Uma Arquitetura de Controle

para Robôs Cooperativos

Christian Manrich

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Uma Arquitetura de Controle para Robôs Cooperativos

Christian Manrich

Fevereiro de 2001

Banca Examinadora:

- Marcel Bergerman (Orientador)
  Instituto Nacional de Tecnologia da Informação
- Maria Beatriz Felgar de Toledo
  Instituto de Computação da Unicamp
- Fabio Gagliardi Cozman
  Escola Politécnica da USP
- Edmundo Roberto Mauro Madeira (Suplente)
  Instituto de Computação da Unicamp

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Manrich, Christian

M317a Uma arquitetura de controle para robôs cooperativos / Christian Manrich -- Campinas, [S.P. :s.n.], 2001.

Orientadores : Marcel Bergerman, Heloisa Vieira da Rocha

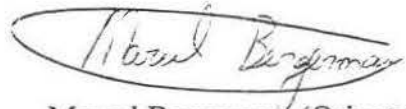
Dissertação (Mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Robótica. 2. Software – Arquitetura. 3. Robôs Móveis. I. Bergerman, Marcel. II. Rocha, Heloisa Vieira da. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

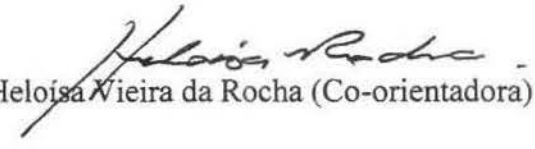
Uma Arquitetura de Controle para Robôs Cooperativos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Christian Manrich e aprovada pela Banca Examinadora.

Campinas, XX de Mês de 2001



Marcel Bergerman (Orientador)



Heloísa Vieira da Rocha (Co-orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 20 de fevereiro de 2001, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Fábio Gagliardi Cozman
USP

Profa. Dra. Maria Beatriz Felgar de Toledo
IC – UNICAMP

Prof. Dr. Marcel Bergerman
CTI

Agradecimentos

Eu gostaria de agradecer a todas as pessoas que contribuíram de forma direta ou indireta para a realização deste trabalho. Em especial eu gostaria de agradecer ...

A meus pais, Silvio e Sati, e meus irmãos, Anny e Eiric, responsáveis pela minha formação como pessoa e que sempre me acompanharam e apoiaram em tudo na vida.

À Raquel, minha guria, pelo incentivo e por tudo o que ela representa para mim.

Ao Marcel, por ter sido não apenas um excelente orientador, mas também um grande amigo e conselheiro com quem pude aprender muito.

Ao grupo do LRV do ITI, principalmente Paulo, Rubens, Reginaldo, Josué e Samuel, pelas inúmeras contribuições durante este o projeto.

A Reid Simmons e David Apfelbaum pela ajuda prestada na utilização das ferramentas por eles desenvolvidas.

Este projeto teve o apoio financeiro da Fapesp através dos processos 97/13384-7 e 99/04254-8.

Resumo

Sistemas multi-robóticos tendem a tornar-se cada vez mais presentes no cotidiano das pessoas, seja na forma de robôs independentes que, para conviverem em um mesmo ambiente, devem se organizar como um sistema único, seja devido à eficiência e confiabilidade que robôs cooperativos possuem.

As vantagens que sistemas multi-robóticos podem oferecer são atingidas através de uma programação relativamente complexa. Arquiteturas de controle para robôs são ferramentas que auxiliam a programação destes sistemas.

Esta dissertação tem como objetivo a implantação de uma arquitetura de controle para robôs cooperativos. Adotou-se como metodologia a definição dos requisitos desejados para a arquitetura, o estudo das arquiteturas existentes na literatura, a adoção daquela que mais se adequa aos requisitos deste projeto, e sua extensão de forma a atender a todos os requisitos propostos.

A validação experimental da arquitetura consiste na implementação de um sistema multi-robótico envolvendo dois robôs móveis para a realização de uma tarefa cooperativa. Nesta tarefa, um dos robôs é o líder e deve guiar o segundo até a posição destino. O segundo robô não é capaz de navegar sozinho e deve seguir o líder.

A principal contribuição deste projeto é a implantação de uma arquitetura para o desenvolvimento de sistemas multi-robóticos em projetos realizados pela comunidade de robótica no país.

Abstract

Systems composed of several robots tend to become ever more present in people's lives. Such systems may come in the form of independent robots, which must be organized as a collective in order to coexist in the same environment, or of efficient and dependable cooperative robots.

The advantages that multi-robot systems offer are reached through a relatively complex programming. Robot control architectures are tools that ease one's job of programming these systems.

In this thesis our objective is to build a control architecture for cooperating robots. The methodology consists of defining the requisites of the architecture, surveying the literature for the existing ones, selecting the one that is most suitable with respect to the requisites, and extending it to make it attend all of them.

The architecture is validated experimentally on a multi-robot system composed of two mobile robots working on a cooperative task. In this task, one of the robots is the leader and must guide the other one to their destination. The second robot lacks the capability to navigate on its own and must follow the leader.

The main contribution of this work is the development of a control architecture for multi-robot systems for the Brazilian robotics community.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

Sistemas que dependem da cooperação entre mais de um robô podem, e em alguns casos devem, ser utilizados. Dudek e sua equipe [1] chamam estes conjuntos de robôs de coletivos. As razões pelas quais o uso de um grupo de robôs se torna necessária dividem-se basicamente em dois tipos.

O primeiro caso é quando os múltiplos robôs já existem. Portanto, um sistema que coordene os robôs deve ser desenvolvido. Há a necessidade de um entendimento entre eles para que possam conviver no mesmo ambiente sem que haja conflito. Evidentemente deve existir um nível razoável de comunicação entre os elementos do coletivo para que possam se relacionar.

Um exemplo deste caso é o controle de veículos em um "*Automated Highway System*" (AHS). Em um AHS, uma vez recebida a informação sobre o destino desejado pelo ocupante de um veículo, o controle de navegação do automóvel deve ser autônomo. O sistema deve planejar a melhor rota e guiar o automóvel. Em [2], Raza e Ioannou mostram como cada veículo pode ser tratado como um elemento de um pelotão (conjunto de veículos locomovendo-se em grupo à mesma velocidade e mantendo-se separados por uma certa distância). Em [3] é mostrado um trabalho em que um veículo aprende a guiar-se em uma rodovia movimentada.

O segundo caso é quando existe uma tarefa complexa a ser realizada. Em algumas situações, o problema não pode ser resolvido por apenas um robô. Em outras, a tarefa torna-se mais fácil se houver um grupo de robôs para realizá-la.

Um exemplo de situação onde há sistemas multi-robóticos mais eficientes que sistemas de apenas um robô, são os "*Automated Guided Vehicles*" (AGV's) [4] utilizados em portos e aeroportos para transporte de cargas. Por um lado, deve haver um entendimento entre eles para que trafeguem organizadamente pelo pátio. Por outro lado, os veículos devem trabalhar em equipe para realizar as tarefas: planejamento de trajetórias, otimização do trabalho, etc.

Uma característica importante de sistemas multi-robóticos é a confiabilidade, pois a falha de um elemento do grupo pode não comprometer a execução da tarefa. Note que o simples fato de haver um grupo de robôs disponíveis para a realização de uma tarefa em comum não garante que a sua execução será mais eficiente ou confiável. Deve haver uma boa coordenação e entendimento entre os elementos do grupo para que o sistema seja melhor que um robô único.

Os exemplos descritos acima ilustram sistemas multi-robóticos homogêneos, isto é, os robôs que compõem os sistemas são funcionalmente equivalentes. Sistemas multi-robóticos podem ser heterogêneos também. Suponha que exista um robô terrestre capaz de realizar um certo trabalho, mas incapaz de encontrar o local onde este deve ser executado. Suponha também um robô aéreo que seja capaz de localizar o alvo do trabalho. O robô aéreo pode informar a localização do alvo e guiar o robô terrestre até o local correto.

Um exemplo prático pode ser um robô terrestre capaz de resgatar uma pessoa em perigo no interior de uma floresta. Este robô não consegue localizar a vítima e precisa da ajuda de um robô aéreo capaz de encontrá-la. Este segundo robô então encontra a pessoa e orienta o robô terrestre até que ele chegue ao local onde o salvamento será feito.

Um projeto que segue esta linha de raciocínio, tendo um robô aéreo e um terrestre, está entre os planos de um projeto maior, o AURORA [5], desenvolvido no Laboratório de Robótica e Visão do Instituto Nacional de Tecnologia da Informação (LRV-ITI).

Outros projetos envolvendo múltiplos robôs estão em desenvolvimento no mundo. No Georgia Institute of Technologie, Ronald Arkin e sua equipe desenvolvem sistemas de

robôs cooperativos. Um dos projetos, por exemplo, consiste em um time de três robôs que trabalham em equipe na limpeza de escritórios [6].

Na Carnegie Mellon University - CMU, o projeto Mercator [7] liderado por Sebastian Thrun e Reid Simmons tem por objetivo o controle de grupos heterogêneos de robôs para a realização de tarefas cooperativas, como construção de mapas, vigilância, e o estabelecimento de uma rede adaptativa de comunicação ponto a ponto.

O projeto DIRA [8] é realizado em conjunto entre o Johnson Space Center - JSC/TRACLabs, o National Institute of Standards and Technology - NIST e a Carnegie Mellon University – CMU. Nele desenvolve-se uma arquitetura para sistemas multi-robóticos. O objetivo do projeto é a construção de uma arquitetura distribuída baseando-se nas arquiteturas desenvolvidas nos três centros de pesquisas envolvidos: TCA da CMU, 3T do TRACLabs e RCS do NIST.

1.2 Objetivo

O projeto descrito neste documento tem como objetivo a implantação de uma arquitetura de controle para robôs cooperativos. Por uma questão de pragmatismo, adotou-se como metodologia o estudo das arquiteturas existentes na literatura, a adoção daquela que mais se adequa aos requisitos deste projeto e sua extensão de forma a atender todos os requisitos propostos. A contribuição deste projeto reside na implantação de uma arquitetura para o desenvolvimento de aplicações multi-robôs em projetos realizados pela comunidade de robótica no país.

A próxima seção contextualiza a contribuição contida neste projeto, classificando as arquiteturas de controle encontradas na literatura e descrevendo algumas das mais utilizadas [9]. Em seguida são apresentados os requisitos deste projeto e a metodologia seguida para a realização do objetivo acima proposto.

1.3 Arquiteturas de *Software* para Controle de Robôs Móveis

As arquiteturas existentes para controle de robôs móveis podem ser divididas em três grupos: arquiteturas deliberativas, arquiteturas reativas e arquiteturas híbridas. Nesta seção, serão apresentadas as características e um exemplo para cada um destes grupos.

As arquiteturas deliberativas permitem a definição explícita das tarefas que devem ser realizadas. Estas arquiteturas decompõem o sistema de controle do robô em uma seqüência ordenada de componentes funcionais. Os dados são primeiramente coletados dos sensores e tratados, eliminando-se os ruídos e resolvendo-se eventuais conflitos entre os diferentes sensores. Em seguida é construído um modelo do ambiente onde o robô poderá operar. Através deste modelo, o robô faz o planejamento da seqüência de ações que deve executar. Esta seqüência de ações é transformada em comandos que são finalmente enviados para os atuadores do robô.

A vantagem das arquiteturas deliberativas é que, nelas, as tarefas são bem definidas. A partir dos dados disponíveis, as ações para a realização da tarefa são planejadas e a tarefa é executada. Porém, este tipo de arquitetura não é muito robusta a mudanças bruscas no ambiente. A construção do modelo e o planejamento muitas vezes tomam um certo tempo e o aparecimento de elementos no ambiente não computados no modelo podem comprometer a execução da tarefa.

Um exemplo de arquitetura deliberativa é a arquitetura NASREM [10]. Nesta arquitetura, a informação percebida passa por uma série de estágios de processamento até que uma instância coerente da situação é obtida. Feito isso, é adotado um plano. Este plano é decomposto sucessivamente por módulos até que as ações possam ser executadas diretamente pelos atuadores (motores, rodas, braços mecânicos, etc.).

As arquiteturas reativas são baseadas em comportamentos e seguem um procedimento bem diferente em comparação com as arquiteturas deliberativas. Os comportamentos são disparados pelos sensores e são camadas de um sistema de controle que trabalham em paralelo. Cada sensor ou conjunto de sensores pode gerar um comportamento diferente, muitas vezes conflitantes. Estes conflitos são resolvidos através

de um esquema de prioridades onde alguns comportamentos são dominantes em relação a outros. Nas arquiteturas reativas não existem modelos do ambiente nem planejamento de uma tarefa que o robô deve executar. Cada comportamento reage aos dados dos sensores e comanda diretamente os atuadores.

A vantagem das arquiteturas reativas é a robustez em relação a mudanças bruscas no ambiente de operação. As novas condições são imediatamente assimiladas pelo robô, que pode reagir rapidamente às mudanças. A desvantagem é a dificuldade em se definir uma tarefa. A tarefa não é definida diretamente, mas é o resultado da soma dos vários comportamentos.

Um exemplo deste tipo de arquitetura é a arquitetura *Subsumption* [11]. Nesta arquitetura, o controle é feito por camadas de controle. Cada camada tem um nível de competência que varia conforme o distanciamento do nível físico. Estas camadas são módulos assíncronos que se comunicam entre si. As camadas de níveis mais altos podem desabilitar a saída das camadas inferiores e assim assumir seus papéis. Um exemplo de níveis de competência seria:

1 - Evitar contato com objetos;
2 - Movimentar-se sem objetivo sem tocar em obstáculos;
3 - "Explorar" o ambiente percebendo lugares a uma certa distância que podem ser alcançados;
4 - Construir um mapa do ambiente e planejar caminhos.

As arquiteturas híbridas possuem características tanto das arquiteturas reativas quanto das deliberativas. As tarefas de baixo nível, como desvio de obstáculos, são executadas de forma reativa, e tarefas de alto nível, como planejamento de trajetória, são executadas de forma deliberativa.

O objetivo deste tipo de arquitetura é aproveitar as vantagens dos dois outros descritos anteriormente. As características reativas da arquitetura permitem que o robô reaja a mudanças inesperadas no ambiente sem que a tarefa como um todo seja comprometida. As características deliberativas permitem que a definição de tarefas de mais alto nível sejam mais fácil e claramente definidas.

Exemplos para este tipo de arquitetura são as arquiteturas LAAS [12] e TDL/TCM [13][14][15][16]. A arquitetura LAAS é organizada em três níveis, sendo dois níveis de decisão e um funcional. O nível mais alto faz o planejamento. O segundo nível recebe tarefas e as transforma em *scripts* (procedimentos compostos por ações elementares). O nível funcional tem contato direto com as primitivas do robô. Este nível é gerenciado e controlado por um Executivo, pois é formado por vários módulos que se comunicam através de mensagens. A LAAS utiliza uma linguagem de programação própria e específica para esta arquitetura.

O TCM é uma arquitetura que fornece um ambiente genérico para sistemas robóticos que precisam mesclar controle deliberativo e reativo. A arquitetura provê também seqüenciamento e decomposição hierárquica de tarefas e gerenciamento de recursos. O TCM pode ser programado utilizando-se linguagens conhecidas como C++ ou Lisp, mas este trabalho é facilitado utilizando-se a linguagem TDL. A linguagem TDL é uma extensão de C++ com decomposição de tarefas, sincronização, monitoramento e tratamento de exceções. Esta arquitetura, por ter sido adotada para o projeto, é descrita mais detalhadamente na seção 3.1.

1.4 Requisitos do Projeto

A meta deste projeto é a implantação de uma arquitetura de controle única para todos os componentes de um sistema composto por múltiplos robôs cooperativos. A arquitetura de controle a ser concebida deve ser reativa a mudanças e incertezas no ambiente, assim como permitir a realização de múltiplas tarefas simultaneamente. É importante também que a arquitetura permita que as funcionalidades do sistema sejam acrescentadas de forma independente e gradativa. Portanto, a arquitetura deve atender aos seguintes requisitos:

- Arquitetura multi-robô: a arquitetura deve suportar sistemas compostos por conjuntos de robôs que se comunicam entre si para a realização de tarefas cooperativas.
- Generalidade: necessidade de ter uma arquitetura que possa ser utilizada em vários projetos, utilizando diferentes robôs.

- Especificidade: a arquitetura deve ser genérica, mas voltada para sistemas na área de robótica, isto é, deve ser específica para a área de robótica mas genérica dentro dela.
- Modularidade: a arquitetura deve permitir a construção de um sistema através da adição de módulos. Desta forma, assim que cada etapa é concluída, o sistema pode ser testado mesmo estando incompleto.
- Arquitetura híbrida: a arquitetura deve ser híbrida. Isso torna possível a construção de um sistema que ao mesmo tempo possa planejar uma tarefa complexa e reagir a situações não previstas durante a execução.
- Disponibilidade: os componentes da arquitetura criados por outros autores devem estar disponíveis para poderem ser utilizadas livremente.
- Suporte: os responsáveis por estes componentes devem fornecer suporte para a solução de problemas quando estes surgirem.

A arquitetura que melhor responde a todos estes requisitos e, portanto, será utilizada neste projeto, é o TDL/TCM. A arquitetura LAAS, apesar de responder a vários dos requisitos, não é disponível para uso público. Além disso, ela possui uma linguagem de programação específica que deve ser aprendida. O TDL/TCM também possui uma linguagem de programação, mas ela é uma extensão de C++, que é amplamente difundida.

As características do TDL/TCM que satisfazem os requisitos do projeto são:

- Generalidade: o TDL/TCM é uma arquitetura para ser utilizada por uma variedade de tipos de robôs. Isso vem de encontro com a necessidade de se ter uma arquitetura genérica na área de robótica.
- Especificidade: o TDL/TCM foi desenvolvido para sistemas robóticos, sendo então, específico para esta área de pesquisa.
- Modularidade: um sistema que utiliza o TDL/TCM pode ser desenvolvido por partes. O sistema pode ser construído e testado incrementalmente. Além disso, subtarefas podem ser substituídas por outras que forneçam os mesmos serviços.
- Arquitetura híbrida: o TDL/TCM não é totalmente deliberativo nem totalmente reativo. Ele é híbrido, apresentando portanto, ambas as características.

- Disponibilidade: o código em linguagem C++ do TDL/TCM está disponível para *download* via ftp anônimo, isto é, os fontes dos programas de instalação são de domínio público e podem ser utilizados livremente.
- Suporte: Os criadores da arquitetura fornecem suporte pessoalmente (via *Internet*) e com rapidez. A solução de problemas e esclarecimento de dúvidas são feitas pelas pessoas mais capacitadas no assunto.

Apesar de responder a quase todos os requisitos exigidos, o TDL/TCM não tem uma característica fundamental para sistemas multi-robóticos: o suporte para comunicação entre diferentes máquinas e processos. Portanto, a arquitetura que será apresentada nesta dissertação herda todas as características do TDL/TCM além de tornar-se capaz de lidar com múltiplos robôs.

1.5 Estrutura da Tese

Este documento está organizado em cinco seções principais e dois anexos. A seção 1 é a introdução. A seção 2 descreve o ambiente de trabalho e os recursos disponíveis para a realização deste projeto. A seção 3 descreve as ferramentas de *software* básicas utilizadas neste trabalho. A seção 4 traz a contribuição deste projeto, onde é descrita a arquitetura de controle para robôs cooperativos e sua validação experimental através de uma aplicação prática. A seção 5 traz a conclusão e sugestões para trabalhos futuros. O apêndice 1 descreve dificuldades que surgem quando se trabalha com robôs reais e o apêndice 2 traz instruções de instalação dos *software* básicos no ambiente Linux.

Capítulo 2

Ambiente Experimental

Esta seção descreve o ambiente experimental de trabalho disponível para a realização deste projeto. Aqui são descritos os robôs utilizados e a infra-estrutura do laboratório onde os experimentos práticos foram realizados.

A parte experimental do projeto foi desenvolvida no Laboratório de Robótica e Visão - LRV do Instituto Nacional de Tecnologia da Informação - ITI. Foram utilizados nas experiências dois robôs móveis fabricados pela Nomadic Technologies Inc. e dois computadores, todos se comunicando através da rede.

A comunicação com os robôs é feita através do protocolo TCP/IP. A conexão dos robôs com a rede é feita via ondas eletromagnéticas com taxa de transmissão de 1,6Mbps. Um RangeLAN2 7510 ligado à rede Ethernet a 10Mbps via par trançado é responsável pela ligação entre os robôs e o restante da rede.

O primeiro robô é um Nomad 200 (Figura 2.1), referenciado neste texto com o nome de Nomad. Ele é equipado com um processador Pentium 100MHz com 64MB de RAM, rodando o sistema operacional Linux com *kernel* 2.0.24.

Figura 2.1: Robô móvel Nomad 200 (Nomad).

Seu conjunto de sensores é constituído por 16 sonares utilizados para medição de longas distâncias (até 10 m), 16 sensores infra-vermelhos para distâncias pequenas (até 60 cm) e 20 sensores de toque, todos dispostos em forma de anel ao redor do robô. Os sonares estão posicionados no topo da torre, os infra-vermelhos na parte inferior da torre e os sensores de toque são divididos em dois anéis localizados na base do robô.

Para a movimentação, o Nomad 200 possui um sistema de 3 rodas utilizado para a translação. Um outro sistema mecânico, que movimenta a torre, possibilita a rotação do robô sem modificar a direção definida pelas rodas. Um integrador (*encoder*) é responsável por calcular a posição do robô em relação à posição inicial através da somatória dos movimentos.

O XR4000 (Figura 2.2) é referenciado aqui com o nome de Theus. Ele é equipado com dois processadores Pentium 166MHz com 64MB de RAM. O sistema operacional é Linux com *kernel* 2.0.35. A comunicação entre os dois processadores é feita através de compartilhamento de memória utilizando o protocolo TCP/IP.

Figura 2.2: Robô móvel XR4000 (Theus).

Theus possui os mesmos tipos de sensores que o Nomad, porém em quantidades diferentes. Ele possui dois anéis de sensores, onde cada um deles possui 24 sonares, 24 sensores infra-vermelhos e 24 sensores de toque. Para os três tipos de sensores existe um anel na parte superior e outro na parte inferior do robô.

A movimentação é realizada de forma diferente do Nomad. Theus possui um sistema holonômico de 4 rodas, tornando possível bruscas mudanças de direção enquanto se movimenta. Sua posição, assim como no Nomad, é registrada por um integrador.

Os dois computadores são PC's Pentium. O primeiro possui um processador Pentium II 300MHz com 128MB de RAM e carregado com o sistema operacional Linux com *kernel* 2.0.36. O outro possui um processador Pentium III 700MHz com 128MB de RAM e seu sistema operacional é Linux com *kernel* 2.2.14.

O esquema do laboratório contendo os computadores e os robôs pode ser visto na figura 2.3.

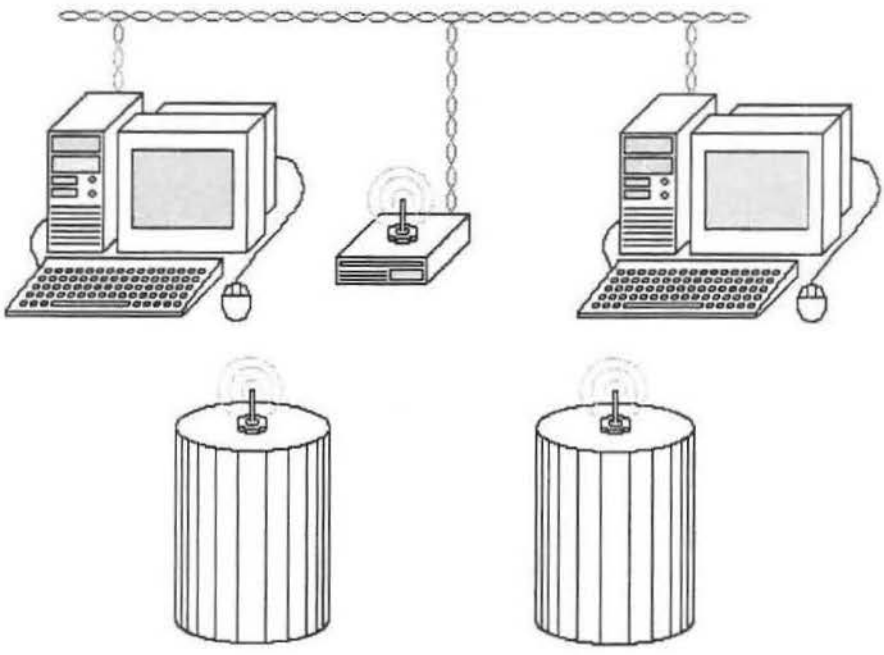


Figura 2.3: Modelo do equipamento experimental.

Capítulo 3

Ferramentas-Base

Esta seção traz a descrição das ferramentas utilizadas na construção da arquitetura de controle para robôs cooperativos. A seção 3.1 descreve a arquitetura para robôs individuais na qual este trabalho está baseado. A seção 3.2 descreve o mecanismo de comunicação que possibilitou a expansão da arquitetura TDL/TCM para sistemas com múltiplos robôs.

3.1 Arquitetura TDL/TCM

O TCM (*Task Control Management*) [13][15][16] foi desenvolvido por Reid Simmons e sua equipe na Carnegie Mellon University. O TCM é uma arquitetura que fornece um ambiente genérico para sistemas robóticos que precisam mesclar controle deliberativo e reativo. A arquitetura provê também seqüenciamento e decomposição hierárquica de tarefas e gerenciamento de recursos.

O TCM pode ser programado utilizando-se linguagens conhecidas como C++ ou Lisp, mas este trabalho é facilitado utilizando-se a linguagem TDL (*Task Description Language*), também desenvolvida pela equipe de Simmons. A linguagem TDL é uma extensão de C++ com decomposição de tarefas, sincronização, monitoramento e tratamento de exceções.

O compilador TDL é um programa escrito em Java que transforma código TDL em C++ puro. Uma biblioteca, utilizada pelo código gerado, contém as funcionalidades do

TCM para gerenciamento de tarefas. Tanto o código gerado quanto a biblioteca utilizada são independentes da plataforma em que são utilizados.

A linguagem TDL foi criada basicamente para permitir a execução de árvores de tarefas (Figura 3.1). As árvores de tarefas são estruturas que refletem a hierarquia das tarefas que estão sendo executadas. Os dois tipos básicos de nós das árvores são gols (nós intermediários) e comandos (folhas). Os gols representam tarefas relativamente complexas que devem ser divididas em subtarefas, que são outros gols ou comandos. Os comandos são tarefas simples que podem ser executadas diretamente pelo robô.

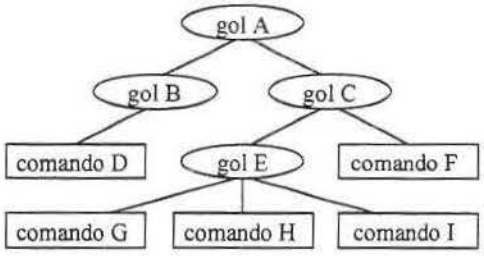


Figura 3.1: Árvore de tarefas.

Cada nó é um pedaço de código que executa uma determinada ação, que pode ser auto-suficiente ou pode ser subdividida gerando filhos na árvore. A criação de filhos é feita dentro do código do nó através da cláusula *spawn*, que pode ser seguida de uma clausula *with* onde são colocadas restrições sobre a subtarefa.

As ações podem conter código condicional, iterativo e até mesmo recursivo, mas a árvore resultante é sempre uma árvore simples, pois os filhos, mesmo quando são nós do mesmo tipo, são nós inteiramente novos na árvore. Note que, como a criação dos filhos se dá dentro do código do nó, o mesmo código pode gerar árvores bem distintas em diferentes execuções.

Os nós das árvores podem assumir diferentes estados quanto ao seu tratamento. Esses estados são:

- desabilitado: as restrições de sincronia (descritas abaixo) ainda não foram satisfeitas.

- habilitado: as restrições de sincronia já foram satisfeitas e o tratamento tem permissão para iniciar.
- ativo: o tratamento iniciou e está em andamento.
- completado: o tratamento terminou e retornou Sucesso ou Falha.

A situação onde o tratamento de um nó pode estar habilitado mas não ativo é quando há escassez de recursos físicos ou computacionais no sistema.

O conceito de tratamento é válido apenas para um nó individualmente, mas existe a necessidade de se definir o estado de uma subárvore como um todo. Os conceitos expansão e execução são introduzidos para determinar os estados de expansão e execução de subárvores. Expansão de um nó corresponde ao tratamento de todos os gols de sua subárvore, enquanto que execução corresponde ao tratamento dos comandos.

As árvores de execução mantém informação sobre a estrutura de tarefas e subtarefas. Apenas com este tipo de informação, a capacidade de representação da linguagem estaria muito limitado, já que em robótica muitas tarefas dependem de sincronia para que possam ser bem executadas. Por exemplo, uma tarefa chamada `fix_it` pode ser dividida em duas subtarefas, `get_tools` e `fix_object`. `get_tools` pega as ferramentas necessárias para consertar um objeto e `fix_object` conserta o objeto. `get_tools` só saberá quais as ferramentas que deve pegar quando `fix_object` já estiver sido planejada e `fix_object` só poderá ser executada quando `get_tools` já tiver sido completada. A linguagem TDL deverá garantir que as ações sejam realizadas na seguinte ordem: planejamento de `fix_object`, planejamento de `get_tools`, execução de `get_tools` e execução de `fix_object`.

Além de gols e comandos, existem outros tipos de nós e os monitores são um deles. Enquanto gols e comandos são tarefas que são executadas apenas uma vez por chamada (através de uma cláusula *spawn*), monitores são tarefas repetitivas. Monitores podem, para uma mesma chamada, serem instanciados várias vezes, periodicamente ou não. Quando o monitor é periódico, ele é disparado a cada intervalo de tempo fixo definido na chamada. O monitor não periódico é disparado conforme a restrição de ativação que é também definido na chamada do monitor.

Outro tipo de nó são as exceções. Exceções em TDL funcionam de forma bem parecida com a utilizada em C++. Quando é atingida uma situação de erro, uma exceção é gerada. Esta exceção é passada de nó para nó até que seja encontrado um que contenha um tratador de exceções capaz de tratá-la. Tratadores de exceções podem ser adicionados incrementalmente nas tarefas durante a programação. Isso facilita o desenvolvimento de um sistema, onde não foram previstas antecipadamente todas, que podem ser muitas, as possíveis situações de erro.

3.2 O Sistema de Comunicação IPC

O *Inter Process Communication* – IPC [17][18] foi criado para facilitar a comunicação entre processos em um sistema heterogêneo. Este *software* de comunicação foi desenvolvido também por uma equipe liderada por Simmons. As linguagens compatíveis com o IPC são C, C++ e Lisp. O IPC também pode operar em diferentes plataformas, incluindo Sun, x86, PPC, etc. e diferentes sistemas operacionais como SunOS, Solaris, Linux, etc.

Um sistema que utiliza IPC consiste em um servidor central independente e um número qualquer de processos específicos da aplicação (módulos) que se comunicam através dele (Figura 3.2). Para fazer a interface com o servidor central, os módulos utilizam uma biblioteca que contém as funções do IPC. Esta biblioteca deve ser ligada a cada programa que fará parte do sistema.

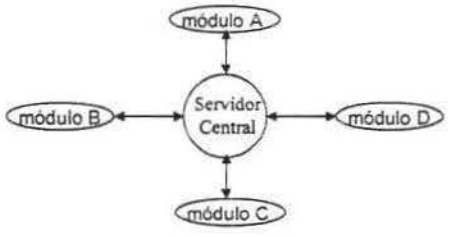


Figura 3.2: Comunicação entre processos com o IPC.

A comunicação via IPC utiliza essencialmente o modelo *publish/subscribe* onde alguns processos (*subscribers*) manifestam o interesse em receber certos tipos de

mensagens, enquanto outros (*publishers*) enviam mensagens para o sistema. Quando o servidor central recebe uma mensagem, ele envia uma cópia para cada *subscriber* que espera mensagens daquele tipo. Processos não precisam apenas enviar ou apenas receber mensagens, eles podem ser *subscribers* para alguns tipos de mensagens e *publishers* para outros. Nada impede também que um processo seja *publisher* e *subscriber* para o mesmo tipo de mensagem, isto é, o próprio processo pode receber uma cópia da mensagem que envia.

Como a recepção é assíncrona, cada *subscriber* deve possuir um *handler* que é uma função que será acionada assim que uma mensagem de um certo tipo for recebida. O processo pode ficar constantemente esperando mensagens chegarem ou pode, sempre que precisar ou puder, verificar se existe alguma mensagem pronta para ser tratada.

O IPC opera também com o modelo de comunicação cliente/servidor, que funciona com mensagens do tipo *query/response*. Uma *query* pode ser tanto com bloqueio quanto sem. Em uma *query* com bloqueio, o processo pára suas atividades até receber a resposta. Quando a resposta chega, a execução continua a partir do ponto onde a *query* foi solicitada. Em uma *query* sem bloqueio, o processo continua a rodar e a mensagem de resposta é tratada por um *handler* quando for recebida.

O IPC traz também mecanismos de empacotamento (*marshalling* e *unmarshalling*). Estes mecanismos servem para a passagem de estruturas complexas como ponteiros, listas ligadas, vetores de tamanho variável, etc. e também proporcionam a passagem de dados entre máquinas com ordenamentos diferentes dos bits.

Capítulo 4

Uma Arquitetura de Controle para Robôs Cooperativos

Nesta seção será apresentada a arquitetura de controle proposta neste projeto. A arquitetura é baseada na arquitetura TDL/TCM já existente para robôs independentes e o sistema de comunicação IPC para troca de mensagens entre processos.

A seção 4.1 mostra a arquitetura TDL/TCM e como ela é aplicada em sistemas multi-robóticos. A seção 4.2 descreve como o IPC completa a arquitetura para possibilitar o trabalho com múltiplos robôs. A seção 4.3 descreve a implantação da arquitetura através da construção de um sistema multi-robótico e a validação experimental do projeto.

4.1 A Arquitetura TDL/TCM e os Sistemas Multi-Robóticos

O TDL/TCM já foi utilizado e validado em vários sistemas robóticos [13], [14], [19], os quais, no entanto, eram de robôs independentes (Figura 4.1). A principal meta deste projeto é a implementação de um sistema com mais de um robô usando a mesma arquitetura.

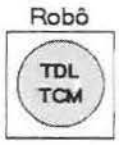


Figura 4.1: Um robô independente.

Existem duas formas que poderiam ser utilizadas para a implantação do sistema utilizando o TDL/TCM. A primeira é ter o TDL/TCM funcionando independentemente dentro de cada robô. Não haveria então comunicação entre os membros do grupo e a cooperação se daria através da somatória das atividades de cada um. A Figura 4.2 mostra um esquema genérico onde o TDL/TCM é utilizado independentemente dentro de cada robô.

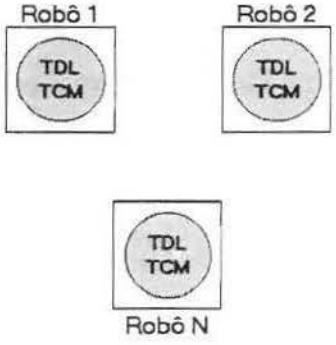


Figura 4.2: Múltiplos robôs independentes com TDL/TCM.

Um sistema onde houvesse realmente cooperação entre os elementos do grupo seria difícil e, para algumas tarefas, até mesmo impossível de ser conseguido. Para um melhor aproveitamento do potencial do grupo, é fundamental que os elementos possam trocar informações entre si.

A segunda maneira de se implementar o sistema multi-robô é, portanto, tornando possível a comunicação entre os membros do coletivo. Para isso é necessário um sistema de troca de mensagens entre processos de máquinas diferentes. O sistema escolhido para este projeto é o IPC.

A Figura 4.3 traz o modelo que é utilizado neste projeto. O sistema possui o TDL/TCM tratando tanto do comportamento individual de cada robô quanto do grupo como um todo. Enquanto isso, o IPC possibilita a comunicação dentro do grupo. A mesma

arquitetura que controla cada robô individualmente é usada para o gerenciamento do coletivo como um todo. Isso elimina as preocupações com problemas que poderiam surgir na implementação de um outro sistema para controlar o grupo.

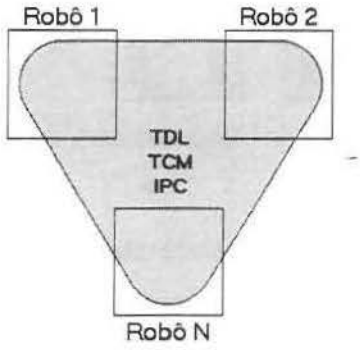


Figura 4.3: Coletivo utilizando TDL/TCM com IPC.

4.2 TDL/TCM com Comunicação entre Robôs

O IPC é um sistema responsável por troca de mensagens entre processos. A comunicação através dele é transparente em relação à localização do processo. Para o programador, é idêntico o mecanismo de troca de mensagens entre processos em uma mesma máquina ou em máquinas diferentes, pois tudo passa pelo servidor central.

Foi detectado, porém, um sério problema com este tipo de centralização. Se, por um lado, o trabalho do programador fica mais fácil, por outro, o servidor central pode tornar-se um gargalo conforme o número de robôs aumenta. Todas as mensagens, mesmo as que interessam apenas a processos dentro de um mesmo robô, são enviadas através do servidor central (Figura 4.4). Este envio faz a mensagem ir para uma outra máquina e em seguida para o destinatário, podendo, em muitos casos, voltar ao mesmo robô.

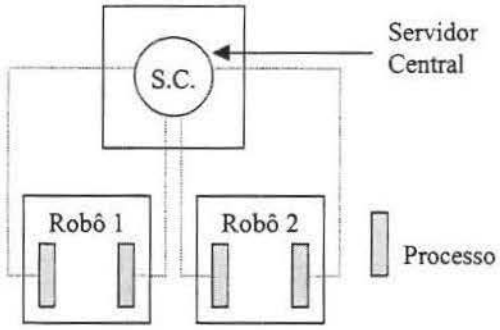


Figura 4.4: Um único servidor central.

A solução deste problema é possível através da definição de contextos. O IPC possibilita que um mesmo processo esteja conectado a mais de um servidor central, um em cada contexto. Portanto, podemos ter vários servidores centrais rodando, um local em cada robô e um global em outra máquina (Figura 4.5).

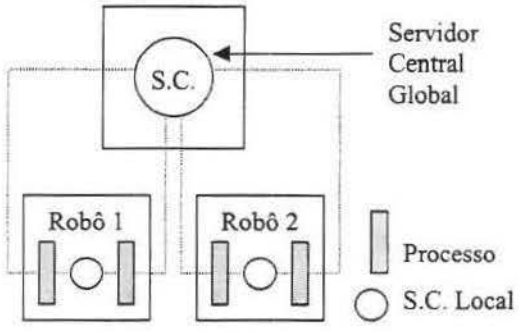


Figura 4.5: Vários servidores centrais.

Programando-se cada processo para ter dois contextos, um local e outro global, pode-se evitar que mensagens pertinentes a apenas processos dentro do mesmo robô sejam enviadas através do servidor central global. O servidor global fica então responsável apenas por mensagens que passam informação de um robô para outro. Os servidores locais controlam as mensagens trocadas entre processos no mesmo robô.

Um processo não precisa necessariamente estar conectado a ambos os servidores. Por exemplo, um módulo que controla os motores do robô pode estar conectado apenas ao servidor local. Isso traz até um certo benefício para a segurança do robô, evitando que processos externos possam controlar diretamente seus movimentos.

4.3 Implantação da Arquitetura

Na seção anterior foi descrita, de forma genérica, a arquitetura de controle proposta neste projeto. Nesta seção é apresentada a implantação da arquitetura. A extensão de uma arquitetura de controle concebida para um único robô para uma arquitetura capaz de lidar com múltiplos robôs, é conceitualmente simples. Neste projeto, como explicado anteriormente, o trabalho consiste em extender a arquitetura TDL/TCM com a inclusão dos métodos fornecidos pelo IPC, de forma que todos os robôs do coletivo possam ser programados para a realização de uma tarefa comum e possam comunicar-se entre si. A concretização deste conceito requer um esforço considerável, que envolve decisões de como tratar o fluxo de mensagens entre os módulos de cada robô e os módulos em diferentes robôs, como programar os robôs de forma coletiva, e como validar experimentalmente a arquitetura implantada.

Como decisão de projeto, optou-se por implantar a arquitetura através de um estudo de caso, unificando-se assim a descrição de sua implantação e de sua validação. Este estudo de caso consiste na implementação de um sistema multi-robótico para a realização de uma tarefa relativamente complexa. Desta forma, ao se concluir a implantação da arquitetura, obteve-se também a validação experimental que confirma a viabilidade de sua utilização na implementação de sistemas com múltiplos robôs.

A seção 4.3.1 descreve a tarefa que o conjunto de robôs irá realizar. A seção 4.3.2 descreve o sistema implementado para a realização da tarefa e a seção 4.3.3 descreve como seus vários módulos funcionam. A seção 4.3.4 descreve como os módulos do sistema se comunicam entre si e a seção 4.3.5 mostra os resultados experimentais obtidos.

4.3.1 Descrição da Tarefa

A tarefa escolhida consiste em transformar um dos robôs em um guia que deverá levar o outro robô até uma determinada posição. O robô líder é o único capaz de seguir uma trajetória até este ponto. O segundo robô é capaz apenas de seguir o líder, sem o qual, fica incapacitado de chegar até o ponto alvo.

A motivação para esta tarefa são situações reais, onde um dos veículos é considerado líder e outros devem segui-lo. Uma situação como esta pode ocorrer, por exemplo, em um sistema de *Automated Guided Vehicles* – AGV's. Um sistema deste tipo pode ser utilizado para transporte de carga em grandes depósitos ou em locais como portos ou aeroportos. A implantação de um sistema com AGV's pode se beneficiar da capacidade dos veículos se organizarem em comboios, aumentando a organização do ambiente e minimizando os cálculos de trajetórias.

Uma outra situação onde este tipo de tarefa é útil é em uma *Automated Highway. Automated Highways* são estradas nas quais automóveis não precisam do auxílio do motorista para segui-la. H. Raza e P. Ioannou [2], por exemplo, descrevem um sistema onde os veículos se agrupam em pelotões que são guiados pela estrada.

No experimento deste projeto, a localização do robô líder é feita através da leitura dos integradores. As distâncias percorridas pelo robô neste experimento são pequenas, o que permite que o erro acumulado nos integradores sejam desprezados. Para o caso de tarefas onde os robôs percorressem distâncias maiores, algum outro tipo de sensor para localização poderia ser utilizado, ou, caso estes não estejam disponíveis, os dados lidos dos integradores poderiam ser processados por um filtro de Kalman, a fim de torná-los mais confiáveis [20].

Assume-se também que o posicionamento relativo entre os robôs seja conhecido. Esta restrição não compromete o projeto, pois é comum, em sistemas reais, o conhecimento das posições iniciais dos robôs. Outros trabalhos com sistemas multi-robóticos reais já foram realizados assumindo-se que o posicionamento relativo dos robôs é conhecido ao se iniciar a tarefa [21].

O posicionamento relativo consiste em três valores: distância entre os robôs, ângulo do líder em relação ao seguidor e o ângulo do seguidor em relação ao líder. Estes valores são fornecidos pelo operador e podem conter uma imprecisão de até ±20cm para a distância e até $\pm 10^{o}$ para os ângulos se a distância for menor que 3m.

Além do posicionamento inicial, o operador deverá fornecer as coordenadas x e y dos pontos alvo pelos quais o líder deverá passar.

A partir deste momento, a tarefa deve ser realizada sem nenhum auxílio do operador. Com a informação de posicionamento inicial, os robôs deverão rotacionar de forma que o

seguidor fique direcionado para o líder e o líder fique direcionado no sentido oposto ao seguidor. Em seguida o seguidor iniciará a aproximação. A aproximação é feita somente uma vez durante a execução da tarefa.

Quando o seguidor se aproximar suficientemente do líder, este poderá começar a se mover em direção ao primeiro ponto alvo e o seguidor a segui-lo. Ao atingir o ponto alvo, o líder já pode ir em direção ao segundo ponto alvo e assim sucessivamente.

4.3.2 Descrição do Sistema

A tarefa será realizada por dois robôs. O líder é o Nomad (robô Nomad 200), enquanto o seguidor é Theus (robô XR4000). Como a forma de programação de cada robô é diferente, verificou-se a necessidade de se criarem módulos básicos que são responsáveis pela interação com os robôs. Cada robô tem dois módulos básicos que levam os nomes de mover e sensors. O módulo mover é responsável pela movimentação, enquanto o módulo sensors fornece os valores lidos pelos sensores do robô. Estes são os únicos módulos que possuem contato direto com os robôs. Os demais devem receber ou enviar informações para os robôs sempre através de um destes módulos.

Como visto na descrição da tarefa, ela é dividida em duas partes principais: a aproximação e o seguimento. A aproximação é realizada através da cooperação entre os módulos gonear em Theus e bringnear em Nomad. Depois de feita a aproximação, Theus passa a seguir o Nomad utilizando o módulo follow e o Nomad navega até o ponto alvo utilizando o módulo fields.

Os módulos detect são responsáveis, em cada robô, por manter atualizada a informação do posicionamento do outro robô em relação a si, isto é, a distância relativa e o ângulo em que o outro robô se encontra. Estas informações são utilizadas pelos módulos follow em Theus e fields no Nomad.

A figura 4.6 mostra como os módulos estão hierarquicamente organizados. As linhas representam as ligações por onde mensagens são trocadas por meio do IPC. Mais adiante será descrito em detalhe o funcionamento de cada um dos módulos que compõem o sistema. Aqui é descrito resumidamente como o conjunto de módulos trabalha para a realização da tarefa.

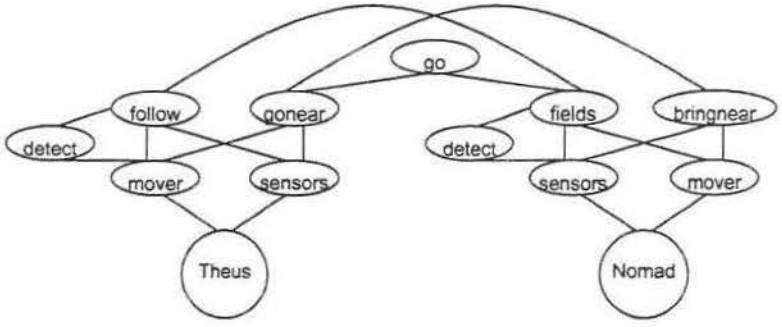


Figura 4.6: Hierarquia dos módulos.

A tarefa é disparada pelo módulo go. Este é responsável por conseguir as informações iniciais necessárias para a tarefa e por coordenar sua execução. Após conseguir as informações iniciais, o módulo go pode disparar a execução da aproximação no módulo gonear. Este, por sua vez, inicia a aproximação de Theus em relação ao Nomad em conjunto com o módulo bringnear. Assim que estes dois módulos concordam que a aproximação foi concluída, o módulo gonear retorna para o módulo go a situação alcançada, isto é, se a aproximação foi bem sucedida ou se houve falha na execução.

Se a aproximação falha, o módulo go aborta a tarefa. Caso contrário, ele dispara a navegação do Nomad através do módulo fields. Este, por sua vez, antes de iniciar a movimentação do robô, avisa Theus para começar a seguí-lo utilizando o módulo follow. Além disso, tanto o módulo fields quanto o módulo follow disparam seus respectivos módulos detect para que a posição relativa entre os robôs seja sempre conhecida.

O Nomad vai então em direção ao primeiro ponto alvo sendo seguido por Theus. Ao atingir este ponto, passa a ir em direção ao próximo e assim por diante. Quando o último ponto alvo é atingido, a tarefa é dada como encerrada e bem sucedida. Problemas podem ocorrer durante a navegação: um ponto alvo pode não ser alcançável, Theus pode se perder pelo caminho, etc. Caso um problema destes ocorra, a tarefa como um todo falha e é abortada.

4.3.3 Descrição dos Módulos

Módulo sensors

Existe um módulo sensors para cada um dos robôs. Eles têm contato direto com os robôs e são responsáveis por fornecer informações sobre os sensores dos mesmos. Os sensores disponíveis para este sistema são: sonares, infra-vermelhos, sensores de toque e integradores (*encoders*).

A leitura dos integradores foi padronizada, para que a programação fosse facilitada. A padronização é feita dentro dos módulos sensors. Os dois robôs fornecem informação de posição e velocidade de forma diferente. A tabela 4.1 mostra como cada robô disponibiliza esta informação e como foi padronizado.

Tabela 4.1: Padronização dos integradores.

Robô	**Posição**	**Velocidade**
Nomad	x, y (décimos de polegadas) θ (décimos de grau)	Translação (0,1pol/s) Rotação (0,1°/s)
Theus	x, y (milímetros) θ (milirradianos)	vx, vy (mm/s) rotação (mrad/s)
Padrão	x, y (milímetros) θ (milirradianos)	Translação (mm/s) Rotação (mrad/s)

Além do problema de unidades diferentes, os robôs possuem um sistema de coordenadas rotacionado um em relação ao outro, mostrado na Figura 4.7. Em Theus, o eixo x fica na direção da parte lateral direita do robô, enquanto no Nomad, o eixo x é para frente.

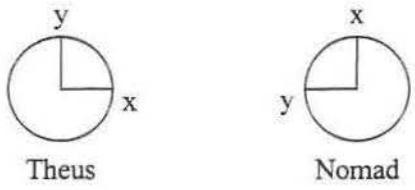


Figura 4.7: Sistemas de coordenadas.

Através de algumas transformações de unidades, sistemas de coordenadas e cálculos de módulo de vetores, pôde-se padronizar a leitura dos integradores. Para os módulos que consultam o módulo sensors, as transformações são transparentes e as informações podem ser utilizadas de forma análoga tanto para um quanto para o outro robô.

No caso dos outros sensores, uma padronização não foi possível. Para este nível de abstração dos dados, isto é, a simples leitura dos valores sem a construção de mapas ou a identificação de obstáculos, não seria simples a utilização de um mesmo tipo de dado para o armazenamento dos mesmos. Além de o número de sensores não coincidir, os valores lidos em alguns casos têm também interpretações diferentes, como pode ser observado na tabela 4.2.

Tabela 4.2: Sensores: disposições e significados diferentes.

Robô	**Sensor**	**Número de sensores**	**Significado dos valores**
Nomad	Sonares	1 anel de 16 sensores	Distância em polegadas
	Infra-vermelhos	1 anel de 16 sensores	Valor proporcional à distância
	Sensores de toque	2 anéis de 10 sensores	Pressionado ou não pressionado
Theus	Sonares	2 anéis de 24 sensores	Distância em milímetros
	Infra-vermelhos	2 anéis de 24 sensores	Valor proporcional ao retorno do sinal
	Sensores de toque	2 anéis de 24 sensores	Pressionado fraco/forte ou não pressionado

Os sonares ainda puderam ser semi-padronizados, pois, apesar de serem diferentes em quantidade, ambos retornam distância. Como são em unidades diferentes, o módulo sensors do Nomad transforma os valores de décimos de polegada em milímetros antes de enviá-los para o módulo que fez a requisição.

Módulo mover

Assim como os módulos sensors, existe um módulo mover para cada um dos robôs. Eles também têm contato direto com os robôs e são responsáveis pela movimentação dos mesmos. Quando um outro módulo deseja movimentar um robô, simplesmente envia uma mensagem e o módulo mover correspondente se encarrega de fazer o robô se mover de acordo com os dados enviados na mensagem.

Foram implementadas quatro maneiras diferentes de se mover os robôs. Apesar dos robôs possuírem maneiras diferentes de programação e movimentação, ambos os módulos mover fornecem os mesmos serviços. No entanto, mesmo sendo as trajetórias e velocidades do movimento equivalentes, devido às diferenças físicas entre os robôs, a forma como o movimento é realizado em cada um deles é diferente.

Os tipos de movimento implementados são baseados nos possíveis movimentos de Theus. Foram feitas adaptações para que o Nomad pudesse executar os mesmos movimentos de Theus, mesmo tendo a característica de ser não-holonômico.

Foi criado um conjunto de parâmetros que contém a velocidade e a aceleração que são utilizadas na movimentação, bem como a posição do robô no momento. Estas informações ficam armazenadas no módulo mover e podem ser atualizadas ou consultadas por outros módulos. Os parâmetros são armazenados e transmitidos em estruturas como a definida abaixo.

```
typedef struct
{
    long x;
    long y;
    long angle;
    long vtrans;
    long vangle;
    long atrans;
    long aangle;
} THEUS_MOVE_PARAMS_TYPE;
```

Se um módulo quiser, por exemplo, alterar apenas os valores de aceleração, basta preencher os campos `atrans` e `aangle` com os valores desejados e os demais campos com a constante `NO_VALUE`. Feito isso, o módulo envia uma mensagem para atualizar os

parâmetros e o módulo mover irá, a partir deste momento, utilizar o novo valor de aceleração.

Os parâmetros são os mesmos para ambos os robôs e os valores são sempre em milímetros, no caso de distâncias, e milirradianos, no caso de ângulos. Porém, existem diferenças entre os robôs, não apenas na unidade dos valores, mas principalmente em como estes valores são utilizados.

Cada robô possui três eixos. Cada eixo tem os valores correntes de posição, velocidade e aceleração (vide tabela 4.3). A única exceção é o eixo de translação do Nomad, onde o valor da posição é dada por um par (x,y) indicando a posição do robô no plano.

Tabela 4.3: Eixos de movimentação dos robôs.

Robô	**Eixo**	**Unidade**	**Descrição**
Nomad	Translação	0,1pol	Translação do robô na direção das rodas
	Direção	0,1°	Direção das rodas
	Torre	0,1°	Ângulo da torre que contém os sensores
Theus	X	Mm	Posição x do robô
	Y	Mm	Posição y do robô
	Rotação	Mrad	Ângulo da torre que contém os sensores

Nomad é um robô não-holonômico, isto é, ele se movimenta sempre na direção das rodas. Para que ele faça, por exemplo, uma curva de 90°, é preciso que ele pare, redirecione as rodas e volte a se mover. Theus, por outro lado, é um robô holonômico. Ele pode fazer esta mesma curva de 90° bruscamente, sem que precise parar seu movimento.

Um detalhe importante é que o conjunto de eixos de Theus tem um campo chamado `Global` que pode ter valor verdadeiro ou falso. Quando este valor é verdadeiro, os eixos x e y são globais e não mudam de direção quando o robô gira. Caso contrário, o sistema de coordenadas rotaciona mantendo o eixo y sempre na direção da parte frontal do robô.

Abaixo são descritos os quatro tipos de movimento e mostrado como cada um destes foi implementado, tanto em Theus quanto no Nomad. As diferenças na forma de programação e das características físicas de cada robô trazem particularidades na execução do mesmo movimento por um ou por outro robô.

TRANSSPEED:

No movimento tipo TRANSSPEED o módulo **mover** recebe dois valores na mensagem. Estes valores representam a velocidade de translação (vtrans) e a velocidade de rotação (vangle). O robô deve mover-se com estas velocidades para frente. Note que a frente do robô muda de direção quando vangle é diferente de zero. Neste caso, o robô segue uma trajetória em forma de circunferência (Figura 4.8).

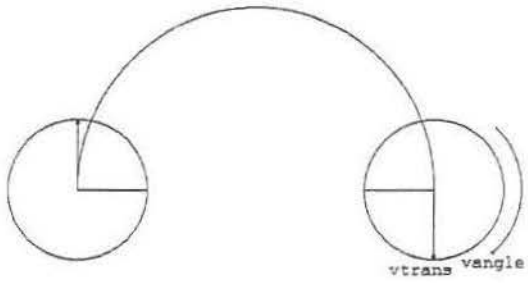


Figura 4.8: Movimento tipo TRANSSPEED.

No Nomad, esta forma de movimentação é a mais natural. Simplesmente atribui-se para o eixo de translação a velocidade vtrans e para os eixos de direção das rodas e da torre a velocidade de rotação vangle.

Em Theus, também é simples este tipo de movimento. O eixo x recebe velocidade zero, enquanto o eixo y recebe a velocidade vtrans e a rotação recebe a velocidade vangle. Além disso, deve-se atribuir ao campo Global dos eixos o valor FALSE para que o eixo y indique sempre a frente do robô.

AXESSPEED:

Neste tipo de movimento, são fornecidos três valores de velocidades, vx, vy e vangle, respectivamente para as velocidades no eixo x, no eixo y e no eixo de rotação. A trajetória do robô é sempre uma linha reta e a torre gira a uma velocidade angular vangle, sem influenciar no movimento do robô (Figura 4.9).

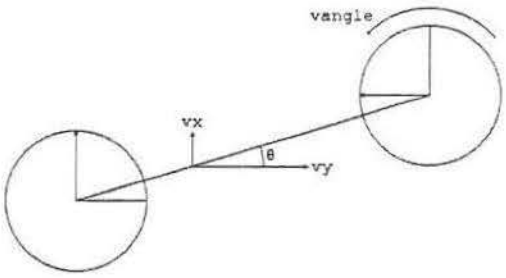


Figura 4.9: Movimento tipo AXESSPEED.

Em Theus, este movimento é implementado da seguinte forma: primeiramente, deve-se atribuir ao campo `Global` dos eixos o valor `TRUE` para que o sistema de coordenadas não mude com a rotação da torre. Em seguida, os eixos x, y e de rotação recebem velocidades com os valores `vx`, `vy` e `vangle` movimentando o robô.

Este mesmo movimento é implementado no Nomad de forma bem diferente. A velocidade de rotação da torre é simples, basta usar a velocidade angular `vangle`. Para o movimento no plano xy, ele deve primeiramente direcionar suas rodas de tal forma que, quando receber um certo valor de velocidade para frente, a decomposição desta velocidade nos eixos x e y tenham respectivamente os valores fornecidos `vx` e `vy`.

O valor desta velocidade é o módulo de um vetor V com componentes `vx` e `vy` e a direção do movimento é a direção (θ) deste mesmo vetor. Portanto, para se gerar um movimento com velocidades `vx` e `vy`, é necessário direcionar as rodas com um ângulo θ em relação ao eixo x e fornecer uma velocidade de translação com o valor |V|. |V| e θ são calculados com as equações abaixo:

$$|V| = \sqrt{vx^2 + vy^2} \quad \text{e} \quad \theta = \text{atan2}(vy, vx) \tag{4.1}$$

onde a função atan2 retorna o arco-tangente de vy/vx, sem o problema da divisão por zero e diferenciando os quadrantes, isto é, o ângulo retornado assume valores entre $-\pi$ e π e não entre $-\pi/2$ e $\pi/2$ como a função arco-tangente comum.

RELPOINT:

Neste tipo de movimento, são fornecidos três valores de posição, x, y e angle, que representam um ponto relativo tendo a posição do robô antes deste movimento como referencial. O robô deve mover-se até estas coordenadas de tal forma que a trajetória até o ponto alvo seja uma reta (Figura 4.10). Além disso, a rotação da torre deve ocorrer simultaneamente ao movimento do robô no plano xy.

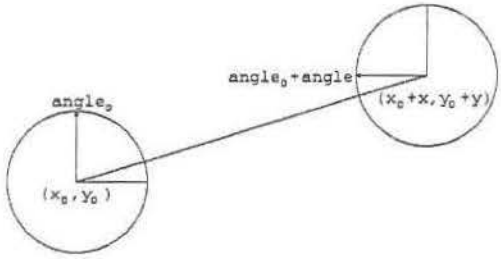


Figura 4.10: Movimento tipo RELPOINT.

Em Theus, isto seria facilmente realizado se não fossem as restrições de simultaneidade. Bastaria simplesmente fornecer as posições alvo para cada eixo e ordenar que o robô se movesse até elas. No entanto, o movimento no plano deve ser uma reta, o que restringe a velocidade em cada eixo. Como a rotação também deve ser simultânea, a velocidade angular também deve ser restringida.

Para facilitar o cálculo das três velocidades, o movimento pode ser considerado uma reta no espaço tridimensional, sendo a rotação da torre o terceiro eixo no espaço. Considerando-se o vetor P de três dimensões, com componentes x, y e angle, tem-se que o movimento neste espaço tridimensional será na mesma direção de P, portanto o vetor velocidade e o vetor aceleração também terão esta mesma direção. Conhecendo-se os módulos dos vetores velocidade e aceleração, pode-se facilmente calcular as componentes destes, pois como são paralelos a P, devem ter componentes proporcionais aos de P, isto é, proporcionais a x, y e angle.

No Nomad, o cálculo das velocidades é análogo ao de Theus e a execução é parecida com o movimento do tipo AXESSPEED do próprio Nomad. Primeiramente calculam-se as componentes do vetor como é feito em Theus. Com isso obtém-se vx, vy e vangle. A

soma vetorial de vx e vy resulta no vetor velocidade, enquanto a de x e y resulta no vetor posição, cujo módulo é a distância até o ponto alvo.

As rodas devem, então, ser colocadas na direção do vetor velocidade e em seguida devem percorrer a distância calculada. Enquanto isso, a torre deve rotacionar com a velocidade calculada vangle até atingir a posição desejada.

ABSPOINT:

Este movimento é praticamente idêntico ao anterior. A única diferença é que os valores fornecidos x, y e angle representam um ponto em relação ao sistema de coordenadas absoluto do robô (Figura 4.11). A realização deste movimento, tanto pelo Nomad quanto por Theus, é feita calculando-se as coordenadas relativas do ponto alvo e utilizando-se os mesmos procedimentos descritos acima.

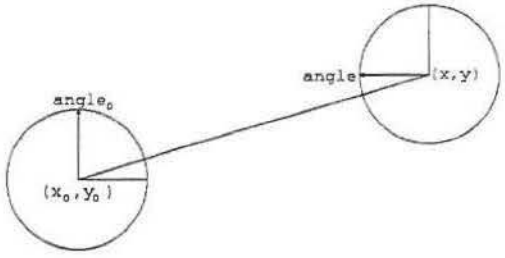


Figura 4.11: Movimento tipo ABSPOINT.

Módulo go

Este é o módulo responsável por coordenar a execução da tarefa como um todo. Nos módulos sensors e mover descritos anteriormente, não faz sentido falar em programação TDL pois são módulos básicos, cujas ações não têm nível de complexidade muito elevado. O módulo go, por outro lado, possui o nível mais alto na hierarquia das tarefas, portanto utiliza a linguagem TDL para quebrar a tarefa em subtarefas que serão executadas pelos outros módulos.

A tarefa realizada pelo módulo go é dividida em 3 subtarefas: LocateRobots, Approach e Follow, executados nesta ordem (Figura 4.12). As subtarefas Approach e Follow são implementadas aqui como um comando TDL, pois as árvores de tarefas que

os executam não estão neste módulo. Os comandos `Approach` e `Follow` simplesmente cuidam da comunicação com os módulos que executam a subtarefa. O comando `Approach` é executado apenas uma vez durante a tarefa, enquanto o comando `Follow` pode ser executado mais de uma vez, dependendo do número de pontos alvo fornecidos pelo operador.

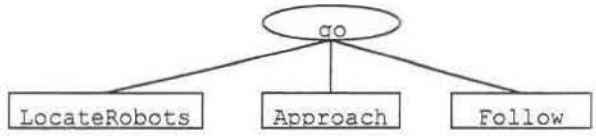


Figura 4.12: Árvore de tarefas do módulo go.

O comando `LocateRobots` é responsável por descobrir a posição relativa inicial dos robôs, isto é, os ângulos e a distância entre eles. A maneira encontrada para a execução deste comando foi deixar que um operador humano entre com os valores pedidos através do teclado. Apesar de parecer exagero utilizar um comando TDL para algo tão simples, isso facilitará um eventual futuro trabalho decorrente de, por exemplo, a incorporação de um novo *hardware* capaz de localizar automaticamente os robôs. Tudo o que o futuro programador precisará fazer será substituir o comando `LocateRobots` por um comando que utilize o novo *hardware*.

Módulos gonear e bringnear

Estes módulos fazem a aproximação inicial entre os dois robôs. Executado em Theus, o módulo gonear é responsável por levá-lo até as proximidades do Nomad, enquanto bringnear, executado no Nomad, auxilia gonear nesta tarefa. As posições relativas entre os dois robôs são conhecidas. Durante a aproximação o Nomad não se move. Theus é quem deve ir em direção ao Nomad, percorrendo uma trajetória retilínea.

As informações sobre posicionamento relativo entre os dois robôs consistem em três valores: D, θ_1 e θ_2, onde D é a distância entre os robôs, θ_1 é o ângulo onde Nomad se encontra em relação a Theus e θ_2 é o ângulo em que Theus se encontra em relação ao Nomad. Veja figura 4.13.

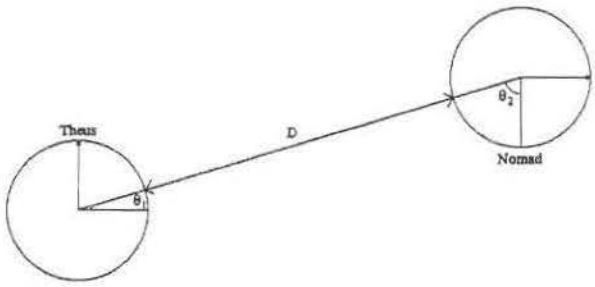


Figura 4.13: Posicionamento relativo entre os robôs.

Com estas informações, os robôs devem primeiramente rotacionar de tal forma que a frente de Theus seja direcionada para a parte de trás do Nomad, isto é, θ_1 deve ser igual a 90° e θ_2 deve ser –90°. Assim que os robôs estiverem direcionados, Theus pode iniciar a aproximação, movendo-se com velocidade constante para frente. Quando Theus detecta o Nomad ou vice-versa através dos sensores infra-vermelhos, a velocidade passa a diminuir até que se atinja uma distância razoável e o robô pare.

A árvore de tarefas do módulo gonear pode ser visto na figura 4.14. A tarefa se divide em 3 subtarefas e possui, associadas ao seu nó principal, dois tratadores de exceção que são acionados caso Theus não consiga aproximar-se do Nomad.

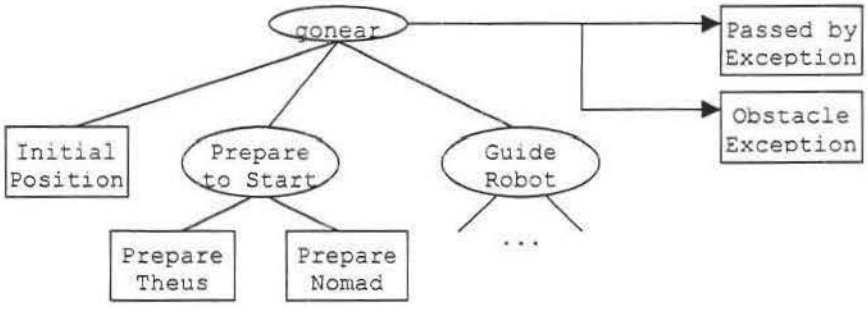


Figura 4.14: Árvore de tarefas do módulo gonear.

A subtarefa `InitialPosition` guarda o posicionamento inicial de Theus. Esta informação é importante, pois durante a aproximação pode-se sempre saber a distância percorrida pelo robô. A subtarefa `PrepareToStart` faz o direcionamento inicial dos

robôs. Ela se divide em outras duas subtarefas, `PrepareTheus` e `PrepareNomad`, que são responsáveis pelo direcionamento de Theus e do Nomad respectivamente.

A subtarefa `GuideRobot` é quem vai efetivamente realizar a aproximação. Ela é repetida várias vezes até que se atinja a posição desejada. `GuideRobot` se divide em cinco outras subtarefas (Figura 4.15). Primeiramente é feita uma requisição ao módulo bringnear para que este calcule a trajetória de Theus levando-se em conta as leituras dos sensores do Nomad. Enquanto isso, Theus calcula também a trajetória utilizando seus próprios sensores. Em seguida, gonear pede para bringnear enviar-lhe a trajetória calculada. É feita então uma combinação entre as trajetórias calculadas e finalmente movimenta-se o robô conforme a trajetória final.

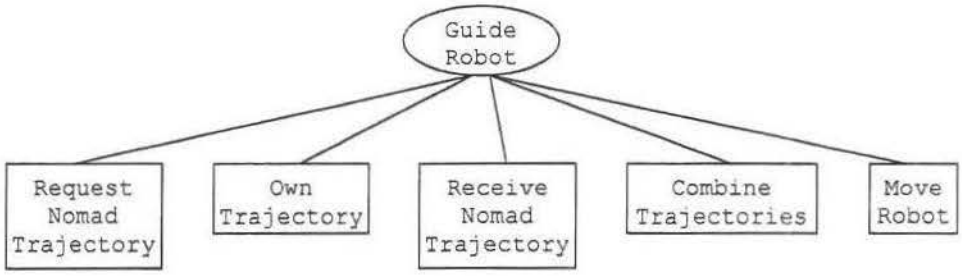


Figura 4.15: Subárvore da tarefa `GuideRobot`.

A trajetória calculada pode ser representada por diversas maneiras, uma seqüência de pontos, velocidades de rotação e translação, etc. Nesta implementação as trajetórias são representadas por uma velocidade de translação, pois como é assumido que os robôs está corretamente direcionado, Theus não deve ter velocidade de rotação. A velocidade é calculada em cada um dos robôs sendo ela proporcional à distância que falta para se atingir a distância desejada, para a qual foi adotado o valor de 20 cm. A combinação das trajetórias, neste caso, é a média aritmética dos valores calculados por um e pelo outro robô.

A palavra `Trajectory` presente nos nomes da maioria das subtarefas é apenas uma maneira genérica para se referenciar o movimento do robô. Nesta implementação, foi utilizada, como informação de trajetória, simplesmente a velocidade com a qual o robô deve mover-se para frente. Esta velocidade depende exclusivamente da distância detectada

nos sensores infra-vermelhos localizados, tanto em Theus quanto no Nomad, nas regiões que podem detectar o outro robô.

O cálculo da velocidade por Theus é feito pela subtarefa `OwnTrajectory` e, quando solicitado, é feito também pelo módulo bringnear no Nomad. As subtarefas `RequestTrajectory` e `ReceiveTrajectory` fazem a comunicação com o módulo bringnear. Estas duas subtarefas entram em operação apenas quando o módulo bringnear detecta a aproximação de Theus e avisa o módulo gonear que a partir deste momento o Nomad poderá auxiliá-lo com o cálculo da trajetória.

Na subtarefa `MoveRobot` é enviado o comando para que o robô se mova conforme a velocidade calculada pelas subtarefas anteriores. Além disso, aqui se verifica se a tarefa de aproximação deve continuar ou ser abortada, isto é, se o robô deve parar de se mover e uma exceção deve ser lançada.

Em duas situações a tarefa deve ser abortada. A primeira é quando o cálculo da velocidade por Theus resulta em zero enquanto o cálculo feito pelo Nomad resulta numa velocidade relativamente alta. Isto indica que Theus já terminou a aproximação mas o Nomad não o está detectando próximo de si. A conclusão que se tira desta situação é que há um obstáculo entre Theus e o Nomad (Figura 4.16). Theus concluiu a aproximação, mas não em relação ao Nomad, e sim em relação a um obstáculo, por isso uma exceção do tipo `ObstacleException` deve ser lançada.

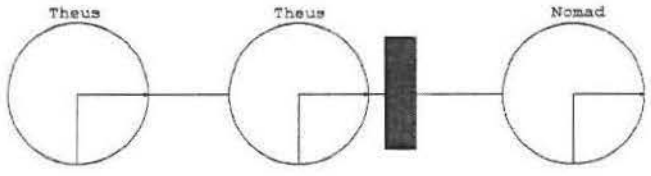


Figura 4.16: Exceção tipo `ObstacleException`.

A segunda situação onde a tarefa deve ser abortada é quando Theus já andou uma distância maior do que a que precisaria para encontrar o Nomad, mas mesmo assim não detecta nada em seu conjunto de sensores. Esta situação pode ocorrer quando a informação sobre o posicionamento inicial dos robôs contiver um erro muito grande, principalmente em θ_1. Theus se moverá na direção errada e nunca encontrará o Nomad (Figura 4.17). Neste

caso, após percorrer uma certa distância e não encontrar o Nomad, uma exceção do tipo `PassedByException` deve ser lançada.

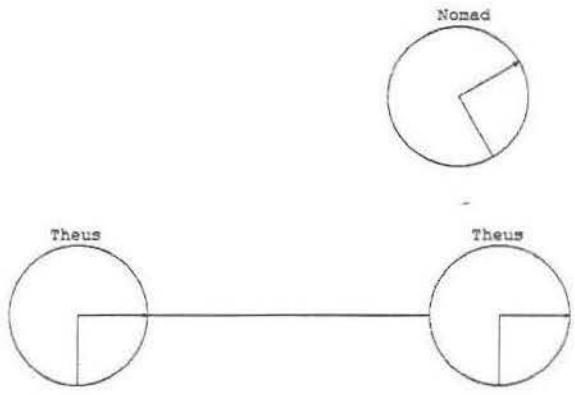


Figura 4.17: Exceção tipo `PassedByException`.

Quando a tarefa de aproximação termina, o módulo gonear envia uma mensagem notificando o módulo go. Juntamente com esta notificação, é informado se a aproximação foi bem sucedida ou se ela falhou, pois a tarefa de seguimento só será realizada se a aproximação for corretamente concluída.

Módulo detect

Existe um módulo detect para cada robô. A função do módulo é fazer o rastreamento do outro robô, isto é, manter a posição do outro robô conhecida o tempo todo. Ao receber uma consulta, o módulo detect de um robô deve respondê-la fornecendo a distância e o ângulo no qual o outro robô se encontra em relação ao primeiro.

O módulo consiste basicamente em um monitor TDL que fica periodicamente renovando a informação da posição do outro robô. Este monitor pode ser inicializado e interrompido por outros módulos. Quando inicializado, deve receber uma sugestão da posição inicial. Esta sugestão é dada fornecendo-se o índice do sensor que tem a maior probabilidade de detectar o outro robô.

Abaixo é descrito como cada um dos robôs detecta o posicionamento do outro através do módulo detect. Devido às diferenças tanto de *hardware* quanto de *software* entre os dois robôs, a forma como cada um deles faz o rastreamento da posição do outro é diferente.

Através de seu módulo detect, o Nomad calcula os valores de D_{Theus} e A_{Theus}, respectivamente sua distância em relação a Theus e o ângulo em que este é detectado pelo Nomad (Figura 4.18).

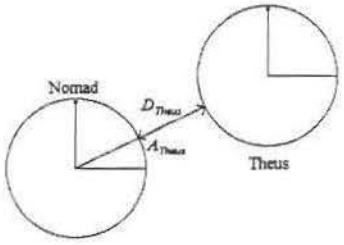


Figura 4.18: Nomad rastreia Theus.

O Nomad utiliza dois tipos de sensores para rastrear Theus, os sonares e os infra-vermelhos. Os sonares têm uma precisão melhor para distâncias maiores, enquanto os infra-vermelhos são mais precisos para distâncias menores. Uma distância limite de 15cm foi determinada através de experimentos práticos. Para distâncias abaixo deste limite são utilizados os infra-vermelhos e para distâncias acima dele são utilizados os sonares. Os sonares têm uma precisão muito melhor que os infra-vermelhos, mas a utilização dos infra-vermelhos foi necessária, pois para distâncias menores que 15cm, o valor retornado pelo sonar deixa de ser confiável.

A cada ativação do monitor é guardado o índice do último sensor central. Assume-se que no tempo de uma iteração Theus não se moverá uma distância maior que a que pode ser detectada em uma vizinhança de 5 sensores, isto é, 112,5°. Procura-se então, o sonar com menor valor nesta vizinhança e este será o novo sonar central.

Se o valor do sonar central for menor que 15cm, utiliza-se para o cálculo da posição de Theus os sensores infra-vermelhos. Caso contrário, utilizam-se os sonares.

Devem ser calculados a distância de Theus (D_{Theus}) e o ângulo (A_{Theus}) em que ele se encontra em relação ao Nomad. Para isso é utilizado o sensor central e o sensor vizinho a este que possui o menor valor. Os valores do sensor central e de seu vizinho serão chamados respectivamente de m_1 e m_2. Com a ajuda de experimentos práticos, foram obtidas as seguintes equações para o cálculo do posicionamento:

$$
\begin{aligned}
D_{Theus} &= (1{,}25m + 8{,}75)m_1 - 60 \\
A_{Theus} &= \left(\frac{654{,}5}{m} - 458{,}1\right) + A_{sc} \\
m &= \frac{m_2}{m_1}
\end{aligned}
\qquad (4.2)
$$

onde A_{sc} é o índice do sensor central em relação ao sistema de coordenadas do Nomad.

A primeira parcela em ambas equações foram resultados de ajustes de curvas sobre dados adquiridos experimentalmente. No cálculo da distância é subtraído o valor 60, pois deve ser levada em conta uma diferença de 60mm entre os raios do anel de *bumpers* e do anel de sonares. Quando os sonares detectam, por exemplo, 150mm, na verdade Theus está a apenas 90mm dos *bumpers* do Nomad. A descrição de como estas equações e as demais utilizadas nos módulos detect foram obtidas está na seção 3 do apêndice 1.

No cálculo do ângulo, a primeira parcela representa apenas o ângulo de Theus em relação ao sensor central. Porém, deve ser retornado o ângulo em relação ao sistema de coordenadas do Nomad. A segunda parcela adiciona, ao ângulo relativo, o ângulo absoluto no qual se encontra o sensor central.

O cálculo com infra-vermelhos é feito com equações que, assim como no caso dos sonares, foram resultados de experimentos práticos.

$$
\begin{aligned}
&D_{Theus} = 40m_1 - 20 \text{ e } A_{Theus} = 0{,}0^\circ \quad se\left(\frac{m_1}{m_2}\right) > 1{,}5 \\
&D_{Theus} = 40m_1 - 40 \text{ e } A_{Theus} = 7{,}5^\circ \quad c.c.
\end{aligned}
\qquad (4.3)
$$

onde m_1 e m_2 são infra-vermelhos adjacentes com menores valores, sendo sempre $m_1 \geq m_2$.

O módulo detect de Theus funciona de forma semelhante. Também baseia-se em um monitor TDL que é acionado periodicamente. Como o Nomad utiliza os sonares, Theus não poderá utilizá-los, pois quando os dois sonares estão ligados ao mesmo tempo, há muita interferência entre eles, resultando em sinais com muito ruído.

A solução adotada foi utilizar exclusivamente os sensores infra-vermelhos. Felizmente a discretização dos infra-vermelhos de Theus não é tão severa quanto a do

Nomad. Por outro lado, a informação lida nos sensores é um valor proporcional ao retorno do sinal infra-vermelho enviado e não a distância ao objeto, o que significa que quanto maior o valor lido, mais perto o objeto se encontra. Para que não seja confundido com a distância, este tipo de medida será chamada de intensidade e denotada por I.

Um outro problema mais sério é que os sensores não são uniformes, isto é, sensores diferentes retornam valores diferentes para objetos posicionados à mesma distância. Foi feito um estudo que possibilitou a uniformização das leituras dos sensores infra-vermelhos em Theus. Este estudo está descrito na seção 2 do apêndice 1.

Assim como no Nomad, é guardada a informação do sensor central. O novo sensor central é procurado numa vizinhança que inclui o central anterior, três vizinhos da direita e três da esquerda, formando um ângulo de 105°, não muito menor que o utilizado no Nomad.

Através de seu módulo **detect**, Theus deverá calcular os valores de I_{Nomad} e A_{Nomad}, respectivamente sua distância em relação ao Nomad e o ângulo em que este é detectado por Theus (Figura 4.19).

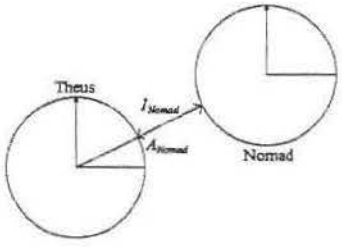


Figura 4.19: Theus rastreia Nomad.

O módulo **detect** de Theus não retorna efetivamente a distância ao objeto, e sim um valor equivalente à intensidade. Este valor é equivalente a se posicionar um infra-vermelho diretamente na direção do Nomad e fazer a leitura no sensor. Porém, os cálculos continuam necessários, pois nem sempre o sensor central está exatamente na direção do Nomad.

Após ser determinado o novo sensor central, pode-se calcular a distância em relação ao Nomad. Este cálculo é feito utilizando-se três sensores, cujas intensidades são chamadas de m_a, m_c e m_p, referindo-se respectivamente ao sensor anterior, sensor central e sensor posterior.

São calculados dois valores para a intensidade e depois é feita uma média ponderada para se chegar ao valor final. Aqui foi utilizada uma média ponderada em relação ao quadrado do valor, que favorece os valores maiores, pois as distâncias calculadas através dos sensores com valores maiores são mais precisos.

$$
\begin{aligned}
I_a &= 0{,}93m_c + 0{,}28m_a \\
I_p &= 0{,}93m_c + 0{,}28m_p \\
I_{Nomad} &= \frac{m_a^2 I_a + m_p^2 I_p}{m_a^2 + m_p^2}
\end{aligned}
\qquad (4.4)
$$

O ângulo A_{Nomad} é calculado utilizando-se os três sensores. Para cada sensor, calcula-se independentemente o ângulo A_i. Cada um destes ângulos é formado por duas parcelas, sendo uma o ângulo relativo entre o sensor e o outro robô, e a outra o ângulo em que se encontra o sensor no próprio robô. O ângulo final é dado pela média ponderada dos três valores A_a, A_c e A_p.

Sabendo-se a distância do Nomad em relação a Theus, pode-se calcular, para cada um dos três sensores, a primeira parcela, que é o valor do ângulo em que Nomad se encontra em relação ao sensor. Para isso utiliza-se a seguinte fórmula:

$$
A_{ri} = \left[\frac{10^8}{3} \left(\frac{I_{Nomad}}{m_i} - 1 \right) \right]^{0,303}, i = [a, c, p] \qquad (4.5)
$$

O ângulo absoluto A_i é calculado somando-se ou subtraindo-se A_{ri} do ângulo A_{si} no qual se encontra o sensor i no sistema de coordenadas de Theus e será sempre constante para cada sensor. Aplicando-se a fórmula para os três sensores, obtém-se a tabela 4.4.

Tabela 4.4: Possíveis valores para o ângulo.

Sensor	*Direita*	*Esquerda*
A	$A_{sa} + A_{ra}$	$A_{sa} - A_{ra}$
C	$A_{sc} + A_{rc}$	$A_{sc} - A_{rc}$
P	$A_{sp} + A_{rp}$	$A_{sp} - A_{rp}$

Para cada linha, apenas uma das opções é válida. Nos casos dos sensores anterior e posterior, a opção válida será sempre o ângulo que estiver mais próximo ao sensor central, portanto os ângulos referentes a estes sensores serão obtidos sempre a partir das seguintes fórmulas:

$$\begin{aligned} A_a &= A_{sa} + A_{ra} \\ A_p &= A_{sp} - A_{rp} \end{aligned} \tag{4.6}$$

O ângulo calculado a partir do sensor central assume apenas um de dois valores. O valor escolhido é o que está mais próximo do sensor vizinho com maior valor, pois sua direção é mais frontal ao Nomad que a do vizinho com valor menor. O ângulo calculado pelo sensor central é, portanto, escolhido da seguinte forma (Figura 4.20):

$$\begin{aligned} A_c &= A_{sc} + A_{rc}, se(m_p > m_a) \\ A_c &= A_{sc} - A_{rc}, c.c. \end{aligned} \tag{4.7}$$

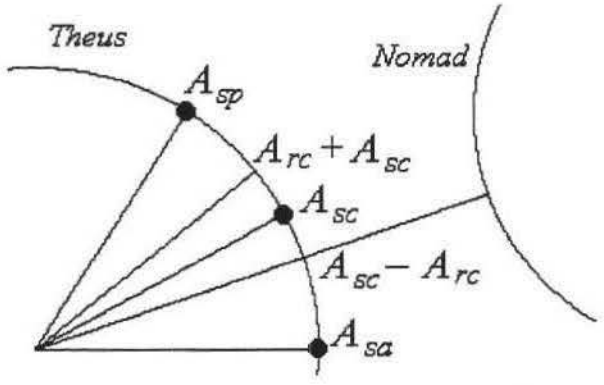


Figura 4.20: Sensor central: escolha do ângulo correto leva em conta os sensores adjacentes.

O valor final do ângulo é obtido através da média dos três valores calculados, ponderado em relação ao quadrado do valor lido no sensor. A média ponderada foi utilizada pelo mesmo motivo descrito para a equação (4.4). O sensor que mais influencia é sempre o central, pois é o que tem sempre o maior valor. O ângulo retornado é, portanto, dado por:

$$A_{Nomad} = \frac{m_a^2 A_a + m_c^2 A_c + m_p^2 A_p}{m_a^2 + m_c^2 + m_p^2} \tag{4.8}$$

Módulo fields

O módulo fields é executado pelo Nomad. O Nomad atuará como líder durante a execução da tarefa, portanto é necessário que ele seja capaz de navegar pelo ambiente. Foi implementado no módulo fields um algoritmo baseado em campos potenciais para a realização desta navegação.

O algoritmo de navegação por campos potenciais implementado aqui é baseado na descrição de Latombe [22]. Neste tipo de algoritmo, o robô é considerado como sendo uma partícula submetida a forças resultantes de campos potenciais. Os campos potenciais são gerados pelo ponto alvo (força de atração) e por obstáculos (forças de repulsão). O robô move-se na direção da força resultante, dada pela soma das forças atrativa e repulsivas, até atingir o ponto alvo.

Este tipo de algoritmo foi escolhido pois é o mais comumente utilizado para navegação sem a necessidade de um mapeamento prévio do ambiente. Como o ambiente para os testes é constantemente modificado, não seria viável a realização de um mapeamento a cada vez que a tarefa fosse realizada.

Esta implementação foi feita através de uma árvore TDL que tem como raiz o nó `Navigate`. Este, por sua vez, se divide em 5 subtarefas, como pode ser visto na figura 4.21.

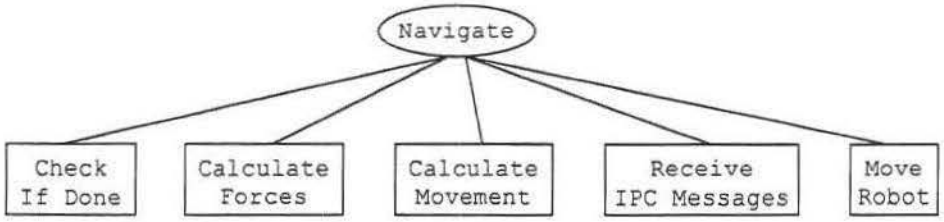


Figura 4.21: Árvore de tarefas do módulo fields.

A primeira subtarefa, `CheckIfDone`, verifica se o robô alcançou ou não a posição alvo. Quando a posição alvo é atingida, a navegação pára e uma mensagem é retornada confirmando a conclusão da tarefa com sucesso. Caso ainda não se tenha chegado ao destino, as outras subtarefas são executadas.

A subtarefa `ReceiveIPCMessages` verifica se há alguma mensagem que deve ser tratada pelo módulo. Em geral são mensagens ordenando que seja interrompida a navegação. Neste caso a navegação pára e uma mensagem é retornada informando a falha na execução da tarefa. O Nomad é efetivamente movimentado quando a subtarefa `MoveRobot` é executada e envia as informações de velocidade para o módulo mover.

`CalculateForces` calcula as forças que o campo potencial impõe ao robô e repassa o valor resultante da força para a subtarefa `CalculateMovement`. Esta, por sua vez, calcula as velocidades de translação e rotação para o robô.

O campo potencial é uma função de R^2 em R. A força que atua sobre o robô é o gradiente desta função potencial e é composta por dois tipos de força, a atrativa e a repulsiva. A força atrativa é exercida pelo ponto alvo, enquanto a repulsiva é exercida pelos obstáculos.

O ponto alvo deforma o campo potencial de maneira não uniforme. Nas proximidades do ponto alvo o campo tem a forma de uma parabolóide. A partir de um certo raio, o campo assume a forma de cone. Com isso, a força atrativa, dada pela derivada do campo potencial atrativo, fica constante na maior parte do ambiente e vai diminuindo proporcionalmente à distância do objetivo quando esta distância é menor que um certo limite.

A transição na função resultante deve ser suave, isto é, ela deve ser derivável no limite entre as duas funções que compõem o campo potencial atrativo. As seguintes equações são utilizadas para o cálculo dos campos potenciais:

$$U_{att} = 0{,}001|\vec{r}|^2, se(|\vec{r}| < 1000)$$
$$U_{att} = 2|\vec{r}| - 1000, c.c. \qquad (4.9)$$

onde U_{att} é o valor do campo potencial e r é o vetor distância em milímetros do robô em relação ao ponto alvo. Na figura 4.22 pode-se ver o valor do campo potencial em relação à distância do robô ao ponto alvo.

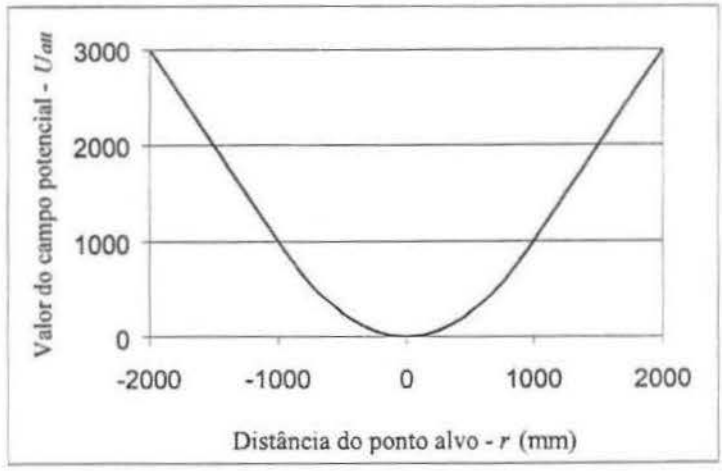


Figura 4.22: Campo potencial devido ao ponto alvo (valor do campo é adimensional).

As forças que atuam sobre o robô devido a este campo, são obtidas através do inverso do gradiente da função potencial. Ou seja, derivando-se as equações do campo potencial, obtêm-se as equações para as forças. Na prática não se calcula o valor do campo potencial, pois para o algoritmo interessam apenas as forças que atuam sobre o robô.

$$\vec{F}_{att} = -0{,}002\vec{r}, se(|\vec{r}| < 1000)$$
$$\vec{F}_{att} = \frac{-2\vec{r}}{|\vec{r}|}, c.c. \qquad (4.10)$$

As forças de repulsão, exercidas pelos obstáculos, são calculados a partir dos sensores infra-vermelhos e dos sonares. O Nomad possui 16 sensores de cada tipo, distribuídos uniformemente ao redor de sua torre. Cada sensor contribuirá com uma certa força de repulsão. A direção desta força depende da posição do sensor em relação à torre e da torre em relação ao ambiente, enquanto o módulo é calculado em função da distância medida por este sensor.

$$\begin{aligned} |U_{rep}(s)| &= \frac{1}{2}\eta\left(\frac{1}{d(s)} - \frac{1}{d_0}\right)^2, se(d(s) < d_0) \\ |U_{rep}(s)| &= 0, c.c. \end{aligned} \tag{4.11}$$

onde $d(s)$ é a distância ao obstáculo detectada pelo sensor s e d_0 é a distância de influência dos obstáculos. Para os sonares, $\eta = 1{,}2.10^7$ e $d_0 = 5000$mm, enquanto para os infra-vermelhos, $\eta = 7.10^7$ e $d_0 = 600$mm. Estes valores de η e d_0 são resultados de ajustes feitos durante experimentos práticos. Conforme o comportamento do robô, os valores eram modificados. Primeiro foi feito um ajuste mais grosseiro em testes simulados e depois um refinamento em testes com o robô real.

A figura 4.23 traz o campo potencial calculado através dos sonares, enquanto a figura 4.24 traz o campo potencial calculado através dos sensores infra-vermelhos, ambos em relação à distância do robô ao obstáculo.

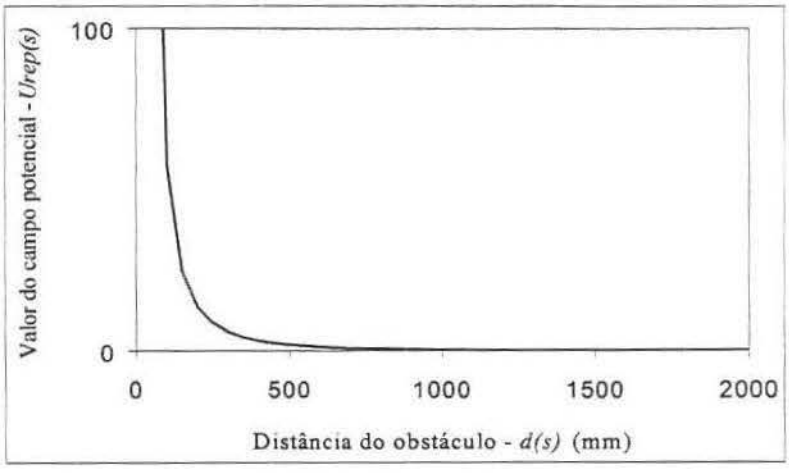


Figura 4.23: Campo potencial parcial para um sonar s (valor do campo é adimensional).

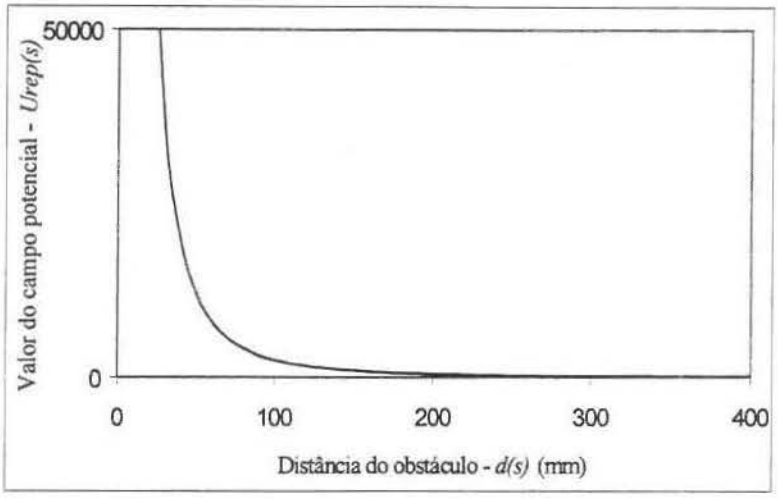


Figura 4.24: Campo potencial parcial para um infra-vermelho s (valor do campo é adimensional).

Cada sensor s contribui com uma parcela $F_{rep}(s)$ na direção oposta à qual ele está posicionado. Esta parcela é calculada através da derivada da função potencial como:

$$\begin{aligned} \left|\vec{F}_{rep}(s)\right| &= \eta\left(\frac{1}{d(s)} - \frac{1}{d_0}\right)\frac{1}{d^2(s)}, se(d(s) < d_0) \\ \left|\vec{F}_{rep}(s)\right| &= 0, c.c. \end{aligned} \tag{4.12}$$

A força repulsiva total é a soma vetorial de todas as parcelas $F_{rep}(s)$, tanto dos sonares quanto dos infra-vermelhos.

$$\vec{F}_{rep} = \sum_{s} \vec{F}_{rep}(s) \tag{4.13}$$

Esta equação funcionaria bem em uma situação genérica, na qual o Nomad tivesse que navegar sozinho. Porém, nesta aplicação, Nomad será seguido por Theus durante a navegação. Neste caso, Theus seria considerado como um obstáculo qualquer, fazendo com que o Nomad se preocupasse em "fugir" de Theus, eventualmente impedindo-o de chegar até o ponto alvo.

Como visto anteriormente, o módulo detect do Nomad mantém a informação sobre a localização de Theus. Com esta informação, o Nomad pode diferenciar Theus dos demais obstáculos e percorrer da melhor maneira seu caminho até o ponto alvo. Esta diferenciação é feita ignorando-se a região dos sensores em que Theus é detectado. Portanto, a somatória das parcelas é dada por:

$$\vec{F}_{rep} = \sum_{s} \vec{F}_{rep}(s), s \notin S_{Theus} \tag{4.14}$$

onde S_{Theus} é o conjunto de sensores infra-vermelhos e de sonares que detectam Theus. Os elementos de S_{Theus} são o sensor direcionado diretamente para Theus e seus quatro vizinhos, dois de cada lado. A área em cinza na figura 4.25 representa a área desconsiderada pelo Nomad na detecção de obstáculos.

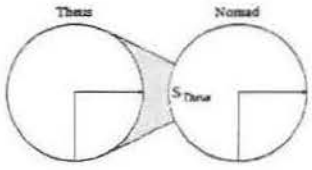


Figura 4.25: Nomad sabe que Theus não é um obstáculo.

Outro problema surge quando há um obstáculo próximo ao ponto alvo. Como a força de atração perto do ponto alvo é menor, as forças repulsivas tendem a sobressair, ficando o robô sem atingir o objetivo, preso em um mínimo local. Isso pode ser resolvido atenuando-se também as forças repulsivas, usando:

$$\vec{F}_{rep} = \mu \sum_{s} \vec{F}_{rep}(s), s \notin S_{Theus}$$

onde

$$\mu = \frac{|\vec{F}|}{1000}, se(|\vec{F}| < 1000) \tag{4.15}$$
$$\mu = 1, c.c.$$

A força total é a soma vetorial das forças atrativa e repulsiva.

$$\vec{F} = \vec{F}_{att} + \vec{F}_{rep} \tag{4.16}$$

O resultado final da força sobre o robô é utilizada pela subtarefa `CalculateMovement`. Nesta subtarefa são calculadas, em função desta força, as velocidades de translação V_T e de rotação V_A do Nomad.

A velocidade de rotação é calculada baseada na diferença entre a direção do robô e a direção do vetor força. Esta diferença é medida em radianos e é referenciada aqui por A_{diff}. A velocidade de rotação é, portanto, dada pela fórmula:

$$V_A = \pm V_{A\max} \sqrt{\frac{|A_{diff}|}{\pi}} \tag{4.17}$$

onde $V_{A\max}$ é a máxima velocidade angular permitida.

O valor de $V_{A\max}$ é lido do arquivo fields.info e pode ser alterado pelo usuário. Quanto maior a diferença entre a direção do robô e a da força, maior será V_A, que pode ser tanto positivo quanto negativo. O sinal de V_A é o mesmo de A_{diff}, pois o robô tenderá sempre a diminuir o valor absoluto desta diferença.

A velocidade de translação é calculada em função da força e também da velocidade de rotação. A velocidade é proporcional ao módulo da força, mas, como o Nomad não é holonômico, esta velocidade nem sempre possui a mesma direção da força. Quando isto ocorre, o Nomad tentará, através da velocidade angular, corrigir seu direcionamento. Portanto, quanto maior a velocidade angular, menor deverá ser a velocidade de translação.

$$V_T = \frac{|\vec{F}|}{|\vec{F}_{\max}|} v(V_A) \tag{4.18}$$

onde $F_{\max}$ é o valor máximo para a força e $v(V_A)$ é uma função exponencial que retorna uma velocidade de translação em função da velocidade de rotação.

$$v(V_A) = V_{T\min}\left(\frac{V_{T\max}}{V_{T\min}}\right)^{vc}, \text{ onde } vc = \frac{V_{A\max} - |V_A|}{V_{A\max}} \tag{4.19}$$

Assim como V_{Amax}, V_{Tmax} e V_{Tmin} estão armazenados no arquivo fields.info e podem ser alterados pelo usuário. O expoente vc é o complemento normalizado do módulo da velocidade de rotação. Isto faz com que a velocidade de translação esteja sempre entre V_{Tmax} e V_{Tmin}, assumindo o valor V_{Tmax} quando V_A for zero e V_{Tmin} quando V_A for V_{Amax}. A figura 4.26 mostra o comportamento da curva para os valores utilizados na prática, onde V_{Tmax} = 100mm/s, V_{Tmin} = 3mm/s e V_{Amax} = 200mrad/s.

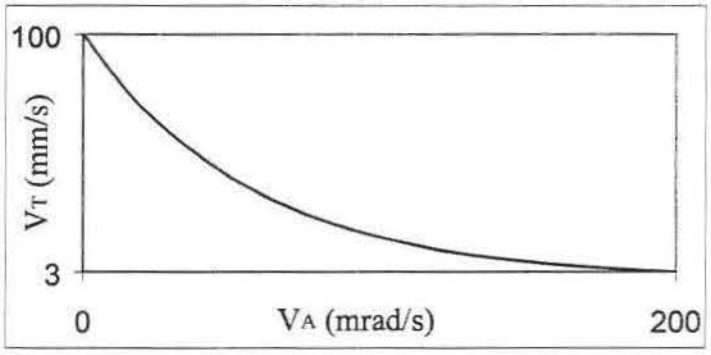


Figura 4.26: Velocidade de rotação x velocidade de translação.

Note que o robô se move rápido apenas quando a velocidade de rotação é bem baixa, isto é, quando o robô está na direção correta. Quando a velocidade de rotação é alta, o robô quase pára até começar a andar na direção indicada pela força.

Módulo follow

O módulo follow é executado por Theus. Theus atuará como seguidor durante a execução da tarefa e é este módulo que contém o algoritmo que torna isto possível.

Esta implementação foi feita através de uma árvore TDL que tem como raiz o nó `Follow`. Este, por sua vez, se divide em 4 subtarefas, como pode ser visto na figura 4.27.

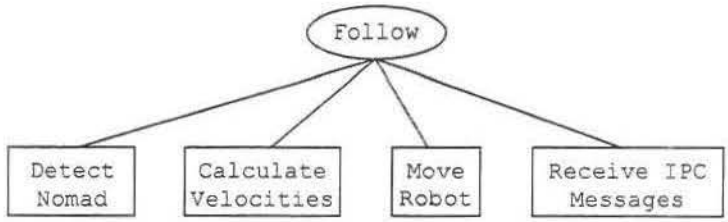


Figura 4.27: Árvore de tarefas do módulo follow.

A subtarefa `ReceiveIPCMessages` é como a de mesmo nome que existe no módulo fields do Nomad e `MoveRobot` movimenta Theus com as velocidades calculadas e fornecidas pela subtarefa `CalculateVelocities`.

A subtarefa `DetectNomad` retorna a posição relativa do Nomad com o auxílio do módulo detect de Theus. O posicionamento do Nomad é caracterizado por dois valores: a distância entre os robôs e o ângulo em que ele se encontra em relação a Theus.

A subtarefa de maior importância neste módulo é a `CalculateVelocities`. Ela deve, a partir da informação sobre o posicionamento do Nomad, calcular as velocidades de rotação e de translação de Theus.

A função deste algoritmo é manter Theus sempre a uma mesma distância do Nomad. Além disso, a frente de Theus deve estar sempre direcionada para o Nomad. Este trabalho é facilitado, pois como Theus é um robô holonômico, estes dois movimentos podem ser realizados independentemente.

A velocidade de rotação de Theus é calculada de forma a trazer o valor do ângulo detectado para 90°, pois esta é a direção da frente do robô (Figura 4.28). Quanto mais distante de 90° é o ângulo, maior é a velocidade de rotação no sentido de reposicionar a frente do robô na direção do Nomad. A velocidade de rotação é portanto dada por:

$$\begin{aligned} V_A &= \pm 15\left(\frac{A_{diff}}{100}\right)^2, se(A_{diff} < 700) \\ V_A &= \pm 150\ln\left(|A_{diff}| - 565{,}7\right), c.c. \end{aligned} \tag{4.20}$$

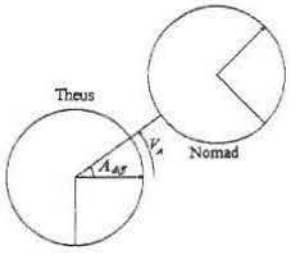


Figura 4.28: Theus gira em direção ao Nomad.

onde V_A é a velocidade de rotação em mm/s e A_{diff} é a diferença entre a frente de Theus e a posição do Nomad em miliradianos. A velocidade aumenta quadraticamente em relação à diferença entre os ângulos até 700mrad. A partir daí, ela aumenta logaritmicamente para que a velocidade não seja alta demais ultrapassando a permitida pelo robô. O valor da velocidade angular em relação à diferença do ângulo pode ser vista na figura 4.29.

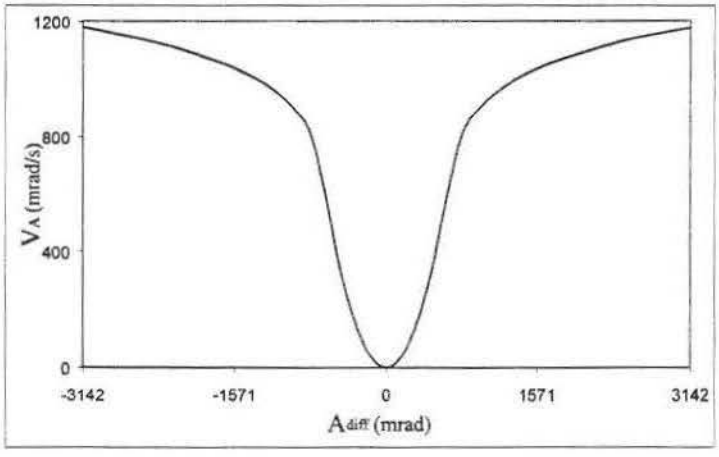


Figura 4.29: Velocidade de rotação em função da diferença angular.

Além da direção correta, Theus deve manter-se também a uma certa distância do Nomad. Para isso, é necessário que ele se mova, distanciando-se quando está muito perto e aproximando-se quando está muito longe.

Como o movimento de translação em Theus é independente do movimento de rotação, a velocidade de translação deve ser vista como um vetor, com módulo e direção.

A direção da velocidade é a direção do Nomad. O módulo da velocidade de translação é calculada a partir da distância em que Theus se encontra do Nomad. É preciso lembrar que esta distância não é uma medida de distância física, e sim um valor

proporcional ao retorno de sinal infra-vermelho que seria lido por um sensor direcionado diretamente para o Nomad. Quanto maior o valor do sinal, menor a distância. Este retorno de sinal é referenciado aqui, assim como no módulo detect do Theus, por I.

Theus deve manter-se a uma distância I_0 do Nomad (Figura 4.30). Na implementação usou-se $I_0 = 4000$, o que corresponde a uma distância em torno de 20cm. A fórmula para a velocidade de translação é muito parecida com a da velocidade de rotação:

$$\left|\vec{V}_{Tfollow}\right| = \pm 200(x)^2, se(|x| < 0{,}6)$$
$$\left|\vec{V}_{Tfollow}\right| = \pm 200\ln(|x| - 0{,}834), c.c. \quad \text{, onde } x = \ln\left(\frac{I_{Nomad}}{4000}\right) \tag{4.21}$$

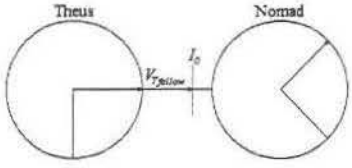


Figura 4.30: Theus se move em direção ao Nomad.

O retorno do sinal infra-vermelho é uma função exponencial da distância, por isso é utilizado o logaritmo natural de I_{Nomad}. O comportamento da velocidade de translação em relação à distância física entre os robôs pode ser vista na figura 4.31.

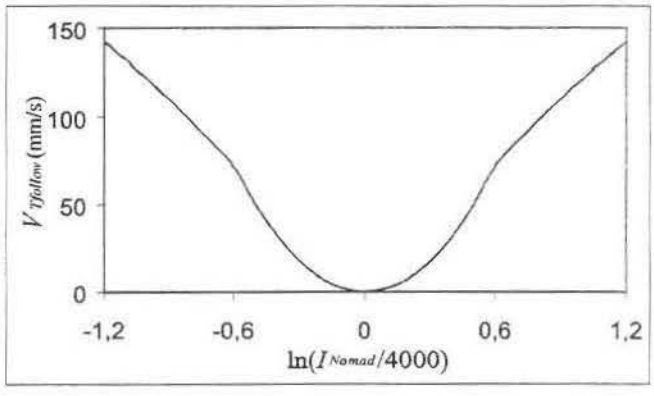


Figura 4.31: Velocidade de translação em função do retorno do sinal (I é adimensional).

Até aqui, o algoritmo funciona para ambientes sem obstáculos, pois a única informação que Theus recebe do mundo exterior é a posição relativa do líder. No entanto,

existem situações em ambientes com obstáculos em que o Nomad pode levar Theus a colidir com algum objeto. A figura 4.32 mostra uma situação em que o Nomad já desviou do obstáculo, mas se Theus o continuar seguindo, irá colidir. Portanto é necessário que Theus também seja capaz de desviar de obstáculos. Para isso, foi acrescentada ao cálculo da velocidade, uma técnica de desvio de obstáculos utilizando os sensores infra-vermelhos. Theus utiliza apenas os sensores que não detectam o Nomad para que este não seja considerado como obstáculo.

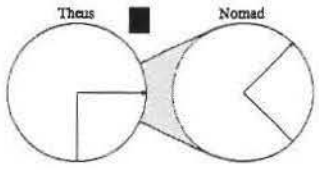


Figura 4.32: Theus é induzido a colidir com um obstáculo.

A técnica consiste em se acrescentar à velocidade já calculada $V_{Tfollow}$, uma velocidade de desvio de obstáculo V_{Tobst}. Esta velocidade é calculada através da soma vetorial das parcelas calculadas com os valores lidos nos infra-vermelhos. Cada infra-vermelho contribui com um vetor na sua direção e com módulo calculado a partir do retorno de sinal captado no sensor.

$$\begin{aligned} \left|\vec{V}_{Tobst}(s)\right| &= 0, se(s \in S_{Nomad}) ou (I(s) < 500) \\ \left|\vec{V}_{Tobst}(s)\right| &= (I(s) - 500)/50, c.c. \\ \vec{V}_{Tobst} &= \sum_{s} \left|\vec{V}_{Tobst}(s)\right| \end{aligned} \tag{4.22}$$

onde S_{Nomad} são os sensores que detectam o Nomad.

A velocidade total é dada por:

$$\vec{V}_T = \vec{V}_{Tfollow} + \vec{V}_{Tobst} \tag{4.23}$$

Esta velocidade deve ser passada para a subtarefa `MoveRobot`, que vai, por sua vez, enviá-la através de uma mensagem para o módulo **mover** de Theus. O tipo de movimento é `AXESSPEED`, onde cada eixo, x, y e de rotação, recebe um valor de velocidade e se move independentemente. O eixo de rotação recebe diretamente a velocidade V_A e a velocidade de translação V_T é decomposta em duas componentes V_x e V_y, que são enviadas respectivamente para os eixos x e y do robô.

Com isso, Theus se move sempre na direção do Nomad, aproximando-se ou afastando-se dele, tentando manter a distância constante. Enquanto isso, sua torre rotaciona tentando manter-se sempre de frente para o Nomad.

4.3.4 Troca de Mensagens

Até agora a descrição do sistema concentrou-se no funcionamento dos módulos que o compõem. Nesta seção será apresentada a comunicação entre os módulos e como as mensagens são trocadas entre eles através do IPC.

O sistema é executado por dois robôs e utiliza um esquema com três servidores centrais como descrito na seção 4.2. São utilizados um servidor global e dois servidores locais, um em cada robô. Mensagens trocadas entre módulos executados no mesmo robô passarão pelos servidores locais, enquanto mensagens trocadas entre módulos de máquinas diferentes passarão obrigatoriamente pelo servidor global.

Os servidores são iniciados antes que o primeiro módulo seja executado. Os servidores devem estar rodando nas máquinas já pré-definidas. O arquivo "hosts.info" é lido por todos os módulos e traz as informações da localização dos servidores, isto é, a máquina em que estão rodando e o número da porta em que estão recebendo as mensagens.

As mensagens estão todas definidas como macros em linguagem C++. Para uma melhor organização, estas definições estão todas armazenadas em arquivos que são incluídos pelos programas em TDL como arquivos ".h" do C++. Foi definido um padrão para a definição das mensagens e dos tipos de dados passados através delas.

Os nomes das mensagens são escritos em letras maiúsculas, sendo as palavras que compõem o nome separadas por *underscore*. Para uma melhor identificação, todo nome de mensagem é terminado com "`_MSG`". Cada mensagem tem associadas a ela mais três

definições que seguem o mesmo padrão de nomenclatura, mudando-se apenas a terminação do nome. A tabela 4.5 mostra os elementos que compõem uma mensagem.

Tabela 4.5: Elementos de uma mensagem do IPC.

Elemento	**Terminação**	**Função**
Nome	_MSG	Identifica uma mensagem
Handler	_HND	Nome da função que trata a mensagem recebida
Tipo dos dados	_TYPE	Estrutura que armazena os dados enviados ou recebidos
Formato dos dados	_FORM	Forma de armazenamento dos dados enviado com a mensagem

Um exemplo de como uma definição de mensagem é escrita, pode ser visto abaixo. Este exemplo define a mensagem que envia informações sobre os sonares de Theus.

```
typedef struct
{
    int isOn;
    long value[24][2];
} THEUS_SONAR_TYPE;
#define THEUS_SONAR_FORM "{int,[long:24,2]}"
#define THEUS_SONAR_MSG "TheusSonarMsg"
#define THEUS_SONAR_HND TheusSonarHnd
```

A seguir, serão descritas as principais mensagens trocadas entre módulos do sistema. As tabelas apresentadas nesta seção seguem alguns padrões de nomenclatura: T refere-se a Theus, N ao Nomad e C a ambos, por exemplo, N sensors refere-se ao módulo sensors do Nomad. O sinal * é utilizado quando mais de um módulo pode exercer a função citada.

As mensagens são caracterizadas aqui por quatro itens: *Message* é o nome da mensagem, *Sender* é o módulo que a envia, *Receiver* é o módulo que a recebe e *Server* é o servidor central pelo qual a mensagem é transmitida.

As tabelas 4.6 e 4.7 mostram as mensagens referentes aos módulos sensors dos dois robôs.

Tabela 4.6: Mensagens recebidas e enviadas pelo módulo sensors do Nomad.

Nomad Message	*Sender*	*Receiver*	*Server*
GET_NOMAD_INTEG_MSG	C *	N sensors	N local/C global
NOMAD_INTEG_MSG	N sensors	C *	N local/C global
GET_NOMAD_SONAR_MSG	C *	N sensors	N local/C global
NOMAD_SONAR_MSG	N sensors	C *	N local/C global
GET_NOMAD_INFRARED_MSG	C *	N sensors	N local/C global
NOMAD_INFRARED_MSG	N sensors	C *	N local/C global
GET_NOMAD_BUMPER_MSG	C *	N sensors	N local/C global
NOMAD_BUMPER_MSG	N sensors	C *	N local/C global

Tabela 4.7: Mensagens recebidas e enviadas pelo módulo sensors de Theus.

Theus Message	*Sender*	*Receiver*	*Server*
GET_THEUS_INTEG_MSG	C *	T sensors	T local/C global
THEUS_INTEG_MSG	T sensors	C *	T local/C global
GET_THEUS_SONAR_MSG	C *	T sensors	T local/C global
THEUS_SONAR_MSG	T sensors	C *	T local/C global
GET_THEUS_INFRARED_MSG	C *	T sensors	T local/C global
THEUS_INFRARED_MSG	T sensors	C *	T local/C global
GET_THEUS_BUMPER_MSG	C *	T sensors	T local/C global
THEUS_BUMPER_MSG	T sensors	C *	T local/C global

As mensagens que começam com GET são requisições feitas por outros módulos para os módulos sensors. As mensagens de mesmo nome sem a partícula GET são as respostas contendo as informações sobre os sensores. NOMAD ou THEUS no nome das mensagens identificam o robô e a partícula seguinte identifica o tipo de sensor requisitado.

Pode-se ver nas tabelas 4.6 e 4.7 que os módulos sensors têm a capacidade de receber requisições tanto através do servidor local quanto do servidor global. Porém, nesta tarefa apenas módulos locais utilizam informações provenientes diretamente dos sensores dos robôs.

As tabelas 4.8 e 4.9 mostram os serviços disponíveis nos módulos mover dos robôs.

Tabela 4.8: Mensagens recebidas e enviadas pelo módulo mover do Nomad.

Nomad Message	*Sender*	*Receiver*	*Server*
SET_NOMAD_MOVE_PARAMS_MSG	N *	N mover	N local
GET_NOMAD_MOVE_PARAMS_MSG	N *	N mover	N local
NOMAD_MOVE_PARAMS_MSG	N mover	N *	N local
NOMAD_MOVE_RELPOINT_MSG	N *	N mover	N local
NOMAD_MOVE_ABSPOINT_MSG	N *	N mover	N local
REPLY_NOMAD_MOVE_MSG	N mover	N *	N local
NOMAD_MOVE_TRANSSPEED_MSG	N *	N mover	N local
NOMAD_MOVE_AXESSPEED_MSG	N *	N mover	N local

Tabela 4.9: Mensagens recebidas e enviadas pelo módulo mover de Theus.

Theus Message	*Sender*	*Receiver*	*Server*
SET_THEUS_MOVE_PARAMS_MSG	T *	T mover	T local
GET_THEUS_MOVE_PARAMS_MSG	T *	T mover	T local
THEUS_MOVE_PARAMS_MSG	T mover	T *	T local
THEUS_MOVE_RELPOINT_MSG	T *	T mover	T local
THEUS_MOVE_ABSPOINT_MSG	T *	T mover	T local
REPLY_THEUS_MOVE_MSG	T mover	T *	T local
THEUS_MOVE_TRANSSPEED_MSG	T *	T mover	T local
THEUS_MOVE_AXESSPEED_MSG	T *	T mover	T local

As mensagens que possuem a partícula PARAMS no nome referem-se aos parâmetros de posição, velocidade e aceleração que são utilizados na movimentação dos robôs. Estes parâmetros são modificados e acessados através de SET e GET. A terceira mensagem de parâmetros é a resposta que o módulo mover envia a quem fez a requisição através de GET.

Os movimentos com velocidades definidas são inicializados através das mensagens TRANSSPEED e AXESSPEED. As mensagens de RELPOINT e ABSPOINT são referentes, respectivamente, aos movimentos até um ponto de coordenadas relativas e absolutas. Estas mensagens são respondidas pelos módulos mover através da mensagem REPLY quando o movimento termina. Todos os tipos de movimentos estão detalhados na descrição do módulo mover, na seção 4.3.3.

Ao contrário do módulo sensor, apenas os módulos locais ao robô podem se comunicar com o módulo mover. Esta decisão foi tomada para que se diminuísse o risco de

mais de um módulo tentar movimentar o robô. Isto não impede, porém, que haja conflitos entre módulos locais para a movimentação do robô, mas é muito mais fácil resolver este problema quando ele não envolve módulos externos ao robô.

As mensagens para se acessar os módulos detect são apresentadas nas tabelas 4.10 e 4.11:

Tabela 4.10: Mensagens recebidas e enviadas pelo módulo detect do Nomad.

Nomad Message	*Sender*	*Receiver*	*Server*
NOMAD_DETECT_THEUS_START_MSG	C *	N detect	N local/C global
NOMAD_DETECT_THEUS_STOP_MSG	C *	N detect	N local/C global
NOMAD_DETECT_THEUS_REQUEST_MSG	C *	N detect	N local/C global
NOMAD_DETECT_THEUS_RESPONSE_MSG	N detect	C *	N local/C global

Tabela 4.11: Mensagens recebidas e enviadas pelo módulo detect de Theus.

Theus Message	*Sender*	*Receiver*	*Server*
THEUS_DETECT_NOMAD_START_MSG	C *	T detect	T local/C global
THEUS_DETECT_NOMAD_STOP_MSG	C *	T detect	T local/C global
THEUS_DETECT_NOMAD_REQUEST_MSG	C *	T detect	T local/C global
THEUS_DETECT_NOMAD_RESPONSE_MSG	T detect	C *	T local/C global

A mensagem de START inicia e a de STOP pára o *tracking* do outro robô. A mensagem de REQUEST é enviada para o módulo detect que responde através da mensagem RESPONSE enviando o posicionamento relativo do outro robô.

O módulo detect poderia ser consultado por qualquer módulo de qualquer robô. Neste sistema, porém, um único módulo por robô utiliza o serviço de *tracking*. O serviço é utilizado pelo módulo fields no Nomad e pelo módulo follow em Theus.

As mensagens apresentadas até aqui são de uso genérico. São enviadas quando um módulo quer utilizar um serviço fornecido por outro módulo. Apesar de tecnicamente não serem diferentes das anteriores, as mensagens apresentadas a seguir são de uso específico da tarefa executada por este sistema. A tabela 4.12 traz as mensagens utilizadas durante a subtarefa de aproximação inicial entre os robôs.

Tabela 4.12: Mensagens para a subtarefa de aproximação dos robôs.

Message	*Sender*	*Receiver*	*Server*
THEUS_GO_NEAR_REQUEST_MSG	C go	T gonear	C global
THEUS_GO_NEAR_RESPONSE_MSG	T gonear	C go	C global
NOMAD_APROACH_PREPARE_REQUEST_MSG	T gonear	N bringnear	C global
NOMAD_APROACH_PREPARE_RESPONSE_MSG	N bringnear	T gonear	C global
NOMAD_APROACH_NOMAD_DETECTED_THEUS_MSG	N bringnear	T gonear	C global
NOMAD_APROACH_TRAJECTORY_REQUEST_MSG	T gonear	N bringnear	C global
NOMAD_APROACH_TRAJECTORY_RESPONSE_MSG	N bringnear	T gonear	C global

O módulo go é marcado como sendo comum aos dois robôs porque não importa onde ele é executado. Ele poderia inclusive ser executado em uma terceira máquina, desde que também conectada à rede.

O módulo gonear é o módulo que controla esta subtarefa. Ele recebe do módulo go o GO_NEAR_REQUEST e só envia a resposta GO_NEAR_RESPONSE quando houver terminado a aproximação. Para o módulo bringnear, ele envia a mensagem PREPARE_REQUEST, que é respondida com um PREPARE_RESPONSE quando o Nomad já estiver corretamente posicionado.

A mensagem NOMAD_DETECTED_THEUS é enviada pelo módulo bringnear quando o Nomad passa a detectar Theus com o auxílio de seus infra-vermelhos. A partir deste ponto, Theus passa a ser ajudado pelo Nomad. O módulo gonear envia mensagens de TRAJECTORY_REQUEST e recebe a ajuda necessária através de mensagens de TRAJECTORY_RESPONSE.

Todas as mensagens passam pelo servidor central global, pois são mensagens trocadas entre módulos executando em máquinas diferentes.

O último conjunto de mensagens é referente à subtarefa de seguimento realizada pelos módulos fields no Nomad e follow em Theus. As mensagens são mostrados na tabela 4.13.

Tabela 4.13: Mensagens para a subtarefa de seguimento.

Message	*Sender*	*Receiver*	*Server*
NOMAD_FIELDS_START_REQUEST_MSG	C go	N fields	C global
NOMAD_FIELDS_START_RESPONSE_MSG	N fields	C go	C global
NOMAD_FIELDS_STOP_REQUEST_MSG	T follow	N fields	C global
NOMAD_FIELDS_STOP_RESPONSE_MSG	N fields	T follow	C global
THEUS_FOLLOW_START_REQUEST_MSG	N fields	T follow	C global
THEUS_FOLLOW_START_RESPONSE_MSG	T follow	N fields	C global
THEUS_FOLLOW_STOP_REQUEST_MSG	N fields	T follow	C global
THEUS_FOLLOW_STOP_RESPONSE_MSG	T follow	N fields	C global

A mensagem que inicia a subtarefa como um todo é a FIELDS_START_REQUEST enviada pelo módulo go que só receberá a resposta FIELDS_START_RESPONSE quando a subtarefa terminar. Assim que o módulo fields do Nomad recebe a mensagem para iniciar, ele envia uma mensagem de FOLLOW_START_REQUEST para o módulo follow de Theus, que responde imediatamente com a mensagem FOLLOW_START_RESPONSE. A partir deste ponto, o Nomad começa a mover-se em direção ao objetivo enquanto Theus o segue.

As mensagens de FIELDS_STOP_REQUEST e FOLLOW_STOP_REQUEST e suas respectivas respostas são utilizadas para interromper a execução do seguimento quando o Nomad chega à posição desejada ou quando algum erro ocorre.

Assim como na subtarefa de aproximação, as mensagens, trocadas entre módulos de máquinas diferentes, são enviadas através do servidor central global.

É interessante mencionar aqui uma situação que foi percebida apenas durante a implementação do sistema. Quando um módulo está tratando uma mensagem, ele não pode receber outra mensagem, mesmo que chame explicitamente a função IPC_listen(). Quando uma mensagem é recebida pelo módulo, é executada uma função para tratar desta mensagem e enquanto esta função não retornar, nenhuma outra mensagem poderá ser tratada.

Este problema foi detectado na implementação dos módulos fields. Na primeira implementação deste módulo, a função main() ficava esperando a chegada de mensagens, até receber a mensagem NOMAD_FIELDS_START_REQUEST_MSG. A função que tratava esta mensagem chamava outras funções que deveriam executar o algoritmo de navegação.

O problema é que o módulo fields precisa receber os valores dos sensores através de mensagens, mas como a função que tratava a primeira mensagem nunca retornava, não havia como receber os dados dos sensores.

A solução encontrada para este problema foi a criação de uma função para tratar a mensagem de START que apenas atualiza um valor em uma variável de estado e retorna imediatamente. O algoritmo entra em ação se esta variável tiver o valor verdadeiro.

4.3.5 Resultados Experimentais

Os resultados experimentais deste trabalho foram sendo atingidos conforme cada etapa da implementação do sistema era concluída. Esta é uma característica da linguagem TDL: pode-se construir um sistema complexo por etapas e realizar testes reais assim que cada etapa é concluída. Aqui serão apresentados os resultados na ordem em que foram obtidos.

Os primeiros módulos a serem implementados foram os módulos sensors de Theus e do Nomad. Os testes foram realizados com o auxílio de módulos que consultam os sensores de ambos os robôs e mostram os valores na tela.

Da mesma forma, quando o módulo mover de cada robô ficou pronto, utilizaram-se módulos auxiliares para o envio de mensagens para a movimentação do robô. Estes módulos para testes são bem simples: o usuário só precisa escolher o tipo de movimento e fornecer os parâmetros necessários de velocidade ou posição.

Com os módulos básicos prontos e testados, iniciou-se a construção do sistema com o módulo mais alto na hierarquia. O módulo go, que coordena a realização da tarefa, foi então implementado, mas ainda não realmente testado, apesar de poder ser executado normalmente.

Os módulos gonear em Theus e bringnear no Nomad foram implementados ao mesmo tempo, pois dependem um do outro para executar a tarefa de aproximação. Os testes agora não precisam de módulos auxiliares, basta iniciar o sistema como se ele estivesse completo. Mesmo sem os módulos follow e fields implementados, o sistema funcionava sem que o módulo go precisasse ser alterado. A tarefa era iniciada, os robôs se alinhavam, aproximavam-se e a tarefa era encerrada sem o seguimento.

O próximo passo foi a implementação do módulo fields. A tarefa agora não terminava após a aproximação. Quando os robôs já estavam próximos, Theus parava e o Nomad iniciava sua navegação até o ponto pré-definido. Theus continuava parado, pois o algoritmo de seguimento ainda não estava implementado.

O módulo detect de Theus foi implementado antes do módulo follow, pois o algoritmo de seguimento precisa de informações sobre o posicionamento relativo do Nomad. O módulo detect foi testado separadamente com a ajuda de um módulo auxiliar que mostra na tela a distância e o ângulo em que se encontra o Nomad, enquanto este se move nas proximidades de Theus com o auxílio de um *joystick*.

O módulo follow foi construído em seguida e testado separadamente. Theus foi posicionado próximo ao Nomad e, quando este é movimentado com o *joystick*, Theus o segue. Ao ser incluído no sistema completo, o teste não foi bem sucedido. O módulo follow estava se comportando como esperado, com Theus sempre seguindo o Nomad. Porém, o módulo fields não guiava o Nomad até o ponto alvo, pois Theus era considerado pelo Nomad como sendo um obstáculo e este se movia sempre na direção oposta a Theus, ignorando o objetivo da navegação.

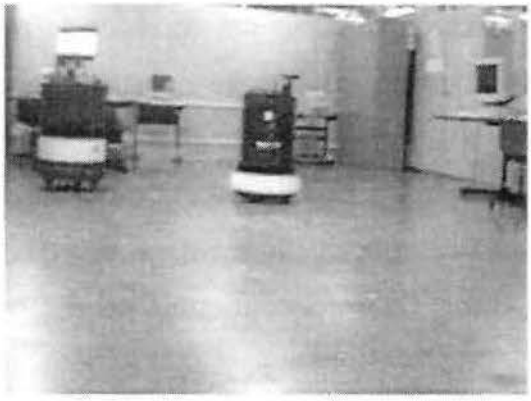
O problema foi solucionado com a implementação do módulo detect do Nomad e uma pequena alteração no algoritmo de navegação, já descrita anteriormente, com a qual o Nomad passa a ignorar os sensores que detectam Theus. Os testes com este sistema foram bem sucedidos.

Com o sistema completo, a tarefa é executada da seguinte forma:

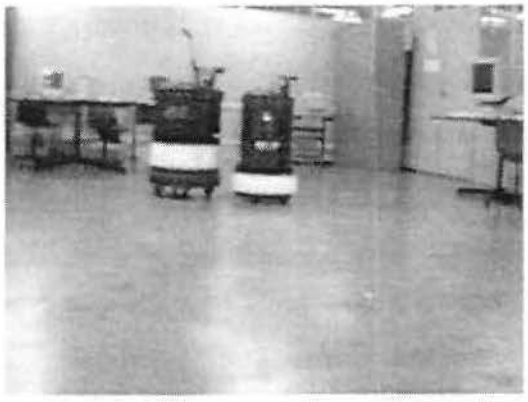
1 - Inicialização dos três servidores centrais. Os servidores são executados de acordo com as informações contidas no arquivo hosts.info.
2 - Inicialização dos módulos no Nomad e em Theus. Deve-se inicializar o sistema executando-se o *script* "start" no Nomad e em Theus. Os *scripts* "start" inicializam os módulos do respectivo robô, que ficam esperando a chegada de mensagens.
3 - módulo go é o último a ser executado. Ele pede que o usuário forneça o posicionamento inicial dos robôs e os pontos alvo que o Nomad deverá atingir.
4 - A partir daqui, o sistema irá executar a tarefa sem que o usuário precise intervir.

Muitos testes, com várias configurações de posicionamento inicial, pontos alvo e ambientes com e sem obstáculos foram realizados com sucesso. Como exemplo, foram gravados dois vídeos em formato "mpg" de execuções de tarefas de navegação e seguimento. Os vídeos estão disponíveis na Internet para download em "http://www.ia.cti.br/~lrv/acrc/".

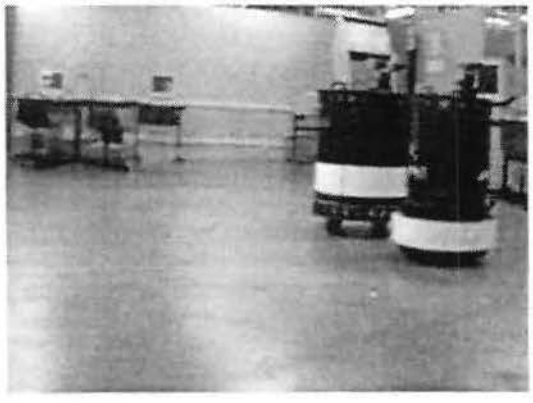
No primeiro exemplo, os dois robôs navegam por um ambiente sem obstáculos passando por uma seqüência de pontos pré-definidos. Na figura 4.33-a, os robôs estão distantes um do outro e não alinhados. São fornecidas informações de posicionamento relativo entre eles para que Theus possa fazer a aproximação inicial (Figura 4.33-b). Nas figuras 4.33-c, 4.33-d e 4.33-e, o Nomad guia Theus ao redor da sala, completando a volta na figura 4.33-f.



(a)



(b)



(c)

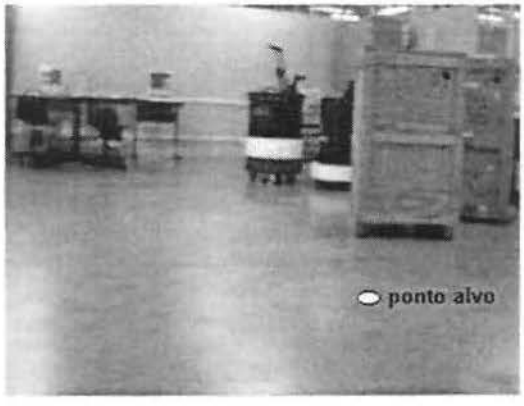
(d)

(e)

(f)

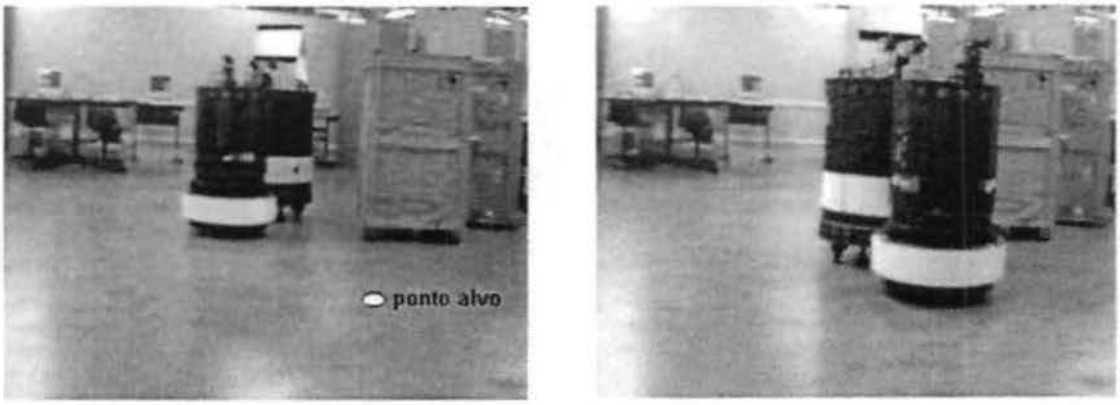
Figura 4.33: Volta ao redor da sala sem obstáculos.

No segundo exemplo, os robôs devem desviar de um obstáculo para alcançar o ponto alvo. Pela figura 4.34-a pode-se ver que não é possível percorrer uma linha reta entre a posição inicial do Nomad e o ponto alvo onde se quer chegar. Os robôs desviam do obstáculo (Figura 4.34-b) e em seguida retomam a trajetória em direção ao ponto alvo (Figura 4.34-c). Completando a tarefa, o Nomad pára exatamente sobre o ponto alvo (Figura 4.34-d).



(a)

(b)



(c) (d)

Figura 4.34: Contorno de um obstáculo para atingir o ponto alvo.

Capítulo 5

Conclusão

5.1 Contribuições

Conjuntos de robôs cooperativos são sistemas cada vez mais comuns na área de robótica. Sistemas com múltiplos robôs são utilizados devido às suas vantagens em relação a sistemas de um único robô. Tais vantagens, como a maior tolerância a falhas e a possibilidade de atuação simultânea e coordenada em pontos distantes entre si, não são características inerentes de sistemas multi-robóticos, mas sim, resultados de um trabalho longe de ser simples, realizado na implementação do sistema.

A implementação de um sistema robótico exige uma programação demasiadamente complexa quando se utiliza uma linguagem de programação genérica. As arquiteturas de controle de robôs são mecanismos que facilitam este tipo de trabalho.

O objetivo alcançado neste projeto foi o estabelecimento de uma arquitetura de controle para o desenvolvimento de sistemas multi-robóticos no país. A metodologia adotada consistiu no levantamento dos requisitos necessários da arquitetura, no estudo das arquiteturas encontradas na literatura, na adoção daquela que mais se adequasse aos requisitos como componente básico e na sua extensão para que atendesse a todos os requisitos estabelecidos. Desta forma, foi adotada a arquitetura TDL/TCM como a mais adequada aos nossos objetivos.

O único requisito que o TDL/TCM não atende é quanto a sua aplicabilidade em sistemas com mais de um robô, pois nele não é prevista a comunicação entre diferentes

processos e máquinas. A contribuição deste projeto consiste, então, na concepção e validação de uma arquitetura baseada no TDL/TCM que, com o auxílio do sistema de comunicação IPC, possibilita o desenvolvimento de sistemas multi-robóticos.

Como validação experimental foi implementado um sistema com dois robôs móveis para a realização de uma tarefa cooperativa. Nesta tarefa, um dos robôs atua como líder e deve guiar o segundo robô através do ambiente. O sistema é formado por módulos que se comunicam entre si. Alguns dos módulos são básicos e são encarregados da movimentação e aquisição de dados dos sensores dos robôs. Os módulos básicos não dependem da tarefa a ser realizada e podem ser utilizados na implementação de outros sistemas.

Em resumo, as principais contribuições deste trabalho são:

- O estabelecimento de uma arquitetura de controle para o desenvolvimento de sistemas de robôs cooperativos no país.
- A implementação de um sistema de dois robôs utilizando esta arquitetura. Os módulos deste sistema, principalmente os básicos, podem ser reaproveitados para outras tarefas.

5.2 Trabalhos Futuros

Nesta seção são apresentados possíveis trabalhos que podem dar continuidade a este projeto. As principais linhas que podem ser seguidas são a utilização da arquitetura de controle no desenvolvimento de outros projetos envolvendo múltiplos robôs e estudos sobre a questão da comunicação quando há um aumento no número de robôs envolvidos.

A arquitetura de controle será utilizada em outros projetos do LRV envolvendo múltiplos robôs. Para projetos que utilizem os mesmos robôs usados neste trabalho, podem-se aproveitar os módulos básicos já desenvolvidos e implementar o novo sistema sobre esta base.

Projetos envolvendo outros robôs também utilizarão a arquitetura de controle apresentada neste trabalho. Pretende-se utilizar esta arquitetura, por exemplo, em um projeto envolvendo um robô aéreo e um terrestre [23]. Neste projeto, um robô aéreo, com

uma visão global do ambiente, fornece informações para um robô terrestre de tal forma a guiá-lo até a localização onde o robô terrestre deverá atuar (vide Figura 5.1).

Figura 5.1: Um robô aéreo e um terrestre operando de forma cooperativa.

Fica em aberto a questão da utilização da arquitetura de controle proposta neste trabalho em sistemas com mais robôs. Quando se tem apenas dois robôs, a comunicação entre eles é feita através de um único servidor central. Com o aumento do número de robôs, a comunicação pode ser feita de diferentes formas.

Pode-se continuar a utilizar apenas um servidor central responsável por toda a troca de mensagens entre robôs, como na figura 5.2. Esta forma é a mais simples, mas se o número de robôs for muito grande, o servidor central pode se tornar um gargalo do sistema.

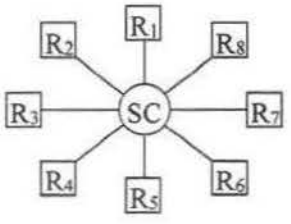


Figura 5.2: Servidor central único.

Uma outra forma de se organizar a comunicação seria através da utilização de vários servidores centrais. Estes podem ser organizados em diversas estruturas diferentes. Pode-se

definir, por exemplo, uma hierarquia de comunicação entre os robôs, como na figura 5.3, ou a formação de grupos de robôs, com a existência de um servidor central intermediando a comunicação entre os grupos e servidores centrais para a comunicação entre os robôs dentro de cada grupo, como na figura 5.4. A adoção de uma ou outra filosofia de comunicação depende, principalmente, de como as responsabilidades pelas tarefas a serem executadas serão divididas entre os robôs do conjunto.

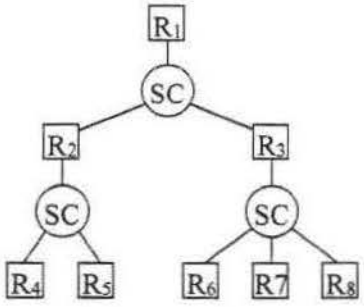


Figura 5.3: Sistema de comunicação hierárquico.

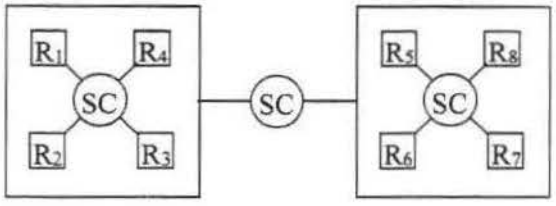


Figura 5.4: Robôs divididos em grupos.

Bibliografia

[1] G. Dudek, M. Jerkins, E. Milios e D. Wilkes. "A Taxonomy for Multi-Agent Robotics." *Autonomous Robots*, vol. 3, 1996, pp. 375-397.

[2] H. Raza e P. Ioannou. "Vehicle Following Control Design for Automated Highway System." *IEEE Control Systems Magazine*, vol. 16, no. 6, Dezembro 1996, pp. 43-60.

[3] R. Sukthankar, S. Baluja e J. Hancock "Evolving an Intelligent Vehicle for Tactical Reasoning in Traffic." *IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, EUA, Abril 1997, pp. 519-524.

[4] R. Alami, S. Fleury, M. Herrb, F. Ingrand e F. Robert. "Multi-Robot Cooperation in the MARTHA Project." *IEEE Robotics and Automation Magazine*, vol. 5, no. 1, Março 1998, pp. 36-47.

[5] A. Elfes, S.S. Bueno, M. Bergerman, J.J.G. Ramos e S.B.V. Gomes. "Project AURORA: Development of an Autonomous Unmanned Remote Monitoring Robotic Airship." *Journal of the Brazilian Computer Society*, vol. 4, no. 3, Abril 1998, pp. 70-78.

[6] R. Arkin. *AAAI Mobile Robot Competition*. Georgia Institute of Technologie, http://www.cc.gatech.edu/aimosaic/robot-lab/research/aaai94.html.

[7] S. Thrun e R. Simmons. *Mercator Project*. Carnegie Mellon University, http://www.cs.cmu.edu/~mercator/index.html.

[8] R. Simmons e S. Singh. *Distributed Robot Architectures – DIRA*. Johnson Space Center TRACLabs, National Institute of Standards and Technology e Carnegie Mellon University, http://www.frc.ri.cmu.edu/projects/dira/.

[9] A.A.D. Medeiros. "A Survey of Control Architectures for Autonomous Mobile Robots." *Journal of the Brazilian Computer Society*, vol. 4, no. 3, Abril 1998, pp. 35-43.

[10] J.S. Albus, H.G. McCain e R. Lumia. *NASA/NBS standard reference model for telerobot control system architecture (NARSEM)*. Technical Report 1235, National Institute of Standards and Technology, EUA, 1989.

[11] R.A. Brooks. "A robust layered control system for a mobile robot." *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, Março 1986, pp. 14-23.

[12] R. Alami, R. Chatila, S. Fleury, M. Ghallab e F.F. Ingrand. "An architecture for autonomy." *International Journal of Robotics Research*, Spring 1998.

[13] R. Simmons e D. Apfelbaum. "A Task Description Language for Robot Control." *IEEE/RSJ International Conference on Inteligent Robots and Systems*, Victoria, Canadá, Outubro 1997, pp. 1931-1937.

[14] R. Simmons, R. Goodwin, K.Z. Haigh, S. Koenig e J. O'Sullivan. "A Layered Architecture for Office Delivery Robots." *First International Conference on Autonomous Agents*, Marina del Rey, EUA, Fevereiro 1997.

[15] D. Apfelbaum. *The Task Description Language*. Technical Report, Carnegie Mellon University, Pittsburgh, PA, EUA, Setembro 1999.

[16] R. Simmons e D. Apfelbaum. *The Task Description Language – TDL.* Carnegie Mellon University, http://www.cs.cmu.edu/~tdl/.

[17] R. Simmons e D. James. *Inter-Process Communication - A Reference Manual.* Carnegie Mellon University, Pittsburgh, PA, EUA, Julho 1997.

[18] R. Simmons. *Inter-Process Communication – IPC.* Carnegie Mellon University, http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html

[19] R. Simmons. "Monitoring and Error Recovery for Autonomous Walking." *IEEE/RSJ International Conference on Inteligent Robots and Systems*, 1992.

[20] G. Welch e G. Bishop. *An Introduction to the Kalman Filter.* Technical Report 95-041, University of North Carolina at Chapel Hill, Department of Computer Science, 1999.

[21] R. Simmons, D. Apfelbaum, W.Burgard, D.Fox, M.Moors, S.Thrun e H.Younes. "Coordination for Multi-Robot Exploration and Mapping." *National Conference on Artificial Intelligence*, Austin, TX, EUA, Agosto 2000.

[22] J.C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers. Boston, EUA, 1991.

[23] A. Elfes, M. Bergerman, J.R.H. Carvalho, E.C. Paiva, J.J.G. Ramos, S.S. Bueno. "Air-Ground Robotic Ensembles for Cooperative Applications: Concepts and Preliminary Results." *International Conference on Field and Service Robotics.* Pittsburgh, PA, EUA, Agosto 1999, pp. 75-80.

Apêndice 1

Trabalho com Robôs Reais

Este anexo mostra as dificuldades encontradas em trabalhos que utilizam robôs reais. A validação experimental feita com robôs reais traz uma série de complicações que não são encontradas em simulação. A seção 1 apresenta das dificuldades que surgiram durante este trabalho e as seções 2 e 3 mostram como algumas delas foram contornadas durante a execução do projeto.

A1.1 Dificuldades no Trabalho com Robôs Reais

Um dos maiores problemas ao se lidar com robôs reais é o tratamento dos valores lidos pelos sensores. Em simulação os sensores são bem comportados e os valores lidos são precisos e confiáveis. No mundo real é diferente: existem ruídos, superfícies de diferentes materiais e formas, sensores não homogêneos, etc.

Os problemas de imprecisão quando se utiliza sensores reais são muito sérios. O valor de um sensor real pode mudar de uma leitura para outra sem que nada tenha se movido ou se modificado no mundo real. Além disso, o valor lido de um sensor pode ser bem diferente do valor lido em outro do mesmo tipo e nas mesmas condições de distância, luminosidade, umidade do ar, superfície refletora, etc. Isto ocorre porque os sensores são componentes físicos e, ao contrário do que ocorre em simulação, onde os sensores são todos perfeitamente idênticos, sensores reais estão sujeitos a imperfeições que os tornam diferentes um do outro.

O problema de imprecisão dos sensores acarreta um outro problema muito sério quando se trabalha com robôs reais: a dificuldade de se conhecer a localização exata do robô no ambiente. Em sistemas simulados isto é muito fácil, pois basta ler as coordenadas em que o robô se encontra e se obtém sua posição. Em sistemas reais, este tipo de informação não é precisa o suficiente.

Muitos robôs reais mantêm a informação de sua localização utilizando os integradores. Integradores são mecanismos que estão ligados à movimentação do robô e, através da somatória das distâncias percorridas, calculam a posição do robô a partir da sua posição inicial. O problema é que os integradores não são perfeitos e a cada interação um pequeno erro é acrescentado. Depois de percorrer uma certa distância, o acúmulo de erro pode se tornar tão grande que a informação deixa de ser confiável (Figura A1.1).

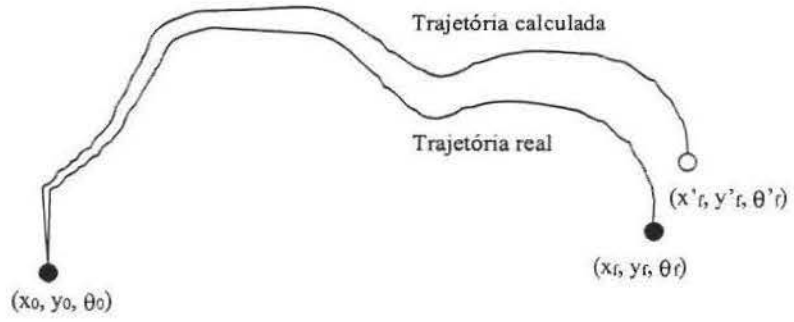


Figura A1.1: Acúmulo de erro nos integradores.

Apesar do acúmulo de erro, os integradores podem ser muito úteis quando se utiliza um único robô para tarefas onde sua posição absoluta não é importante. Neste caso, pode-se sempre ignorar toda a trajetória anterior e utilizar apenas a diferença de posição em relação à última medição. Se as medições forem feitas em intervalos relativamente curtos, os erros são bem pequenos e não prejudicam a tarefa.

No caso do sistema multi-robótico apresentado neste trabalho, o posicionamento absoluto dos robôs em relação ao ambiente pode ter um certo acúmulo de erro, mas o posicionamento relativo entre os robôs, mesmo que não seja extremamente preciso, não pode ter distorções muito grandes com o passar do tempo. Neste caso, os integradores não podem ser utilizados pois o erro acumulado em um dos robôs pode ser muito diferente do

erro acumulado no outro. Por este motivo decidiu-se utilizar os sensores de infra-vermelho e sonares, menos precisos, mas que não acumulam erro com o tempo.

Para tarefas onde o posicionamento absoluto do robô é importante, algum outro mecanismo deve ser utilizado, como por exemplo, *landmarks* ou GPS.

Robôs reais estão também sujeitos a problemas cotidianos que são simplesmente inimagináveis em simulação, tais como incompatibilidade de *software* e de *drivers*, e problemas com *hardware*.

Problemas como um disco rígido que deixa de funcionar ou uma bateria cuja vida útil termina antes do previsto podem atrasar o andamento ou até provocar alterações em um projeto. Durante este projeto, como não poderia ser diferente, muitas dificuldades deste tipo surgiram.

Um disco rígido dentro do robô Theus apresentou defeitos e teve que ser substituído. O Linux teve que ser reinstalado juntamente com todos os *drivers* para os componentes do robô. Esta reinstalação não foi fácil, pois a versão do Linux utilizada era diferente da anterior e vários problemas de incompatibilidade tiveram que ser resolvidos.

Nesta mesma época, a placa de rede via rádio do robô também apresentou problemas. A solução foi simples: a placa foi substituída. Porém, muito tempo foi gasto até que fosse identificada a origem do do problema, pois tudo estava sendo reinstalado, inclusive os *drivers* para a placa de rede.

O mais recente grande acidente ocorrido com o robô foi com as placas responsáveis pelo recarregamento das baterias. Um pico de tensão na rede elétrica provocou um superaquecimento das placas que acabaram sendo destruídas por fogo. Novas placas foram encomendadas e substituíram as danificadas, mas os quase dois meses entre o incidente e a troca das placas prejudicou totalmente o cronograma do projeto, pois todo o trabalho realizado em simulação durante este período teve que ser praticamente refeito para o robô real, dada a diferença de condições entre o programa simulador e a realidade.

Vários outros problemas menos significantes, e que não têm a necessidade de serem detalhados neste texto, vieram a ocorrer durante a execução deste projeto. Tudo o que foi descrito nesta seção tem a intenção de mostrar que o trabalho com robôs de verdade é muito mais que a simples aplicação, em sistemas reais, de resultados obtidos em sistemas simulados.

A1.2 Tratamento dos Sensores Infra-vermelhos

Nesta seção será descrito o problema da leitura dos sensores infra-vermelhos, o mais interessante enfrentado durante este projeto. Os sensores infra-vermelhos captam a quantidade de luz infra-vermelha refletida pelo objeto. Quanto maior o retorno, mais perto do robô está o objeto. Porém, a reflexão é sensível à cor do objeto, portanto, objetos à mesma distância, mas com cores diferentes, refletem quantidades diferentes de luz.

Quanto mais claro o objeto, maior a quantidade de luz refletida por ele, e portanto, maior é a eficiência do sensor infra-vermelho que o detecta. Para que um melhor aproveitamento dos sensores infra-vermelhos fosse obtida, os robôs foram equipados com faixas brancas ao redor de suas torres na altura dos sensores do outro robô, que podem ser vistas na figura A1.2. Com isso, os robôs podem detectar-se um ao outro com maior precisão e a maiores distâncias.

Figura A1.2: Faixa branca na altura dos infra-vermelhos.

Outro problema encontrado foi a heterogeneidade dos sensores infra-vermelhos de Theus. Quando seu módulo detect teve que ser projetado, foi feito um estudo de como os sensores infra-vermelhos detectam o Nomad, para que a partir destes dados experimentais, o algoritmo pudesse ser desenvolvido.

Para este experimento foi escrito um programa que coleta dados dos infra-vermelhos e gira o robô. O robô gira um certo ângulo e em seguida, com o robô parado, é feita a coleta de dados lendo-se várias vezes todos os sensores e calculando-se a média das várias leituras para cada sensor. Este procedimento é repetido várias vezes até que o robô complete uma volta inteira. As médias dos valores lidos são armazenados em uma matriz bidimensional onde cada linha representa um sensor e cada coluna representa um ângulo.

Um experimento com o Nomad posicionado a 20 cm de Theus foi realizado e os dados podem ser vistos na figura A1.3. Neste gráfico estão mostrados todos os dados coletados em uma volta do robô. Cada linha representa um sensor e os picos são as regiões onde os sensores detectaram o Nomad.

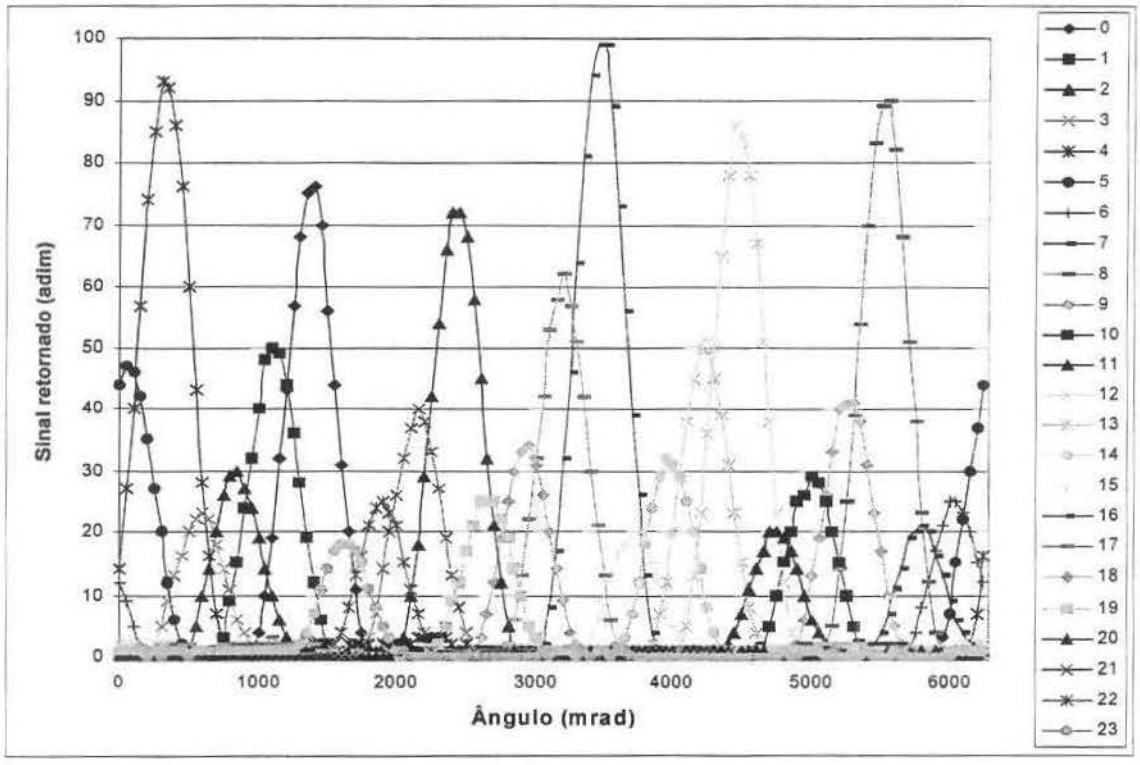


Figura A1.3: Sensores infra-vermelhos de Theus detectando o Nomad

Foi notado que os dados seguem um padrão interessante. A cada conjunto de quatro sensores adjacentes o sinal vai aumentando gradativamente e depois volta a ser baixo no sensor seguinte. Depois de perceber este padrão, verificou-se que os sensores no robô estão fisicamente agrupados em grupos de quatro e que os de sinal mais baixo são sempre os da

extremidade mais distante. A figura A1.4 mostra como os sensores são agrupados e como varia a intensidade da luz refletida em cada sensor.

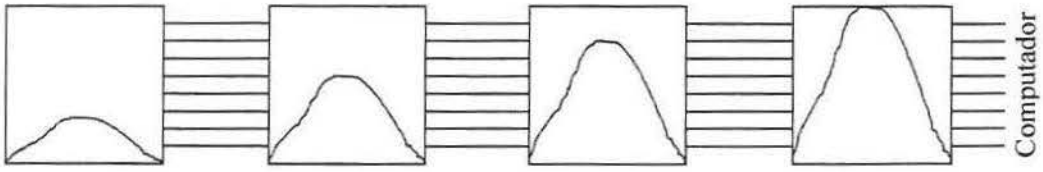


Figura A1.4: Disposição dos sensores e sua relação com a força do sinal.

Estas distorções tiveram que ser corrigidas. As curvas têm o mesmo formato, apenas a intensidade do sinal que muda. Portanto, para se obter curvas homogêneas, basta encontrar um multiplicador para cada um dos sensores. O multiplicador deve ser um número inversamente proporcional ao valor máximo da curva. Na figura A1.5 temos os valores normalizados, isto é, divididos pelo valor máximo de cada curva.

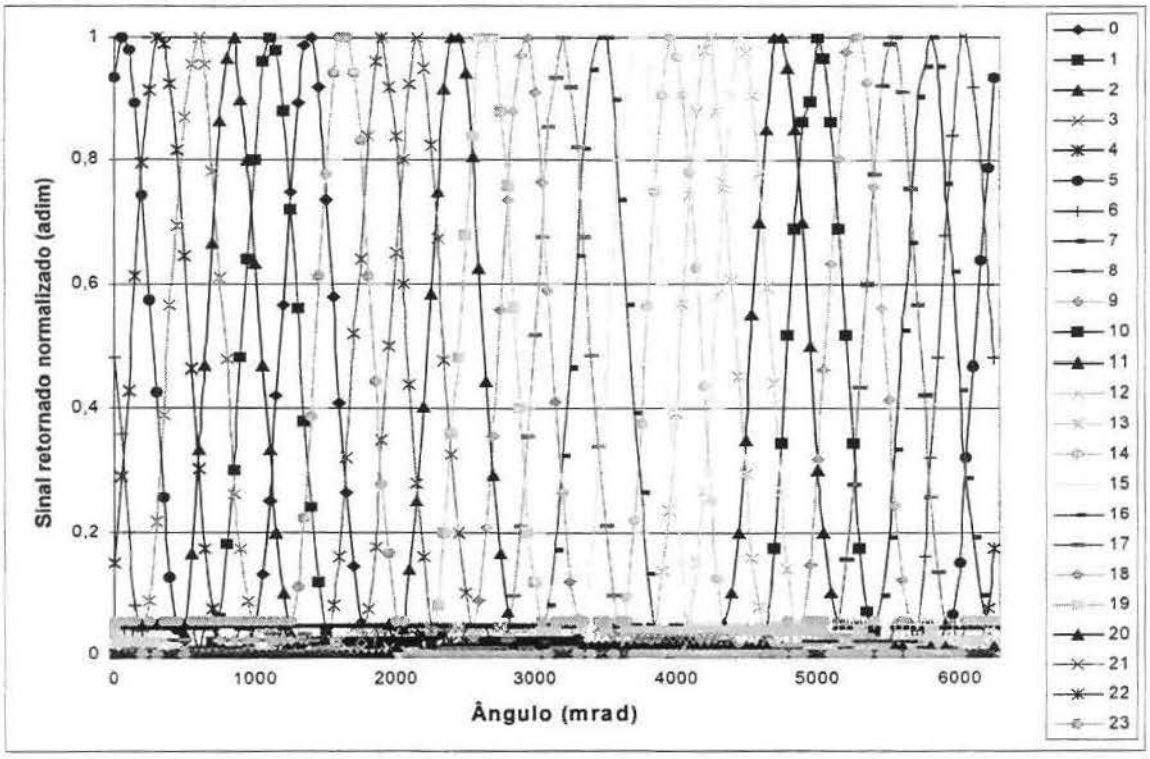


Figura A1.5: Valores normalizados dos sensores.

Para valores variando entre zero e um, seria necessário utilizar tipos de dados *float* ou *double*. Para que as variáveis pudessem continuar a ser do tipo inteiro, foram aplicados multiplicadores inteiros mil vezes maiores que os anteriores. Para confirmar a validade dos multiplicadores, foram coletadas independentemente mais duas seqüências de dados. Ao se aplicar os multiplicadores nestes dois conjuntos, obteve-se um bom resultado, que pode ser visto na figura A1.6.

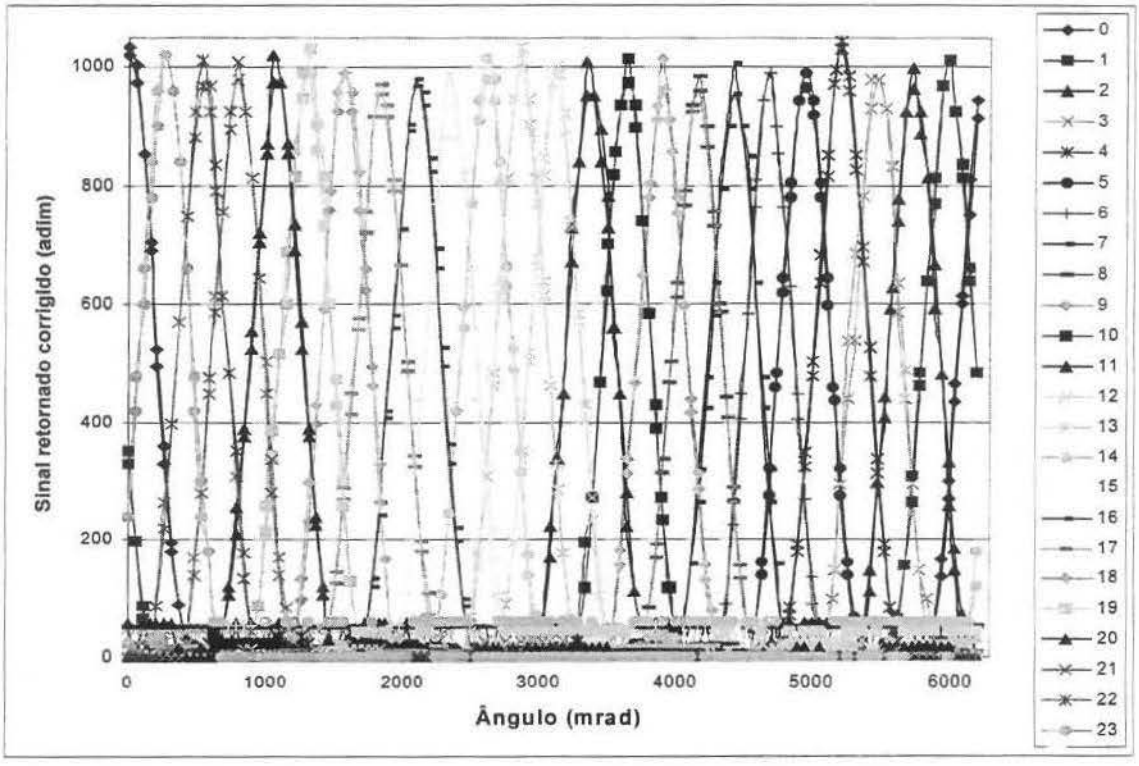


Figura A1.6: Aplicação de multiplicadores em outros conjuntos de dados.

Esta solução não eliminou problemas como ruído ou a influência de cores dos objetos na determinação da distância, mas permitiu tratar os sensores infra-vermelhos de forma equivalente. Um algoritmo que utiliza estes sensores não precisa mais levar em conta de qual dos sensores a informação é proveniente, o que traz uma facilidade muito grande para a idealização e implementação dos algoritmos.

O ruído dos sensores infra-vermelhos teve que ser levado em conta no algoritmo de seguimento. Quando o Nomad ficava parado, Theus também deveria permanecer parado,

mas ao contrário disso, ele ficava fazendo pequenos movimentos para frente, para trás, para um lado e para o outro.

Isto ocorria porque existia apenas um ponto (ângulo, distância) onde, segundo o algoritmo, Theus deveria permanecer parado. Porém, devido a ruídos que provocavam pequenas variações na leitura dos sensores, o algoritmo entendia que Theus deveria continuamente ajustar sua posição. O problema só era percebido quando o robô líder estava parado, pois quando o Nomad estava se movendo, o comportamento de Theus era normal.

Para evitar que Theus ficasse "dançando", foram definidas regiões onde ele pudesse permanecer parado mesmo não estando no ponto ótimo. Para o ângulo, definiu-se uma tolerância de 150 miliradianos para mais ou para menos em torno do ponto ótimo. Para a distância, o robô poderia ficar a algo em torno de 10cm do ponto ótimo. Estes valores para a tolerância mal podem ser percebidos visualmente e não prejudicam o algoritmo de seguimento.

A1.3 Detecção Utilizando Sensores de Distância

Os módulos detect, que são responsáveis por manter conhecida a localização relativa entre os robôs, utilizam equações para o cálculo destas informações a partir dos valores lidos dos sensores. A obtenção destas equações foi resultado do trabalho experimental descrito nesta seção.

Esta descrição está na seção de trabalho com robôs reais, pois como os sensores são reais, é muito difícil a obtenção de um modelo matemático preciso, ao contrário do que ocorre em simulação, onde o próprio sensor já é um modelo matemático. As equações foram obtidas totalmente a partir de dados experimentais. Estes dados são a leitura dos sensores com o outro robô posicionado a diferentes distâncias e diferentes ângulos.

A detecção de Theus feita pelo Nomad é feita por dois tipos de sensores, os sonares e os infra-vermelhos. Analisando-se os dados, percebeu-se que os sonares são mais fiéis à realidade que os infra-vermelhos, portanto são utilizados na maior parte das vezes. Porém, notou-se que os sonares nunca retornam um valor menor que 13cm. Se a distância for maior ou igual a 15cm, os sonares são eficientes, mas quando a distância de Theus é menor que

esta, os sonares perdem sua precisão e os dados deixam de ser confiáveis. Portanto, para distâncias menores que 15cm utilizam-se os infra-vermelhos.

Para distâncias maiores ou iguais a 15cm, utilizam-se os sonares. Tanto para a distância quanto para o ângulo, são utilizados dois sensores para o cálculo. Um deles é o central, cujo valor (m_1) é o menor dentre os que detectam Theus e o outro é seu adjacente de menor valor (m_2).

Notou-se que havia uma relação entre o quociente m_1/m_2 e o ângulo em que se encontrava Theus. Como os dados contêm muito ruído, foi determinado que as equações deveriam ser bem simples, pois a exatidão de equações mais complexas seria destruída pela imperfeição dos sensores. Assumiu-se, então, esta relação como sendo linear:

$$A = a\left(\frac{m_1}{m_2}\right) + b \qquad \text{(A1.1)}$$

onde a e b são constantes e A é o ângulo em que se encontra Theus em relação ao sensor central.

As constantes a e b receberam os valores que melhor ajustaram a curva aos dados experimentais. Os valores obtidos foram $a = 654{,}5$ e $b = 458{,}1$.

O cálculo da distância é baseado na leitura do sonar central. Porém, nem sempre o sonar central está exatamente na direção de Theus, por isso considera-se também o sonar adjacente de menor valor. Utilizou-se, então, uma função linear em relação às duas variáveis m_1 e m_2.

$$D_s = am_1 + bm_2 \qquad \text{(A1.2)}$$

onde a e b são constantes e D_s é a distância de Theus em relação aos sonares.

Os valores para as constantes que melhor se adaptam aos dados experimentais foram $a = 8{,}75$, $b = 1{,}25$. O valor de D_s é a distância de Theus em relação ao anel de sonares do Nomad. Porém, o tamanho efetivo do Nomad é determinado pelo anel de *bumpers*, cujo raio é 60mm maior que do anel dos sonares (vide Figura A1.7). Portanto, é necessário que se subtraia esta diferença do valor calculado.

$$D = D_s - 60 \tag{A1.3}$$

onde D é a distância real entre os dois robôs.

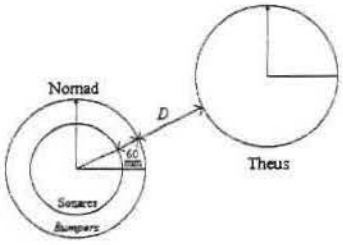


Figura A1.7: Distância efetiva é em relação ao anel de *bumpers*.

Os dados adquiridos através dos infra-vermelhos têm uma imprecisão muito grande pois as distâncias são discretizadas em 40mm. Portanto, as equações para o cálculo da distância e ângulo utilizando os sensores infra-vermelhos são mais simplificadas que as vistas acima. Nas equações, são utilizados dois sensores, um central, cujo valor é m_1 e seu adjacente de menor valor, m_2. O sensor adjacente é utilizado apenas para determinar qual conjunto de constantes será utilizado.

$$\begin{aligned} D &= am_1 - b \\ A &= c \end{aligned} \tag{A1.4}$$

onde D é a distância entre os dois robôs, já considerando o anel de bumpers, A é o ângulo de Theus em relação ao sensor central e a, b e c são constantes nas equações, mas podem assumir valores que dependem da relação entre m_1 e m_2.

Cada constante pode assumir dois valores diferentes, dependendo do quociente $m = m_1 / m_2$. Quando m>1,5 as constantes são: a = 40, b = -20 e c = 0,0°. Caso contrário, a = 40, b = -40 e c = 7,5°. Quando Theus está na direção do sensor central, utiliza-se o primeiro conjunto de constantes e quando Theus está na direção entre os dois sensores, utiliza-se o segundo conjunto.

Com estas duas equações, a discretização é bem severa, mas é muito difícil conseguir uma melhor devido à imprecisão dos infra-vermelhos. Apesar disto, este método é melhor que a utilização de apenas um sensor. Utilizando-se dois sensores infra-vermelhos, foi possível diminuir a discretização de 40mm para 20mm para a distância e de 15,0° para 7,5° para o ângulo, metade da distância em graus entre sensores.

O cálculo do posicionamento do Nomad feito por Theus é bem diferente. Primeiramente ele utiliza apenas os sensores infra-vermelhos, pois os sonares permanecem desligados durante a execução da tarefa para que não interfiram com os sonares do Nomad, que é utilizado para o algoritmo de navegação.

Em segundo lugar, a distância não é medida em milímetros e sim através de um valor proporcional à intensidade de sinal infra-vermelho captado no sensor que, para que não seja confundido com distância proprimente dita, será denotado aqui por I. Os valores de intensidade são previamente ajustados com o auxílio dos multiplicadores encontrados no estudo dos sensores infra-vermelhos descrito anteriormente.

Outra diferença é que os cálculos não são independentes entre si. É necessário calcular a distância antes do ângulo, pois o cálculo do ângulo leva em consideração a distância do Nomad.

Como Theus possui mais sensores no anel, eles se tornam mais próximos entre si, possibilitando a utilização de três sensores infra-vermelhos para os cálculos. Os valores destes três sensores serão denotados por m_c, m_a e m_p, referindo-se respectivamente ao sensor central, adjacente anterior e adjacente posterior.

Para o cálculo da distância, encontrou-se uma equação que utiliza dois sensores adjacentes, sendo um deles o central.

$$I_x = am_c + bm_x \qquad \text{(A1.5)}$$

onde I_x é a distância do Nomad e m_x pode ser m_a ou m_p.

Ajustando-se a equação aos dados experimentais, foram obtidas as seguintes constantes: a = 0,93 e b = 0,28. Os dados utilizados neste ajuste de curva foram obtidos realizando-se a leitura de dois sensores adjacentes, variando-se a distância e o ângulo do Nomad em relação ao Theus.

Como são calculados dois valores para a distância, I_a e I_p, estes valores devem ser combinados de alguma forma. Quanto maior o valor de m_i, onde m_i pode ser m_c, m_a ou m_p, maior é a precisão da leitura, portanto, a combinação deve favorecer ao máximo os sensores de valores maiores. Uma boa maneira de se combinar os valores favorecendo os sensores mais precisos é através de uma média ponderada em relação ao quadrado dos valores dos sensores. Como o sensor central é utilizado nas duas equações, a ponderação fica sendo apenas em relação aos dois sensores adjacentes.

$$I = \frac{m_a^2 I_a + m_p^2 I_p}{m_a^2 + m_p^2} \tag{A1.6}$$

O cálculo do ângulo é o que possui a equação mais complexa, mas devido às características dos dados experimentais, uma equação muito simplificada acabaria deteriorando demais as informações. O ângulo é calculado utilizando a distância, que já foi calculada e a informação dos mesmos três sensores utilizados anteriormente.

Uma curva que tem um formato parecido com a curva dos dados experimentais é dada pela seguinte função:

$$A_{ri} = \left[a\left(\frac{I}{m_i} - 1 \right) \right]^b \tag{A1.7}$$

onde a e b são constantes, A_{ri} é o ângulo do Nomad em relação ao sensor i, I é a distância do Nomad calculada através dos infra-vermelhos e m_i assume os valores de m_c, m_a e m_p.

Ao se ajustar a curva desta função aos dados reais, os valores obtidos para as constantes foram: $a = 10^8/3$ e $b = 0{,}303$. Os dados utilizados aqui são equivalentes a várias curvas sobrepostas como as da figura A1.7, onde o robô faz leituras dos infra-vermelhos variando-se o ângulo em que o Nomad se encontra.

A forma como os três ângulos calculados são combinados já está explicado na seção 4.3.3 na parte onde é descrito o módulo detect de Theus.

Apêndice 2

Instalação da Arquitetura

A2.1 Instalação do TDL/TCM

A arquitetura TDL/TCM tem a característica de poder ser utilizada em diferentes plataformas e sistemas operacionais. Isso é possível porque o *software* é fornecido em forma de código fonte na linguagem C++. A instalação é feita com a compilação do código, que gera a biblioteca que contém as funcionalidades do TCM, o compilador TDL e os demais componentes necessários para a utilização da arquitetura.

Um arquivo compactado contém os arquivos fonte necessários para a compilação e instalação do TDL e do TCM. Descompactando este arquivo, são obtidos os diretórios "tdl" e "tcm", onde estão localizados respectivamente os arquivos.

O TDL/TCM depende de algumas ferramentas de apoio que devem ser previamente instaladas. Estas ferramentas podem ser encontradas para livre *download* na rede:

- Java 1.1.X ou mais recente.
- JavaCC 0.8pre2 ou mais recente.
- Netscape's IFC Java library, versão 1.1.1 ou mais recente.

As variáveis de sistema PATH e CLASSPATH devem ser alteradas para incluir as informações de onde estarão localizados os diretórios tdl e tcm, bem como as ferramentas de apoio. Feito isso, deve-se compilar primeiramente o TCM e em seguida o TDL. O

TDL/TCM estará então pronto para ser utilizado na escrita de programas na linguagem TDL.

A compilação de um programa TDL envolve duas etapas, geração de um código C++ puro a partir do código TDL e a geração de um executável a partir do código C++. Um arquivo Makefile pode auxiliar a compilação dos programas. Um exemplo de Makefile que veio junto com os arquivos fonte foi modificado para incluir as bibliotecas dos dois robôs na compilação dos arquivos.

A2.2 Instalação do IPC

A instalação do IPC é feita de forma análoga ao TDL/TCM. A descompactação do arquivo original irá gerar o diretório ipc com os arquivos fonte. Deve-se então compilar o código e atualizar a variável de sistema PATH com a localização do servidor central.

Foi escrito e disponibilizado no LRV um documento com os detalhes para a instalação do TDL/TCM e também do IPC. Este documento traz também um pequeno exemplo da utilização do IPC e o Makefile para a programação dos robôs XR4000 e Nomad 200 utilizando TDL.