 e a Figura 13 mostram os sinais de controle u (velocidade linear) e w (velocidade angular) gerados pelo controlador de posição final, que permitem ao robô evitar obstáculos, enquanto a Figura 14, mostra a orientação do robô. Observa-se, nesta simulação, grande variação nos valores de velocidade angular e velocidade linear, e também na orientação do robô. Esta oscilação é decorrente do método de desvio utilizado, no qual a trajetória de fuga não é fator preponderante. Esta oscilação também deve-se ao fato de que a componente lateral da força não produz rotação do ponto de destino.

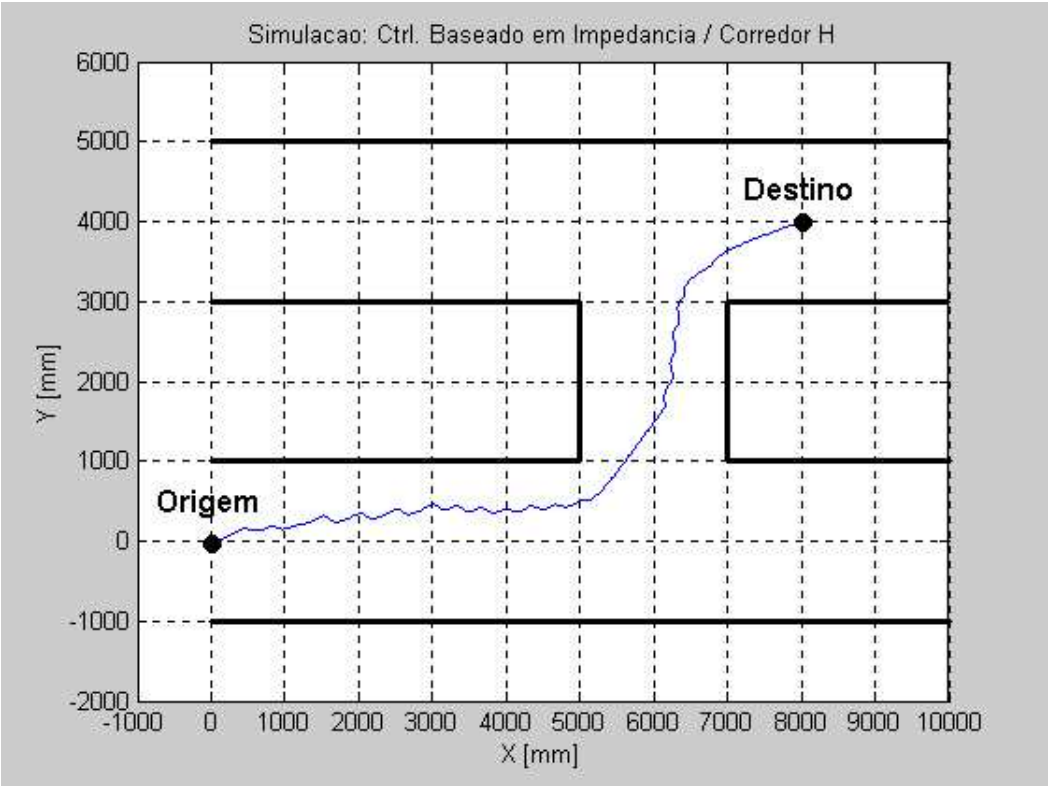

Figura 11: Trajetória do robô.

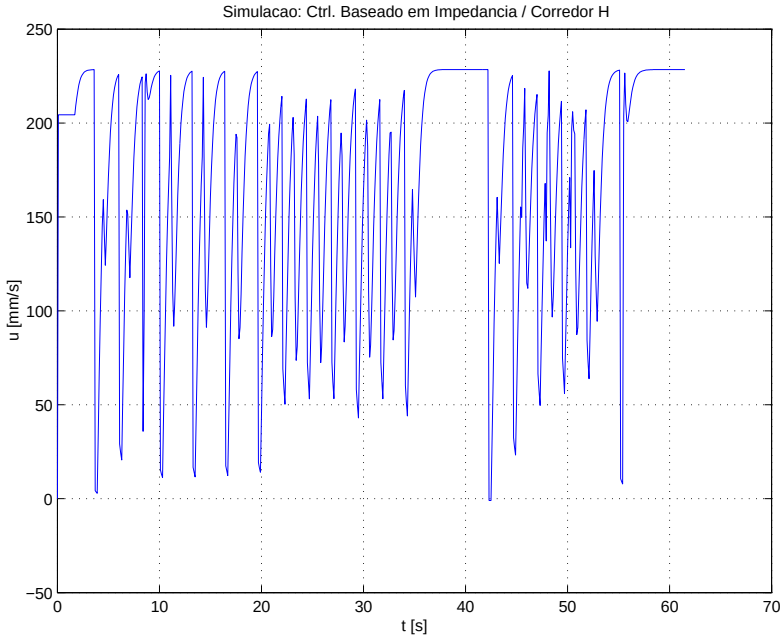

Figura 12: Velocidade linear do robô.

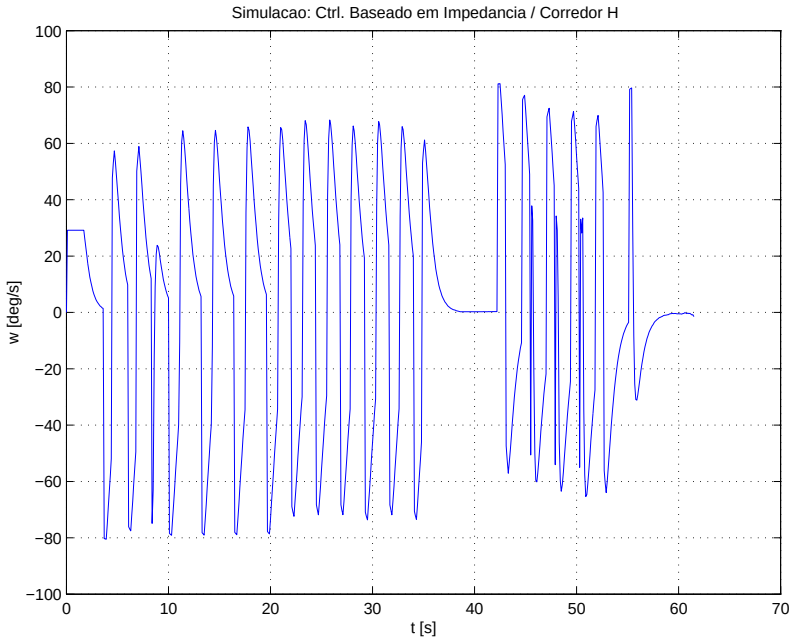

Figura 13: Velocidade angular do robô.

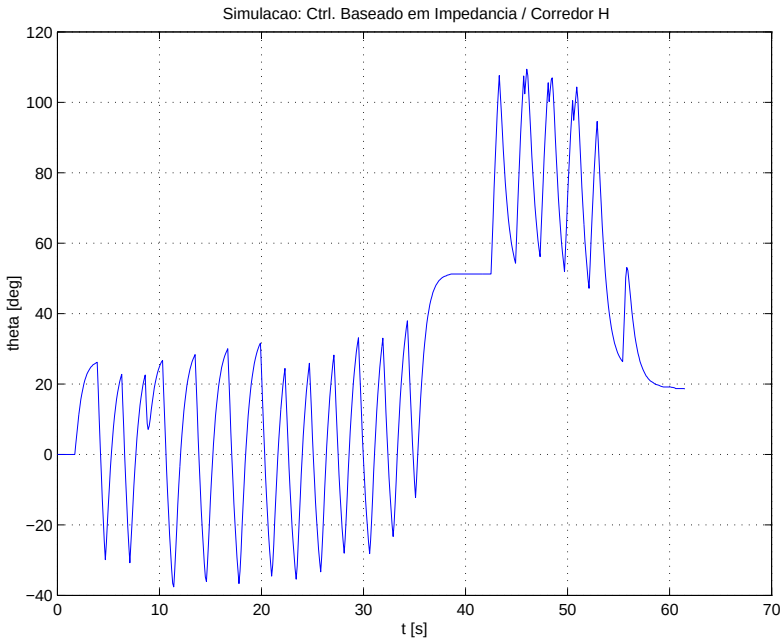

Figura 14: Orientação do robô.

3 O Sistema de Controle Proposto

Como verificado nas simulações anteriores (Figura 11), a arquitetura de controle baseada em impedância é uma solução para o problema de desvio de obstáculos, embora nem sempre assegure um caminho suave e eficiente quando o robô evita os obstáculos. Eficiente, no caso, no sentido de menor gasto de energia, menor desgaste dos motores e menor tempo de execução da tarefa.

A arquitetura proposta neste capítulo é apresentada como uma solução para essas deficiências da arquitetura de controle baseada em impedância original.

3.1 A Nova Arquitetura Proposta

A proposta da nova arquitetura de controle se baseia em trajetórias evasivas tangentes ao obstáculo detectado, utilizando parte da estratégia original, que possui estabilidade comprovada. A idéia desta arquitetura proposta é manter o controlador de posição final (laço interno) e alterar o laço externo, referente ao desvio de obstáculos. O controlador de impedância será substituído por um outro, que é proposto neste trabalho, o qual continuará gerando um ângulo de desvio φ para a reorientação do robô. No entanto, o ângulo φ não será mais calculado em função da força repulsiva $\mathbf{F}$, mas sim em função da posição angular do obstáculo em relação ao robô. O ângulo de desvio deve ser tal que force o veículo a se orientar numa direção paralela ao obstáculo.

A vantagem principal desta arquitetura de fuga está no fato de o robô seguir os contornos do obstáculo, como será aqui demonstrado, descrevendo trajetórias adequadas, sob o ponto de vista da navegação em presença de obstáculos [22].

Para melhor descrição da arquitetura proposta, é apresentada a Figura 15, na qual está representada a situação em que o robô detecta um obstáculo, no caso uma parede.

Quando o robô entra na zona de repulsão, o ângulo β (entre a sua orientação corrente e a parte do obstáculo mais próxima) é medido, e, conhecendo-se a orientação do robô em relação ao alvo real (ângulo α), o ângulo φ, que propicia um desvio tangencial, é determinado como

$$\varphi = \left(\frac{\pi}{2} - |\beta|\right) \cdot reverseSign(\beta) - \alpha, \tag{3.1}$$

onde o valor de α indica o menor ângulo que o robô deve girar para orientar-se com o alvo real (Figura 15). A operação $reverseSign(x)$ expressa o sinal negado do valor x, de módulo unitário. O ângulo β é obtido através das posições angulares, previamente conhecidas, dos sensores ultra-sônicos, dispostos em anel ao redor do robô.

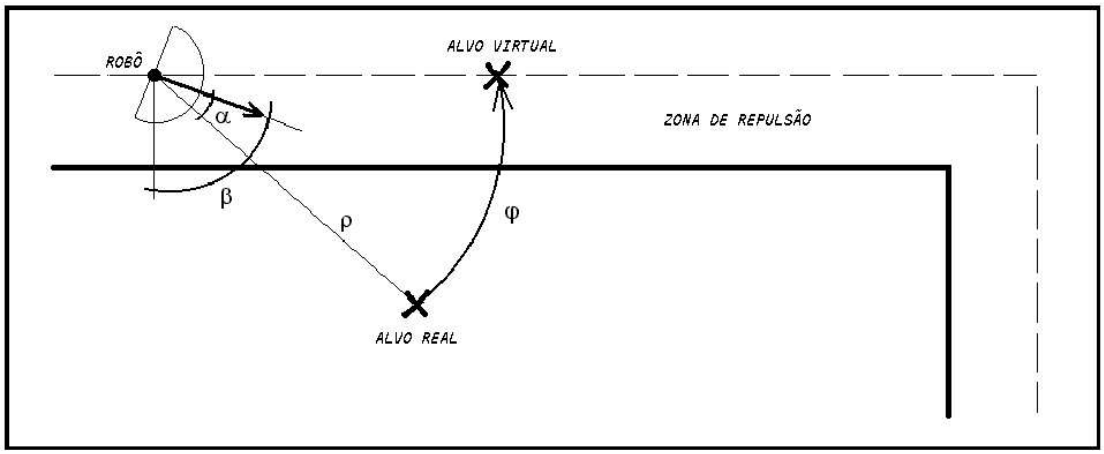

Figura 15: Obtenção do ângulo φ.

Para escolha da zona de repulsão deve-se considerar principalmente a velocidade de percurso do robô (velocidade linear), bem como sua capacidade de giro (velocidade angular), visto que este deve ser capaz de desvencilhar-se dos obstáculos em tempo hábil, sem que haja colisão. Uma vez estabelecido, este valor não precisa ser alterado, salvo se houver alterações nos valores máximos de velocidade. Neste trabalho, as zonas de repulsão utilizadas foram estabelecidas via análise de simulações.

Outro fator importante, ainda sobre zona de repulsão, é que se houver interferência destas zonas, isto não representará problema à navegação do robô. As zonas de repulsão apenas informam ao sistema de controle quando um obstáculo deve ser considerado como de risco ou não. Um experimento ilustrando tal situação é apresentado no final desta Dissertação.

Depois de calculado o desvio φ, este valor é inserido na matriz de rotação (Equação 2.11) e o alvo real é rotacionado, gerando um *alvo virtual*. Este novo ponto é passado ao controlador de posição final, que reorienta o robô, de forma a atingir o novo objetivo. Na

ausência de obstáculos, o desvio sofrido pelo alvo real é nulo, e o robô navega normalmente em direção ao alvo estabelecido. Esta arquitetura está representada na Figura 16.

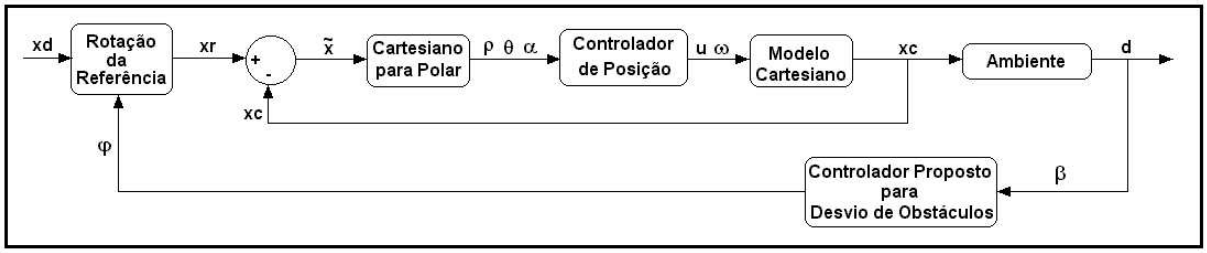

Figura 16: Diagrama de blocos para o sistema de controle proposto.

Quanto à estabilidade do sistema, sua análise se restringe à análise do controlador de posição final, já que o controlador proposto, assim como o de impedância, gera apenas novos valores de referência para o controlador de posição final. É importante mencionar que o controlador proposto atua em um laço muito mais lento do que o laço do controlador de posição final, e por isso este último é capaz de seguir a variação do ponto de destino. A análise de estabilidade em questão foi apresentada parcialmente na Seção 2.2, e é feita de forma completa em [14][20]. Portanto, comparando-se os diagramas em bloco das duas arquiteturas de controle, pode-se perceber que o desvio tangencial de obstáculos também é estável no sentido de Lyapunov, ou seja, é assegurado que tal estratégia permite ao robô atingir seu ponto de destino.

3.2 Simulações Utilizando a Nova Arquitetura

Algumas simulações, demonstrando a validade da nova proposta, são apresentadas. Incialmente é abordado o problema de navegação em corredores estreitos e, a seguir, é feita uma demonstração de como o método é utilizado para seguir corredores e paredes, enquanto evita colisões.

3.2.1 Navegação em Corredores Estreitos

Um problema existente em sistemas baseados na Teoria de Campos Potenciais (Método da Impedância original, por exemplo) reside na dificuldade de navegação em corredores e passagens estreitas. Nessas situações, as forças exercidas pelos obstáculos geram uma resultante que tende a impedir a passagem do robô e/ou fazê-lo oscilar.

A arquitetura proposta nesta Dissertação não faz uso dessa teoria, possuindo um comportamento bem mais suave nestas situações. Isto é ilustrado pelo exemplo a seguir, em que é simulada a navegação do robô em um corredor que se estreita ao longo do percurso, tal como ocorre na situação representada na Figura 3, onde é apresentado o problema de oscilação existente no método VFF.

Nesta simulação o robô parte do ponto (0,0) em direção ao ponto destino (9000, -250) (dimensões em mm) através de um corredor que possui, inicialmente, $1,5$ m de largura. Durante seu trajeto, ele encontra uma perturbação (representada pelo estreitamento abrupto do corredor) e, em seguida, um corredor de 1 m de largura — lembrando que o veículo ocupa praticamente a metade desse espaço. Depois de percorrer os 9 m de corredor, praticamente não há oscilações bruscas na trajetória resultante descrita pelo robô, como visto na Figura 17. Os ganhos utilizados no controlador de posição final foram $k_u = 0,3$ e $k_\omega = 0,9$. A zona de repulsão foi estabelecida em 30 cm. As Figuras 18, 19 e 20 apresentam as velocidades linear e angular e a orientação do robô ao longo da trajetória, respectivamente.

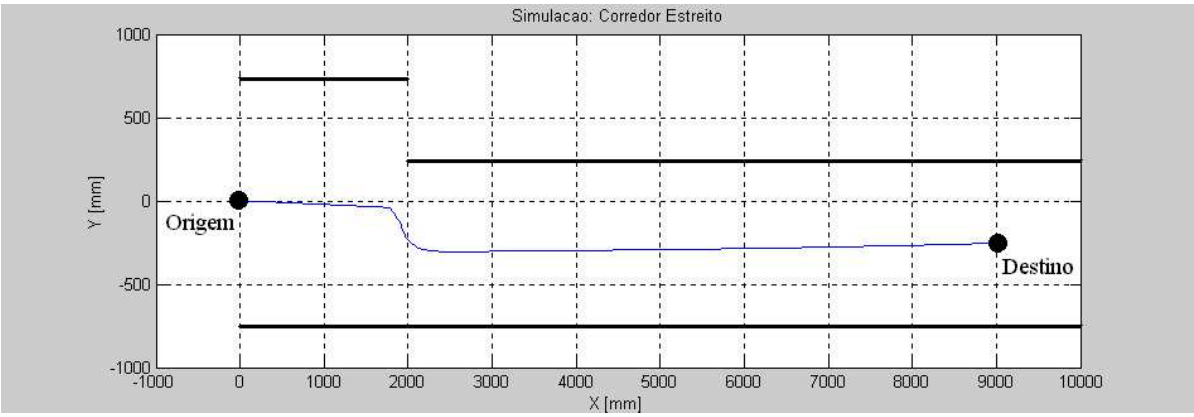

Figura 17: Trajetória do robô ao longo do corredor estreito.

3.2.2 Navegação em Corredores e Seguimento de Paredes

O objetivo, nesta simulação, é alcançar o ponto de coordenadas (8000,4000) (valores expressos em milímetros), evitando os obstáculos presentes na trajetória. A Figura 21 mostra a trajetória descrita pelo robô numa seqüência de corredores, até chegar ao ponto destino. Assim que as paredes são detectadas, o robô se orienta, de modo a ficar numa posição paralela às mesmas. A distância a partir da qual o robô entra na zona de repulsão foi estabelecida em 70 cm. A Figura 22 e a Figura 23 mostram os sinais de controle u (velocidade linear) e w (velocidade angular) gerados pelo controlador de posição final,

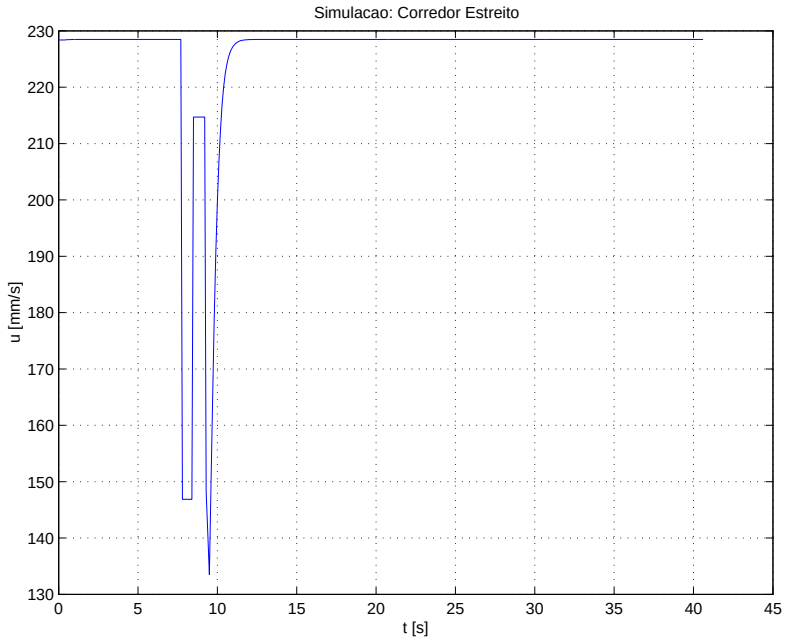

Figura 18: Velocidade linear do robô no corredor estreito.

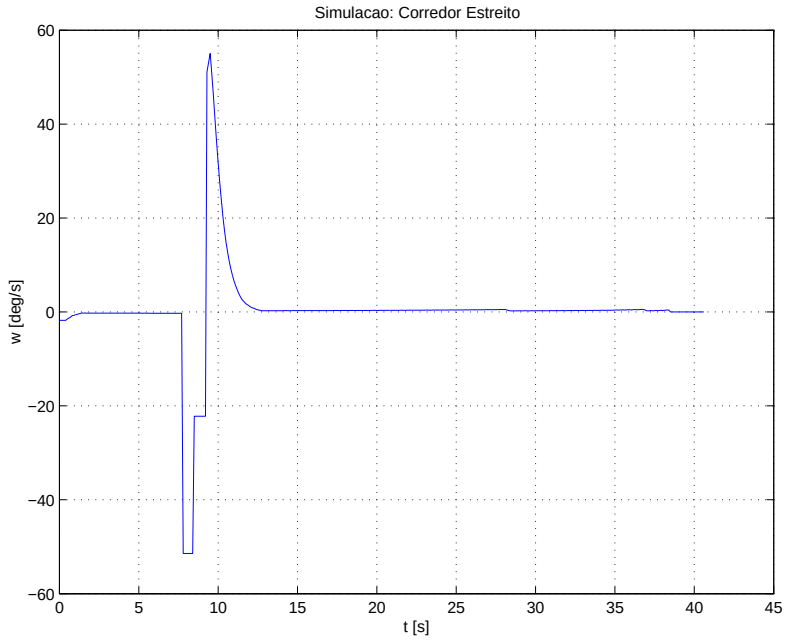

Figura 19: Velocidade angular do robô no corredor estreito.

que permitem ao robô evitar obstáculos. A Figura 24 mostra o comportamento do robô através da evolução temporal de sua orientação.

Uma observação importante acerca deste exemplo é que durante os períodos nos quais o robô evita obstáculos, principalmente entre os instantes de tempo 4 s e 23 s, os valores de velocidade linear u, velocidade angular w e orientação do robô θ permanecem prati-

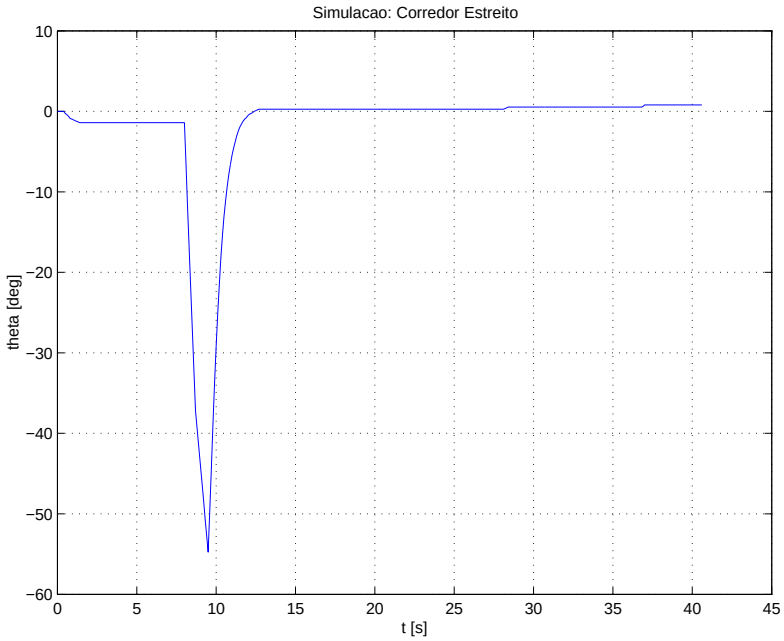

Figura 20: Orientação do robô no corredor estreito.

camente constantes, sendo que a velocidade angular fica com valores muito próximos de zero. Adicionalmente, como se pode ver na Figura 21, a trajetória executada é bastante adequada.

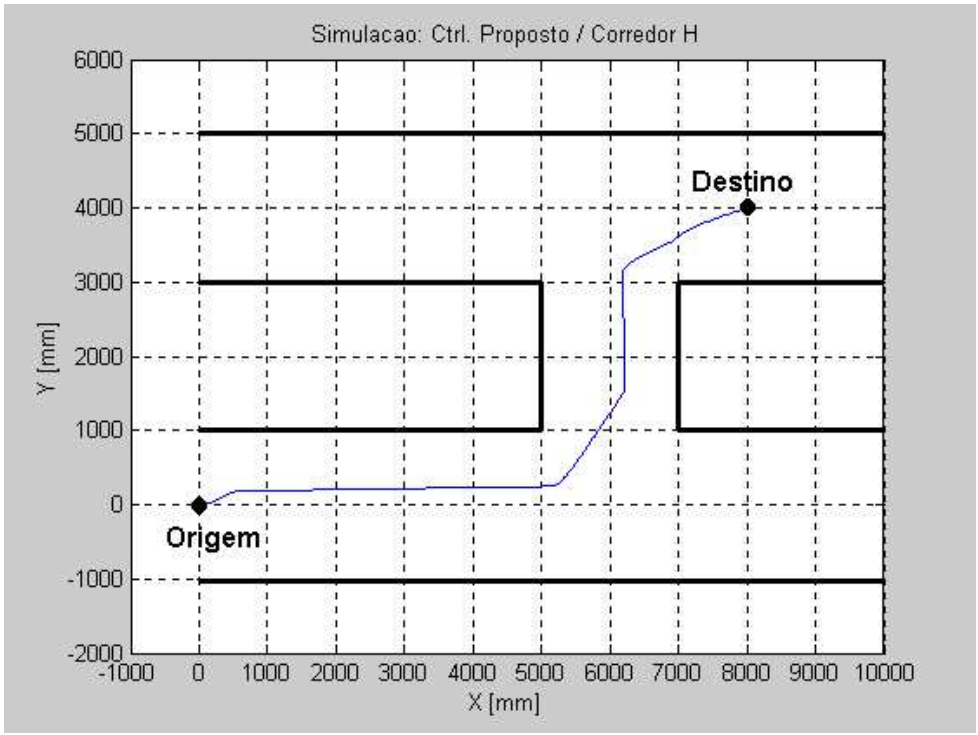

Figura 21: Trajetória do robô.

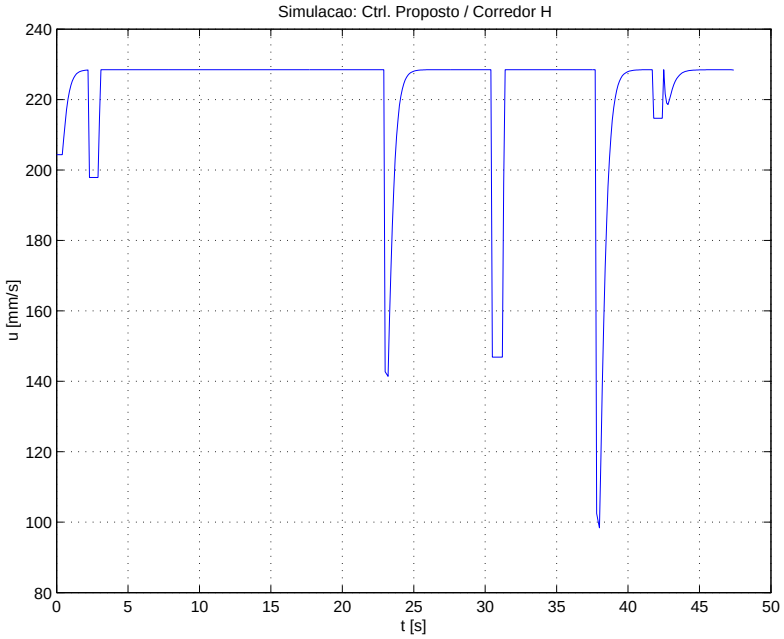

Figura 22: Velocidade linear do robô.

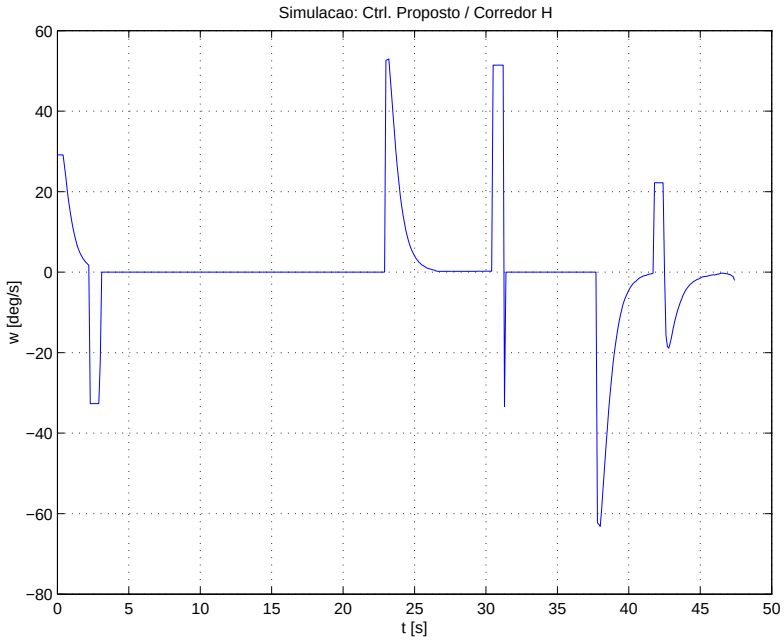

Figura 23: Velocidade angular do robô.

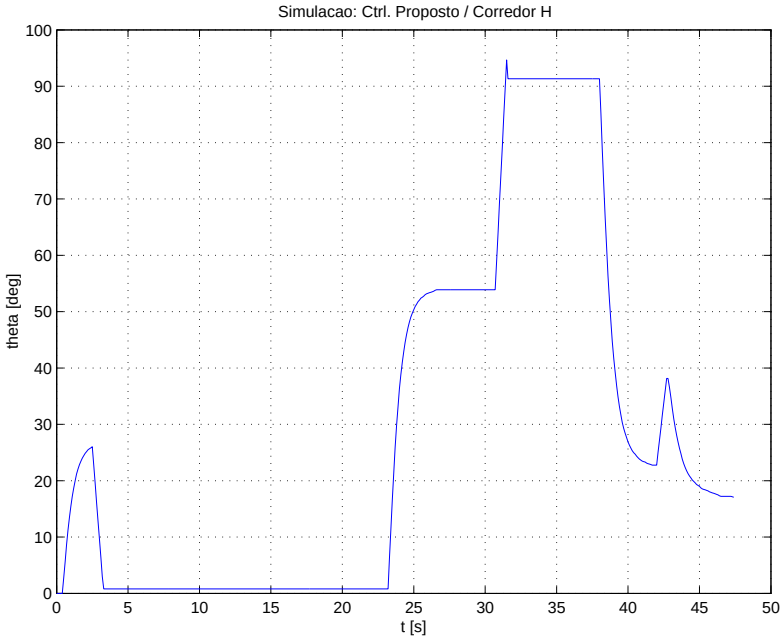

Figura 24: Orientação do robô.

4 Resultados e Discussões

Este capítulo é dedicado à discussão dos exemplos simulados apresentados nos capítulos 2 e 3, assim como à discussão de resultados obtidos a partir da implementação dos sistemas correspondentes a bordo de um robô móvel comercialmente disponível.

4.1 O Simulador Utilizado

Os resultados simulados apresentados nos Capítulos 2 e 3 foram obtidos usando o *SRIsim*, simulador que acompanha o robô *PIONEER 2-DX*. Este simulador utiliza modelos de erro realísticos para os sonares e *encoders* das rodas, fazendo com que os resultados de simulações e experimentos fiquem razoavelmente próximos. A partir do início do movimento do robô, o sistema "lê" os dados dos sensores ultra-sônicos e executa o sistema de controle em ciclos que se repetem a cada 100 ms. A interface deste simulador é apresentada na Figura 25.

Aliado ao *SRIsim* está o *software Mapper*, uma ferramenta para criação e edição de *layouts* bidimensionais, propiciando diferentes condições de teste e melhor caracterização do ambiente no qual o robô irá navegar. Este *software* permite ainda definir o ponto de partida do robô (sempre caracterizado nas simulações pelo par cartesiano (0,0)), bem como sua orientação. Depois de criados (ou modificados), esses *layouts* podem ser importados pelo simulador. Uma imagem desta ferramenta é mostrada na Figura 26.

Outro detalhe importante é que as versões utilizadas, tanto do *SRIsim*, quanto do *Mapper*, são compatíveis com o sistema operacional Linux, uma vez que a idéia é implantar o sistema de controle proposto no robô móvel, o qual utiliza este sistema operacional.

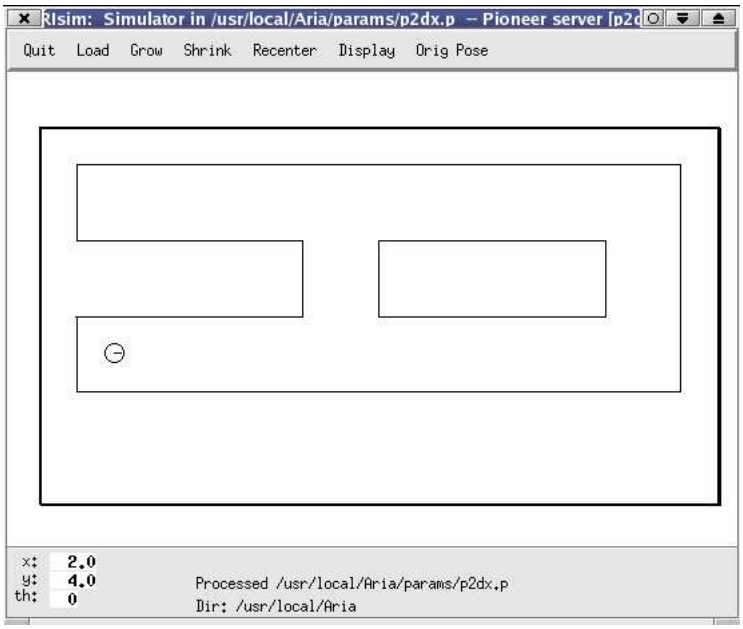

Figura 25: Simulador *SRIsim* utilizado.

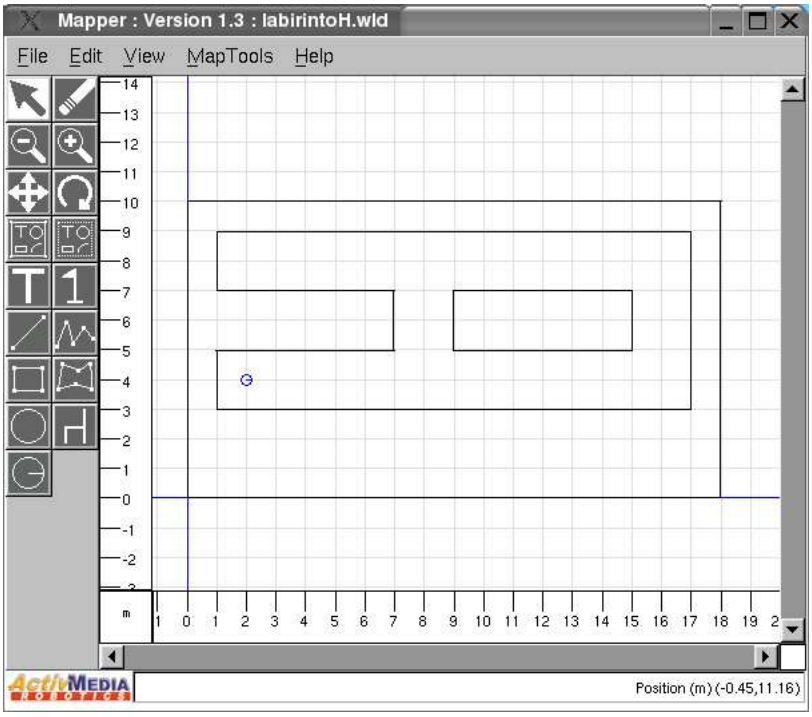

Figura 26: *Mapper*: Ferramenta para criação/edição de mapas.

4.2 Discussão dos Resultados da Simulação

Analisando-se as figuras relativas aos resultados associados à arquitetura de controle baseada em impedância original e à nova arquitetura de controle aqui discutida, podem-

se perceber algumas diferenças marcantes entre as duas abordagens. Primeiramente, enquanto o robô evita obstáculos, a arquitetura aqui proposta assegura uma velocidade linear constante, com o que não há aceleração e frenagem desnecessárias dos motores, reduzindo seu desgaste e o consumo de bateria.

O desvio tangencial do obstáculo também permite que o ponto de destino seja alcançado em menor tempo, já que a trajetória é melhorada, evitando-se oscilações desnecessárias na orientação do robô, assim como o perfil de velocidade linear. De fato, a velocidade angular do robô permanece nula, e sua orientação permanece constante, enquanto ele evita as paredes superior e lateral direita, presentes na Figura 21, por exemplo, assim como sua velocidade linear é máxima.

Em relação ao exemplo relacionado a corredores estreitos (ver Figura 17) é importante notar que os valores de velocidade angular, velocidade linear e orientação do robô permanecem razoavelmente constantes enquanto o veículo navega pelo corredor estreito. Mais ainda, quando são comparadas as trajetórias obtidas nas Figuras 17 e 3, torna-se claro, nesta situação, o bom desempenho e a superioridade da arquitetura aqui proposta, em relação a métodos que utilizam a teoria de campos potenciais, tal como o VFF.

Assim sendo, a arquitetura aqui proposta para evitar obstáculos é extremamente atrativa. Por fim, o próprio exemplo correspondente à Figura 21 permite visualizar que o mesmo sistema de controle aqui proposto permite ao robô seguir paredes e navegar ao longo de corredores.

4.3 A Implementação a Bordo do Robô

Além dos resultados de simulação apresentados nos capítulos anteriores, a arquitetura de controle apresentada na Figura 16 foi implementada a bordo de um robô móvel, e vários experimentos foram realizados, com o objetivo de validar a estratégia aqui proposta.

Nos experimentos, foi utilizada a plataforma comercial *PIONEER 2-DX*, que faz parte de uma família de robôs móveis da *ActivMedia Robotics*, cuja arquitetura foi originalmente desenvolvida por Kurt Konolige do *SRI International, Inc.* e por pesquisadores da Universidade de *Stanford*.

4.3.1 Hardware

A estrutura básica deste robô (Figura 27) é composta por um microcontrolador Siemens 88C166, de 20 MHz, com 32 $KBytes$ de *FLASH ROM* e 32 $KBytes$ de *RAM* dinâmica [23]. Para tarefas que demandem maior capacidade de processamento, existe um computador de bordo com processador PENTIUM MMX, de 266 MHz, bem como um painel com conexões disponíveis para periféricos (teclado, mouse, monitor e conexão de rede). Uma câmera de vídeo Sony PTZ (*Pan-Tilt-Zoom*) está integrada ao robô, para aplicações que utilizem visão computacional, além de um sistema de rádio Ethernet para comunicação remota.

Em relação ao sensoriamento, o robô apresenta, além da câmera, 16 transdutores de ultra-som, tipo eletrostáticos, distribuídos radialmente, conforme a Figura 28. A seqüência dos disparos é a seguinte: um sonar frontal e um sonar traseiro são gatilhados simultaneamente, esperam o eco por no máximo 40 ms (na prática são 32 ms), e um novo par de sonares é gatilhado. Este procedimento tem a função de evitar o fenômeno descrito como *crosstalk*, no qual o sinal de eco de um sonar é recebido por outro transdutor vizinho, gerando medidas errôneas. Embora o circuito responsável por gerar os sinais emitidos pelos sonares trabalhe a 25 Hz, a freqüência de gatilhamento dos sonares é de aproximadamente 3 Hz. A faixa de operação dos sonares está entre 10 cm (na prática, 20 cm) e 5 m, aproximadamente. Esta sensibilidade pode ser alterada ajustando-se os ganhos no módulo específico.

Ainda em relação ao *hardware*, vale a ressalva sobre dispositivos de proteção do veículo utilizado. O Sistema Operacional do robô (*P2OS*), que é executado no microcontrolador, possui um *watchdog* que monitora a comunicação cliente-servidor. Se esta comunicação for interrompida por mais de 2 s (valor nominal, mas que é configurável), o movimento do robô é cancelado, até que a comunicação seja restaurada. Ele também faz a monitoração do travamento das rodas. Se o *driver* gera um pulso PWM igual ou maior que um dado valor configurado e as rodas não giram, o fornecimento de potência ao motor é interrompido por um determinado período de tempo (também configurável), e só depois é reestabelecido [23].

No que tange à saturação dos controladores, cabe lembrar que o sistema proposto faz uso de restrições nas equações (Equações 2.5) dos sinais de controle (velocidade linear u e a velocidade angular ω), limitando estes sinais. Na plataforma utilizada (*PIONEER 2-DX*), esses valores máximos são de 2200 mm/s e 500°/s, respectivamente.

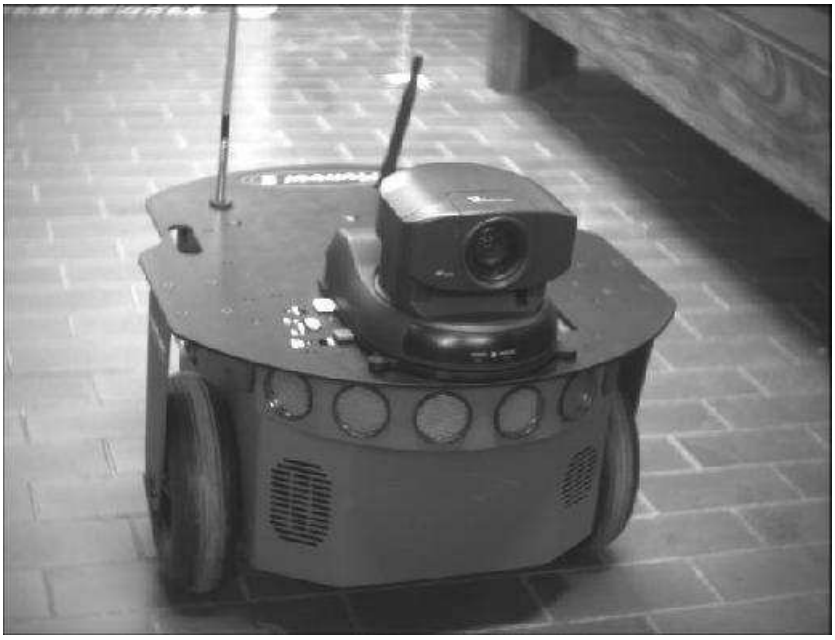

Figura 27: Robô móvel *PIONEER 2-DX*.

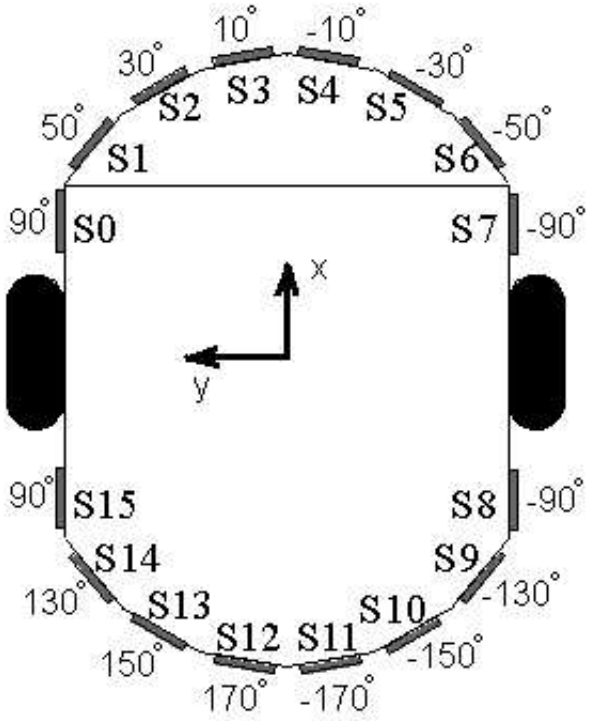

Figura 28: Distribuição dos sonares no robô *PIONEER 2-DX*.

4.3.2 Software

Quanto à programação do robô, ela pode ser feita de duas formas: programação direta do microcontrolador 88C166 ou através do uso de interfaces de alto nível, como ARIA ou

SAPHIRA. Por ser mais amigável e possuir grande flexibilidade quanto à programação (além de outras vantagens descritas adiante), a opção escolhida foi a interface ARIA. Neste caso, rotinas clientes são criadas e devem estabelecer uma conexão tipo cliente-servidor com um servidor, que pode ser o Simulador *SRIsim* ou o *P2OS* (Sistema Operacional em execução na *ROM* do 88C166) [24].

ARIA (*ActivMedia Robotics Interface for Application*) é uma interface de programação de código aberto com orientação a objetos para aplicações com robôs móveis da *ActivMedia*. Ela contém classes e funções que proveêm o suporte necessário, desde o acesso a baixo nível e o controle de robôs e efetuadores até funções de alto nível, como teleoperação e controle através de ações de comportamento (*behavioral actions*). ARIA foi projetada para servir como uma base versátil para programas de alto nível. A interface ARIA é distribuída sob Licença Pública GNU, significando que se algum trabalho que utilize ARIA é distribuído, todo o código fonte do mesmo deve ser disponibilizado em conjunto.

Saphira, por outro lado, é um ambiente de alto nível, tendo ARIA como base, permitindo que suas classes e funções sejam utilizadas quando programando em Saphira. A interface Saphira é desenvolvida e mantida pela *SRI* e não pela *ActivMedia*. Saphira não possui código aberto.

Dentre as vantagens de se utilizar a interface ARIA, podem ser citadas a sua constante atualização (*software* e documentação), o fato de ser completamente *open-source*, permitindo ajuste e adequação do código-fonte para melhor atender a determinadas aplicações, e, por fim, a grande disponibilidade de suporte, principalmente através de listas de discussão.

Nesta Dissertação, as aplicações-cliente foram escritas em C++, tendo como base a interface ARIA e o Sistema Operacional Linux. Esses clientes utilizavam ora o Simulador, ora o *P2OS* (no microcontrolador) como servidor.

O sistema operacional Linux foi utilizado no desenvolvimento deste trabalho, tanto nos PCs que geraram simulações, quanto no PC embarcado no robô. Este sistema possui várias características desejáveis em aplicações com robôs móveis, dentre elas: pode ser executado sem problemas em máquinas sem teclado e *display*; os *drivers* de dispositivo possuem uma interface padrão e podem ser escritos usando apenas um compilador C, ao contrário de outros sistemas que requerem *kits* específicos para desenvolvimento de *drivers* (*Driver Development Kit - DDK*), os quais nem sempre estão prontamente disponíveis; por ter código-fonte aberto, pode ser modificado para melhor atender aos requerimentos específicos de *hardware* e possui um sistema de rede muito flexível e completo [25].

Quanto à programação das aplicações-cliente, a arquitetura Saphira/Aria disponibiliza dois tipos de rotinas de usuário. O primeiro tipo utiliza micro-tarefas (*micro-tasks*), que são executadas em modo síncrono. Este processo se dá através da criação de uma classe contendo uma função (método) a ser executada pelo robô a cada 100 *ms* e um *functor*, um tipo de ponteiro que aponta para o método criado. Quando a classe é inicializada, o *functor* é passado para o robô, que, por sua vez, armazena o ponteiro em uma lista. A cada 100 *ms* o robô executa todas as funções (em ordem de prioridade) para as quais os *functors* da lista apontam. Com a realização de alguns testes foi possível perceber alguns pontos importantes desse tipo de implementação: a ordem de prioridades realmente é cumprida; se uma tarefa de baixa prioridade está sendo executada e o tempo de 100 *ms* se esgota, mesmo que haja uma tarefa de maior prioridade pronta, esta somente será executada depois que toda a lista de tarefas é executada, independente do tempo que seja gasto. Portanto, nenhuma restrição temporal é garantida. Para uso desta implementação é necessário que as rotinas sejam escritas de modo que todas as tarefas se ajustem dentro de um período de 100 *ms*.

O outro tipo de rotina de usuário é apresentado na forma de *rotinas assíncronas*. Essas rotinas são independentes do ciclo síncrono de 100 *ms*, e seu uso deve ser amplamente explorado para aplicações que demandem grande esforço computacional, como aplicações em visão, por exemplo. A Figura 29 mostra o esquema de execução das tarefas de usuário, sejam elas síncronas ou assíncronas.

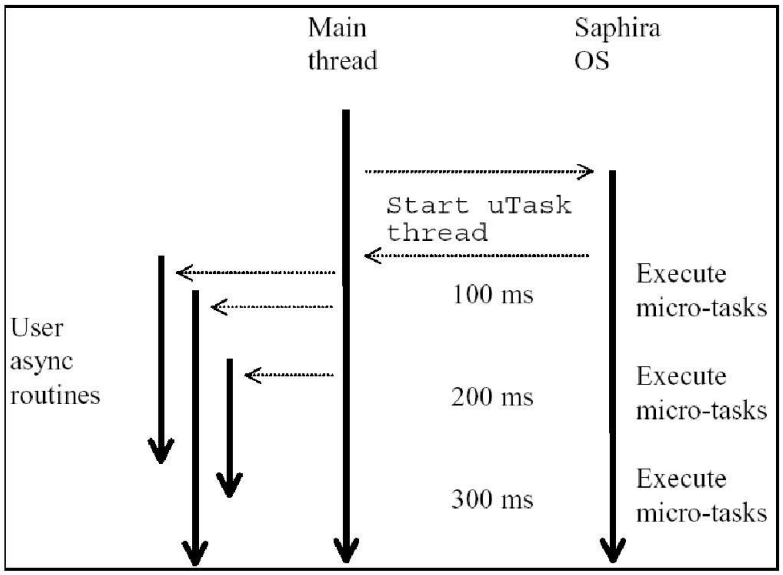

Figura 29: Execução concorrente do Saphira OS e tarefas assíncronas dos usuários [23].

Nos experimentos realizados durante o desenvolvimento desta Dissertação, as rotinas

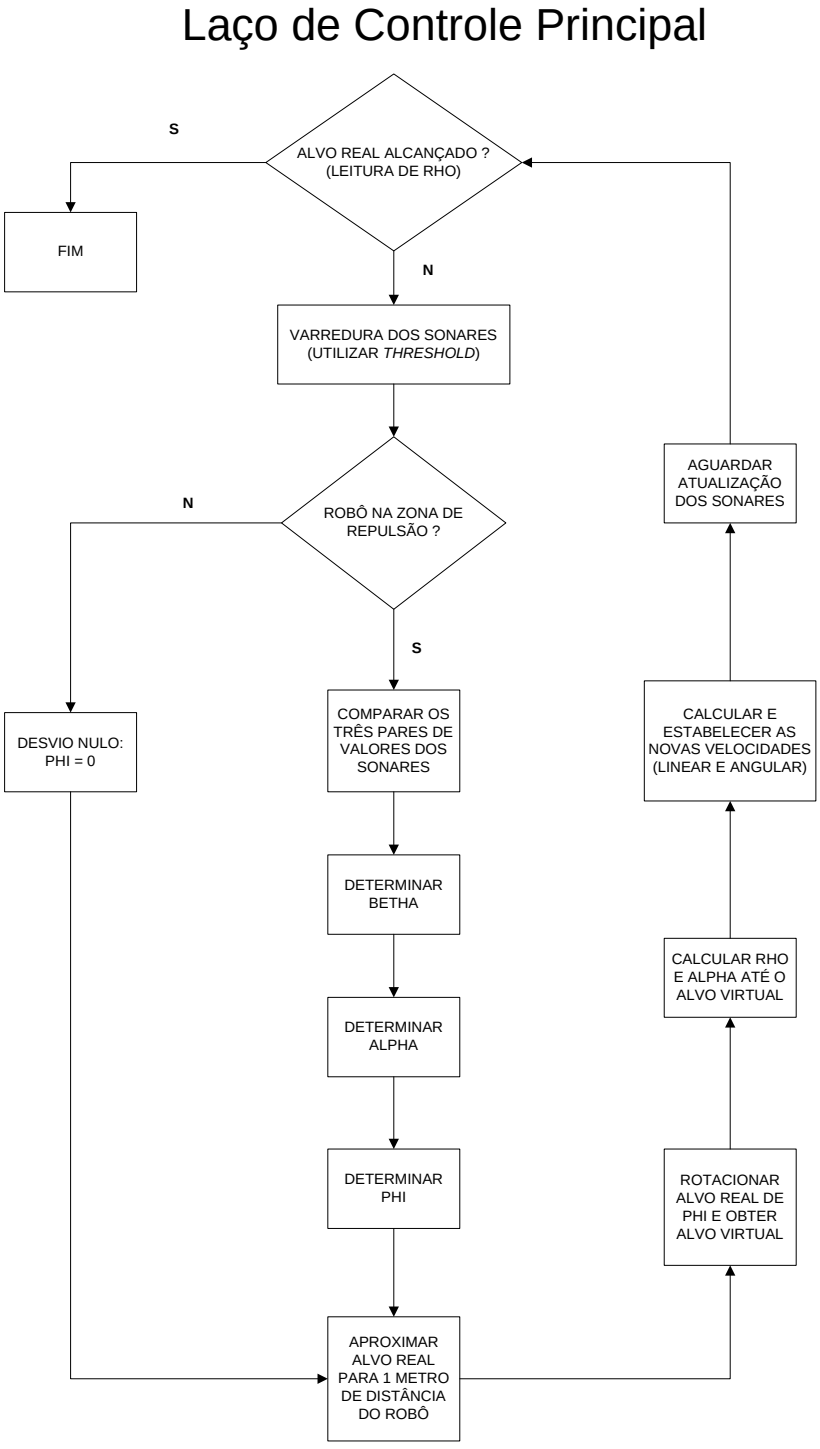

Figura 30: Laço de controle principal.

de controle do robô foram implementadas usando a forma assíncrona. No laço de controle principal da aplicação-cliente, uma rotina se encarrega da medição do tempo gasto até um

determinado ponto deste laço, e depois faz com o que o sistema aguarde até a próxima amostragem dos sensores. Esta estrutura pode ser visualizada na Figura 30. Os programas são apresentados no Apêndice.

4.4 Resultados Experimentais

Para validação da proposta apresentada nesta Dissertação, foram realizados alguns experimentos. Para tal, foi utilizada a plataforma comercial *PIONEER 2-DX*, já descrita.

4.4.1 Experimento 1

Este experimento é relativo à navegação em corredores e seguimento de paredes. Foi montada uma configuração em forma de H com corredores possuindo larguras de 2 e 3 metros, conforme a Figura 31. Ao robô foi designada a tarefa de navegar até o ponto de coordenadas $(9000mm, 5000mm)$, partindo do ponto (0,0). A zona de repulsão foi estabelecida em 70 cm. Inicialmente são apresentados os resultados do experimento e, em seguida, são mostrados os gráficos da simulação correspondente. Por fim, é apresentado um gráfico das trajetórias descritas pelo robô durante a simulação e o experimento (Figura 39), comparando os resultados obtidos.

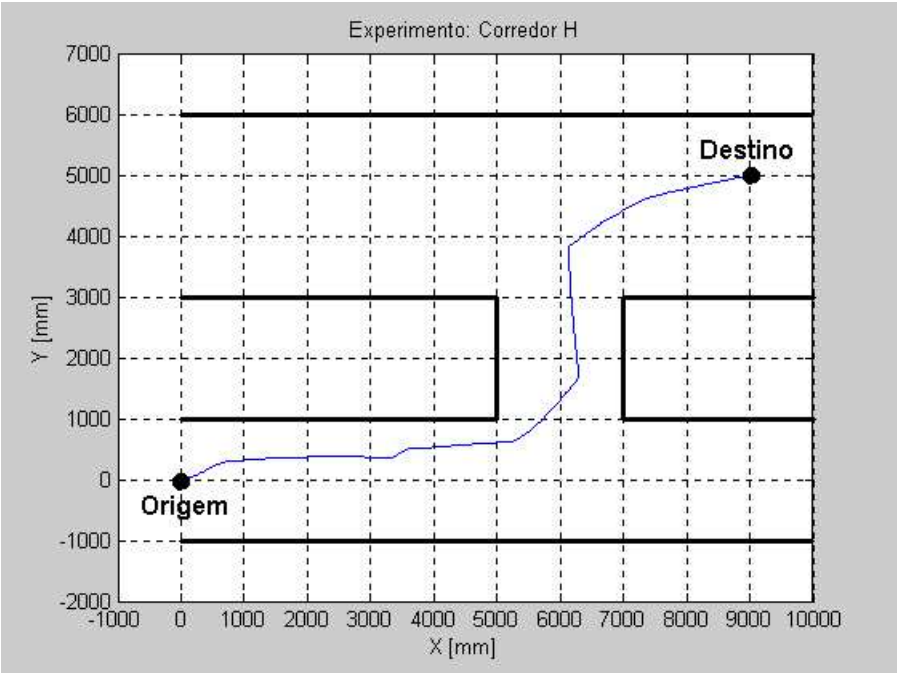

Figura 31: Trajetória do robô.

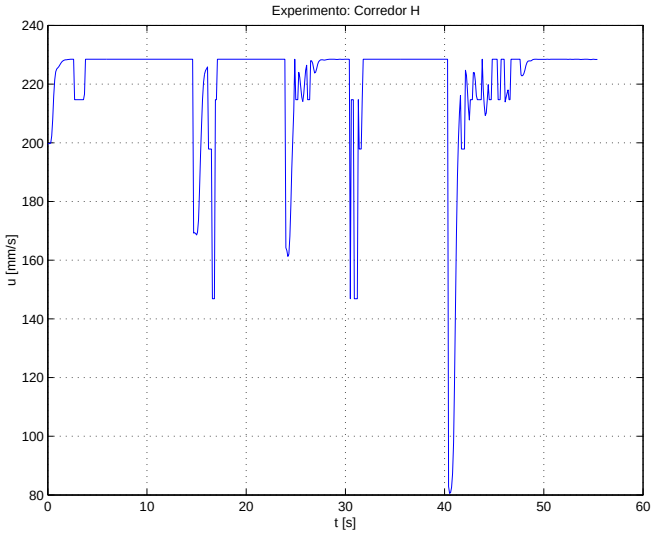

Figura 32: Velocidade linear u.

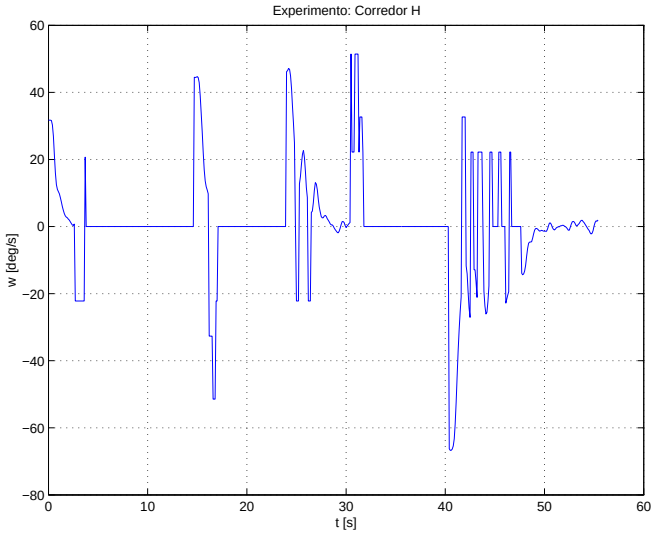

Figura 33: Velocidade angular ω.

4.4.2 Experimento 2

Este experimento mostra o comportamento do robô quando submetido a uma navegação *tipo U* e com presença de obstáculos, além das paredes dos corredores. A configuração de corredor utilizada é a mesma do experimento anterior, diferenciando-se apenas em relação ao ponto de destino, que agora possui coordenadas $(1000mm, 5000mm)$, além do acréscimo de dois obstáculos cilíndricos de raios 40 cm e 30 cm situados em $(0mm, 2500mm)$ e $(6000mm, 3000mm)$, respectivamente. O *layout* do ambiente de trabalho correspondente a este experimento é representado na Figura 40, juntamente com a trajetória que o robô descreve. A zona de repulsão é de 70 cm.

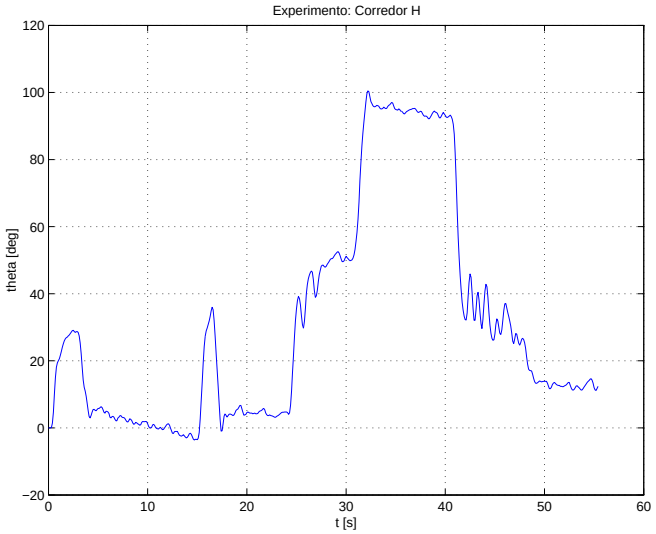

Figura 34: Orientação do robô.

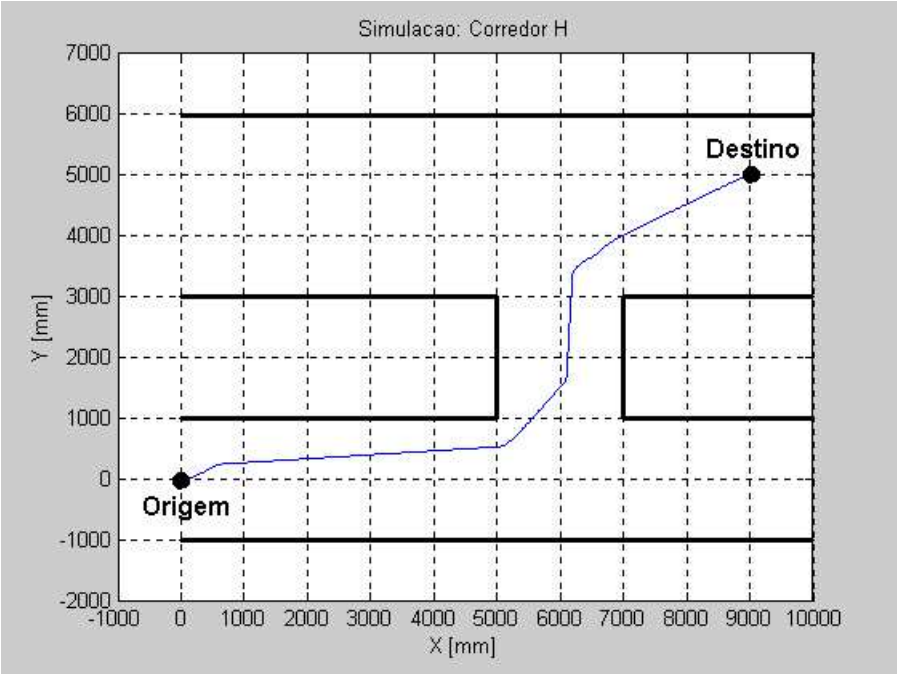

Figura 35: Trajetória do robô.

A seguir, são apresentados resultados de simulação para o mesmo problema, utilizando, inicialmente, o controlador proposto (Figura 41), e, em seguida, o controlador baseado em impedância (Figura 42).

Através da Figura 43 pode-se fazer uma comparação das trajetórias descritas pelo robô, no experimento e nas duas simulações.

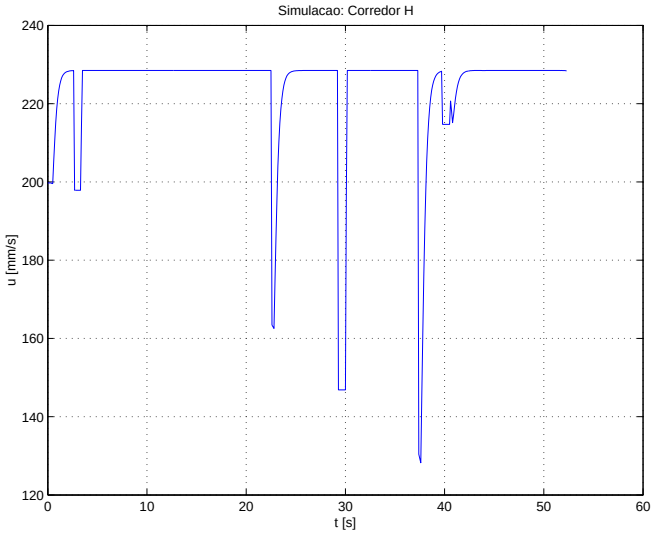

Figura 36: Velocidade linear u.

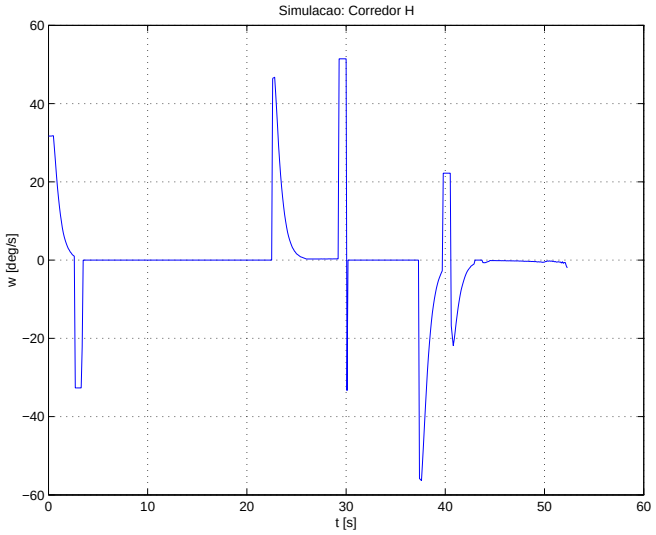

Figura 37: Velocidade angular ω.

4.5 Discussão dos Resultados dos Experimentos

Como ocorrido em algumas simulações, um detalhe interessante se repete no Experimento 1. Em determinadas situações, quando o robô se depara com um obstáculo e se reorienta de modo a tangenciá-lo, às vezes não consegue ficar totalmente paralelo ao mesmo, permanecendo um desvio de poucos graus. Isso pode ser observado na simulação apresentada na Figura 17 e também neste experimento, quando o robô passa pelo marco $X = 1000$ (Figura 31). Nesses casos, embora o robô descreva uma trajetória retilínea, esse caminho não é exatamente paralelo à parede. Isto se deve ao fato de que, quando o

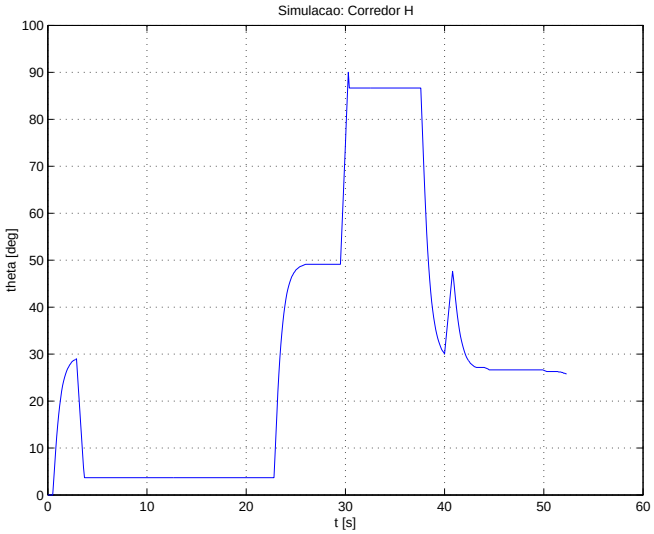

Figura 38: Orientação do robô.

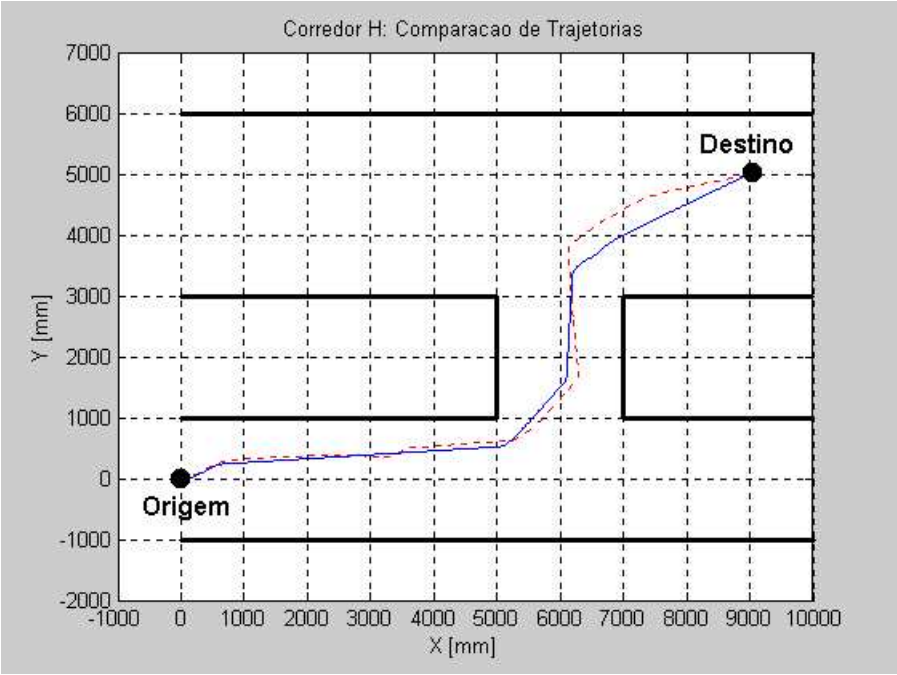

Figura 39: Posição do robô. Linha tracejada: experimento. Linha contínua: simulação

veículo se reorienta e fica quase paralelo ao obstáculo, o transdutor que registra o menor valor é o lateral, gerando um ângulo β de 90 ou -90 graus. Nesta situação, o algoritmo para desvio de obstáculos entende que o robô está na orientação correta. No entanto, há um desvio de alguns graus, fazendo com que o robô continue seu caminho como se estivesse paralelo ao obstáculo, ainda que não esteja.

Ainda considerando o Experimento 1, após passar pelo marco $X = 1000$ e em virtude

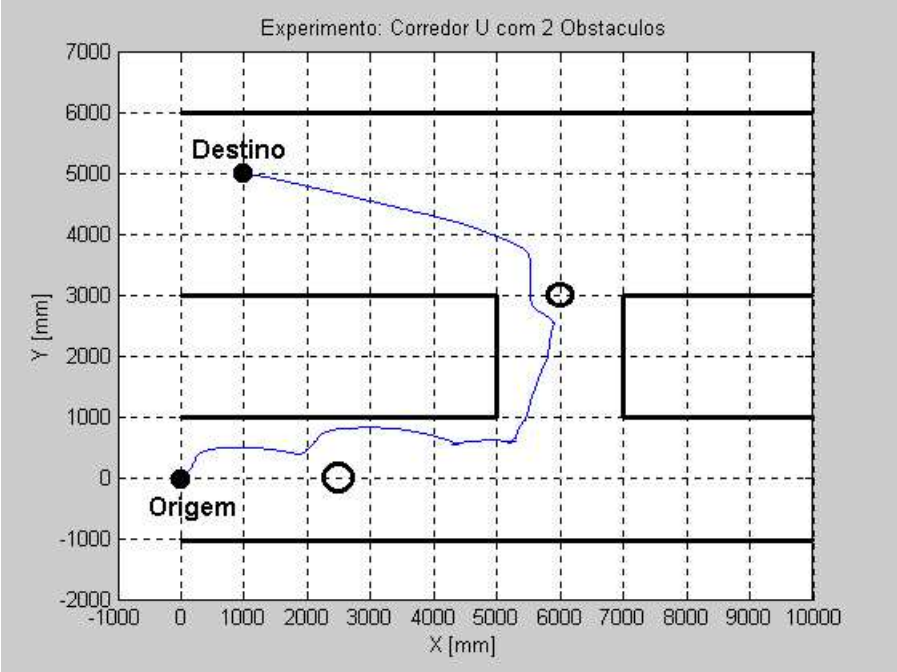

Figura 40: Experimento utilizando o controlador proposto.

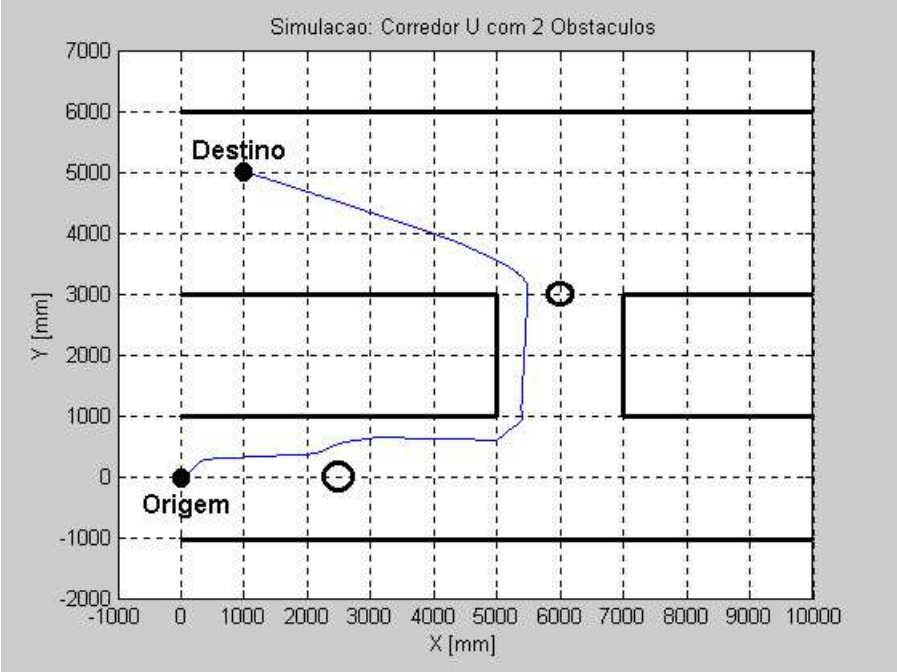

Figura 41: Simulação utilizando o controlador proposto.

de não ficar completamente paralelo ao obstáculo, o robô afasta-se lentamente dele, até chegar próximo ao marco $X = 3500$ (Figura 31). Neste instante, ele se encontra fora do zona de repulsão e, novamente, o alvo real passa a governar sua trajetória, ou seja, o robô se orienta e inicia o percurso até o seu destino. No entanto, ele novamente entra na zona

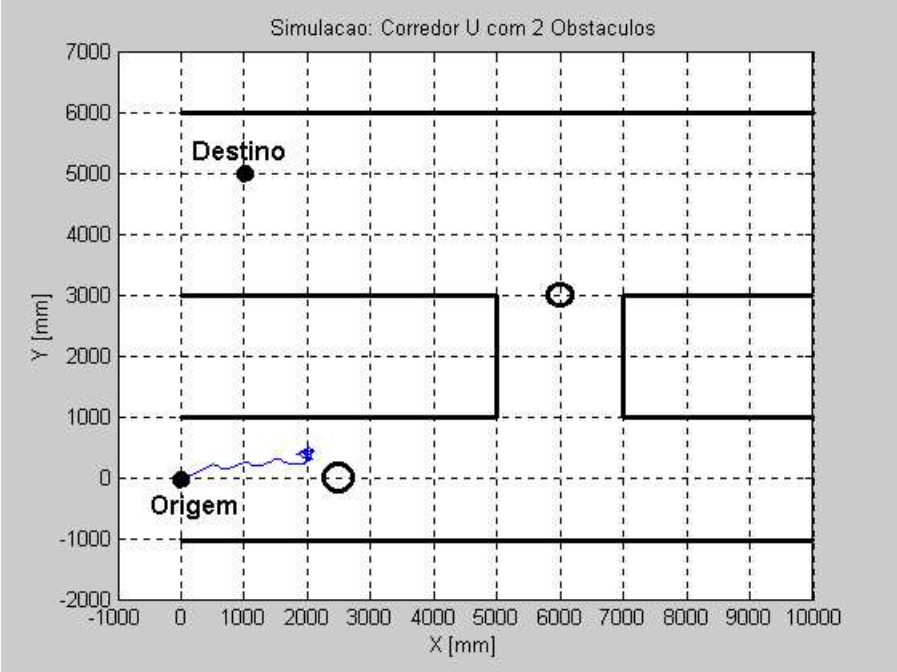

Figura 42: Simulação utilizando o controlador baseado em impedância.

de repulsão, fato que gera nova orientação de forma a ficar paralelo à parede.

A imprecisão no momento de se orientar de forma totalmente paralela ao obstáculo (ver, por exemplo, Figura 17) é proveniente da dificuldade encontrada na obtenção do ângulo β. Isto se deve, principalmente, à distribuição irregular dos sensores ao redor do robô (Figura 28). A medida precisa deste valor é fundamental para um desvio angular correto e controlado — no caso, um desvio tangencial.

No intuito de amenizar esta deficiência, sempre que o ângulo β é necessário, é feita uma varredura nos sonares para verificar qual deles indica maior proximidade de obstáculos. Uma vez encontrado este sonar, os transdutores vizinhos são consultados, de modo a saber qual par, dentre as duas possibilidades resultantes da combinação do sonar em questão seus dois vizinhos, possui valor mais próximo. O novo valor de β é obtido através da média dos valores de posição angular dos sensores desse par. Assim, mesmo que não haja um sonar, por exemplo, a zero graus (exatamente à frente do robô), se os sensores de número 3 e 4 (Figura 28) tiverem medidas bem próximas, a indicação será de um provável obstáculo logo à frente. A localização dos transdutores no robô é fixa e é obtida através de funções da interface ARIA.

O Experimento 2 possui uma particularidade interessante. Devido à localização dos pontos de origem e destino, e também em função da disposição das paredes, o robô deve

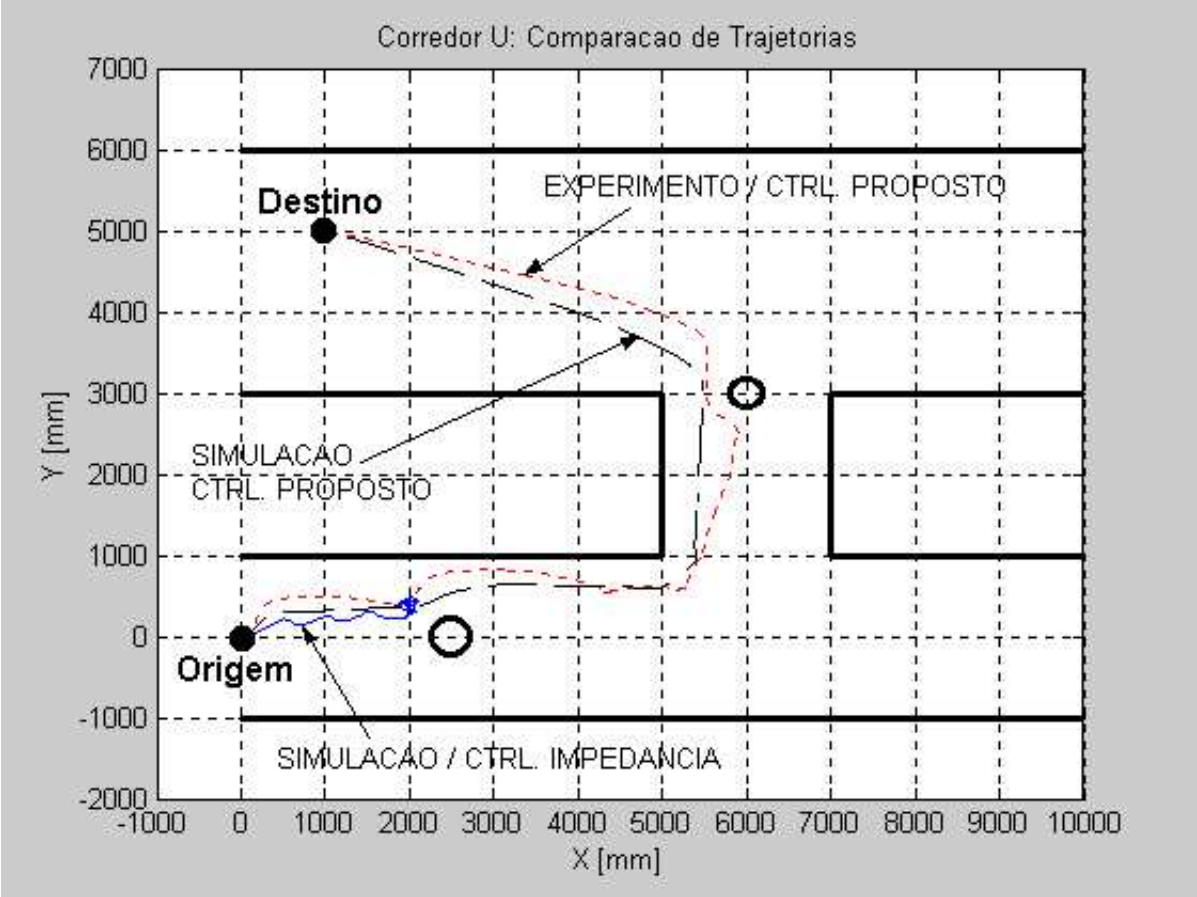

Figura 43: Comparação dos resultados.

percorrer uma trajetória em forma de U. Nesta situação, o robô tende a voltar assim que a distância ao ponto destino começa a aumentar ao invés de diminuir. Isso porque o controlador de posição final tende a minimizar o valor de ρ. Portanto, num certo trecho o robô tem que se afastar do alvo, para depois voltar a dele se aproximar. Na Figura 41 isto ocorre quando o robô cruza o ponto de abcissa $X = 1000mm$, e a partir daí ele começa a afastar-se do alvo. Assim é que tal ponto representa um ponto de mínimo local, pois ali o robô está a uma distância mínima do alvo, e dele passa a se afastar. Em outras palavras, o sistema proposto permite que o robô saia de pontos de mínimo local existentes ao longo de sua trajetória. Isto se deve ao fato de que o alvo passa a ocupar uma posição fictícia em frente ao robô, na presença de obstáculos, de forma que o sistema passa a contornar o obstáculo como objetivo principal.

Neste experimento também é observado o comportamento do robô quando em presença de obstáculos no corredor. Assim que eles são detectados o robô inicia uma reorientação, de forma a tangenciá-los.

Outra questão importante apresentada neste experimento refere-se à zona de repulsão, melhor chamada de zona de desvio de obstáculos. As zonas de desvio utilizadas foram de 70 *cm*. Em certas situações existe inclusive interferência entre essas zonas, o que não representou qualquer problema ao desempenho do robô.

No caso dos dois obstáculos presentes nos corredores, as zonas de repulsão correspondentes às paredes e aos objetos cilíndricos têm uma região de intersecção, como se pode inferir do *layout*. Mesmo neste caso, como se pode ver na Figura 40 e na Figura 41, o sistema proposto é capaz de fazer com que o robô ultrapasse o gargalo, o que não ocorre no caso do controle baseado em impedância, como se vê na Figura 42. Também vale ressaltar que experimentos realizados com o controle baseado em impedância também tiveram o mesmo desfecho verificado na Figura 42, ou seja, o robô não conseguiu ultrapassar o primeiro obstáculo.

Erros de odometria foram constantes durante os experimentos, sendo "cumulativos". Em alguns casos, pontos de destino não eram alcançáveis, pois, de acordo com a odometria do robô, eles estavam atrás de paredes. Uma fusão de sinais provenientes dos *encoders* e dos sonares talvez pudesse ser utilizada para obtenção de dados mais confiáveis. Entretanto, a proposta deste trabalho é justamente voltada para ambientes dinâmicos, nos quais medidas odométricas sem um mapeamento prévio não seriam muito atrativas.

Os pequenos erros, mostrados na Figura 39 e na Figura 43, se devem principalmente a erros relativos à odometria, praticamente inexistentes no simulador, mas constantes na plataforma móvel. Além disso, a baixa precisão dos sonares influencia no instante em que um obstáculo é detectado, podendo gerar, portanto, reorientações do robô em pontos diferentes. No entanto, mesmo com as dificuldades encontradas, as trajetórias descritas foram consideradas adequadas e de acordo com o objetivo proposto neste trabalho.

Variações na velocidade linear do robô ocorrem em virtude deste valor ter dependência com o ângulo α, ou seja, quando o alvo virtual sofre uma mudança grande de posição, essa variação angular do alvo se manifesta através da diminuição da velocidade linear.

Os comportamentos *Evitar Paredes* e *Navegação em Corredores* ficaram evidentes tanto na simulação quanto no experimento. Essas ações são obtidas apenas realizando o desvio de obstáculos, no caso representados pelas paredes do corredor, de forma tangencial.

5 Conclusões

Uma arquitetura que permite a um robô móvel desviar-se de obstáculos em seu caminho de maneira eficiente é apresentada nesta Dissertação. Ao invés de simplesmente afastar-se do obstáculo, o robô procura tangenciá-lo, até sobrepassá-lo. A partir daí, ele volta a buscar o seu alvo, até que novo obstáculo surja em seu caminho.

A arquitetura proposta é implementada através de um controlador, cuja estrutura é adaptada de uma arquitetura baseada em impedância, diferindo apenas na forma como o ângulo de orientação do robô é modificado: ele agora é calculado de forma que o robô se orienta paralelamente ao contorno do obstáculo. O controlador assim implementado é estável no sentido de Lyapunov (assim como o controlador baseado em impedância original), o que quer dizer que o robô sempre alcançará seu objetivo final, independente dos obstáculos em seu caminho (sempre que o ponto de destino for alcançável). Com base nos exemplos apresentados, assim como em outros que foram também implementados, as vantagens da nova arquitetura proposta, o desvio tangencial de obstáculos, são visíveis: a navegação é bem mais suave e eficiente (no sentido de que o tempo gasto é menor, assim como o consumo de energia e o desgaste dos motores).

Entretanto, para determinar com maior precisão a posição angular dos obstáculos em relação ao robô — medição do ângulo β — é necessária uma melhor resolução nas medidas provenientes do anel de sensores ultra-sônicos. Isto pode ser obtido com um maior número de sensores e uma melhor distribuição angular dos mesmos ao redor do robô, o que não se pode obter com o equipamento utilizado, no qual o espaçamento angular entre os sensores não é constante. Note-se, ainda, que o custo associado à utilização de mais sensores não é proibitivo.

Ainda como pontos importantes desta arquitetura podem ser citados: capacidade de guiar o robô em algumas situações de mínimos locais e através de obstáculos próximos entre si; flexibilidade quanto ao tipo de sensor utilizado, permitindo, *a priori*, que qualquer sensor para medição de distância possa ser empregado neste sistema e a versatilidade

em se tratando de comportamentos, já que a arquitetura apresentada faz, com um só controlador, o papel de quatro controladores (evitar obstáculos, controle de posição final, seguimento de corredores e seguimento de paredes).

Assim, uma técnica eficiente para a navegação de robôs móveis em ambientes semi-estruturados, baseada somente em sonar, é proposta, a qual pode ser implementada mesmo em robôs de baixo custo, já que não demanda grande esforço computacional.

Como proposta de continuidade deste trabalho pode-se considerar a questão mencionada no início desta Dissertação, referente a sua aplicação em cooperação de robôs. A técnica proposta poderia ser utilizada como opção de arquitetura de baixo nível (reativa) e, em função de sua baixa demanda computacional, poderia ser executada em um microcontrolador ou como tarefa de um *kernel* de tempo real (*uC-OS*, por exemplo).

Por fim, uma outra aplicação interessante é dada pela utilização desta arquitetura como apoio a um sistema de construção automática de mosaicos. Neste sistema, o robô pode, por exemplo, percorrer trajetórias relativamente paralelas a uma parede, enquanto várias imagens vão sendo adquiridas, permitindo a montagem do mosaico.

Referências

[1] EVERETT, H. R. *Sensors for mobile robots: theory and application*. [S.l.]: A K Peters, 1995.

[2] ARKIN, R. C. Behavior-based robot navigation for extended domains. *Adaptive Behavior*, MIT Press, v. 1, n. 2, p. 201–225, 1992.

[3] KUC, R.; BARSHAN, B. Navigating vehicles through an unstructured environment with sonar. In: *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*. Scottsdale, AZ: IEEE, 1989. v. 3, p. 1422–1426.

[4] BORENSTEIN, J.; KOREN, Y. Obstacle avoidance with ultrasonic sensors. *IEEE Journal of Robotics and Automation*, v. 4, n. 2, p. 213–218, abril de 1988.

[5] MORAVEC, H.; ELFES, A. E. High resolution maps from wide angle sonar. In: *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*. [S.l.]: IEEE, 1985. p. 116–121.

[6] ELFES, A. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, v. 3, n. 3, p. 249–265, junho de 1987.

[7] MORAVEC, H. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, v. 9, n. 2, p. 61–74, 1988.

[8] KHATIB, O. Real time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, v. 5, n. 1, p. 90–98, 1986.

[9] THORPE, C. *Path Relaxation: Path Planning for a Mobile Robot*. Pittsburgh, PA, abril de 1984.

[10] KROGH, B.; THORPE, C. Integrated path planning and dynamic steering control for autonomous vehicles. In: *Proceedings of 1986 IEEE International Conference on Robotics and Automation (ICRA '86)*. [S.l.]: IEEE, 1986. v. 3, p. 1664 – 1669.

[11] NEWMAN, W.; HOGAN, N. High speed robot control and obstacle avoidance using dynamic potential functions. In: *Proceedings of 1987 IEEE International Conference on Robotics and Automation (ICRA '87)*. [S.l.]: IEEE, 1987. v. 4, p. 14 – 24.

[12] BROOKS, R. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, v. 2, n. 1, p. 14–23, março de 1986.

[13] ARKIN, R. C. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, p. 92–112, agosto de 1989.

[14] CARELLI, R.; SECCHI, H.; MUT, V. Algorithms for stable control of mobile robots with obstacle avoidance. *Latin American Applied Research - An International Journal*, v. 29, n. 3/4, p. 191–196, 1999.

[15] KOREN, Y.; BORENSTEIN, J. Potential field methods and their inherent limitations for mobile robot navigation. In: *Proceedings of the IEEE Conference on Robotics and Automation*. Sacramento, California: IEEE, 1991. v. 2, p. 1398–1404.

[16] BORENSTEIN, J.; KOREN, Y. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 19, n. 5, p. 1179–1187, setembro/outubro de 1989.

[17] BORENSTEIN, J.; KOREN, Y. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, v. 7, n. 3, p. 278–288, 1991.

[18] SECCHI, H.; CARELLI, R.; MUT, V. Discrete stable control of mobile robots with obstacles avoidance. In: *International Conference on Advanced Robotics, ICAR'01*. Budapeste, Hungria: IEEE, 2001. p. 405–411.

[19] AICARDI, M. et al. Closed loop steering of unicycle-like vehicles via lyapunov techniques. *IEEE Robotics and Automation Magazine*, v. 2, n. 1, p. 27–35, março de 1995.

[20] SECCHI, H. A. *Control de Vehículos Autoguiados con Realimentación Sensorial*. Dissertação (Mestrado) — Facultad de Ingeniería de la Universidad Nacional de San Juan, San Juan, Argentina, julho de 1998.

[21] HOGAN, N. Impedance control: An approach to manipulation. *ASME Journal of Dynamic Systems, Measurement, and Control*, v. 107, p. 1–23, março de 1985.

[22] MARAVALL, D.; LOPE, J. de. Integration of artificial potential field theory and sensory-based search in autonomous navigation. In: *5th International Federation of Automatic Control, IFAC'2002*. Barcelona, Spain: IEEE, 2002.

[23] ROBOTICS, A. *Pioneer 2 Mobile Robots – Computer and Software Manual*. [S.l.], março de 2001.

[24] KONOLIGE, K. G. *Saphira and Aria Software Manual*. Menlo Park, California, setembro de 2001.

[25] LAGES, W. F. Linux as a software platform for mobile robots. Manuscrito obtido em http://www.eletro.ufrgs.br/ fetter/papers.html. 1998.

[26] KOHOUT, B. Challenges in real-time obstacle avoidance. In: *AAAI Spring Symposium on Real-Time Autonomous Systems*. Palo Alto, CA: AAAI, 2000.

[27] MUSLINER, D. J. et al. The chalenges of real-time ai. *IEEE Computer*, v. 28, n. 1, p. 58–66, janeiro de 1995.

APÊNDICE A – Considerações sobre Sistemas de Tempo Real

Um requisito básico para um robô móvel é que ele seja capaz de locomover-se por um ambiente, seja por teleoperação ou de forma autônoma. Para tanto, é necessário o uso de sensores e atuadores, propiciando maior interação com o meio e, conseqüentemente, que o robô reaja apropriadamente a alterações no ambiente em que trabalha.

Geralmente, a aquisição de dados por um sensor não é tarefa complicada. O ponto de maior importância é que em robôs móveis, na maioria dos casos, esses dados não podem ser adquiridos e armazenados para posterior análise. Eles devem ser amostrados e processados imediatamente, pois são utilizados no cálculo dos sinais de controle para acionamento dos atuadores. Devido a motivos relacionados à estabilidade do sistema de controle, esses cálculos devem ser efetuados a uma taxa constante, chamada taxa de amostragem do sistema. O uso de um sistema de tempo real torna-se interessante justamente como um meio de satisfazer as restrições temporais impostas pelo sistema do robô.

Robôs de pequeno porte podem utilizar uma EPROM com um sistema ou programa carregado. Pode-se mencionar, como um exemplo relacionado, uma placa que foi desenvolvida no Departamento de Engenharia Elétrica da UFES. Ela provê uma plataforma para construção de pequenos robôs móveis, e é baseada no microcontrolador MSP430 (*Texas Instruments*). Neste *hardware* é possível executar, por exemplo, o *uC-OS*, um *kernel* de tempo real completo. Robôs maiores e com maior demanda computacional geralmente possuem *hardware* baseado em PC, daí a necessidade de um sistema operacional compatível com estas máquinas. Dentre as opções de Sistema Operacional capazes de atender a essa compatibilidade, o Linux foi escolhido como exemplo por ter sido utilizado em todas as fases de desenvolvimento deste trabalho.

Embora o *kernel* padrão do Linux não suporte aplicações de tempo real, há uma extensão chamada *Real Time Linux*. A idéia desta extensão é implementar um escalonador preemptivo que compartilha o processador entre as tarefas de tempo real e o *kernel* padrão.

Para satisfazer as restrições temporais, o *kernel* padrão é então configurado como a tarefa de menor prioridade [25].

Uma observação importante acerca do uso de sistemas de tempo real em aplicações de robótica móvel, conforme discutido por Kohout [26], é justamente quanto ao uso do termo. De acordo com Musliner et al. [27], uma definição para este tipo de sistema pode ser dada por:

> Computação em tempo real não significa construção de sistemas "rápidos", mas sim a construção de sistemas que são previsivelmente "rápidos o bastante" para agir em seu ambiente de uma forma bem especificada.

Sistemas que utilizam restrições temporais severas, cujo não atendimento pode gerar situações catastróficas (perda de vida humana, por exemplo), são denominados *Sistemas de Missão-Crítica* ou *Hard Real Time Systems*. Sistemas em que o não cumprimento de uma restrição temporal, em conseqüência de uma falha, não são desastrosos, são denominados *Soft Real Time Systems*.

Ainda de acordo com Kohout [26], muitos artigos publicados na área de robótica móvel utilizam erroneamente o termo *Sistema de Tempo Real*, relacionando-o à natureza dinâmica de um dado problema. Em outras palavras, se um robô móvel faz aquisição e processamento de dados em tempo de execução (enquanto navega, de forma *on-line*), alguns autores atribuem a este sistema a característica de tempo real, quando não necessariamente o são.

Nesta Dissertação não foi utilizado um sistema de tempo real, dado que o sistema operacional da plataforma móvel utilizada (*P2OS*) não possui essas características. Além disso, a aplicação em questão não requer tamanha exigência quanto às restrições temporais. No entanto, para tarefas mais sofisticadas, é de extrema importância a ponderação a respeito do uso de um *kernel* de tempo real. Nesse caso, uma solução seria utilizar um versão do *uC-OS*, por exemplo, desenvolvida para o microcontrolador *Siemens* 88C166 utilizado no robô.

APÊNDICE B – Código-fonte do Controlador Proposto

```
// Andre Ferreira

#include "Aria.h"
#include <sys/time.h>
#define PI 3.14159265
#define __USE_TIME_DLY__
#define __GRAPHICS__
#define __USING_ROBOT__

double AlphaCalc(double steering, double target)
{
  double alpha;
  int sinalInv;
  alpha = (target - steering);
  if (fabs(alpha) > 180)
  {
    if (alpha > 0)
      sinalInv = -1;
    else
      sinalInv = 1;
    alpha = (360 - fabs(alpha)) * sinalInv;
  }
  alpha = alpha * PI / 180; // [rad]
  return alpha;
}
```

```
#ifdef __USE_TIME_DLY__
int microseconds(void)
{
  struct timeval tval;
  struct timezone tz;
  int us;
  gettimeofday( &tval, &tz );
  us=tval.tv_usec;
  return us;
}
#endif

int main()
{
  ArRobot robot;
  ArSonarDevice sonar;
  ArKeyHandler handler;
  ArTcpConnection tcpConn;
  ArSerialConnection serConn;

  // Starting up Aria
  Aria::init();

  // KeyHandler
  Aria::setKeyHandler(&handler);
  robot.attachKeyHandler(&handler);

  // Sonar
  robot.addRangeDevice(&sonar);

  // Trying default TCP connection
  tcpConn.setPort();

  if (tcpConn.openSimple())
  {
```

```
    // we could get to the sim, so set the robots device connection to sim
    printf("Connecting to simulator through tcp.\n");
    robot.setDeviceConnection(&tcpConn);
  }
  else
  {
    serConn.setPort();
    printf("Could not connect to simulator, connecting to robot through
                                                                  serial.\n");
    robot.setDeviceConnection(&serConn);
  }

  // try to connect, if it fails exit
  if (!robot.blockingConnect())
  {
    printf("Could not connect to robot... exiting\n");
    Aria::shutdown();
    return 1;
  }
  printf("ROBOT READY !\n");
  // Turn on the robot motors
  robot.runAsync(true);
#ifdef __USING_ROBOT__
  ArUtil::sleep(10000);
#endif
  // INITIALIZATION READY !!!
  // ###########################################################################

  // ***************************************************************************
  // ********************** FINAL POSE CONTROLLER ************************
  // ***************************************************************************

#ifdef __GRAPHICS__
  FILE *pOutput;
  if ((pOutput=fopen("output","a+"))==NULL)
```

```
  {
    printf("Cannot open file.\n");
    exit(1);
  }
  int count = 0;
#endif

  // PROPOSED CONTROLLER VARs
  double sensorRange;
  int shorttestSensorIdx;
  double d; // shorttest distance to obstacle
  double betha; // angle between robot orientation and obstacle direction
  double safeDistance = 700; // [mm] Distance to start Obstacle Avoidance
  double x_rot, y_rot, x_current, y_current, x_near, y_near;
  int reverseSignal;
  double phi, mih;

  // FINAL POSE CONTROLLER VARs
  double rho,alpha,steering,finalPosition,u,w;
  ArPose target(9000,5000,0); // [mm,mm,deg]
  ArPose targetTmp(target.getX(),target.getY(),target.getTh()); // [mm,mm,deg]
  ArPose currentPose;

  // Gains
  double Ku = 0.3; //u_max;
  double Kw = 0.9; //(w_max - (Ku*0.01))/PI;//Kw = (w_max - (Ku*0.5))/PI;

  double d_left,d_right,d_central
  double difLeftCentral,difRightCentral,difLeftRight,menor;
  double angleLeft,angleRight,angleCentral;
  int sensorLeftNumber,sensorRightNumber,sensorCentralNumber;

  currentPose = robot.getPose();
  rho = (currentPose.findDistanceTo(targetTmp))/1000; // [m]
```

```
  while (rho > 0.03) // [m]
  {
#ifdef __USE_TIME_DLY__
  int ti = microseconds();
#endif
    // Use real target
    steering = robot.getTh(); // [deg]
    finalPosition = currentPose.findAngleTo(target);//Tmp); // [deg]
    if (finalPosition < 0)        finalPosition += 360;
    alpha = AlphaCalc(steering, finalPosition); // [rad]
    // ************************************************************************
    // ************************ PROPOSED CONTROLLER ************************
    // ************************************************************************
    // Let's find the frontSonar(0-7) with the shorttest distance to obstacle...
    d = safeDistance; // [mm]
    shorttestSensorIdx = -1;
    for (int j=0;j<8;j++)
    {
      sensorRange = (robot.getSonarReading(j)->getRange());  // [mm]
      // Threshold
      if ( (sensorRange < 1000) && (sensorRange > 200) )
      {
        if ( sensorRange < d )
        {
          d = sensorRange;
          shorttestSensorIdx = j;
        }
      }
    }

  // If in DangerZone --> Compare sonar readings

  if ( d < safeDistance )
  {
      if (shorttestSensorIdx == 0)
```

```
    sensorLeftNumber = 15;
  else
    sensorLeftNumber = shorttestSensorIdx-1;
 sensorRightNumber = shorttestSensorIdx+1;
 sensorCentralNumber = shorttestSensorIdx;

// [mm]
d_left = (robot.getSonarReading(sensorLeftNumber)->getRange());
d_right = (robot.getSonarReading(sensorRightNumber)->getRange());
d_central = (robot.getSonarReading(sensorCentralNumber)->getRange());

// [deg]
angleLeft = robot.getSonarReading(sensorLeftNumber)->getSensorTh();
angleRight = robot.getSonarReading(sensorRightNumber)->getSensorTh();
angleCentral = robot.getSonarReading(sensorCentralNumber)->getSensorTh();

difLeftCentral = fabs(d_central - d_left);
difRightCentral = fabs(d_central - d_right);
difLeftRight = fabs(d_right - d_left);

// BETHA
if (difLeftCentral < difRightCentral)
{
  menor = difLeftCentral;
  betha = (angleLeft + angleCentral) / 2;
}
else
{
  menor = difRightCentral;
  betha = (angleRight + angleCentral) / 2;
}

if (difLeftRight < menor)
{
  menor = difLeftRight;
```

```
      betha = (angleLeft + angleRight) / 2;
    }

    // Find the reverseSignal of betha
    if (betha > 0)
      reverseSignal = -1;
    else
      reverseSignal = 1;
    // Determinar PHI
    phi = ( (( ( (90 - fabs(betha)) * reverseSignal) * PI) / 180) - alpha);
// [rad]
  }
  else // If out DangerZone
  {
    // PHI
    phi = 0;
    betha = 0;
  }

  // Current robot position
  x_current = robot.getX();
  y_current = robot.getY();

  finalPosition = finalPosition * PI / 180; // [rad]

  // Find the target 1m ahead
  x_near = x_current + 1000 * cos(finalPosition);
  y_near = y_current + 1000 * sin(finalPosition);

  // Rotate the target around the robot (phi must be in [rad])
  x_rot = (x_near-x_current) * cos(phi) - (y_near-y_current) * sin(phi)
                                                          + x_current;
  y_rot = (x_near-x_current) * sin(phi) + (y_near-y_current) * cos(phi)
                                                          + y_current;
  targetTmp.setX(x_rot); // Here we change the location of the target
```

```
targetTmp.setY(y_rot); // Here we change the location of the target

// ************************************************************************
// Calculate velocities u,w ---> use virtual target
currentPose = robot.getPose();
rho = (currentPose.findDistanceTo(targetTmp))/1000; // [m]
steering = robot.getTh(); // [deg]
finalPosition = currentPose.findAngleTo(targetTmp); // [deg]
if (finalPosition < 0)
  finalPosition += 360;
alpha = AlphaCalc(steering, finalPosition); // [rad]

u = Ku * tanh(rho) * cos(alpha); // [m/s]

w = Kw * alpha + Ku * (tanh(rho)/rho) * sin(alpha) * cos(alpha); // [rad/s]

// Set new velocities
u = u * 1000; // [mm/s]
w = w * 180 / PI; // [deg/s]
robot.setVel(u);
robot.setRotVel(w);

// Take care with obstacles not detected !!!
// ########################################
for (int j=3;j<5;j++) // Using only front sonars (3,4)
{
  sensorRange = (robot.getSonarReading(j)->getRange());  // [mm]
  if ( sensorRange < 200 )
  {
    robot.stopRunning();
    break;
  }
}
if ( ! robot.isRunning() )
{
```

```
    printf("SAFETY TASK.\n");
    break;
  }
  // ########################################
#ifdef __GRAPHICS__
  // Write the data to a file
  fprintf(pOutput,"\n\t%d\t%f\t%f\t%f\t%f\t%f\t%f\t%f",count,u,w,
          steering,robot.getX(),robot.getY(),x_rot, y_rot);
  count++;
#endif

#ifdef __USE_TIME_DLY__
  int tf = microseconds();
  int dif = ( 100000 - (tf-ti) ) / 1000; // [ms]
  ArUtil::sleep(dif);
#endif

  currentPose = robot.getPose();
  rho = (currentPose.findDistanceTo(target))/1000; // [m]
}

#ifdef __GRAPHICS__
  fclose(pOutput);
#endif

  robot.disconnect();
  Aria::shutdown();
  return 0;
}
```