GUSTAVO ANDRÉ NUNES FERREIRA

**DESENVOLVIMENTO DE UMA ARQUITETURA DE
CONTROLE BASEADA EM OBJETOS PARA UM ROBÔ
MÓVEL AQUÁTICO**

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

São Paulo
2003

GUSTAVO ANDRÉ NUNES FERREIRA

**DESENVOLVIMENTO DE UMA ARQUITETURA DE
CONTROLE BASEADA EM OBJETOS PARA UM ROBÔ
MÓVEL AQUÁTICO**

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

Área de Concentração:
Engenharia Mecatrônica e
Sistemas Mecânicos

Orientador:
Prof. Dr.
Ettore Apolônio de Barros

São Paulo
2003

FICHA CATALOGRÁFICA

Ferreira, Gustavo André Nunes

Desenvolvimento de uma arquitetura de controle baseada em objetos para um robô móvel aquático. São Paulo, 2003.
137 p.

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

1. Robôs 2. *Softwares* Orientado a Objeto 3. Sistemas de Tempo Real I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e Sistemas Mecânicos II. t.

A meus pais, que muito se esforçaram
por minha educação e a quem devo
muito das virtudes que possuo.

Agradecimentos

Agradeço inicialmente o financiamento da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), tanto através do projeto individual de mestrado “Arquiteturas de Controle Aplicadas a Robôs Móveis Oceânicos” (proc. FAPESP 2000/01121-6), como através do projeto temático "Comportamento em Ondas de Embarcações de Alto Desempenho" (proc. FAPESP 1997/13090-3), financiamento sem o qual este projeto não seria possível.

Agradeço também a orientação do Prof. Dr. Ettore Apolônio de Barros, juntamente com os demais professores que fazem parte do projeto temático já citado.

Agradeço a ajuda inestimável do amigo Flávio José Aguiar Soares, pela sua supervisão na construção do modelo ITTC em fibra de vidro, juntamente com os técnicos do Departamento de Engenharia Naval do Instituto de Pesquisas Tecnológicas (IPT), pela sua cooperação nos diversos ensaios realizados no decorrer de todo o projeto e pelo estímulo e amizade como colega de laboratório.

Agradeço aos estagiários Daniel Barbuto Rossato e Hélder Flores Silva pelo trabalho realizado nas implementações eletrônicas deste projeto e ao estagiário Cassius Resende Duarte pelos estudos iniciais com a unidade inercial e o equipamento DGPS.

Agradeço a ajuda dos mestrandos Fábio Villas Boas e Fernanda de Sá Teixeira nos ensaios de finalização deste projeto.

E, finalmente, agradeço aos meus familiares e em especial a meus pais pelo amor, carinho e esforço que propiciou as condições necessárias para que eu realizasse este trabalho.

Resumo

Este trabalho trata do estudo de concepções de arquitetura do controle aplicadas aos robôs móveis autônomos e da proposição de um delas à instrumentação e controle em tempo real de um modelo de embarcação naval de alto desempenho. Tal veículo remotamente operado foi desenvolvido como parte das atividades do projeto temático "Comportamento em Ondas de Embarcações de Alto Desempenho" (proc.Fapesp 1997/13090-3). Realizou-se uma investigação dos diversos paradigmas de inteligência artificial que orientaram a evolução dos robôs móveis autônomos até o presente momento e, em particular, as concepções baseadas em modelos sócio-antropológicos e computacionais (teoria de agentes e orientação a objetos) através de sua aplicação à implementação de um sistema de aquisição e controle orientado a objetos, modelado através da UML (*Unified Modeling Language*), para o veículo mencionado. Testes de validação da arquitetura do controle foram realizados, sendo obtidos resultados experimentais que permitiram análises a respeito da dinâmica, manobrabilidade e navegação do veículo, as quais sugerem vários aperfeiçoamentos para o sistema de *hardware* e *software* em trabalhos futuros.

Abstract

This work deals with the study of control architecture approaches applied to autonomous mobile robots, and proposes one of them for the control system of a self-propelled high speed ship model. Such unmanned vehicle was developed for the research project “Comportamento em Ondas de Embarcações de Alto Desempenho” (proc. FAPESP 1997/13090-3). A number of artificial intelligence paradigms, related to the autonomous robot evolution up to now, were investigated. Models based on the socio-anthropological paradigm and the corresponding computer science approaches, i.e. agent theory and object-oriented modeling, were emphasized. Object-oriented control software based on the UML (Unified Modeling Language) was designed for the real-time embedded system of the ship model. Validation tests of the control architecture were carried out. Experimental results, related to vehicle dynamics, maneuverability and navigation were acquired by the embedded system and analyzed in this work. These results suggest a number of improvements for future works on the software and hardware systems.

Sumário

Lista de Figuras	iv
Lista de Tabelas	viii
1 Introdução	1
1.1 Motivação	1
1.2 Aplicação Específica	2
1.3 Objetivos da Dissertação	3
1.4 Estrutura da Dissertação	3
2 Visão Geral sobre Robôs Móveis Autônomos	5
2.1 Arquiteturas de Controle para Robôs Móveis Autônomos	8
2.1.1 Arquiteturas Hierárquicas	11
2.1.2 Arquiteturas Heterárquicas	13
2.1.3 Arquiteturas <i>Subsumption</i>	14
2.1.4 Arquiteturas Híbridas	15
2.2 Agentes e Desenvolvimento Baseado em Objetos	16
2.2.1 Definição de Agente	17
2.2.2 Conceito de Objeto	19
2.2.3 Arquitetura de Controle Proposta	20
3 Implicações de Tempo Real na Arquitetura Proposta	23
3.1 Sistemas de Tempo Real	24
3.2 Características Gerais de Sistemas de Tempo Real	25
3.2.1 Resposta Temporal	25

3.2.2	Simultaneidade	26
3.2.3	Previsibilidade	27
3.2.4	Dependabilidade	27
3.3	Sistemas Operacionais de Tempo Real	28
3.3.1	Núcleos de Tempo Real	29
3.3.2	Multi-Processos	30
3.3.3	Comunicação Interprocessos	31
3.3.4	Algoritmos de Escalonamento	32
3.4	Escolha da Plataforma de <i>Hardware</i> e <i>Software</i>	33
3.4.1	Dispositivos de Computação Embarcada	34
3.4.2	Sistema Operacional de <i>Software</i>	35
4	Proposta de Arquitetura de <i>Software</i>	40
4.1	Métodos Orientados a Objetos	42
4.2	<i>UML</i> , Linguagem Unificada de Modelagem	43
4.3	<i>UML</i> de Tempo Real	46
4.4	<i>UML</i> e Agentes	48
4.5	Modelagem da Arquitetura de Controle Proposta	49
4.5.1	Diagramas de Classes	50
4.5.2	Diagramas de Gráfico de Estados e de Atividades	53
4.5.3	Diagramas de Interação	55
5	Implementações e Resultados	66
5.1	Estrutura de <i>Hardware</i> e <i>Software</i> Implementada	68
5.1.1	Casco ITTC em escala reduzida	69
5.1.2	Microcomputador de Bordo PC/104	69
5.1.3	Servos-Sistemas e Atuadores	70
5.1.4	Sensores	70
5.1.5	Equipamentos de Comunicação	74
5.1.6	Baterias	76
5.2	Ensaio Realizados	76
5.2.1	Ensaio Preliminares	78

5.2.2	Ensaio com Piloto Automático	94
5.2.3	Ensaio de Navegação	118
6	Conclusões e Sugestões para Trabalhos Futuros	127
	Referências Bibliográficas	132

Lista de Figuras

2.1	<i>Pioneer</i> , UAV utilizado na Guerra do Golfo e nos ataques ao Afeganistão.	6
2.2	<i>Sojourner</i> , veículo de exploração enviado ao planeta Marte.	7
2.3	<i>Martin</i> , AUV desenvolvido pela <i>Maridan ApS</i> destinado a inspeção de oleodutos e cabos submarinos.	7
2.4	OTTER, <i>Ocean Technologies Testbed for Engineering Research</i>	12
2.5	Diagrama esquemático da arquitetura de controle do OTTER.	12
2.6	Pteroa-150, AUV desenvolvido no <i>Institute of Industrial Science da University of Tokyo</i>	13
2.7	Diagrama esquemático da arquitetura de controle do Pteroa-150.	13
2.8	Arquitetura de controle do AUV <i>Sea Squirt</i>	14
2.9	<i>Phoenix</i> , AUV de arquitetura híbrida.	16
2.10	Diagrama esquemático da arquitetura de controle do <i>Phoenix</i>	16
2.11	Agente interagindo com ambiente através de sensores e atuadores.	18
2.12	Diagrama simplificado da arquitetura de controle proposta.	20
3.1	Arquitetura de <i>hardware</i> escolhida.	35
3.2	Equipamentos de computação embarcada baseados no barramento PC/104.	36
3.3	O <i>microkernel</i> QNX coordenando os gerenciadores do sistema.	38
4.1	Origem do Método UML.	44
4.2	Diagrama de classes da arquitetura de <i>software</i> proposta.	50
4.3	Tipos de mensagens do sistema: <i>TMsg</i> e suas especializações.	51
4.4	Tipos de estruturas de dados presentes na arquitetura de <i>software</i> proposta.	52
4.5	Diagrama de Gráfico de Estados do agente <i>Gerenciador</i>	54
4.6	Diagrama de Gráfico de Estados do agente <i>Monitor</i>	56

4.7	Diagrama de Atividades dos processos de monitoramento do agente <i>Monitor</i> . . .	57
4.8	Diagrama de Gráfico de Estados do agente <i>Atuador</i>	58
4.9	Diagrama de Gráfico de Estados do agente <i>Sensor</i>	59
4.10	Diagrama de Gráfico de Estados do agente <i>Radio</i>	59
4.11	Diagrama de Colaboração para a determinação de referências para os atuadores.	62
4.12	Diagrama de Colaboração para o monitoramento da arquitetura.	63
4.13	Diagrama de Seqüência mostrando um cenário de execução de missão.	64
4.14	Diagrama de Seqüência mostrando um cenário de monitoramento de um sensor.	65
5.1	Embarcação utilizada em testes ITTC.	66
5.2	Modelo ITTC em escala reduzida.	67
5.3	Diagrama esquemático do modelo em escala de navio.	68
5.4	Bússola eletrônica TCM2.	71
5.5	Equipamento de posicionamento global.	72
5.6	Unidade inercial VG600AA.	73
5.7	Equipamento <i>wireless</i> LAN da <i>JRC Ltda.</i>	75
5.8	Antena externa direcional NZA-4200, bateria externa NBB-476 e antena externa onidirecional NZA-200.	75
5.9	Equipamento para telecomando de aeromodelos.	75
5.10	Posição do centro de gravidade (LCG) do veículo, coincidente com seu centro de flutuação (LCB).	77
5.11	Sistema de coordenadas adotado e movimentos medidos em ensaios.	78
5.12	Velocidade angular ω_x no ensaio em repouso.	79
5.13	Velocidade angular ω_y no ensaio em repouso.	79
5.14	Velocidade angular ω_z no ensaio em repouso.	80
5.15	Movimento de avanço A_x no ensaio em repouso.	80
5.16	Movimento lateral A_y no ensaio em repouso.	81
5.17	Movimento de arfagem A_z no ensaio em repouso.	81
5.18	Rotações dos propulsores em ensaio de vibração.	83
5.19	Movimento de avanço A_x em ensaio de vibração.	83
5.20	Movimento lateral A_y em ensaio de vibração.	84

5.21	Movimento de arfagem A_z em ensaio de vibração.	84
5.22	Velocidade angular ω_x em ensaio de vibração.	85
5.23	Velocidade angular ω_y em ensaio de vibração.	85
5.24	Velocidade angular ω_z em ensaio de vibração.	86
5.25	Dados de movimento do ensaio em águas calmas.	87
5.26	Dados de deslocamento angular do ensaio em águas calmas.	87
5.27	Ângulo de rumo em ensaio sob comando manual.	90
5.28	Rotação dos motores de propulsão em ensaio sob comando manual.	91
5.29	Corrente instantânea em motores de propulsão em ensaio sob comando manual.	91
5.30	Ângulo de rumo em ensaio sob comando manual.	92
5.31	Rotação dos motores de propulsão em ensaio sob comando manual.	92
5.32	Resposta típica de um ensaio em <i>zig-zag</i>	93
5.33	Ângulo de rumo e leme do modelo em escala de navio em ensaio de <i>zig-zag</i>	95
5.34	Diagrama de blocos representando modelo do veículo mais controlador PI.	96
5.35	Visão superior do modelo em escala de navio.	97
5.36	Detalhe mostrando bússola eletrônica TCM2, equipamento <i>wireless</i> LAN e unidade inercial VG600AA.	97
5.37	Detalhe mostrando motores de propulsão, servo <i>drives</i> , PC104 (centro) e unidade inercial VG600AA (à direita).	98
5.38	Detalhe mostrando conjunto de baterias (esquerda) utilizadas, servo <i>drive</i> (direita) e mecanismo para posicionamento dos lemes.	98
5.39	Ângulo de rumo em Ensaio I.	103
5.40	Rotação dos motores de propulsão em Ensaio I	103
5.41	Corrente instantânea em motor de propulsão em Ensaio I.	104
5.42	Corrente instantânea em motor de propulsão em Ensaio I.	104
5.43	Esforço de controle em Ensaio I	105
5.44	Deslocamentos angulares em Ensaio I	105
5.45	Movimento em <i>Roll</i> ω_x em Ensaio I.	106
5.46	Movimento em <i>Pitch</i> ω_y em Ensaio I.	106
5.47	Movimento em <i>Yaw</i> ω_z em Ensaio I.	107
5.48	Movimento de avanço A_x em Ensaio I.	107

5.49	Movimento lateral A_y em Ensaio I.	108
5.50	Movimento de arfagem A_z em Ensaio I.	108
5.51	Ângulo de rumo em Ensaio IV.	112
5.52	Rotação dos motores de propulsão em Ensaio IV.	112
5.53	Corrente instantânea em motor de propulsão em Ensaio IV.	113
5.54	Corrente instantânea em motor de propulsão em Ensaio IV.	113
5.55	Esforço de controle em Ensaio IV.	114
5.56	Deslocamentos angulares em Ensaio IV.	114
5.57	Movimento em <i>Roll</i> ω_x em Ensaio IV.	115
5.58	Movimento em <i>Pitch</i> ω_y em Ensaio IV.	115
5.59	Movimento em <i>Yaw</i> ω_z em Ensaio IV.	116
5.60	Movimento de avanço A_x em Ensaio IV.	116
5.61	Movimento lateral A_y em Ensaio IV.	117
5.62	Movimento de arfagem A_z em Ensaio IV.	117
5.63	Sistema GPS.	118
5.64	Sistema de coordenadas ECEF e ENU.	120
5.65	Vista superior o modelo em escala de navio.	121
5.66	Detalhe mostrando antena e receptor GPS, PC104 e motores de propulsão. . . .	121
5.67	Posição X e Y do veículo em Ensaio 1.	125
5.68	Ângulo de rumo em Ensaio 1.	125
5.69	Posição X e Y do veículo em Ensaio 2.	126
5.70	Posição X e Y do veículo em Ensaio 3.	126

Lista de Tabelas

2.1	Propriedade desejáveis para um agente.	18
2.2	Agentes da arquitetura proposta e respectivas descrições <i>PAGE</i>	21
4.1	Descrição do comportamento interno do agente <i>Gerenciador</i>	55
4.2	Descrição do comportamento interno do agente <i>Monitor</i>	60
4.3	Descrição do comportamento interno do agente <i>Atuador</i>	61
4.4	Descrição do comportamento interno do agente <i>Sensor</i>	61
4.5	Descrição do comportamento interno do agente <i>Radio</i>	62
5.1	Resumo das informações a respeito da bússola eletrônica TCM2.	72
5.2	Resumo das informações a respeito do equipamento GPS.	73
5.3	Resumo das informações a respeito da unidade inercial VG600AA.	73
5.4	Posições relativas da unidade inercial e da bússola eletrônica em relação ao centro de gravidade (LCG) do veículo.	77
5.5	Valores médios e desvios médios dos dados de ensaio em repouso.	82
5.6	Valores médios e desvios médios dos dados de ensaio de vibrações.	82
5.7	Valores médios de dados do ensaio em águas calmas.	88
5.8	Ganhos utilizados para o controlador PI.	96
5.9	Resumo dos resultados do Ensaio I.	100
5.10	Resumo dos resultados do Ensaio II.	109
5.11	Resumo dos resultados do Ensaio III.	110
5.12	Resumo dos resultados do Ensaio IV.	111

Capítulo 1

Introdução

Robôs móveis servem como plataformas de testes para sistemas mecatrônicos. O desenvolvimento de tais máquinas depende de conceitos das mais diversas áreas do conhecimento, dentre elas da teoria da cognição, do controle e automação, do sensoriamiento, da engenharia elétrica e mecânica.

Num robô móvel, o grau de autonomia depende invariavelmente da implementação a bordo de uma diversidade de funções que venham a colaborar com os objetivos da sua missão. Tais funções podem ser classificadas em relação a conceitos tais como percepção do ambiente externo, análise e avaliação do andamento da missão, planejamento de ações e a sua execução, dentre outros.

O sistema de *hardware* e *software* responsável pela execução, controle e coordenação destas funções em tempo-real é denominada de arquitetura de controle do robô. Este tópico, crucial no desenvolvimento de todo robô autônomo, corresponde ao tema desta dissertação de mestrado.

1.1 Motivação

Dentre os diversos temas que trazem motivações para o desenvolvimento das atividades realizadas neste trabalho de dissertação de mestrado, podem ser citados os seguintes:

- a investigação na literatura especializada dos diversos paradigmas que orientaram o de-

envolvimento de arquiteturas de controle destinadas aos robôs móveis autônomos até o presente momento, em especial dos veículos autônomos submarinos;

- a elaboração de abordagens baseadas em modelos sócio-antropológicos e computacionais, tais como a teoria de agentes e orientação a objetos, na concepção de novas arquiteturas de controle destinadas a robôs móveis autônomos;
- o estudo de sistemas de tempo-real, caracterização de seus aspectos e o levantamento de como estes são abordados no desenvolvimento das arquiteturas de controle dos robôs móveis autônomos.

1.2 Aplicação Específica

As atividades deste programa de pesquisa constituem uma parte importante do projeto temático "Comportamento em Ondas de Embarcações de Alto Desempenho" (proc. Fapesp 1997/13090-3), que é diretamente relacionada a testes experimentais para a investigação e validação de modelos de comportamento de embarcações de alta velocidade.

A idéia é utilizar modelos em escala reduzida (ver Figura 5.2) para a validação de métodos de predição numérica e analítica desenvolvidos dentro do projeto temático.

Se por um lado tais modelos em escala de navios propiciam a coleta de informações a respeito da dinâmica e manobrabilidade das embarcações reais, por outro lado servem como uma fase de desenvolvimento de menor custo para robôs móveis autônomos (quando comparados a implementações submarinas de veículos submarinos autônomos e veículos submarinos remotamente operados), constituindo assim bancadas de testes para a implementação e validação da arquitetura de *software* proposta neste trabalho.

É preciso ressaltar o papel relevante deste trabalho numa pesquisa mais ampla na área de tecnologia submarina. Esta área possui grande importância no panorama atual do país dada a situação de capacitação atingida pela engenharia nacional no que tange à exploração petrolífera *off-shore*. Ressalta-se também a utilidade do tema como base de desenvolvimento de sistemas mecatrônicos e de automação em geral.

1.3 Objetivos da Dissertação

Os estudos e implementações realizados dentro deste trabalho têm como principais metas as seguintes:

- propor uma arquitetura de controle híbrida, flexível, composta por uma comunidade de agentes que se comunicam através de troca de mensagens, destinada a aplicações de robôs móveis autônomos;
- apresentar uma modelagem orientada a objetos desta arquitetura, buscando ilustrar os aspectos que conferem a esta arquitetura características de um sistema de tempo-real multi-agentes;
- desenvolver o sistema de controle e aquisição de dados de uma plataforma de testes voltada a pesquisa em controle de robôs móveis, particularmente para a coleta de informações para o projeto temático e a validação da arquitetura de controle proposta, vislumbrando futuramente aplicações em veículos oceânicos autônomos.

1.4 Estrutura da Dissertação

A presente dissertação de mestrado inicia-se, no capítulo 2, com uma breve visão geral a respeito dos robôs móveis autônomos. Primeiramente, é apresentada uma classificação para os diversos tipos de implementações destes veículos segundo o ambiente de operação dos mesmos. A seguir, é definido o que vem a ser uma arquitetura de controle de um robô móvel autônomo, enumerando-se as diversas características desejáveis para esta arquitetura de controle e como elas são classificados segundo dois aspectos: a presença ou não de um modelo do ambiente operacional do veículo e a relação de comando entre os diversos módulos que constituem a arquitetura. Aspectos sobre a teoria de agentes e orientação a objetos que contribuem para o desenvolvimento de novas arquiteturas são então analisados, finalizando-se o capítulo

com a apresentação de uma proposta de arquitetura de controle híbrida, composta por uma comunidade de agentes que se comunicam através de troca de mensagens.

O capítulo 3 fornece noções básicas a respeito dos sistemas de tempo real, buscando contextualizá-los dentro das aplicações de veículos autônomos e justificar as abordagens adotadas na elaboração da arquitetura de controle proposta por este trabalho para o tratamento dos requisitos de tempo real. Inicialmente, é apresentada uma discussão a respeito das características gerais destes sistemas, seguida de uma seção a respeito de sistemas operacionais de tempo real. Finalizando o capítulo, apresenta-se a plataforma de *hardware* e *software* escolhida para a implementação da arquitetura de *software* proposta por este trabalho.

Tal arquitetura é detalhada no capítulo 4, que trata do tema modelagem baseada em objetos. Inicialmente é apresentada uma visão geral da UML, *Unified Modeling Language*, linguagem e método de modelagem orientado a objetos escolhida para a utilização neste projeto, além das extensões desta linguagem destinadas para a aplicação específica em sistemas multi-agentes (*Agent UML*) e em sistemas de tempo real (*Real-Time UML*).

No capítulo 5 são apresentados os diversos equipamentos de *hardware* e montagens eletrônicas utilizados para a plataforma de aplicação da arquitetura proposta no capítulo anterior, juntamente com a descrição dos ensaios executados e análise de seus resultados, tanto para os estudos de identificação dinâmica previstos no projeto temático ao qual esta pesquisa é vinculada, como para a avaliação de características da arquitetura proposta.

Finalizando este trabalho, o capítulo 6 contém algumas conclusões dos estudos aqui realizados, juntamente com sugestões para trabalhos futuros objetivando a continuidade das investigações realizadas nesta pesquisa.

Capítulo 2

Visão Geral sobre Robôs Móveis Autônomos

Na sociedade atual é crescente a necessidade de se realizar tarefas com eficiência e precisão. Paralelamente, existem tarefas a serem realizadas em lugares onde a presença humana se torna difícil, arriscada e até mesmo impossível, como é o caso do fundo do mar ou a imensidão do espaço. Para realizar essas tarefas, se faz cada vez mais necessária a presença de dispositivos (robôs), que realizam essas tarefas sem riscos a vida humana e com eficiência e precisão tipicamente superiores às alcançadas pelo homem. A robótica, é a área da ciência que se preocupa com o desenvolvimento de tais dispositivos. Em particular, o desenvolvimento de robôs móveis autônomos é o que diretamente mais contribui para a realização de atividades até então inviáveis ao ser humano. Exemplos destas atividades incluem a exploração da superfície de outros planetas, a remoção de minas terrestres e submarinas, a limpeza de acidentes nucleares, a exploração de vulcões, entre outras.

Um robô móvel autônomo pode ser definido como um veículo provido de um sistema sensorial e um sistema atuador, gerenciado por uma arquitetura de controle capaz de executar missões definidas pelo usuário (Ridao, Batlle, Amat e Roberts, 1999). Em geral, um robô móvel autônomo apresenta fonte de energia própria e depende de computadores de bordo para a execução de missões. Tais missões consistem de uma série de instruções pré-programadas e modificáveis em tempo-real por meio de dados ou informações provenientes dos sensores do veículo (Valavanis, Gracanin, Matijasevic, Kolluru e Demetriou, 1997).

Fundamentalmente, todo robô móvel autônomo deve desempenhar um conjunto comum de funções que requerem o uso de dispositivos de entradas (sensores) para detectar sua orientação e um conjunto de dispositivos de saída como controles de superfície ou propulsores que ajustem ou mantenham a orientação. No centro deste sistema está o controlador do veículo, representado na maioria dos casos por microcontroladores ou microcomputadores. Um conjunto mínimo de dispositivos de saída deve incluir algum tipo de controle móvel de superfície e um sistema de propulsão. Entretanto, muitos controladores de robôs móveis autônomos terão, em adicional ao conjunto de dispositivos de entrada e saída, um conjunto de dispositivos específicos necessários para satisfazer as exigências do sistema ou missão específica (Henrickson e Dzielski, 1995), tais como sonares, sensores infra-vermelho, sistemas de posicionamento via satélite (GPS), etc.

De acordo com o tipo de ambiente de execução das missões e para que função são destinados, os robôs móveis autônomos apresentam algumas diferenciações quanto aos requisitos para sua implementação, dispositivos de entrada e saída necessários, etc. Assim, tomando-se como parâmetro o ambiente de execução das missões, é possível classificar os robôs móveis autônomos em:

- Veículos Aéreos Não Tripulados (*Unmanned Aerial Vehicles - UAVs*) surgiram durante a Segunda Guerra Mundial e têm sido destinados principalmente para uso militar (reconhecimento e espionagem). Um exemplo é o UAV *Pioneer* (Reid e Valley, 1996) (ver Figura 2.1), utilizado pela Marinha dos Estados Unidos na Operação Tempestade no Deserto (Guerra do Golfo) e, mais recentemente, nos ataques de retaliação ao Afeganistão;

Figura 2.1: *Pioneer*, UAV utilizado na Guerra do Golfo e nos ataques ao Afeganistão.

- Veículos Terrestres Não Tripulados (*Unmanned Ground Vehicles - UGVs*), incluem veículos teleoperados e autônomos, têm sido usados em aplicações civis e militares, com

Figura 2.2: *Sojourner*, veículo de exploração enviado ao planeta Marte.

Figura 2.3: *Martin*, AUV desenvolvido pela *Maridan ApS* destinado a inspeção de oleodutos e cabos submarinos.

destaque a exploração de superfície planetária como o *Sojourner* (Matijevic, 1997) (ver Figura 2.2), desenvolvido pelo *Jet Propulsion Laboratory Mars Rover* e que foi enviado em missão ao planeta Marte;

- Veículos Submarinos Não Tripulados (*Unmanned Underwater Vehicles - UUVs*) incluem veículos operados remotamente e veículos submarinos autônomos:
 - Veículos Operados Remotamente (*Remotely Operated Vehicles - ROVs*), são presos a um navio suporte através de um cabo umbilical que pode enviar energia e sinais de controle para o veículo, ao mesmo tempo que retorna imagens e outros dados sensoriais para o navio suporte. São particularmente importantes quando a localização de destino submarino é incerta ou quando as condições ambientais são arriscadas para uma missão tripulada. Os ROVs são continuamente controlados e navegados por um operador humano.
 - Veículos Submarinos Autônomos (*Autonomous Underwater Vehicles - AUVs*) são veículos submarinos não tripulados que carregam sua própria fonte de energia e microcomputadores de bordo para a execução de missões pré-definidas. Um AUV pode ser lançado por navios simples e menores (em comparação aos ROVs) ou até mesmo em cais, uma vez que não possui cabo umbilical. Também possibilitam operações a distâncias significativas de um navio suporte ou plataforma.

A área de atuação mais fecunda na atualidade para estes tipos de veículos, particularmente para os ROVs, é a indústria de exploração de petróleo *offshore*. Outras aplicações incluem remoção de minas, instalação e inspeção de cabos submarinos, entre outras. Como exemplos de implementações destes veículos cita-se o *Martin* (Madsen, Christensen e Lauridsen, 2000) (ver Figura 2.3), o *Autonomous Benthic Explorer (ABE)* (Yoerger, Bradley, Cormier, Ryan e Walden, 1999), o *Ocean Technologies Testbed for Engineering Research (OTTER)* (Wang e Rock, 1996), o *Ocean Voyager II* (Smith, 1994), o *Omni-Directional Intelligent Navigator (ODIN)*, o *Sea Squirt* (Bellingham e Consi, 1990), o *GARBI* (Amat, Monferrer, Batlle e Cufi, 1999), o *Phoenix* (Marco, Healey e McGhee, 1996), entre outros.

Em função de suas características de intenso processamento computacional, grande quantidade de dados sensoriais e necessidade de execução em tempo real, o projeto da arquitetura de controle é o principal problema para o desenvolvimento de robôs móveis autônomos. Para o caso de veículos autônomos submarinos (AUVs), este problema torna-se mais complexo devido às limitações de comunicação e energia inerentes às condições ambientais dos meios aos quais estes veículos se destinam (Valavanis et al., 1997).

2.1 Arquiteturas de Controle para Robôs Móveis Autônomos

De forma simplificada, o que proporciona autonomia aos chamados robôs móveis autônomos é o sistema de *hardware* e *software* responsável pela execução, controle e coordenação em tempo-real das diversas funções que colaboram para o alcance dos objetivos da missão. Este sistema é que será aqui denominado por *arquitetura de controle*.

Tomando-se como base estudos recentes a respeito do assunto, particularmente os publicados em (O'Reilly, 2000; Ridao et al., 1999; Valavanis et al., 1997; Hall e Adams, 1992), pode-se citar como características desejáveis em uma boa arquitetura de controle destinada a robôs móveis autônomos:

1. **Extensibilidade:** a inclusão de novos sensores, atuadores e módulos funcionais deve

ser facilitada, sem que seja necessário grandes modificações na estrutura já existente. E ainda, os novos sistemas devem ser capazes de interagir com os componentes previamente existentes.

2. **Generalidade:** os componentes da arquitetura devem ser suficientemente genéricos para que possam ser intercambiados entre diferentes veículos, ou substituídos por componentes equivalentes sem modificações significativas. Esta característica é desejável tanto para o *hardware* quanto para o *software* da arquitetura.
3. **Previsibilidade:** o comportamento como um todo do sistema deve ser facilmente previsível com base nos objetivos da missão e nos dados fornecidos pelo sistema sensorial. Em resumo, o sistema não deve se comportar de forma aleatória.
4. **Modelo do ambiente:** em aplicações de veículos autônomos estes modelos codificam as estruturas estáticas (paredes e obstáculos fixos) do ambiente. Isto permite determinar a posição absoluta do veículo e, por conseguinte, planejar trajetórias ótimas. Em robótica submarina, o modelo do ambiente consiste geralmente de um contorno da superfície submersa.
5. **Reatividade:** ambientes operacionais reais são dinâmicos. Isto é especialmente verdade para aplicações submarinas onde o meio é freqüentemente desconhecido. O veículo deve ser reativo o suficiente para sobreviver neste tipo de ambiente. Para isto, ações planejadas a partir do modelo do ambiente devem ser convenientemente fundidas com reações a situações não previstas no modelo.
6. **Deteção e recuperação de erros:** erros podem aparecer durante a execução de uma missão, especialmente em uma missão de longo alcance. A arquitetura deve prover um mecanismo de deteção de erros e processos de recuperação definidos pelo usuário que evitem o cancelamento da missão. Quando a recuperação é impossível, os módulos defeituosos devem ser indicados como danificados de forma a evitar que sejam utilizados em objetivos subseqüentes.
7. **Tolerância a falhas:** sempre que o nível de redundância dos componentes do veículo é suficiente, qualquer mecanismo de reposição dos componentes falhos por componentes

equivalentes é desejável. Para missões de longo alcance, tolerância à falhas é de grande importância.

8. **Programabilidade:** o usuário deve ser provido de ferramentas para especificação e verificação de missões. Enquanto linguagens de especificação de missões são adequadas para desenvolvedores, interfaces gráficas são mais adequadas para usuários finais.

Na literatura sobre robôs móveis, o robô *Shakey*, apresentado pelo *Stanford Research Institute*, é tido como uma das primeiras implementações de robôs autônomos. Este robô podia navegar e empurrar objetos em um ambiente de várias salas. Seu *hardware* consistia de um carro móvel, do tamanho de um pequeno refrigerador, com sensores táteis, uma câmera de vídeo e um telêmetro óptico. O *software* consistia de programas necessários para a análise do panorama visual (o robô podia reconhecer caixas, entradas e cantos de salas), um planejador (o robô podia planejar seqüências de ações para atingir objetivos) denominado STRIPS, *STanford Research Institute Problem Solver* (Fikes, Hart e Nilsson, 1971), e de programas para converter seus planos em ações sobre o ambiente.

Desde então, um grande número de arquiteturas de controle foram implementadas, seja em veículo autônomos terrestres, veículos autônomos submarinos, veículos destinados à exploração planetária e outros. Porém, em todos os casos, características comuns são encontradas nestas arquiteturas, o que possibilita a classificação das mesmas segundo vários critérios como a distribuição da abstração ao longo das unidades de controle, a presença ou não de modelos e representações do ambiente, a disponibilidade de conhecimento pelas unidades, etc (Hall e Adams, 1992). Tomando-se como base a presença ou não de um modelo do ambiente operacional do veículo, foi apresentado em Ridao et al. (1999) a classificação das arquiteturas em:

- **Arquiteturas Deliberativas:** estas arquiteturas são baseadas principalmente em um processo de planejamento e em um modelo do ambiente de execução da missão. Esta é especificada como um conjunto de objetivos a serem alcançados. Através do modelo do ambiente, a missão é planejada e um sistema de controle executa o plano de ação obtido. Arquiteturas desenvolvidas segundo esta abordagem geralmente apresentam comportamento previsível, porém não se adaptam com facilidade a ambientes muito dinâmicos, nos quais as mudanças ocorrem rapidamente;

- **Arquiteturas Comportamentais:** também conhecidas como arquiteturas reativas, são divididas nos diversos comportamentos que se deseja para o veículo. Normalmente, uma missão é descrita como uma seqüência de fases com um conjunto de comportamentos ativos. Os comportamentos reagem continuamente aos estímulos detectados pelo sistema de percepção. O comportamento global do veículo surge da combinação dos comportamentos elementares ativos;
- **Arquiteturas Híbridas:** estas arquiteturas são as que possuem maior número de implementações. Elas unem características das arquiteturas deliberativas e comportamentais, com elementos determinísticos sendo utilizados para proporcionar ao sistema comportamentos previsíveis e com elementos reativos sendo utilizados para se prover ações de resposta mais rápidas em situações que o sistema não é capaz de prever.

A classificação anterior, apesar de completa e relevante para uma análise conceitual das arquiteturas de controle, não permite um detalhamento maior a respeito da implementação das mesmas. Neste sentido, Valavanis et al. (1997) dá mais atenção a outro tipo de abordagem. Neste caso, o que foi tomado como base para a classificação das arquiteturas foi a relação de comando entre os diversos módulos que constituem o veículo. Segundo esta abordagem, as arquiteturas de controle para robôs móveis autônomos foram classificadas em: hierárquicas, heterárquicas, *subsumption* e híbridas. Nas próximas subseções estas classificações serão melhor detalhadas, sendo apresentadas suas vantagens e desvantagens, assim como exemplos de implementações das mesmas.

2.1.1 Arquiteturas Hierárquicas

Estas arquiteturas utilizam uma abordagem hierárquica para dividir o sistema em níveis. Os níveis superiores são responsáveis pelos objetivos gerais da missão, enquanto os níveis inferiores são responsáveis por solucionar problemas específicos para a realização da missão. Estas arquiteturas consistem de uma estrutura serial em que a comunicação direta é possível apenas entre dois níveis adjacentes. Os níveis superiores enviam comandos para os níveis inferiores e, como resposta, recebem destas informações de retorno. O fluxo de informação decresce da base

para o topo da hierarquia. Como vantagem, estas arquiteturas apresentam uma representação bem definida da estrutura. Isto faz com que seja mais fácil verificar a controlabilidade e estabilidade, ou seja, é possível a avaliação de desempenho da arquitetura. A desvantagem destas arquiteturas é a falta de flexibilidade e, como consequência, uma tentativa de modificar alguma funcionalidade requer modificações significativas de todo o sistema. Desde que não exista comunicação direta entre o controle dos níveis superiores e os periféricos (sensores, atuadores) dos níveis inferiores, o tempo de resposta (sensor de entrada - ação do sistema) é grande, e a integração/fusão dos sensores é difícil. Como consequência, sistemas que usam estas arquiteturas não demonstram comportamento dinâmico real quando exigido em situações imprevistas.

Figura 2.4: OTTER, *Ocean Technologies Testbed for Engineering Research*.

Figura 2.5: Diagrama esquemático da arquitetura de controle do OTTER.

Como exemplos desta arquitetura cita-se o *Ocean Technologies Testbed for Engineering Research - OTTER* (Wang e Rock, 1996) (ver Figura 2.4), desenvolvido conjuntamente pelo *Monterey Bay Aquarium Research Institute* e *Stanford University*. Este veículo implementa uma arquitetura com três camadas hierárquicas (ver Figura 2.5). A camada inferior, denominada *Servo Level*, implementa o controle dos motores através de malhas de controle clássicas. A camada intermediária, denominada *Task Level*, é responsável pela codificação e sequenciamento das tarefas. A camada superior, denominada *Organization Level*, proporciona a interface com o usuário via telas gráficas.

2.1.2 Arquiteturas Heterárquicas

Ao contrário das arquiteturas hierárquicas, as arquiteturas heterárquicas usam uma estrutura paralela onde todos os módulos do sistema podem comunicar-se diretamente entre si, sem níveis intermediários ou supervisores. A grande vantagem deste esquema é sua flexibilidade. Desde que o conhecimento e as informações sensoriais possam ser facilmente acessados por quaisquer componentes do sistema, estas arquiteturas também são adequadas para aplicações de processamento paralelo. A desvantagem é que, devido à falta de supervisão, a comunicação entre os módulos pode ser muito intensa e a controlabilidade torna-se um problema.

Há poucos exemplos na literatura de implementações que utilizem puramente esta abordagem. Uma delas é apresentada em De Barros (1994), arquitetura desenvolvida no *Institute of Industrial Science* da *University of Tokyo* e implementada no AUV Pteroa-150 (ver Figura 2.6). Tal arquitetura foi modelada como uma comunidade de agentes (De Barros, Maeda e Miyajima, 1995), entidades autônomas que coordenam suas ações através de protocolos de comunicação (ver Figura 2.7).

Figura 2.6: Pteroa-150, AUV desenvolvido no *Institute of Industrial Science* da *University of Tokyo*.

Figura 2.7: Diagrama esquemático da arquitetura de controle do Pteroa-150.

2.1.3 Arquiteturas *Subsumption*

As arquiteturas *subsumption* (ou controle por níveis) consiste de procedimentos trabalhando em paralelo, sem um nível superior supervisor. Procedimentos são níveis da arquitetura de controle que são ativados por sensores executando ações. Um nível pode ter prioridade em relação a outro. Embora os níveis ainda trabalhem em paralelo, o procedimento com maior prioridade pode suprimir o outro procedimento. Uma vez que o procedimento de maior prioridade é ativado apenas por um curto período de tempo, o procedimento de menor prioridade reassume o controle. Dados e controles são distribuídos através de todas os níveis, e cada nível processa sua própria informação, isto é, não há uma estrutura de dados global. Como vantagens, este esquema possui flexibilidade, robustez e uma baixa sobrecarga computacional. Estas arquiteturas apresentam comportamento dinâmico real. As desvantagens das arquiteturas *subsumption* incluem a dificuldade de sincronização entre os procedimentos, complexidade do sistema com grande número de procedimentos e a falta de um nível de controle superior. Conseqüentemente, é difícil a verificação do sistema e a realização de testes de estabilidade e ajuste.

Figura 2.8: Arquitetura de controle do AUV *Sea Squirt*.

Como resultado das limitações anteriores, implementações práticas de veículos autônomos usam modificações das arquiteturas *subsumption* com a adição de funções de controle nos níveis superiores (formalização, tabela de estados). Neste contexto, um exemplo é o veículo *Sea Squirt* (Bellingham e Consi, 1990), desenvolvido pelo *Massachusetts Institute of Technology*. Neste veículo, implementou-se uma arquitetura de controle dividida em dois níveis (ver Figura

2.8): o nível superior implementando uma tabela de estados e o nível inferior uma estrutura de controle multi-camadas. O problema de sincronização associado à arquiteturas de controle multi-camadas foi superado através da adição de um nível superior de controle que ativa apenas os comportamentos apropriados para cada fase específica da missão. A tabela de estados indica o estado atual do veículo e garante que os comportamentos são ativados no tempo certo e com a prioridade certa.

2.1.4 Arquiteturas Híbridas

As arquiteturas híbridas são uma combinação das arquiteturas hierárquicas, heterárquicas e *subsumption*. O sistema é dividido em dois níveis, superior e inferior, os quais usam diferentes níveis de abstração. O nível superior usa algum tipo de arquitetura hierárquica para implementar as estratégias e funcionalidades de sua missão. O nível inferior usa tanto arquiteturas heterárquicas como *subsumption* para controlar os subsistemas de *hardware*. Na arquitetura *subsumption*, comandos do nível superior são traduzidos em procedimentos correspondentes, os quais são então ativados. Na arquitetura heterárquica, o nível inferior consiste de vários módulos desempenhando operação normais do sistema. Uma situação de emergência pode ativar a supervisão do nível superior para assumir o controle. As vantagens deste esquema é que, embora preserve as vantagens das arquiteturas hierárquicas, a flexibilidade dos níveis inferiores é alcançada. A desvantagem deste esquema é que a verificação formal do sistema não é facilmente realizável.

Como exemplo de implementações desta abordagem têm-se o AUV *Phoenix* (Marco et al., 1996), desenvolvido pela *Naval Post-graduate School*, de *Monterey*. O controle deste veículo é baseado em uma arquitetura de *software* híbrida de três níveis: estratégico, tático e de execução. A arquitetura hierárquica é utilizada entre os três níveis, enquanto a arquitetura *subsumption* é usada no nível de execução. O nível estratégico utiliza o Prolog como linguagem de especificação da missão baseada em regras. Já o nível tático consiste de um conjunto de funções em linguagem C, que realizam o interfaceamento com os predicados Prolog e retornam valores *booleanos* quando chamadas. A comunicação entre o nível tático e o controlador do nível de execução é feita através de passagem de mensagens assíncronas via rede. No nível de

Figura 2.9: *Phoenix*, AUV de arquitetura híbrida.

Figura 2.10: Diagrama esquemático da arquitetura de controle do *Phoenix*.

execução, os atuadores e sensores são operados de forma síncrona, em resposta aos comandos do nível superior. Finalizando, a arquitetura do *Phoenix* incorpora ainda procedimentos de detecção e recuperação de erros.

2.2 Agentes e Desenvolvimento Baseado em Objetos

De maneira geral, grande parte das pesquisas em arquiteturas de controle para robôs móveis autônomos da atualidade têm buscado as vantagens de abordagens híbridas como forma de superar as limitações enfrentadas pelas outras abordagens. Neste contexto, a utilização de conceitos como *orientação a objetos* e *agentes* figuram como campos fecundos, tanto no desenvolvimento quanto na implementação de novas arquiteturas.

Empregando os conceitos de sistemas orientados a objetos, (Zheng, Jackson e Kao, 1990; Rodseth, 1991; Kwak e Thornton Jr, 1994; O'Reilly, 2000) trazem exemplos de implementações de arquiteturas híbridas para robôs móveis autônomos, particularmente veículos autônomos submarinos. Nestas abordagens, procura-se acrescentar as vantagens proporcionadas pelas técnicas de projeto baseadas em objetos à implementação das arquiteturas de controle.

Por outro lado, no campo de agentes inteligentes, Müller (1996) cita uma abordagem que

tem rendido várias implementações nos últimos anos denominada arquiteturas multi-camadas (*layered architectures*). Esta abordagem híbrida figura como uma ferramenta valiosa para o projeto de sistemas que apresentem tanto propriedades reativas quanto deliberativas. A idéia principal desta abordagem é estruturar as funcionalidades da arquitetura em dois ou mais níveis que interagem entre si de forma a se alcançar comportamentos coerentes da arquitetura como um todo.

Nas seções seguintes é apresentado um rápido resumo a respeito do conceito de agente e sua relação com o desenvolvimento de sistemas baseado em objetos, procurando identificar melhor a relação entre os conceitos objeto e agente, por entender que ambos têm muito a contribuir no campo das arquiteturas de controle para robôs móveis autônomos. Isto também ajudará a contextualizar a arquitetura proposta por este trabalho.

2.2.1 Definição de Agente

O conceito de agente é apresentado dentro da literatura de uma maneira bastante diversificada. De fato, a definição de agente está longe de apresentar uma uniformidade mesmo entre especialistas, cabendo aqui o comentário encontrada em Franklin e Graesser (1996). Para os autores desta referência, as definições de agente existentes na literatura especializada surgem do conjunto de exemplos de agentes que é adotado pelos seus respectivos autores, o que explica a diversidade de definições. Evidentemente, este fato não impede que se encontre idéias comuns entre as diversas definições existentes, idéias estas que muitas vezes se complementam.

Seja qual for a definição adotada, observa Russel e Norvig (1995) que o conceito de agente deve ser uma ferramenta para a análise de sistemas e não uma caracterização absoluta que divide o mundo em agentes e não-agentes.

E são os próprios autores de Russel e Norvig (1995) que fornecem uma definição ao mesmo tempo simples e abrangente para o conceito de agente, a qual será adotado neste trabalho:

"Um agente é qualquer coisa que pode perceber seu ambiente através de sensores e agir sobre este ambiente através de atuadores."

A Figura 2.11 ilustra a idéia geral apresentada pela definição anterior. Nota-se que a tarefa

de se determinar o que vem a ser o ambiente de execução do agente, sua forma de perceber este ambiente, assim como a maneira de agir sobre este, é deixada para o leitor.

Figura 2.11: Agente interagindo com ambiente através de sensores e atuadores.

Em Franklin e Graesser (1996) tem-se uma análise das diversas definições de agente encontradas na literatura especializada, inclusive a apresentada anteriormente. A partir destas diversas definições, foi possível a enumeração de algumas propriedades que podem ser atribuídas a um agente. Tais propriedades são listadas na Tabela 2.1.

Tabela 2.1: Propriedade desejáveis para um agente.

Propriedade	Significado
Reatividade	Responde em um determinado tempo às mudanças no ambiente.
Autonomia	Exerce controle sobre suas próprias ações.
Orientado a Objetivos (pro-ativo)	Não age simplesmente em resposta ao ambiente.
Temporalmente Contínuo	Está continuamente em processo de execução.
Sociabilidade	Comunica-se com outros agentes, possivelmente pessoas.
Aprendizagem (adaptabilidade)	Muda seu comportamento baseado em suas experiências prévias.
Mobilidade	Capaz de locomover-se em seu ambiente.
Flexibilidade	Não age segundo um roteiro.
Carater	Possui <i>personalidade</i> e estado <i>emocional</i> .

Dentre a gama de propriedades contidas na Tabela 2.1, as cinco primeiras são de maior

importância para a proposta de arquitetura de controle apresentada neste trabalho. De fato, Franklin e Graesser (1996) chega a considerar essenciais as quatro primeiras propriedades para se definir um agente. Aqui, acrescentamos a sociabilidade, pois a capacidade de comunicar-se com os demais agentes é central na abordagem da arquitetura proposta.

2.2.2 Conceito de Objeto

Em Wooldridge (1999) encontra-se a seguinte definição para o conceito de objeto:

"Objetos são entidades computacionais que encapsulam algum estado, são capazes de executar ações ou métodos neste estado e comunicar-se através de troca de mensagens."

Tal definição apresenta alguma similariedade com a apresentada anteriormente na seção 2.2.1 para definir um agente, o que torna pertinente a seguinte questão: que relação pode ser observada entre os conceitos objeto e agente? Esta é uma pergunta que pesquisadores destas duas áreas têm formulado e que os autores de Wooldridge (1999) procuram deixar mais clara.

À primeira vista, desenvolvedores de sistemas orientados a objetos podem ter a impressão que esses dois conceitos se equivalem. Entretanto, uma importante distinção entre eles é quanto ao grau de autonomia observado em um e em outro ¹. Desde que um objeto disponibiliza alguns de seus métodos para acesso público, tal objeto passa a não ter mais controle sobre a execução destes métodos, podendo estes serem chamados por qualquer um dos objetos do sistema. No caso de agentes, não há chamadas de métodos e sim a requisições de ações entre agentes, ficando a decisão de execução ou não da ação, assim como o controle sobre a mesma, por conta do agente proprietário.

Evidentemente, a distinção acima não invalida a utilização de métodos orientados a objetos para o desenvolvimento de sistemas multi-agentes. Pelo contrário, com o entendimento adequado da relação entre os conceitos em questão é possível melhor utilizá-la na análise e projeto de sistemas baseados em agentes.

¹ Em Wooldridge (1999) são citadas duas outras distinções entre os conceitos objeto e agente, uma quanto a flexibilidade e outra quanto ao fluxo de controle da execução. Porém, tais distinções não apresentam relevância diante dos novos desenvolvimentos dentro do campo de sistemas orientados a objetos, particularmente da programação orientada a objetos concorrentes.

2.2.3 Arquitetura de Controle Proposta

Como mencionado anteriormente, as implementações em arquitetura de controle para robôs móveis autônomos da atualidade têm adotado abordagem híbridas em função de suas vantagens em relação às outras abordagens. Tal abordagem invariavelmente divide a arquitetura em níveis de abstração distintos, adotando estruturas hierárquicas para implementar as estratégias e funcionalidades da arquitetura e estruturas heterárquicas e/ou *subsumptions* para interfacear as primeiras estruturas com os sub-sistemas de *hardware* e *software* da arquitetura.

É dentro deste contexto que figura a proposta de arquitetura de controle apresentada neste trabalho: aliar os conceitos de sistemas multi-agentes com a utilização de metodologias de desenvolvimento orientado a objetos para o projeto e implementação de uma arquitetura de controle para robôs móveis autônomos.

Figura 2.12: Diagrama simplificado da arquitetura de controle proposta.

A ideia geral desta proposta é estruturar a arquitetura em uma comunidade de agentes, onde agentes de maior abstração têm a responsabilidade de planejar e gerenciar as ações do

sistema, enquanto agentes de menor abstração cuidam dos sub-sistemas de *hardware* e *software*. A Figura 2.12 apresenta de forma simplificada a proposta de arquitetura de controle.

Para uma descrição inicial de cada um dos agentes apresentados na Figura 2.12, pode-se utilizar a forma sugerida por Russel e Norvig (1995), denominada *PAGE* (acrônimo da frase em inglês *Percepts, Actions, Goals, Environment*). A Tabela 2.2 apresenta tal descrição.

Tabela 2.2: Agentes da arquitetura proposta e respectivas descrições *PAGE*.

Agente	Percepções	Ações	Objetivos	Ambiente
<i>Gerenciador</i>	Informações do sistema operacional e dos agentes.	Buscar informações dos agentes, planejar ações, solicitar e gerenciar a execução de ações.	Planejamento e gerenciamento do veículo.	Arquitetura de <i>software</i> do veículo.
<i>Monitor</i>	Informações do sistema operacional e dos agentes.	Disponibilizar informações do veículo para o usuário.	Monitoramento do veículo.	Arquitetura de <i>software</i> do veículo.
<i>Atuador</i>	Dados do atuador e mensagens de outros agentes.	Buscar dados do atuador, manter referência desejada.	Controle do atuador.	Arquitetura de <i>hardware</i> e <i>software</i> do veículo.
<i>Sensor</i>	Dados do sensor, mensagens de outros agentes.	Adquirir e interpretar sinais do sensor	Fornecer informações sensoriais externas.	Arquitetura de <i>hardware</i> e <i>software</i> do veículo.
<i>Radio</i>	Sinais do receptor, mensagens de outros agentes.	Adquirir sinais do receptor, traduzí-los em comandos.	Comando manual do veículo.	Arquitetura de <i>hardware</i> e <i>software</i> do veículo.

Na Figura 2.12 é possível observar a nítida divisão da arquitetura proposta, que reflete na descrição apresentada na Tabela 2.2 através da coluna de ambiente. Os agentes de maior abstração não interagem diretamente com a estrutura da *hardware* da arquitetura, sendo esta uma atribuição dos agentes de menor abstração.

Outro detalhe a ser ressaltado é a comunicação entre agentes representada por arcos. Esta é realizada de forma direta entre agentes, sem intermediários. Uma característica herdada

da abordagem heterárquica e do paralelismo proporcionado pela estrutura multi-agentes. As extremidades dos arcos são valorados, indicando relações do tipo *1 para 1*, *1 para n* ou *n para n*. Por exemplo, *1* agente *Gerenciador* comunica-se com *n* agentes *Atuadores*.

A presença de uma entidade supervisora, representada pelo agente *Gerenciador*, imprime um caráter hierárquico entre os níveis de abstração da arquitetura proposta.

Capítulo 3

Implicações de Tempo Real na Arquitetura Proposta

A expressão *de tempo real* (do inglês *realtime*) tem sido relacionada à diferentes tipos de sistemas na literatura em geral. É possível encontrar esta expressão em textos sobre os mais variados assuntos, de sistemas de controle de usinas nucleares à *softwares* para transações bancárias. Outros exemplos de temas em que a expressão *de tempo real* aparece com frequência são: monitoramento de experimentos em laboratórios, controle de plantas de processos industriais, *softwares* de controle de vôo para aviões e veículos espaciais, gerenciamento de banco de dados corporativos, sistemas de controle de tráfego urbano, além da robótica em geral, incluindo aqui robôs móveis autônomos, tema alvo deste projeto.

Neste capítulo são apresentados os conceitos relacionados a sistemas de tempo real, as características que diferenciam estes tipos de sistemas dos demais sistemas em geral e, principalmente, uma análise das implicações destas características na escolha de plataformas de *hardware* e *software* embarcado, visando implementações de arquiteturas de controle destinadas a robôs móveis autônomos.

3.1 Sistemas de Tempo Real

É comum encontrar, dentro de textos que utilizam a expressão *de tempo real*, o adjetivo *real* sendo mencionado em oposição ao tempo simulado, o que gera algumas confusões no momento em que se necessita fazer diferenciações entre sistemas de tempo real e outros sistemas computadorizados. Este fato faz com que alguns artigos que abordam o assunto busquem uma definição o mais concisa e sem ambigüidades possível do que vem a ser um sistema de tempo real. Um exemplo encontra-se em Halang (1992) e Hoogeboom e Halang (1992), que citam a norma de padronização alemã DIN (*DIN 44 300 A2: Informationsverarbeitung*, 1985) para a caracterização destes sistemas. Segundo esta norma, uma operação em tempo real é:

"o modo de operação de um sistema de computador em que os programas para processamento dos dados que chegam do ambiente externo estão permanentemente prontos, de modo que seus resultados estejam disponíveis dentro de períodos de tempo predeterminados; o tempo da chegada dos dados podem ser aleatoriamente distribuídos ou determinados *a priori* em função das diferentes aplicações."

Como a definição acima, existem muitas disponíveis na literatura (Farines, Fraga e De Oliveira, 2000; Piveropoulos, 2000; Kirner, 1997; Kurki-Suonio, 1994; Panzieri e Davoli, 1993; Stankovic, 1992), umas mais formais, outras menos completas, outras mais específicas para determinadas aplicações. Entretanto, a idéia central é que um sistema de *tempo real* é *aquele que reage às entradas externas e, de uma maneira oportuna, afeta o ambiente em que opera*.

Este capítulo pretende fornecer noções básicas a respeito dos sistemas de tempo real, buscando contextualizá-los dentro das aplicações de robôs móveis autônomos e justificar as abordagens adotadas na elaboração da arquitetura de controle proposta por este trabalho para o tratamento dos requisitos de tempo real. Inicialmente será apresentado uma discussão a respeito das características geral destes sistemas, seguida de uma seção a respeito de sistemas operacionais de tempo real. Finalizando o capítulo, apresenta-se a plataforma de *hardware* e *software* escolhida para a implementação da arquitetura de *software* proposta por esta pesquisa.

3.2 Características Gerais de Sistemas de Tempo Real

Independentemente da área de aplicação em que um sistema de tempo real é utilizado, seja no gerenciamento de um banco de dados corporativo ou no controle de um processo químico industrial, uma característica comum é a estrita ligação a ser mantido entre processos computacionais e processos físicos. Desta relação resulta algumas características comuns para sistema de tempo real, tais como: resposta temporal (*timeliness*), simultaneidade (*simultaneousness*), previsibilidade (*predictability*) e dependabilidade (*dependability*) (Hoogeboom e Halang, 1992).

Nas sub-seções seguintes cada uma destas características serão individualmente detalhadas, sendo ainda apresentadas breves análises de como elas são comumente tratadas em aplicações de robôs móveis autônomos (particularmente, veículos autônomos submarinos).

3.2.1 Resposta Temporal

A característica que melhor diferencia sistemas de tempo real de outros tipos de sistemas é a capacidade de fornecer resultados não somente corretos (*correctness*), mas também dentro de prazos de tempo (*deadlines*) mais ou menos rígidos impostos pelo ambiente. Esta característica é conhecida como resposta temporal (*timeliness*).

Juntamente com os requisitos de tempo, uma importante consideração a ser feita é o custo associado à não satisfação de um requisito de tempo específico. Esta consideração leva a distinção entre sistemas de tempo real crítico (*hard real-time systems*) e não crítico (*soft real-time systems*) (Farines et al., 2000).

Considere um sistema de tempo real em que uma dada tarefa foi executada corretamente, porém em um tempo superior ao permitido por seus requisitos de tempo. Se os resultados da execução de tal tarefa ainda podem ser usados pelo sistema, embora com restrições em relação a sua confiabilidade, diz-se que o sistema é de tempo real não crítico. Em contrapartida, para o caso em que os resultados obtidos com atraso são potencialmente prejudiciais ao processo como um todo, podendo inclusive causar acidentes, diz-se que o sistema é de tempo real crítico.

Outras formas de distinção entre sistemas de tempo real crítico e não crítico, tais como

tempo de reação típico do sistema ou capacidade de interação com o ambiente, também são encontradas na literatura (Hoogeboom e Halang, 1992). Entretanto, estas distinções não fazem nenhuma referência em relação ao custo das falhas provocadas pelo não cumprimentos dos requisitos de tempo do sistema, o que diminui a importância da distinção.

Em aplicações de veículos autônomos, a característica de resposta temporal está ligada ao tipo de estratégia utilizada na elaboração da arquitetura de controle do veículo. Como já mencionado na análise de arquiteturas para robôs móveis autônomos presente no Capítulo 2, estruturas hierárquicas não apresentam resposta temporal adequada em situações imprevistas, o que desencoraja a utilização direta desta abordagem visando missões em ambientes desconhecidos.

Já no caso de estruturas heterárquicas há uma melhoria da resposta temporal através da divisão do sistema em diversos módulos, cada qual com responsabilidades distintas. Neste caso observa-se a existência de módulos com requisitos de tempo real crítico e não crítico, mesmo que o sistema como um todo continue de tempo real crítico. Nas arquiteturas *subsumption* tais requisitos são utilizados como critério para a especificação de prioridades entre os diversos comportamentos que integram o sistema.

3.2.2 Simultaneidade

Uma importante característica observada em sistema de tempo real é a simultaneidade (*simultaneousness*). Segundo esta característica um sistema de tempo real pode receber entradas de duas ou mais fontes independentes e, por sua vez, tem que responder a cada uma delas dentro de determinados limites de tempo. Para satisfazer tais requisitos, é necessário que o sistema possua alguma forma de processamento paralelo.

Diferentes implementações são possíveis para se obter processamento paralelo em sistemas de tempo real, sendo duas principais alternativas o uso de processamento paralelo em processadores distintos e a multiprogramação sobre um único processador.

Ambas soluções citadas acima são adotadas em implementações de robôs móveis autônomos, sendo normalmente encontradas coexistindo dentro da plataforma escolhida para a implementação da arquitetura de controle do veículo.

Em geral, as unidades de processamento de maior capacidade utilizarão multiprogramação para implementar as funcionalidades centrais da arquitetura (planejamento, gerenciamento e monitoramento das tarefas), enquanto processadores menores serão encarregados de tarefas dedicadas, tais como a implementação de malhas de controle para os atuadores ou a aquisição e o tratamento de dados dos sensores do veículo.

3.2.3 Previsibilidade

Previsibilidade é a capacidade de se afirmar algo a respeito do comportamento futuro de um determinado sistema através de suas especificações de projeto (requisitos de tempo). Tal afirmação pode ser determinística ou probabilística. Para o caso da previsibilidade determinística, garante-se que todos os requisitos de tempo serão cumpridos. Já na previsibilidade probabilística, tem-se probabilidades para o cumprimento de cada um dos requisitos de tempo.

A característica de previsibilidade é um requisito que determina implicações em todos os níveis de um sistema de tempo real. Em sistemas robóticos, particularmente em aplicações de robôs móveis autônomos, em que a quantidade de sub-sistemas de diferentes naturezas é considerável, a previsibilidade é uma característica especialmente complexa de ser analisada. Ademais, o tipo de abordagem adotada para a arquitetura de controle do veículo, se deliberativa, comportamental ou híbrida, irá contribuir em mais ou menos na complexidade dessa análise.

3.2.4 Dependabilidade

A dependabilidade ¹ é o termo usado dentro do campo dos sistemas de tempo real para indicar um requisito geral de confiança de um sistema. Ela pode ser decomposta em: correção (*correctness*) e robustez (*robustness*).

¹ A referência bibliográfica De Hollanda (1993) apresenta uma discussão a respeito da tradução apropriada para o termo *dependability*. Tanto a NBR 5462 quanto a NBR ISO 9000 sugerem a tradução como dependabilidade. Já Veríssimo, em Veríssimo e De Lemos (1989), propõem a tradução como *confiança no funcionamento*.

- **Correção:** segundo este requisito, o comportamento funcional de um sistema deve satisfazer a especificação. Este é, evidentemente, um requisito geral para todos os sistemas, mas para sistemas de tempo real há uma forte relação entre correção e desempenho. Neste caso, uma incorreção pode ser em relação a resultados errados ou a resultados corretos fornecidos no tempo errado.
- **Robustez:** o conceito por trás deste requisito é que o sistema deve permanecer dentro de um estado previsível mesmo que o ambiente de operação não corresponda às especificações (isto é, as entradas fornecidas pelo ambiente não estão dentro de uma dada faixa), ou quando uma parte do sistema (um dispositivo periférico, por exemplo) falha.

Dois aspectos pelos quais a confiança de um sistema deve ser abordada são a *tolerância a falhas* e a *detecção e recuperação de erros* ². Nas arquiteturas de controle dos robôs móveis autônomos isto é feito primariamente através de algum mecanismo de verificação via *software*, tais como *watchdogs*. Em alguns veículos é possível encontrar a redundância de processadores, o que permite a implementação de mecanismos de tolerância a falhas baseados em *hardware* (Badr, Byrnes, Brutzman e Nelson, 1992). Em outros casos, em que uma estrutura distribuída de multiprocessadores é implementada, a recuperação de erros pode ser tratada com maior facilidade, desde que falhas em um determinado processador poderiam ser funcionalmente corrigidas carregando e compartilhando os módulos de *software* do processador defeituoso em outros processadores.

3.3 Sistemas Operacionais de Tempo Real

Tipicamente, aplicações de *software* são construídas a partir dos serviços oferecidos por um sistema operacional. Por conseguinte, o atendimento dos requisitos temporais exigidos pela aplicação depende não somente do código implementado, mas também da colaboração do

² Um melhor detalhamento com relação aos conceitos de tolerância a falhas e detecção e recuperação de erros pode ser encontrado nas referências bibliográficas Singh e Murugesan (1990) e Heimerdinger e Weinstock (1992).

sistema operacional no sentido de permitir o cumprimento dos requisitos inerentes aos sistemas de tempo real.

Em Panzieri e Davoli (1993), um *sistema operacional de tempo real* é definido como um sistema de *software* responsável pelo gerenciamento dos recursos de *hardware* (processadores, dispositivos de comunicação e armazenamento de dados, etc) de um sistema de tempo real.

Esta definição situa indiretamente os sistemas operacionais de tempo real dentro da definição de sistemas de tempo real apresentada anteriormente e os diferenciam dos sistemas operacionais convencionais principalmente pela previsibilidade temporal.

Sistemas operacionais convencionais são construídos com o objetivo de apresentar um bom comportamento médio, distribuindo os recursos do sistema de forma justa entre as tarefas e os usuários, não existindo uma preocupação com previsibilidade temporal. Já em sistemas operacionais de tempo real, especial atenção é dedicada ao comportamento temporal, com os serviços do sistema sendo definidos não somente em termos funcionais, mas também em termos temporais (Farines et al., 2000).

Alguns itens que são abordados sempre que se discute sistemas operacionais destinados à aplicações de tempo real são: núcleos de tempo real, execução de multi-processos, comunicação interprocessos (troca de mensagens e memória compartilhada) e algoritmos de escalonamento. Estes itens serão discutidos nas subseções seguintes.

3.3.1 Núcleos de Tempo Real

Muitas vezes os requisitos temporais da aplicação são tão rigorosos que o sistema operacional é substituído por um simples núcleo de tempo real, não incluindo serviços como sistema de gerenciamento de arquivos ou gerenciamento sofisticado de memória. Núcleos de tempo real oferecem uma funcionalidade mínima, mas são capazes de apresentar excelente comportamento temporal.

O grupo de trabalho POSIX.13³ estabeleceu quatro perfis padronizados para núcleos de tempo real: o núcleo mínimo, o tipo controlador, o tipo dedicado e o multi-propósitos. Cada

³ Os grupos de trabalho POSIXs (*Portable Operating System Interface*) foram criados pelo IEEE (*Institute of Electrical and Eletronic Engineers*), com o intuito que estabelecer padrões para sistemas operacionais baseados no UNIX. Maiores detalhes podem ser encontrados em Corrêa e Friedrich (1997).

um destes perfis engloba seu antecessor acrescentando mais serviços, maior capacidade de processamento e características de maior porte a saber:

- *Perfil Mínimo*: inclui aplicações de sistemas dedicados. Nestes sistemas não existe memória de massa (discos ou fitas magnéticas), e a aplicação se resume a um único processo com serviços de interrupções geradas pelo relógio de tempo real ou por dispositivos externos.
- *Perfil Controlador*: acrescenta dispositivos de entrada e saída estruturados (padronizados), sistemas de arquivos e armazenamento de dados e códigos em discos virtuais em memória RAM.
- *Perfil Dedicado*: adiciona características de multi-processamento e de gerenciamento de acesso a memória compartilhada.
- *Perfil Multi-Propósitos*: consiste da totalidade das expansões POSIX.1 e POSIX.4, incluindo capacidades de interligação em rede, sistemas de gerenciamento de janelas e sistemas de armazenamento de massa de alta velocidade.

3.3.2 Multi-Processos

Aplicações de tempo real são usualmente organizadas na forma de várias *threads* ou processos concorrentes. Logo, um requisito básico para os sistemas operacionais de tempo real é oferecer suporte para processos e *threads*.

Processos ou tarefas são abstrações que incluem: um espaço de endereçamento próprio (possivelmente compartilhado), um conjunto de arquivos abertos, um conjunto de direitos de acesso, um contexto de execução formado pelos valores contidos nos registradores do processador, entre outros atributos.

Threads são tarefas simples que possuem como únicos atributos aqueles associados com o contexto de execução. O chaveamento entre duas *threads* de uma mesma tarefa é muito mais rápido que o chaveamento entre duas tarefas.

Algumas vezes é encontrado na literatura (Farines et al., 2000; Piveropoulos, 2000; Panzneri e Davoli, 1993) a classificação dos processos (tarefas) em um sistema de tempo real como

periódicos, aperiódicos e esporádicos. Processos periódicos são aqueles que são executados em intervalos regulares de tempo. Estes processos, usados geralmente em aplicações como processamento de sinais e controle, são tipicamente caracterizados por requisitos de tempo críticos (*hard deadlines*). Processos aperiódicos são aqueles os quais a execução no tempo não pode ser antecipadamente determinada, sendo iniciados pela ocorrência de algum evento externo ou interno, como por exemplo comandos do operador. Estes processos são usualmente caracterizados por requisitos de tempo não críticos (*soft deadlines*). Para o caso em que tais requisitos são críticos (ocorrência de falhas no sistema, por exemplo), o termo processos esporádicos é utilizado.

3.3.3 Comunicação Interprocessos

Uma aplicação de tempo real é tipicamente um programa concorrente, formado por processos e/ou *threads* que se comunicam e se sincronizam. Existem duas grandes classes de soluções para programação concorrente: troca de mensagens e memória compartilhada.

Troca de mensagens é uma forma síncrona de comunicação entre processos. Tipicamente, este tipo de comunicação consiste de um processo enviando uma mensagem à um outro processo, aguardando em seguida alguma prova de seu recebimento, potencialmente uma mensagem de retorno.

Outra forma de comunicação através de mensagens são os *proxies*. Eles são formas especiais de mensagens, especialmente adequados para a notificação de eventos. Neste caso, não há a necessidade de confirmação de recebimento.

Uma outra forma ainda de comunicação através de mensagens são os *sinais*, uma forma tradicional de comunicação assíncrona interprocessos utilizada há vários anos em muitos sistemas operacionais. A norma POSIX especifica um rico conjunto de sinais padrão destinados à comunicação interprocessos.

Através da alocação de regiões de memória para processos distintos é possível a utilização do recurso de memória compartilhada, um segundo tipo de solução para a comunicação interprocessos. Dentro deste tipo de solução estão as variáveis compartilhadas, os *pipes*⁴, as filas

⁴ *Pipes* são regiões de memória destinadas à transição de dados entre dois ou mais processos cooperativos.

de mensagens, dentre outras formas de entrada e saída de dados.

Outra forma de comunicação interprocessos que pode ser considerada como compartilhamento de memória são os *semáforos*. Seu conceito é simples e muito similar a um *flag*. Antes de um processo tentar acessar um recurso, ele primeiro verifica se o semáforo associado com o recurso está livre. Em caso afirmativo, o processo vai em frente com o acesso. Em caso contrário, o processo é bloqueado até que o recurso esteja livre.

3.3.4 Algoritmos de Escalonamento

A correta programação da comunicação e sincronização dos processos através dos mecanismos citados no item anterior garante o comportamento funcional adequado do sistema. Entretanto, não garante o seu comportamento temporal, que é tarefa dos algoritmos de escalonamento. *Escalonamento* (*scheduling*) é a ordem com que são alocados os recursos entre os processos do sistema, ordem esta que é determinada pelos algoritmos de escalonamento.

Tipicamente qualquer sistema operacional dispõe de escalonamento baseado em prioridades. Entretanto, a maioria dos sistemas operacionais de propósito geral inclui mecanismos que reduzem automaticamente a prioridade na medida que um determinado processo consome tempo de processador (*aging*), com o intuito de diminuir o tempo médio de resposta no sistema. Porém, em sistemas operacionais de tempo real isto não é desejado.

A escolha do algoritmo de escalonamento adequado para uma aplicação de tempo real específica deve levar em conta vários fatores, tipicamente os relacionados com os requisitos da aplicação. Por exemplo, um processo de tempo real podem ser preemptivo (*preemptable*) ou não preemptivo (*non-preemptable*). Um processo preemptivo é aquele em que sua execução pode ser interrompida por outro processo e retomada mais tarde. Já um processo não preemptivo deve executar até sua finalização, sem interrupções. Assim, tanto algoritmos preemptivos quanto não preemptivos devem ser propostos.

Os algoritmos de escalonamento também podem ser classificados como estáticos e dinâmicos (Farines et al., 2000; Panzieri e Davoli, 1993). Um algoritmo de escalonamento estático é

Eles são normalmente usados quando dois processos desejam executar em paralelo com movimentação de dados entre eles em uma única direção.

aquele em que a ordem de execução dos processos é computada *a priori*. Isto requer um prévio conhecimento das características dos processos. Por outro lado, um algoritmo de escalonamento dinâmico determina o escalonamento em tempo de execução. Assim, o escalonamento estático é caracterizado por um baixo custo de processamento. Em contrapartida, ele é inflexível e requer uma completa previsibilidade do ambiente no qual é empregado. Em vez disto, o escalonamento dinâmico exige alto custo de processamento, podendo porém adaptar-se às mudanças no ambiente.

Fornecedores de sistemas operacionais de tempo real costumam divulgar métricas para mostrar como o seu sistema suporta aplicações de tempo real. Estas métricas refletem a prática da construção de aplicações de tempo real ligadas à desempenho. Uma métrica muito utilizada é o tempo para chaveamento entre dois processos. Este tempo inclui salvar os registradores do processo que está executando e carregar os registradores com os valores do novo processo. Não inclui o tempo necessário para decidir qual processo vai executar, o que depende do algoritmo de escalonamento.

Em função de sua importância e da diversidade possível de abordagens, o tema algoritmos de escalonamento possui um grande número de pesquisadores dedicados ao assunto, gerando uma vasta literatura (Panzieri e Davoli, 1993; Audsley e Burns, 1990; Ramamritham e Stankovic, 1994; Stankovic, Spuri, Di Natale e Buttazzo, 1995).

3.4 Escolha da Plataforma de *Hardware* e *Software*

Uma das características marcantes em aplicações de robôs móveis autônomos é a necessidade de execução de tarefas em tempo real, especificamente na aquisição de dados sensoriais e utilização dos mesmos no controle e manutenção de sua trajetória. Conseqüentemente, a estrutura escolhida como plataforma para a implementação da arquitetura de controle do veículo deverá ser adequada às características dos sistemas de tempo real detalhadas na seção 3.2.

Pensando neste aspecto e aproveitando as experiências relatadas na literatura especializada, concebeu-se a estrutura de *hardware* e *software* que será apresentada na duas subseções a seguir.

3.4.1 Dispositivos de Computação Embarcada

A escolha dos equipamentos de bordo a serem utilizados em uma aplicação de robôs móveis autônomos está diretamente ligada à abordagem que será utilizada na elaboração da arquitetura de controle do mesmo. Em Valavanis et al. (1997) é citado duas possibilidades de abordagem, uma primeira decorrente do projeto de sistemas baseados em microcontroladores e uma segunda do projeto de sistemas de controle baseados em microcomputadores embarcados.

No entanto, o projeto de sistemas de controle baseados em microcomputadores embarcados apresenta vantagens significativas em relação à segunda abordagem, pois minimiza riscos, custos, assim como o tempo de desenvolvimento e provê uma plataforma pronta para execução de aplicações específicas de *software* (Valavanis et al., 1997). Tais vantagens incentivaram a escolha desta abordagem para a plataforma de *hardware* da arquitetura de controle proposta.

A divisão em níveis de abstração da arquitetura proposta na seção 2.2.3 também reflete na plataforma de *hardware* aqui apresentada. Elevar o nível de abstração em uma aplicação de *software* acarreta na elevação da carga computacional requerida para sua execução. Assim, atividades como planejamento, gerenciamento e monitoramento das ações em um veículo autônomo serão invariavelmente as de maior consumo computacional da arquitetura de controle do veículo.

Optou-se então pela utilização de uma estrutura distribuída, com um microcomputador de bordo com barramento do tipo PC/104 sendo utilizado essencialmente para as atividades mais nobres da arquitetura, agregado a servo-sistemas responsáveis pelo controle dos atuadores do sistema. Segundo esta idéia, o microcomputador de bordo PC/104 fornece informações de referência para os servo-sistemas, enquanto estes retornam informações sobre o estado dos atuadores do veículo. A Figura 3.1 ilustra esta idéia.

Os microcomputadores com barramento do tipo PC/104 são equipamentos modulares, robustos e de dimensões reduzidas. Eles são especialmente desenvolvido para aplicações embarcadas. Em geral, os fabricantes destes equipamentos fornecem configurações padrões, além de módulos avulsos para expansão de sistemas (ver Figura 3.2).

A estrutura mostrada na Figura 3.1 permite um melhor enfrentamento do requisito de simultaneidade da aplicação de tempo real, por possibilitar o paralelismo de processamento para a execução das diversas funcionalidades do sistema. Uma melhor descrição desta estrutura

Figura 3.1: Arquitetura de *hardware* escolhida.

de *hardware* é apresentado no Capítulo 5.

3.4.2 Sistema Operacional de *Software*

Na maioria das vezes em que os projetistas de um sistema precisam fazer uma escolha entre diferentes soluções a serem utilizadas em um projeto, uma comparação entre as diversas opções disponíveis comercialmente é freqüentemente utilizada. No caso da escolha de um sistema operacional a ser utilizado como plataforma de *software* em uma aplicação de tempo real isto não é diferente. Entretanto, neste caso específico, uma comparação adequada geralmente não é possível.

A dificuldade em se estabelecer comparações entre sistemas operacionais de tempo real reside em vários fatores, muitos deles fora do alcance dos projetistas do sistema. Em Farines et al.

Figura 3.2: Equipamentos de computação embarcada baseados no barramento PC/104.

(2000) é possível encontrar uma boa lista destes fatores, dos quais cita-se como mais relevantes os seguintes: as diferentes abordagens de escalonamento existentes; as diferentes métricas publicadas pelos fabricantes de sistemas operacionais de tempo real, que são obtidas em plataformas diferentes e muitas vezes não são completas; os detalhes internos sobre o núcleo dos sistemas normalmente não estão disponíveis; cada sistema possui diferentes níveis de conformidade com os padrões; a política de licenciamento e o custo associado variam muito.

Diante da gama de fatores mencionados anteriormente, fatores estes que dificultam uma comparação entre diferentes tipos de sistemas operacionais de tempo real, os requisitos da aplicação alvo acabam sendo determinantes no momento em que se necessita realizar uma escolha. Assim, um sistema operacional que apresenta ótimos recursos de multi-processamento muitas vezes não será a melhor escolha, por exemplo, para uma aplicação em que o requisito maior é a baixa latência para o atendimento a interrupções. Este fato torna ainda mais importante a necessidade de se ter, com clareza, os requisitos da aplicação.

Para o caso da aplicação abordada neste projeto, uma arquitetura de controle destinada a robôs móveis autônomos, o sistema operacional de *software* a ser utilizado deve evidentemente satisfazer os requisitos de toda aplicação de tempo real. Porém, particularmente três destes requisitos podem ser destacados como segue:

- **Multi-Processamento:** este é um dos requisitos básicos das aplicações robóticas em geral. Especificamente tratando de robôs móveis autônomos, via de regra tem-se, simultaneamente à aquisição de dados de vários dispositivos de entrada (sensores), a necessidade

do planejamento e gerenciamento das ações do veículo, assim como do monitoramento e comando a distância;

- **Comunicação Inter-Processos:** como consequência do multi-processamento e da abordagem adotada para sua implementação neste trabalho, uma comunidade de agentes que possuem responsabilidades específicas dentro da arquitetura de controle (ver seção 2.2.3), alguma forma de comunicação inter-processos torna-se necessária, neste caso implementada através de troca de mensagens;
- **Comunicação via Rede:** este requisito é uma das soluções possíveis para a realização do monitoramento e comando a distância. Diante dos importantes avanços da tecnologia de redes e protocolos de comunicação, este tipo de solução tem trazido grandes contribuições para o campo dos sistemas distribuídos.

De posse destes requisitos e pensando-se na estrutura de *hardware* mostrada na Figura 3.1, chegou-se a escolha de um sistema operacional com perfil multi-propósito, no caso o sistema operacional de tempo real QNX, para a execução no microcomputador de bordo PC/104. Tal sistema operacional é melhor detalhado na seguinte.

Sistema Operacional de Tempo Real QNX

O sistema operacional QNX foi o primeiro produto da *QNX Software Systems Limited*, empresa fundada em 1980. Inicialmente, a empresa chamava-se QSSL (*Quantum Software Systems*) e seu produto chamava-se QUNIX (*Quick UNIX*). Dois anos após a fundação da empresa, uma versão inicial do QNX foi lançada no mercado como o primeiro sistema operacional multi-usuários e multi-tarefas para microcomputadores IBM. Em 1984 o sistema já tinha integrado recursos de LAN e podia ser executado em máquinas da classe AT no modo protegido. Por volta de 1990, foi lançado a versão QNX 4, suportando operações em 32 *bits* e o padrão POSIX. Esta é a versão do QNX que será abordada nesta seção.

O sistema operacional QNX foi idealizado para aplicações de tempo real. Ele provê recursos de multi-processamento, escalonamento preemptivo baseado em prioridades, além de rápido chaveamento entre processos, itens essenciais em sistemas de tempo real.

Outra característica interessante do QNX é a flexibilidade, possibilitando ao usuário montar um sistema operacional personalizado, disponibilizando somente os recursos que este julgar necessários às suas aplicações.

Além do recurso de mensagens, o QNX provê todos os recursos de comunicação inter-processos apresentados a seção 3.3.3, inclusive o recurso de memória compartilhada com acesso exclusivo garantido através de semáforos. O QNX suporta apenas semáforos baseados em memória, apesar de não ser o único tipo definido pela POSIX ⁵.

Figura 3.3: O *microkernel* QNX coordenando os gerenciadores do sistema.

O QNX, nesta sua versão 4, consiste de um pequeno núcleo (*microkernel*) ligado à um grupo de processos atuando em cooperação (ver Figura 3.3). O *Microkernel* QNX é muito pequeno e dedicado a apenas duas funções essenciais:

- **Manipulação de Mensagens:** o *microkernel* manipula todas as mensagens entre os processos através de todo o sistema;
- **Escalaonamento:** o escalonador é uma parte do *microkernel* e é ativado sempre que um processo muda de estado como resultado de uma mensagem ou interrupção. Todos os serviços do QNX, exceto aquele providos pelo *microkernel*, são dirigidos por processos QNX padrões.

Em relação ao escalonamento, o QNX provê três métodos distintos: escalonamento FIFO (*First In, First Out*), escalonamento *Round-Robin* e escalonamento adaptativo. Cada processo no sistema pode executar utilizando-se de qualquer um destes métodos ⁶.

⁵ A POSIX define também semáforos do tipo nomeados (*named semaphores*). Já a extensão POSIX.1b define as variáveis *mutexes* e condicionais, que também não são disponíveis no QNX (Kolnick, 1998).

⁶ Maiores detalhes a respeito dos métodos de escalonamento presentes no sistema operacional QNX podem ser encontrados em (QNX, 1997).

Como a maioria dos serviços do sistema operacional é provida através de processos padrões QNX, é uma questão muito simples aumentar o próprio sistema operacional: o desenvolvedor pode escrever novos programas para prover novos serviços.

Para aplicações de controle embarcado, a modularidade do sistema operacional QNX permite que o desenvolvedor elimine processos desnecessários do sistema. Com o acréscimo de um pequeno módulo de rede QNX, um sistema embarcado pode tornar-se uma extensão de rede transparente de uma ambiente QNX maior para aplicações distribuídas, inicializada por meio da ROM ou da rede.

Através do *microkernel*, o QNX pode transformar uma rede de computadores em uma única máquina lógica para as aplicações, indiferentemente de quantos computadores estejam ligados pela rede. Aplicações desenvolvidas por esta máquina lógica única serão executadas sem alterações, não importando o número de computadores que foram escolhidos para servir as tarefas da aplicação.

Tal flexibilidade é possível porque o QNX permite que aplicações sejam projetados como um conjunto de processos cooperando-se e comunicando-se em uma única máquina. Estes processos podem ser configurados para executarem ao longo da rede, enquanto o QNX provê mensagens de rede transparentes entre estes processos. A rede QNX permite que qualquer processo use qualquer recurso em qualquer computador dentro da mesma. Múltiplos *links* redundantes entre os nós da rede proporcionam proteção contra falhas da rede (QNX, 1997).

As vantagens citadas acima incentivaram a escolha do sistema operacional de tempo real QNX como plataforma *software* para o desenvolvimento da arquitetura de controle proposta neste projeto.

Capítulo 4

Proposta de Arquitetura de *Software*

Os sistemas mecatrônicos em geral situam-se, sem nenhuma dúvida, entre os mais avançados e complexos sistemas da engenharia moderna. Particularmente é nos robôs móveis autônomos, tema central deste projeto, que esta complexidade torna-se mais aparente.

Tipicamente um robô móvel autônomo possui um variado conjunto de sensores para a percepção e coleta de informações do mundo externo e um outro conjunto, não muito menos variado, de atuadores responsáveis pela interação do sistema com o ambiente externo.

Tal interação é usualmente governada por um sistema de *software*, freqüentemente denominada de arquitetura de controle do veículo. Em função de suas características de intenso processamento computacional, grande quantidade de dados sensoriais e necessidade de execução em tempo real, o projeto da arquitetura de controle é a parte mais crítica do desenvolvimento de veículos autônomos (Valavanis et al., 1997) e que, por isto mesmo, deve demandar maior esforço de planejamento, exigindo mais que qualquer outra parte do sistema de um método para seu desenvolvimento.

O recurso da modelagem é um dos métodos de projeto mais difundidos dentro da engenharia, pois freqüentemente não é possível compreender um problema dentro da engenharia em sua totalidade. Assim, um modelo é essencialmente uma simplificação da realidade. Eles são construídos com a finalidade de se compreender melhor um sistema o qual tem-se interesse em desenvolver.

Em linhas gerais, a modelagem de um sistema deve contemplar as seguintes necessidades:

- possuir alguma forma de especificação do sistema - uma solução para determinado proble-

ma deve ser capaz de satisfazer todas as suas necessidades, as quais são determinadas através da busca dos requisitos do sistema junto aos interessados em seu desenvolvimento (clientes, usuários finais, etc...);

- possibilitar o desenvolvimento integrado por uma equipe - geralmente sistemas complexos são desenvolvidos por uma equipe, usualmente composta por especialistas em mecânica, eletrônica, computação. Tal equipe é responsável pela construção de partes distintas do sistema, mas que interagem entre si, o que requer um desenvolvimento cooperativo de forma sintonizada;
- proporcionar várias visões do sistema - nenhum modelo único é suficiente para a modelagem de um sistema de certo grau de complexidade. Qualquer sistema não-trivial será melhor investigado por meio de um pequeno conjunto de modelos quase independentes. Cada modelo poderá ser expresso em diferentes níveis de abstração, em decorrência das diversidades de interesses requeridos do modelo;
- documentar todo o projeto - a documentação inteligível de todo o sistema é particularmente importante, pois tal documentação será utilizada sob diferentes visões de interesse, por diferentes especialistas da equipe de desenvolvimento, cada qual preocupado com diferentes aspectos do projeto.

De forma a atender as necessidades acima, surgiu dentro do campo da engenharia de *software* diferentes métodos de modelagem, cada qual com sua linguagem própria, com o intuito de organizar e padronizar as atividades de desenvolvimento de sistemas. Inicialmente, uma visão tradicional adotava a perspectiva de um algoritmo, em que o bloco de construção do *software* era o procedimento ou a função. Entretanto, tal visão foi substituída por uma perspectiva orientada a objetos, em que o bloco de construção é o objeto ou a classe.

Com o passar dos anos, a experiência tem provado a conveniência da utilização de métodos orientados a objetos para a construção de sistemas em todos os tipos de domínios de problemas, abrangendo todos os graus de tamanho e complexidade. Dentro do campo de sistemas mecatrônicos, Muir e Horner (1998), López-Orozco, De la Cruz, Domínguez, Besada e Polo (1998), O'Reilly (2000) e Loffler, Chitrakaran e Dawson (2001) trazem exemplos da utilização desta nova perspectiva de modelagem.

4.1 Métodos Orientados a Objetos

Grande parte das linguagens de modelagem orientada a objetos surgiu entre a metade da década de 1970 e o final da década de 1980, à medida que o pessoal envolvido com metodologia, diante de aplicações cada vez mais complexas, começou a experimentar métodos alternativos de análise e projeto.

Segundo Booch, Rumbaugh e Jacobson (1999), a quantidade de métodos orientados a objetos aumentou de pouco mais de 10 para mais de 50 durante o período de 1989 a 1994, crescimento em parte alimentado pela constatação de que muitos usuários tiveram dificuldade para encontrar uma linguagem de modelagem capaz de atender inteiramente às suas necessidades.

Com o passar dos anos e o acúmulo de experiência, novas gerações destes métodos começaram a surgir, com um claro destaque de alguns como o *Booch* (Booch, 1993), o *Object-Oriented Software Engineering* de Jacobson (Jacobson, Christerson, Jonsson e Overgaard, 1992) e o *Object Modeling Technique - OMT* de Rumbaugh (Rumbaugh, Blaha, Premerlani, Eddy e Lorenson, 1991). Tais métodos eram completos, porém cada qual com suas respectivas vantagens e desvantagens. Em Pronk e Tercero (1994) encontra-se um estudo comparativo destes métodos, além de outros da mesma geração, apontando suas vantagens e desvantagens.

Uma massa crítica de idéias começou a tomar forma por volta da metade da década de 1990, quando Grady Booch, Ivar Jacobson e James Rumbaugh começaram a adotar, cada qual em seu método, idéias provenientes dos outros métodos e que vinham sendo reconhecidos mundialmente como os principais métodos orientados a objetos.

Os esforços para a criação da *UML* (*Unified Modeling Language*) se iniciaram oficialmente em outubro de 1994, quando Rumbaugh se juntou a Booch na empresa *Rational* e elaboraram o esboço da versão 0.8 do Método Unificado (como então era chamado) que foi lançado em outubro de 1995. Mais ou menos na mesma época, foi a vez de Jacobson se associar à *Rational*, o que resultou no lançamento dos documentos da versão 0.9 da *UML* em junho de 1996. Com a aceitação da comunidade de engenharia de *software* em geral, estabeleceu-se um consórcio de *UML*, surgindo daí a definição da *UML 1.0*. Tal definição foi oferecida para padronização ao *OMG* (*Object Management Group*). Em novembro de 1997, a versão 1.1 revisada da *UML* foi adotada pelo *OMG*, órgão este que ficou responsável pela manutenção da *UML*.

4.2 *UML*, Linguagem Unificada de Modelagem

A *UML* é uma linguagem para especificação, construção, visualização e documentação de sistemas orientados a objeto. Tal definição vai de encontro com as características tidas como necessárias, na seção 4, para uma boa modelagem de um sistema.

A *UML* pode especificar um sistema, pois possibilita a criação de modelos relativamente precisos e completos que, com o desenvolvimento da linguagem, tendem a minimizar ambigüidades.

A *UML* pode ser utilizada para a construção de um sistema, pois, apesar de não ser uma linguagem de programação, seus modelos podem ser diretamente conectados a várias linguagens de programação.

Através da *UML* pode-se visualizar um sistema, pois sendo uma linguagem gráfica, em que para cada símbolo de sua notação existe uma semântica bem definida, um desenvolvedor poderá usar a *UML* para escrever seu modelo e qualquer outro desenvolvedor será capaz de interpretá-lo sem ambigüidades.

A *UML* é uma linguagem para documentação, uma vez que proporciona além da descrição da arquitetura do sistema, a expressão de requisitos e a realização de testes.

Como é de se supor pelo breve histórico apresentado na seção anterior, a notação da *UML* é uma união de sintaxe gráfica de vários métodos, com certo número de símbolos removidos (porque são confusos, supérfluos ou pouco usados) e com outros símbolos adicionados. O resultado é uma única, comum e ampla linguagem de modelagem utilizável por desenvolvedores de sistemas orientado a objetos.

A Figura 4.1, adaptada de Rumbaugh, Jacobson e Booch (1999), mostra os diversos métodos com suas respectivas contribuições para a formação da *UML*. O método *OOSE* (Jacobson et al., 1992) é uma abordagem orientada a *use case* que fornece excelente suporte para especificação de requisitos. As técnicas *OOSE* utilizadas pela *UML* foram o Diagrama de *Use Cases* e *Packages*. O *OMT* (Rumbaugh et al., 1991), especialmente expressivo para a análise de sistemas de informação, contribuiu com a representação do Diagrama de Classes e Diagrama de Estados. Do *BOOCH* (Booch, 1993), a *UML* utilizou a idéia de Diagrama de Estados, Diagrama de Classes, Diagrama de Objetos (que originou o Diagrama de Colaboração), Diagrama de Processos (que

originou o Diagrama de Implantação) e o Diagrama de Módulos (que originou o Diagrama de Componentes). O método *Fusion* também colaborou com o Grafo de Interação de Objetos. O *Statecharts* de *Harel*, contribuiu para a criação do Diagrama de Atividades.

Figura 4.1: Origem do Método UML.

Para compreender a *UML* é preciso formar um modelo conceitual da linguagem e isso pressupõe aprender os seus três elementos principais: os blocos de construção básicos, as regras que determinam como esses blocos poderão ser combinados e alguns mecanismos comuns aplicados na *UML*. Um melhor detalhamento a respeito destes três elementos principais da linguagem podem ser encontrados em Booch et al. (1999).

Dentre os blocos de construção básicos da *UML*, encontram-se os já mencionados diagramas, que consistem de uma apresentação gráfica de um conjunto de elementos, geralmente representada como gráficos de vértices (classes, interfaces, componentes, etc) e arcos (relacionamentos).

Diagramas são usados para visualizar um determinado sistema sob diferentes perspectivas. Uma vez que nenhum sistema complexo pode ser compreendido em sua totalidade a partir de uma única perspectiva, os métodos de modelagem definem um conjunto de diagramas distintos que permitem dirigir o foco da modelagem para aspectos diferentes de um sistema de maneira independente. No caso da *UML*, é definido um total de nove tipos de diagramas.

Tais diagramas podem ser divididos em dois grupos: os diagramas responsáveis pela captação dos aspectos estáticos (estruturais) de um sistema e os diagramas responsáveis pela captação

dos aspectos dinâmicos (comportamentais).

Diagramas Estruturais

Os aspectos estáticos de um sistema podem ser considerados como uma representação de seu esqueleto e estrutura relativamente estáveis. Assim, em um sistema de *software*, os diagramas estruturais conterão classes, interfaces, colaborações, componentes e nós.

Os diagramas estruturais da *UML* são organizados em função dos principais grupos de itens encontrados na modelagem de um sistema, a saber:

- **Diagrama de Classes** - mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos. São diagramas sempre encontrados em modelagens de sistemas orientados a objetos.
- **Diagrama de Objetos** - mostra um conjunto de objetos e seus relacionamentos, ilustrando as estruturas de dados, registros estáticos de instâncias dos itens encontrados nos diagramas de classes.
- **Diagrama de Componentes** - mostra um conjunto de componentes e seus relacionamentos. Estes diagramas estão relacionados aos diagramas de classes, pois tipicamente um componente mapeia uma ou mais classes, interfaces ou colaborações.
- **Diagrama de Implantação** - mostra um conjunto de nós e seus relacionamentos, ilustrando a visão estática da implantação de uma arquitetura.

Diagramas Comportamentais

Os aspectos dinâmicos de um sistema podem ser considerados como uma representação de suas partes que sofrem alterações. Em um sistema de *software* correspondem ao fluxo de mensagens ao longo do tempo e a movimentação física de componentes em uma rede.

Dentro da *UML*, os diagramas comportamentais são organizados a partir das principais maneiras disponíveis para se fazer a modelagem da dinâmica de um sistema, a saber:

- **Diagrama de Caso de Uso** - mostra um conjunto de casos de uso, atores e seus relacionamentos, ilustrando a visão estática do caso de uso de um sistema. São importantes principalmente para a organização e modelagem dos comportamentos de um sistema.
- **Diagrama de Seqüência** - mostra um conjunto de objetos e as mensagens enviadas e recebidas por estes objetos, dando ênfase à ordenação temporal das mensagens.
- **Diagrama de Colaboração** - mostra um conjunto de objetos, as conexões existentes entre esses objetos e as mensagens enviadas e recebidas pelos objetos, dando ênfase à organização estrutural dos objetos de enviam e recebem mensagens.
- **Diagrama de Gráfico de Estados** - mostra uma máquina de estados, que consiste de estados, transições, eventos e atividades, ilustrando a visão dinâmica de um sistema. São importantes para se fazer a modelagem do comportamento de uma interface, classe ou colaboração.
- **Diagrama de Atividades** - mostra um conjunto de atividades, o fluxo seqüencial ou ramificado de uma atividade para outra e os objetos que realizam ou sofrem ações. São importantes para se fazer a modelagem da função de um sistema.

4.3 *UML* de Tempo Real

A *UML* é uma linguagem de modelagem de finalidade geral que provê uma extensa base conceitual para um amplo espectro de aplicações, espectro este que abrange inclusive os sistemas de tempo real caracterizados anteriormente no Capítulo 3 deste trabalho.

Já em sua versão 1.1, versão que foi adotada como padrão pelo *OMG* (*Object Management Group*), a *UML* apresentava algumas extensões destinadas a aplicações em áreas específicas de conhecimento, dentre elas sistemas de tempo real. Tais extensões surgiram em grande parte de contribuições advindas de métodos previamente existentes destinados à modelagem orientada a objetos em campo específicos. No caso de sistemas de tempo real, por exemplo, cita-se o *ROOM* (Selic, Gullekson e Ward, 1994) e o *OCTOPUS* (Awad, Kuusela e Ziegler, 1996).

Entretanto, o próprio *Bran Selic*, criador do *ROOM*, comenta que, embora existam várias extensões chamadas de tempo real que acrescentam novos conceitos à *UML*, a experiência tem provado que isto não é necessário. A modelagem de tempo real pode ser inteiramente realizada através da especialização apropriada da base conceitual já existente na *UML* (Selic, 1999).

Não existem diagramas especiais dentro da *UML* para expressar características de tempo real em um sistema. Ao invés disso, informações da modelagem de tempo-real podem ser inseridas a *UML*, especialmente em diagramas comportamentais.

A referência bibliográfica Douglass (2000) apresenta as definições presentes na *UML* que podem especificar as informações relevantes de um sistema de tempo real, tais como:

- Tempo: especificações de tempo e restrições temporais são melhor definidas em diagramas de seqüência, nos quais o tempo é o aspecto mais relevante. Entretanto, os diagramas de atividades e grafo de estados também podem conter restrições temporais em seus arcos. Notas também podem ser utilizadas para indicar requisitos temporais.
- Concorrência: é descrita através de classes ativas nos diagramas de classes. As classes ativas são classes cujos objetos têm um ou mais processos ou *threads*, diferindo de uma classe comum pelo fato de que seus objetos representam elementos cujo comportamento é concorrente com o de outros elementos. Os diagramas de grafo de estados tratam a concorrência através do recurso de subestados concorrentes. Já nos diagramas de atividades, a concorrência por ser representada através da bifurcação do fluxo de controle, gerando fluxos concorrentes.
- Eventos Assíncronos: a *UML* suporta mensagens assíncronas enviadas entre objetos ou inicializadas por eventos assíncronos no ambiente.
- Sincronização: é descrita de forma trivial em diagramas de atividades (através da união de fluxos de controle concorrentes) e grafo de estados (através de restrições nas transições de estados). Pode também ser descrita como propriedades das classes como operações ou como classes/estereótipos ¹ que definem mecanismos como semáforos, monitores e regiões críticas.

¹ Estereótipos são recursos utilizados para a ampliação do vocabulário da *UML*, permitindo a criação de novos tipos de blocos de construção que são derivados dos já existentes, mas específicos a determinados problemas.

- Distribuição: os diversos processos de um sistema distribuído são descritos nos diagramas de implantação, que representa os vários nós do sistema, seus respectivos processos em execução, além dos dispositivos utilizados.

Para a modelagem da arquitetura de *software* proposta, pretende-se sempre que possível priorizar a utilização dos recursos da base conceitual da *UML*, deixando a utilização de extensões apenas em casos onde a não utilização destas prejudicar o entendimento de aspectos relevantes da modelagem.

4.4 *UML* e Agentes

No passado, as pesquisas dentro do campo da engenharia de *software* orientado a agentes não obtiveram grande aceitação pela indústria de desenvolvimento de *software*. O grande motivo, segundo Bauer, Müller e Odell (2001), foi a falta de tato com que foi tratado a questão por parte dos pesquisadores dentro do meio acadêmico.

Buscando mudar tal situação, estabeleceu-se uma cooperação entre a *FIPA*² e o *OMG*, que permitiu o estabelecimento de relações entre a tecnologia de agentes e padrões do desenvolvimento de *software* orientado a objetos (no caso, a *UML*). Como primeiro resultado desta cooperação foi apresentada uma proposta de extensão para a *UML*, a *Agent UML* (Odell, Parunak e Bauer, 2000), que atualmente encontra-se em fase de discussão junto a um dos grupos de trabalho da *OMG*, o *Agent PSIG*³ (*Agent Platform Special Interest Group*).

Os defensores da *AUML* alegam que a *UML* ainda não provê um base conceitual suficiente para a modelagem de agentes e sistemas baseados em agentes. Isto, segundo Odell et al. (2000), basicamente por duas razões: primeiramente porque, comparado a objetos, agentes possuem mais fortemente o conceito de autonomia pois podem tomar a iniciativa e ter controle sobre como são processadas as requisições externas; em segundo, agentes não agem apenas isoladamente, mas em cooperação ou coordenação com outros agentes.

² A *FIPA*, *Foundation of Intelligent Physical Agents* foi criada para promover a especificação de padrões no desenvolvimento de sistemas multi-agentes voltados para fins comerciais e industriais. Maiores informações podem ser encontradas em <http://www.fipa.org>.

³ Maiores detalhes podem ser encontrados em <http://www.omg.com>.

Em virtude de não haver de fato uma extensão da *UML* para a modelagem de sistemas multi-agentes e sim propostas em discussão junto aos organismos responsáveis pela padronização, optou-se pela utilização unicamente das idéias básicas contidas nos artigos que propõem a *AUML*. De forma geral, tais artigos (Odell et al., 2000; Bauer, 2001) apresentam o conceito de agente como uma extensão de objetos ativos, exibindo tanto autonomia dinâmica (a habilidade de iniciar uma ação sem requisições externas) quanto determinística (a habilidade de recusar ou modificar uma requisição externa). E é neste sentido que a noção de agente será utilizada na modelagem da arquitetura de *software* proposta neste trabalho.

4.5 Modelagem da Arquitetura de Controle Proposta

A vasta gama de diagramas apresentada na seção 4.2 confere à *UML* uma base conceitual mais completa em relação aos métodos que contribuíram para sua formação. Entretanto, este mesmo ponto positivo da *UML* pode gerar, em contrapartida, um complicante a mais para o ato de modelar: o grande número de diagramas disponíveis pode confundir o projetista. Assim, deve-se ter em mente que não é necessário a construção de todos os diagramas disponíveis na *UML* e sim apenas os realmente necessários para se transmitir os aspectos de interesse em uma modelagem. Em verdade, a escolha dos diagramas a serem construídos é um ponto chave no ato de modelar uma aplicação através da *UML*.

Para o caso da modelagem da arquitetura de *software* proposta neste trabalho, optou-se pela elaboração de três grupos distintos de diagramas, buscando captar diferentes aspectos de interesse. Inicialmente apresenta-se uma visão estrutural da arquitetura, contendo diagramas de classes adaptados para incluírem a noção de agentes, noção esta central para a proposta de arquitetura abordada. Em um segundo momento, procura-se modelar o comportamento interno de cada agente através da utilização de diagramas de gráfico de estados e de atividades. Finalizando, apresenta-se as interações entre agentes fazendo-se uso de diagramas de colaboração e de seqüência.

4.5.1 Diagramas de Classes

Um diagrama de classes mostra essencialmente as classes das quais é composto o sistema modelado, juntamente com as relações estruturais existentes entre estas classes. Por isto, estes diagramas são centrais em uma modelagem orientada a objetos.

A modelagem da proposta de arquitetura apresentada neste trabalho inicia-se com o diagrama de classes mostrado na Figura 4.2. Com é possível observar, este diagrama não apresenta apenas classes de objetos, mas também agentes, entidades autônomas que cooperam entre si dentro da arquitetura de *software* proposta buscando atingir os objetivos da missão. Este recurso foi utilizado para mostrar as relações estruturais existentes entre agentes e entre classes de objetos e agentes. Agentes neste diagrama não são entidades instanciáveis, mas sim entidades independentes com execução própria que, para sua existência, instanciam classes de objetos.

Figura 4.2: Diagrama de classes da arquitetura de *software* proposta.

Basicamente foi utilizado três tipos de relacionamentos distintos no diagrama da Figura 4.2. O primeiro deles é o de *generalização/especialização* e pode, por exemplo, ser observado entre os agentes *Atuador*, *Propulsao* e *Leme*. Este relacionamento ocorre entre entidade gerais (no caso citado, agente *Atuador*) e entidades mais específicas (agentes *Propulsao* e *Leme*), indicando que as entidades específicas herdam as características da entidade geral.

Um segundo tipo de relacionamento é o de *agregação*. De acordo com este relacionamento cada agente *Monitor* é composto de 1 a *n* instâncias da classe *classMonitor*. Observa-se que esta relação não aparece entre agentes pois estes não são entidades instanciáveis como já mencionado.

Finalizando, o último relacionamento encontrado no diagrama da Figura 4.2 é definido a partir do esteriótipo «troca de mensagens». Este relacionamento indica a forma de interação utilizado pelos agentes da arquitetura. Aqui também foi utilizado o recurso da valoração dos arcos que representam a associação, atribuindo com isto um aspecto quantitativo ao relacionamento. Assim, o agente *Gerenciador* trocará mensagens com *n* agentes *Monitor*. De forma geral, existirá comumente apenas um agente *Monitor* em execução dentro da arquitetura, o que levaria a indicação de uma relação de 1 para 1 na associação de troca de mensagens entre este agente e o *Gerenciador*. Porém, é possível que se deseje monitorar o sistema de pontos distintos (dois terminais de computador diferentes, por exemplo), o que justifica a utilização de uma relação 1 para *n*.

Figura 4.3: Tipos de mensagens do sistema: *TMsg* e suas especializações.

Já foi mencionado que a interação entre agentes é realizada através de troca de mensagens. Torna-se então necessário ilustrar tais mensagens e como estas se relacionam entre si. Para isto, elaborou-se um outro diagrama com a mesma semântica de um diagrama de classes, porém contendo no lugar de classes de objetos, os tipos de mensagens da arquitetura. Este diagrama é apresentado na Figura 4.3. Como é possível notar observando este diagrama, todas as mensagens são especialização de um único tipo de mensagem genérico (*TMsg*), que possui um atributo *Tipo* que as diferenciam. Este atributo é definido através de uma enumeração, que lista os diversos tipos de mensagens existentes dentro da arquitetura de *software*. Dependendo do tipo de mensagem, as especializações de *TMsg* possuem diferentes atributos. A especialização *TMsgAtuador*, por exemplo, possui o atributo *DadosAtuador* que foi definido como sendo do tipo *TDataAtuador*, uma estrutura de dados que contém as variáveis necessárias para os atuadores da arquitetura (*Propulsao* e *Leme*).

Figura 4.4: Tipos de estruturas de dados presentes na arquitetura de *software* proposta.

A utilização de estruturas de dados, além de melhorar a organização das variáveis utilizadas em um sistema, é conveniente pois possibilita o envio de todas as informações de um agente através de uma única mensagem. Pensando nisto organizou-se para cada agente da arquitetura proposta um tipo de estrutura de dados, as quais são apresentadas no diagrama da Figura 4.4.

4.5.2 Diagramas de Gráfico de Estados e de Atividades

Os diagramas de gráfico de estados mostram máquinas de estados, dando ênfase ao fluxo de controle de um estado para outro. Uma máquina de estados é um comportamento que especifica as seqüências de estados pelos quais uma entidade passa durante seu tempo de vida em resposta a eventos, juntamente com suas respostas a esses eventos.

Por outro lado, os diagramas de atividades mostram um conjunto de atividades e o fluxo de controle de uma atividade para outra. As atividades efetivamente resultam em alguma ação, que ocasionam uma mudança de estado do sistema ou o retorno de um valor.

Os dois tipos de diagramas acima são ideais para a modelagem de aspectos comportamentais de um sistema. Por este motivo, foram escolhidos para a descrição do comportamento interno dos agentes da arquitetura de *software* proposta. Assim, para cada agente apresentado no diagrama da Figura 4.2, foi elaborado um diagrama de gráfico de estados, sendo ainda utilizados diagramas de atividades para o detalhamento dos estados mais relevantes.

Para melhor organizar a explicação dos diagramas que serão apresentados nesta seção optou-se pela utilização de tabelas, as quais conterão os diversos estados possíveis para o agente (que representam o conjunto de situações possíveis em que o agente satisfaz alguma condição, realiza alguma atividade ou aguarda algum evento durante o decorrer de sua existência), acompanhado de uma breve descrição dos mesmos. Assim, apresenta-se a seguir a Figura 4.5 e a Tabela 4.1 referentes ao agente *Gerenciador*, a Figura 4.6 e a Tabela 4.2 referentes ao agente *Monitor*, a Figura 4.8 e a Tabela 4.3 referentes ao agente *Atuador*, a Figura 4.9 e a Tabela 4.4 referentes ao agente *Sensor*, e finalmente, a Figura 4.10 e a Tabela 4.5 referentes ao agente *Radio*, completando a totalidade dos agentes da arquitetura desenvolvida.

De modo geral, cada agente de arquitetura possui algum conjunto de responsabilidades que leva-os à execução de determinado conjunto de procedimentos, como a manutenção da referência

Figura 4.5: Diagrama de Gráfico de Estados do agente *Gerenciador*.

de um atuador ou então a aquisição periódica dos dados de um sensor para a disponibilização das informações para os demais agentes da arquitetura. Seja qual for a responsabilidade do agente, ele deve estar constantemente pronto para responder solicitações externas dos demais elementos da arquitetura, sendo proposto a utilização de processos concorrentes ou multi-*threads* para a implementação desta característica.

Tabela 4.1: Descrição do comportamento interno do agente *Gerenciador*.

Estado	Descrição
Inicializando Gerenciador	Inicialmente, o <i>Gerenciador</i> registra-se no sistema operacional para que possa ser localizado pelos demais agentes da arquitetura. Logo em seguida, cria os agentes necessários para a realização da missão e instancia objetos de suas classes componentes. Com o termino das atividades anteriores, o agente transita para o estado de execução em modo manual.
Executando em Modo Manual	Neste estado o agente atualiza os valores de referência para os atuadores do sistema, solicitando-os ao agente <i>Radio</i> em intervalos de tempo definidos por um <i>timer</i> .
Executando Missão	O comando <i>Executar Missão</i> ocasiona a transição para este estado. Inicialmente é lido as instruções do arquivo de missão, previamente construído pelo usuário. Em seguida, uma a uma as instruções são interpretadas e executadas de forma a se atingir os objetivos da missão. Com o termino da missão ou o cancelamento da mesma pelo operador, há a transição deste estado para a execução em modo manual.
Finalizando Sistema	O comando <i>Sair</i> faz com que este agente encerre a execução da arquitetura. São enviadas mensagens de finalização para todos os agentes da arquitetura, é apagado o registro no sistema operacional e finalizada a execução deste agente.

4.5.3 Diagramas de Interação

Em um sistema multi-agente, como é o caso da arquitetura proposta neste trabalho, as diversas entidades que compõem o sistema possuem cada qual responsabilidades próprias com objetivos particulares. Entretanto, para que seja possível o alcance do objetivo do sistema como um todo, a execução de uma determinada missão definida *a priori* pelo usuário, os diversos objetivos individuais devem ser coordenados de forma cooperativa.

Assim, na modelagem de sistema multi-agentes é essencial a apresentação das formas de cooperação existentes entre as diversas entidades que compõem o sistema, suas formas de sincronização de atividades por meio de diagramas de interação.

Um diagrama de interação mostra as interações de um determinado conjunto de entidades e seus relacionamentos, incluindo as mensagens que poderão ser trocadas entre tais entidades

Figura 4.6: Diagrama de Gráfico de Estados do agente *Monitor*.

no contexto de um cenário que ilustra um comportamento.

Os diagramas de seqüência e os de colaboração são dois exemplos de diagramas de interação existentes na *UML*. Enquanto um diagrama de seqüência dá ênfase à ordenação temporal das mensagens trocadas entre entidades, um diagrama de colaboração, por sua vez, dá ênfase à organização estrutural das entidades que trocam mensagens.

As Figuras 4.11 e 4.12 apresentam dois exemplos de diagramas de colaboração elaborados durante a modelagem da arquitetura proposta, o primeiro ilustrando as trocas de mensagens para a determinação de referências para os atuadores e o segundo as trocas de mensagens para o monitoramento da arquitetura. Em geral, estes diagramas apresentam uma entidade principal (agente *Gerenciador* no caso da Figura 4.11 e agente *Monitor* no caso da Figura

Figura 4.7: Diagrama de Atividades dos processos de monitoramento do agente *Monitor*.

4.12) procurando realizar uma determinada tarefa através da utilização de recursos de outras entidades, numerando-se os procedimentos realizados para o recebimento destes recursos de forma cronológica a que eles são realizados.

Exemplos de diagramas de seqüência são apresentados nas Figuras 4.13 e 4.14, que mostram, respectivamente, um exemplo de cenário de execução de missão definida pelo usuário e um exemplo de cenário de monitoramento de um sensor. Estes diagramas são similares aos apresentados pelas Figuras 4.11 e 4.12 por enfocarem os mesmos cenários de execução. Por outro lado, eles são complementares por apresentarem as informações sob outra ênfase. Enquanto nos diagramas de colaboração o aspecto temporal era ilustrado apenas pela enumeração cronológica dos eventos, nos diagramas de seqüência ele é o essencial.

Basicamente, os diagramas de seqüência apresentam os elementos que cooperam em um determinado cenário ligados a eixos verticais. As cooperações entre entidades são representadas através de arcos, dispostos de forma cronológica ao longo dos eixos, ligando os pares de entidades que se relacionam.

Nestes diagramas encontram-se as melhores possibilidades de se ilustrar características de

Figura 4.8: Diagrama de Gráfico de Estados do agente *Atuador*.

tempo real dentro da modelagem através da *UML*. Requisitos de tempo (como $dt1$, $dt2$ e $dt3$ no diagrama da Figura 4.13) e restrições temporais (como $[dt2 \ll dt1]$ no diagrama da Figura 4.14) podem ser apresentados através de notas vinculadas aos arcos do diagrama. Tais arcos possuem terminadores e traços distintos para ilustrar diferentes tipos de interações como por exemplo as chamadas de métodos e as mensagens assíncronas, periódicas e de retorno.

Figura 4.9: Diagrama de Gráfico de Estados do agente *Sensor*.

Figura 4.10: Diagrama de Gráfico de Estados do agente *Radio*.

Tabela 4.2: Descrição do comportamento interno do agente *Monitor*.

Estado	Descrição
Inicializando Monitor	Este agente inicia sua execução criando uma interface gráfica no terminal de vídeo do computador, na qual serão apresentadas diversas sub-janelas destinados ao monitoramento de cada agente da arquitetura. A partir deste estado, o fluxo de execução deste agente bifurca-se em dois subestados concorrentes.
Monitorando Processos	Neste estado, diversos processos, cada qual responsável pelo monitoramento de um agente da arquitetura, estão constantemente atualizando as informações em suas respectivas sub-janelas no terminal de vídeo do operador. O diagrama apresentado na Figura 4.7 mostra com mais clareza as atividades realizadas em cada um destes processos. Comandos do operador via teclado interrompem momentaneamente a execução destes processos para que tais comandos possam ser interpretados e atendidos.
Amostrando Dados	Inicialmente a execução neste estado aguarda o comando do operador para o início da aquisição de dados. Tal aquisição consiste no armazenamento em arquivo das informações dos diversos agentes do sistema, com tempo de amostragem definida por um <i>timer</i> .
Configurando Parâmetro p/ Aquisição de Dados	Neste estado é efetuada pelo operador a seleção das informações que se deseja armazenar em arquivo durante a execução da missão.
Finalizando Agente	Após a solicitação de encerramento feita pelo operador, o fluxo de execução do agente une-se novamente neste único estado. Os processos concorrentes são finalizados e a execução do agente é encerrada.

Tabela 4.3: Descrição do comportamento interno do agente *Atuador*.

Estado	Descrição
Inicializando Atuador	O agente <i>Atuador</i> inicializa instanciando o objeto <i>classAtuador</i> , registrando-se logo em seguida no sistema operacional para que possa ser localizado pelos demais agentes do sistema.
Aguardando Mensagens	Após a inicialização, o agente <i>Atuador</i> transita para este estado de espera, onde permanece até que receba alguma mensagem válida.
Tratando MSG_SETPOINT	Neste estado atualiza-se o valor de referência. Tal referência é enviada pelo agente <i>Gerenciador</i> e é usada na malha de controle do atuador. Como retorno são enviadas as informações do atuador ou, quando for o caso, mensagens de erro.
Tratando MSG_DATAREQUEST	Neste estado são atendidas solicitações de informações a respeito do atuador. Diferencia-se do estado anterior por não alterar a referência do atuador.
Finalizando Agente	Após solicitações feita pelo agente <i>Gerenciador</i> por meio de mensagens, a execução deste agente é finalizada.

Tabela 4.4: Descrição do comportamento interno do agente *Sensor*.

Estado	Descrição
Inicializando Sensor	A inicialização deste agente consiste, de forma resumida, da instanciação de sua classe componente (<i>classSensor</i>) e do seu registro no sistema operacional.
Aguardando Mensagens	Vide mesmo estado na Tabela 4.3.
Tratando MSG_DATAREQUEST	Após um requisição feita por algum agente da arquitetura, é realizada neste estado a aquisição de dados do sensor. Caso ocorra algum erro na aquisição, uma mensagem de erro correspondente é retornada. Do contrário, os dados são tratados e retornados para o agente que realizou a solicitação.
Finalizando Agente	Vide mesmo estado na Tabela 4.3.

Tabela 4.5: Descrição do comportamento interno do agente *Radio*.

Estado	Descrição
Inicializando Radio	Após instanciar sua classe componente e registrar-se no sistema operacional, o agente vincula uma interrupção de <i>hardware</i> para a aquisição dos sinais de rádio.
Aguardando Mensagens	Há apenas dois tipos de mensagens tratados por este agente: mensagens de requisição das informações do rádio e mensagens solicitando a finalização deste agente.
Tratando Interrupção	Com a ocorrência de uma interrupção de <i>hardware</i> vinculada ao receptor de rádio, o fluxo de execução deste agente é desviado para uma rotina de tratamento, que adquire e interpreta as informações dos canais de rádio, atualizando as variáveis do agente.
Finalizando Agente	Neste estado é desativada a interrupção vinculada ao rádio, apagado o registro do agente no sistema operacional e realizado os demais procedimentos para finalização do agente.

Figura 4.11: Diagrama de Colaboração para a determinação de referências para os atuadores.

Figura 4.12: Diagrama de Colaboração para o monitoramento da arquitetura.

Figura 4.13: Diagrama de Seqüência mostrando um cenário de execução de missão.

Capítulo 5

Implementações e Resultados

Os estudos desenvolvidos dentro deste trabalho de pesquisa, além de suas destinações motivadas por investigações nas áreas de robôs móveis autônomos (particularmente veículos submarinos autônomos), estão vinculados desde sua concepção ao projeto temático "Comportamento em Ondas de Embarcações de Alto Desempenho" (proc. Fapesp 1997/13090-3), projeto este em desenvolvimento conjunto pelas seguintes instituições: Departamento de Engenharia Naval e Oceânica e Departamento de Engenharia Mecatrônica e Sistemas Mecânicos, ambos da Escola Politécnica da Universidade de São Paulo, Instituto de Pesquisas Tecnológicas (IPT) e Marinha do Brasil.

Tal projeto tem como objetivo o estudo do comportamento no mar e manobrabilidade de embarcações de superfície. Para tanto prevê a utilização de modelos em escala reduzida de navios autopropelidos em testes de identificação da dinâmica e manobrabilidade, os quais

Figura 5.1: Embarcação utilizada em testes ITTC.

Figura 5.2: Modelo ITTC em escala reduzida.

são realizados através de manobras padronizadas tais como o *giro* e o *zig-zag* (De Barros e Ferreira, 2000; De Barros e Ferreira, 2001).

O alvo principal dos estudos deste projeto temático é o modelo ITTC (*International Towing Tank Conference*), casco usado para comparações de resultados produzidos por diversos institutos de pesquisa internacionais. Este tipo de casco é utilizado em embarcações de semiplaneio destinadas ao transporte de passageiros e apresenta como principais dimensões as seguintes: 23,7 m de Lpp, 5,2 m de largura, 2,4 m de pontal e 0,8 m de calado (ver Figura 5.1).

A idéia é utilizar modelos em escala reduzida (ver Figura 5.2) para a validação de métodos de predição numérica e analítica desenvolvidos dentro do projeto temático.

Se por um lado tais modelos em escala de navios propiciam a coleta de informações a respeito da dinâmica e manobrabilidade das embarcações reais, por outro lado servem como uma fase de desenvolvimento de menor custo para robôs móveis autônomos (quando comparados a implementações submarinas de veículos submarinos autônomos e veículos submarinos remotamente operados), constituindo assim bancadas de testes para a implementação e validação da arquitetura de *software* proposta neste trabalho de dissertação de mestrado.

Este capítulo apresenta os diversos equipamentos de *hardware* e montagens eletrônicas utilizados para a implementação da proposta de arquitetura contida no capítulo 4, juntamente com a descrição dos ensaios realizados para a coleta de dados, tanto para os estudos de identificação dinâmica previstos no projeto temático citado anteriormente, como para a validação da arquitetura proposta.

5.1 Estrutura de *Hardware* e *Software* Implementada

A Figura 5.3 apresenta um diagrama esquemático contendo o modelo em escala reduzida de navio, juntamente com o conjunto de dispositivos e montagens eletrônicas desenvolvidos como parte das atividades do projeto temático ao qual este trabalho é vinculado. Esta estrutura também será utilizada para a validação das propostas apresentadas neste trabalho de dissertação de mestrado.

Figura 5.3: Diagrama esquemático do modelo em escala de navio.

Como é possível observar na Figura 5.3, o sistema como um todo é composto de duas porções. A primeira delas, embarcada no casco confeccionado em fibra de vidro, é constituída pelo microcomputador de bordo, servo-sistemas para controle dos atuadores (hélices e leme), sensores diversos (bússola eletrônica, receptor GPS e unidade inercial), equipamento de comunicação *wireless* LAN, receptor de radiomodelismo e baterias. A segunda porção, esta em terra, é representada por um microcomputador portátil, um outro equipamento *wireless* LAN e um controle de radiomodelismo, os quais são utilizados para o monitoramento e comando a distância do modelo de navio.

Nas subseções seguintes cada um destes componentes é separadamente detalhado.

5.1.1 Casco ITTC em escala reduzida

O casco, em escala reduzida, do modelo ITTC (ver Figura 5.2) foi construído em fibra de vidro pelo Departamento de Engenharia Naval do Instituto de Pesquisas Tecnológicas (IPT) com as seguintes dimensões: 2m de Lpp, 0,43 m de largura, 0,20 m de pontal e 0,07 m de calado. Tais medidas representam uma redução de 1 para 11,85 do casco ITTC em tamanho natural.

O processo de construção deste casco é inteiramente artesanal, podendo ser resumido da seguinte forma. Primeiramente é necessário a elaboração de uma matrix *macho* em madeira. A partir desta primeira matriz, realiza-se a deposição de mantas de lã de vidro e resina de forma a se obter uma segunda matriz, esta porém *fêmea* e de fibra de vidro. Finalmente, nova deposição de mantas de lã de vidro é realizada, obtendo-se assim o casco desejado.

5.1.2 Microcomputador de Bordo PC/104

A principal unidade de processamento do veículo, responsável pela computação de alto nível da arquitetura e que é embarcada no modelo em escala de navio, é um microcomputador de barramento PC/104. Este tipo de equipamento é encontrado comercialmente na forma de módulos, configuráveis de acordo com as necessidades da aplicação de interesse. Neste projeto escolheu-se uma configuração fornecida pela empresa *Real Time Devices USA, Inc.* contendo os seguintes módulos:

- CPU de 64 bits, com faixa de temperatura de operação estendido para -40 a +85o C (sem a necessidade de *cooler* para evitar superaquecimento do processador), clock de 233 MHz, duas portas seriais RS232/422/485, porta paralela bidirecional ECP, interface para teclado e *mouse* e *flash* BIOS;
- controlador VGA com interface para monitor ou painel de cristal líquido;
- entradas e saídas analógicas e digitais, conversor analógico-digital, cronômetros/contadores e controle digital;

- disco rígido de 3.2 Gbytes, com conector para *slave drive* e controlador de disco flexível;
- controlador NE2000 *Ethernet*;
- expensor com quatro interfaces seriais;
- fonte de alimentação com entrada em 8-30 Vdc e 75 W.

O fabricante disponibiliza os módulos PC/104 já encapsulados em carcaças de alumínio, o que melhora a praticidade e robustez destes equipamentos, protegendo-os do meio externo.

5.1.3 Servos-Sistemas e Atuadores

Dois tipos de servos-sistemas distintos são utilizados para o controle dos atuadores do modelo em escala de navio:

- **Propulsão** - este tipo de servo-sistema realiza o controle da rotação dos motores de corrente contínua responsáveis pela propulsão dos hélices da embarcação. Duas malhas de controle PI (proporcional-integral) independentes realizam o controle dos motores a partir de referências de velocidade, fornecidas pelo microcomputador PC/104, e sinais de retorno de rotação, fornecidos por tacogeradores acoplados aos motores;
- **Leme** - este tipo de servo-sistema realiza o controle de posição do leme. Uma malha de controle PID (proporcional-integral-derivativo) recebe referências do PC/104 e retorno de posição de um *encoder* acoplado ao motor de corrente contínua utilizado para o posicionamento do leme do veículo.

5.1.4 Sensores

O conjunto de sensores a serem utilizados nos ensaios previstos neste trabalho incluem uma bússola eletrônica, dois receptores GPS e uma unidade inercial. Todos estes equipamentos

possuem pré-processamento de seus dados, o que permite o fornecimento de suas informações via interface serial RS232.

A seguir será descrito cada um destes equipamentos.

Bússola Eletrônica

A TCM2 (ver Figura 5.4) é uma bússola eletrônica fabricada pela empresa *PNI Corporation* que combina um magnetômetro de três eixos e um sensor de inclinação de dois eixos em um único módulo de baixo consumo de energia.

A combinação das informações destes dois sensores por meio de um microprocessador permite que este módulo forneça, além do ângulo de rumo (*yaw*) com um erro de $0,5^\circ$, informações sobre os ângulos de *pitch* e *roll* (erro de $0,2^\circ$) e os valores do campo magnético nas três dimensões (erro de $0,2 \mu\text{T}$) a uma taxa de até 30 Hz. A Tabela 5.1 resume essa informações.

Figura 5.4: Bússola eletrônica TCM2.

Por possuir um microprocessador, a TCM2 pode ser programada pelo usuário através de uma interface serial RS232, permitindo a seleção do modo de operação (*Standby Mode* e *Continuous Sampling Mode*), a escolha das informações que se deseja amostrar, além da calibração do equipamento.

Tabela 5.1: Resumo das informações a respeito da bússola eletrônica TCM2.

Frequência de Amostragem Máxima	30 Hz
Erro no Ângulo de <i>Yaw</i>	$\pm 0,5^\circ$
Erro nos Ângulos de <i>Pitch</i> e <i>Roll</i>	$\pm 0,2^\circ/\text{s}$

Receptor GPS

Os equipamentos de posicionamento global (GPS) que se encontram disponíveis para utilização neste trabalho consistem de dispositivos adquiridos do fabricante canadense *Novatel*. Eles constituem de dois receptores GPS, duas antenas e um aplicativo de *software* destina à configuração dos equipamentos.

Figura 5.5: Equipamento de posicionamento global.

Os receptores GPS são do modelo *ProPak II* (ver Figura 5.5), que possui 12 canais para recepção dos sinais transmitidos pelos satélites do sistema GPS e fornece dados de posicionamento com erros inferiores a 15 metros a uma taxa de 5 Hz, quando operado em modo convencional (isto é, sem correções diferenciais). Operando em modo diferencial, este erro cai para 10 centímetros. A Tabela 5.2 resume essa informações.

Tabela 5.2: Resumo das informações a respeito do equipamento GPS.

Frequência de Amostragem Máxima	5 Hz
Erro em Modo Convencional	> 15 m
Erro em Modo Diferencial	> 1 cm

Unidade Inercial

A unidade inercial VG600AA (ver Figura 5.6) utilizada neste projeto foi adquirida da empresa *Crossbow Technology, Inc.* Esta unidade combina um giroscópio de fibra ótica com um conjunto de acelerômetros, disponibilizando via interface serial RS232 a uma frequência de até 80 Hz os ângulos de *pitch* e *roll* (com erro inferior a $0,5^\circ$), as taxas de variação angulares em *pitch*, *roll* e *yaw* (com erro inferior a $0,03$ graus/s), e as acelerações no três eixos (com erro inferior a $8,5$ mg). A Tabela 5.3 resume essa informações.

Figura 5.6: Unidade inercial VG600AA.

Tabela 5.3: Resumo das informações a respeito da unidade inercial VG600AA.

Frequência de Amostragem Máxima	80 Hz
Erro nos Deslocamentos Angulares	$\pm 0,5^\circ$
Erro nas Velocidade Angulares	$\pm 0,03^\circ/\text{s}$
Erro nas Medidas de Aceleração	$8,5\text{mg}$

5.1.5 Equipamentos de Comunicação

Um dos requisitos básicos de toda arquitetura de controle destinada a aplicações de robôs móveis autônomos é a possibilidade de monitoramento e comando a distância. Mesmo que se vislumbre a operação do veículo de forma totalmente autônoma, a presença de alguma meio de comando manual com capacidade de sobrepor-se aos mecanismos de decisão da arquitetura deverá invariavelmente existir.

A satisfação do requisito mencionado acima passa em geral pela utilização de algum tipo de comunicação via rádio frequência. Para o caso da arquitetura de controle proposta neste trabalho, optou-se pela utilização de dois sistemas de rádio-comunicação distintos, um destes destinado exclusivamente para o monitoramento do sistema embarcado e o outro para o comando a distância. Para o primeiro, escolheu-se dispositivos de tecnologia *wireless LAN*, enquanto para o segundo sistema optou-se por equipamentos de rádio-controle desenvolvido para utilização em aeromodelos.

Equipamentos *Wireless LAN*

Para a escolha dos equipamentos *wireless LAN* utilizados nesta aplicação dentre os diversos modelos existentes comercialmente, três requisitos foram de especial relevância: a distância de comunicação máxima requerida, que em situações limites pode ser de até 1 quilometro; o consumo de energia e as dimensões físicas do equipamento, que devem ser o quanto menor possível em função das restrições da aplicação.

Tendo-se em mente os requisitos anteriores, optou-se pela escolha dos equipamentos *wireless LAN* fornecidos pelo fabricante japonês *JRC Ltda.* (ver Figuras 5.7 e 5.8). Estes equipamentos são de fácil utilização, transparentes para o desenvolvedor do sistema de *software* e, após testes realizados em campo e em laboratório, satisfizeram as expectativas iniciais.

JRL-210MS / 210BS

Figura 5.7: Equipamento *wireless* LAN da *JRC Ltda.*

Figura 5.8: Antena externa direcional NZA-4200, bateria externa NBB-476 e antena externa onidirecional NZA-200.

Rádio de Aeromodelos

Como solução rápida para o problema do comando a distância do modelo em escala utilizado neste projeto, adotou-se um sistema de rádio comumente utilizado para o comando de aeromodelos (ver Figura 5.9).

Figura 5.9: Equipamento para telecomando de aeromodelos.

Este equipamento, desenvolvido pelo fabricante japonês *Futaba*, possui seis canais de rádio-frequência independentes para enviar os comandos de suas alavancas em modulação digital.

5.1.6 Baterias

A definição de qual a melhor bateria para uma determinada aplicação depende de uma série de parâmetros. A correta seleção requer um razoável conhecimento das características de demanda dos equipamentos a serem alimentados bem como dos modelos de baterias disponíveis comercialmente.

Para este projeto, após um levantamento do total de demanda de energia requerida para a alimentação dos equipamentos e do valor de massa livre disponível para utilização com baterias, optou-se pela tecnologia de baterias de *Níquel-Metal-Hidreto* (*Ni-Mh*), sendo utilizadas células de carga *4/5A2000*, as quais foram associadas de forma a se obter um par de baterias de 12V e 16Ah.

Os equipamentos embarcados no modelo em escala são alimentados nas tensões nominais de 24, 12 e 5 Volts. Por este motivo foi conveniente projetar baterias com tensão nominal de 12V, dando-se assim maior flexibilidade para a utilização das mesmas. A associação de um par destas baterias em série produz os 24V requeridos, enquanto a tensão de 5V pode ser facilmente conseguida através de reguladores de tensão.

5.2 Ensaios Realizados

Utilizando a estrutura de *hardware* e *software* apresentada nas seções anteriores deste capítulo, foram realizados três conjuntos de ensaios distintos com o modelo ITTC. Além do objetivo comum, validação da arquitetura de *software* proposta por este trabalho, cada conjunto de ensaios tiveram objetivos específicos. Os ensaios preliminares verificaram a integração da estrutura de *hardware* e *software* implementada além de fornecerem elementos de análise dos resultados de ensaios posteriores. Nos ensaios com piloto automático, o modelo navega num rumo constante durante a aquisição das variáveis dinâmicas do seu movimento. Finalmente, nos ensaios de navegação comprovou-se que a arquitetura implementada é capaz de adquirir, em tempo real, as informações de posicionamento fornecidas pelo equipamento DGPS.

Como considerações iniciais, é necessário lembrar que em todas as análises realizadas com

os dados obtidos nos ensaios deste trabalho foi desconsiderada a transformação dos deslocamentos angulares através da matriz de ângulos de *Euler*. Levando-se em conta que, nos testes realizados, tais deslocamentos foram pequenos, esta matriz torna-se aproximadamente igual a matriz identidade, o que propicia a desconsideração efetuada. Somente acelerações diretamente medidas pela unidade inercial foram analisadas. Para o cálculo das acelerações do centro de gravidade do veículo, a contribuição da aceleração centrípeta da unidade bem como a originária de sua aceleração angular devem ser consideradas com base nas velocidades angulares medidas pela unidade e as distâncias entre essa e o centro de gravidade do veículo (vide Tabela 5.4).

A Figura 5.10 apresenta a posição do centro de gravidade do veículo, cuja posição longitudinal coincide com a posição do seu centro de flutuação (LCG=LCB). As coordenadas são medidas em relação a um sistema cartesiano com origem no fundo da popa do modelo, na intersecção do plano de simetria longitudinal com o costado do modelo (conforme indica a Figura 5.10). Já a Figura 5.11 representa o sistema de coordenadas adotado, juntamente com as velocidades angulares ω_x , ω_y e ω_z nos eixos x, y e z (fixos na unidade inercial), respectivamente, assim como as acelerações a_x , a_y e a_z .

A Tabela 5.4 reúne as posições relativas da unidade inercial e da bússola eletrônica em relação ao centro de gravidade apresentado.

Figura 5.10: Posição do centro de gravidade (LCG) do veículo, coincidente com seu centro de flutuação (LCB).

Tabela 5.4: Posições relativas da unidade inercial e da bússola eletrônica em relação ao centro de gravidade (LCG) do veículo.

Item	Posição em X	Posição em Y	Posição em Z
Bússola Eletrônica TCM2	1,11 m	0,0 m	0,045 m
Unidade Inercial VG600AA	0,67 m	0,0 m	0,025 m

Figura 5.11: Sistema de coordenadas adotado e movimentos medidos em ensaios.

5.2.1 Ensaios Preliminares

Dentro desta seção de ensaios preliminares, reuniram-se três tipos de ensaios. No ensaio em repouso registraram-se erros e variações da unidade inercial. No ensaio de vibrações tentou-se averiguar o efeito das vibrações provocadas pelo sistema propulsor do veículo nos dados fornecidos pela unidade inercial. No ensaio em águas calmas, procurou-se determinar o desnível existente entre a unidade inercial e a bússola eletrônica para que os dados destes dois sensores pudessem ser comparados. O último tipo de ensaios apresentado nesta seção denominou-se de ensaios em comando "manual", que visou testar a comunicação entre o operador e o veículo através de comandos e dados transmitidos por rádio.

Ensaio em Repouso

Neste ensaio, realizado em laboratório, foram coletados os dados fornecidos pela unidade inercial em situação de repouso, visando registrar os erros e variações em suas medidas.

Os gráficos mostrados nas Figuras 5.12, 5.13 e 5.14 apresentam as velocidades angulares coletados neste ensaio. Já os gráficos das Figuras 5.15, 5.16 e 5.17 apresentam os movimentos de avanço, lateral e arfagem, respectivamente.

Foi calculado os valores médios e desvios para os dados deste ensaio, reunindo-os na Tabela 5.5.

Figura 5.12: Velocidade angular ω_x no ensaio em repouso.

Figura 5.13: Velocidade angular ω_y no ensaio em repouso.

Figura 5.14: Velocidade angular ω_z no ensaio em repouso.

Figura 5.15: Movimento de avanço A_x no ensaio em repouso.

Figura 5.16: Movimento lateral A_y no ensaio em repouso.

Figura 5.17: Movimento de arfagem A_z no ensaio em repouso.

Tabela 5.5: Valores médios e desvios médios dos dados de ensaio em repouso.

Variável	Média	Desvio Médio
A_x	-0,0631 g	0,92 mg
A_y	0,0183 g	1,01 mg
A_z	-0,9954 g	0,69 mg
ω_x	0,3160 graus/s	0,3681 graus/s
ω_y	-0,0615 graus/s	0,1103 graus/s
ω_z	0,1253 graus/s	0,4069 graus/s

Ensaio de Vibração

Ainda em laboratório, realizou-se a aquisição dos dados fornecidos pela unidade inercial, buscando-se constatar a influência de vibrações de alta frequência nas medidas deste sensor oriundas da propulsão do modelo ITTC. Para isto, colocaram-se os propulsores em funcionamento com uma referência de rotação específica (a mesma utilizada na maioria dos testes posteriores) e adquiriram-se os dados citados anteriormente. Os resultados deste testes iniciais em bancada são apresentados nos gráficos presentes nas Figuras 5.18 a 5.21.

Tabela 5.6: Valores médios e desvios médios dos dados de ensaio de vibrações.

Variável	Média	Desvio Médio
A_x	-0,0630 g	19,1 mg
A_y	0,0098 g	9,2 mg
A_z	-0,9973 g	15,3 mg
ω_x	0,2689 graus/s	0,3836 graus/s
ω_y	-0,1134 graus/s	0,1361 graus/s
ω_z	0,0428 graus/s	0,4190 graus/s

Figura 5.18: Rotações dos propulsores em ensaio de vibração.

Figura 5.19: Movimento de avanço A_x em ensaio de vibração.

Figura 5.20: Movimento lateral A_y em ensaio de vibração.

Figura 5.21: Movimento de arfagem A_z em ensaio de vibração.

Figura 5.22: Velocidade angular ω_x em ensaio de vibração.

Figura 5.23: Velocidade angular ω_y em ensaio de vibração.

Figura 5.24: Velocidade angular ω_z em ensaio de vibração.

Os valores médios e desvios médios calculados para cada uma das variáveis apresentadas nos gráficos anteriores, foram reunidos na Tabela 5.6. Comparando-se os desvios das velocidades angulares neste ensaio, com os desvios médios calculados no ensaio em repouso presentes na Tabela 5.5, observa-se que as vibrações provocadas pelo sistema propulsor afetam mais marcadamente as medidas de A_x , A_y e A_z , ficando as medidas de velocidades angulares com poucas alterações.

Ensaio em Águas Calmas

Neste tipo de ensaio procurou-se determinar os valores de desnível entre a unidade inercial VG600AA e a bússola eletrônica TCM2. Para isto, adquiriu-se os dados fornecidos pela unidade inercial e pela bússola eletrônica com o modelo em escala de navio colocado em repouso na margem da raia olímpica da USP. Tais dados são mostrados nos gráficos das Figuras 5.25 e 5.26.

Figura 5.25: Dados de movimento do ensaio em águas calmas.

Figura 5.26: Dados de deslocamento angular do ensaio em águas calmas.

A partir dos dados presentes nos gráficos mencionados anteriormente, obteve-se valores médios para cada uma das variáveis que são apresentadas na Tabela 5.7.

Tabela 5.7: Valores médios de dados do ensaio em águas calmas.

Variável	Média
Deslocamento angular em <i>Roll</i>	1,28°
Deslocamento angular em <i>Pitch</i>	-4,10°
A_x	-0,0458 g
A_y	0,0171 g
A_z	-0,9987 g

Com os valores médios apresentados na Tabela 5.7 é possível obter os desniveis θ_d em *Pitch* e ϕ_d em *Roll* entre a unidade inercial e a bússola eletrônica, utilizando-se para isto as seguintes relações:

$$\overline{A_x} = g.\text{sen}(\overline{\theta}) \quad (5.1)$$

$$\overline{A_y} = g.\text{sen}(\overline{\phi}) \quad (5.2)$$

$$\overline{A_z} = -g.\text{cos}(\overline{\theta}) \quad (5.3)$$

onde

$\overline{A_x}$: movimento de avanço médio;

$\overline{A_y}$: movimento lateral médio;

$\overline{A_z}$: movimento de arfagem médio;

$\overline{\theta}$: deslocamento angular médio em *Pitch*;

$\overline{\phi}$: deslocamento angular médio em *Roll*.

Das equações acima e dos valores médios presentes na Tabela 5.7, determinou-se um desnível $\theta_d = -1,34 \pm 0,69^\circ$ da bússola eletrônica TCM2 em relação à unidade inercial VG600AA (o desnível ϕ_d calculado ficou abaixo do erro atribuído às medidas dos sensores).

Ensaaios em Comando "Manual"

Em um segundo momento, ainda nos teste preliminares da arquitetura de *hardware* e *software*, procurou-se comandar através de rádio-controle o modelo em escala de navio para que o mesmo navegasse em linha reta com uma referência fixa de rotação para os motores de propulsão. Este tipo de ensaios tinha como objetivo inicialmente a aquisição de informações para os estudos de identificação da dinâmica do casco ITTC previstos no projeto temático ao qual este trabalho está vinculado, além de testar a integração do rádio-controle com o sistema embarcado de manobra e propulsão.

Alguns resultados destes ensaios são apresentados nos gráficos das Figuras 5.27, 5.28 e 5.29 para fins ilustrativos.

Pelo gráfico da Figura 5.27 observa-se que não foi possível a manutenção do rumo da embarcação manualmente. Isto porque o sistema como um todo esteve sujeito a um conjunto de perturbações que interferem no rumo do veículo, dentre elas destacaram-se o vento e as ondas provocadas pelo mesmo. Diante de tais perturbações, o comando manual do veículo em um rumo constante é dificultado. Este fato incentivou à elaboração de um piloto automático (controle de rumo) para o veículo, para só então efetuar a aquisição de dados para os estudos de identificação dinâmica do casco ITTC (ver Seção 5.2.2).

Do gráfico da Figura 5.28 observa-se uma outra perturbação relevante no rumo do veículo e também uma limitação da propulsão do mesmo: os propulsores não atingiram a rotação de referência. O consumo de corrente pelos motores durante a execução do ensaio possibilita explicar esta limitação. Tal consumo foi, em média, de 8,25 A para o motor esquerdo e 4,76 A para o motor direito. Tem-se que a corrente máxima recomendada pelo fabricante dos motores utilizados para uso contínuo é de 3,8 A. Assim, procurou-se operar o veículo em uma velocidade em que as limitações de torque dos motores de propulsão não comprometessem na manutenção da rotação dos mesmos. Um detalhe a ser observado é que a corrente média reduz

Figura 5.27: Ângulo de rumo em ensaio sob comando manual.

progressivamente em ambos os motores, o que indica a redução da carga das baterias.

Após a realização de alguns ensaios com o veículo operação em velocidade reduzida, observou-se a manutenção segura (com consumo de corrente próximo do recomendado pelo fabricante) da rotação dos motores em 1500 rpm. Os gráficos apresentados nas Figuras 5.30 e 5.31 mostram os dados obtidos com o veículo operando com esta rotação em seus propulsores.

Têm-se, como esperado, uma melhoria na manutenção do rumo do veículo com a redução da rotação dos propulsores, constatada através do gráfico da Figura 5.30.

Ensaio de Manobrabilidade

Neste tipo de ensaio, o modelo em escala de navio executou manobras padronizadas (no caso, manobras de *zig-zag*) pré-programadas, objetivando a identificação da manobrabilidade

Figura 5.28: Rotação dos motores de propulsão em ensaio sob comando manual.

Figura 5.29: Corrente instantânea em motores de propulsão em ensaio sob comando manual.

Figura 5.30: Ângulo de rumo em ensaio sob comando manual.

Figura 5.31: Rotação dos motores de propulsão em ensaio sob comando manual.

do veículo necessária para a realização do controle de rumo do mesmo.

Através destes ensaios, utilizados desde a década de 1960 com este fim, procura-se determinar uma função de transferência aproximada que relacione as informações de ângulo do leme do veículo e de variação de rumo com que este está navegando.

Figura 5.32: Resposta típica de um ensaio em *zig-zag*.

A partir de dados coletados em ensaios de *zig-zag* (ver Figura 5.32) é possível obter aproximações da relação de manobrabilidade do modelo por até equações de primeira ordem da forma:

$$\frac{r(s)}{\delta(s)} = G_m = \frac{K}{T.s + 1} \quad (5.4)$$

onde

r : taxa de variação do ângulo de rumo da embarcação;

δ : ângulo do leme da embarcação;

K: ganho de regime permanente;

T: constante de tempo.

Admitindo-se o modelo em primeira ordem anterior, utiliza-se (Nomoto, Taguchi, Honda e Hirono, 1957):

$$\begin{aligned}
T.[1 + e^{\frac{-t_e}{T}} - e^{\frac{-(t_e-t_1)}{T}} - e^{\frac{-(t_e-t_2)}{T}}] &= t_e - (t_1 + t_2) \\
T.[1 + e^{\frac{-(t'_e-t_2)}{T}} - e^{\frac{-(t'_e-t_2-2.t_1)}{T}} - e^{\frac{-(t'_e-t_4)}{T}}] &= t'_e - (t_1 + t_4) \\
T.[1 + e^{\frac{-(t''_e-t_4)}{T}} - e^{\frac{-(t''_e-t_4-2.t_1)}{T}} - e^{\frac{-(t''_e-t_6)}{T}}] &= t''_e - (t_1 + t_6) \\
&\dots
\end{aligned} \tag{5.5}$$

e

$$\begin{aligned}
\frac{1}{K} &= \frac{\delta_0}{\theta_e.t_1} \cdot [(t_1 + t_2).t_e - \frac{1}{2} \cdot (t_e^2 + t_1^2 + t_2^2)] \\
\frac{1}{K} &= \frac{\delta_0}{\theta'_e.t_1} \cdot [-(t_1 + t_4).t'_e + \frac{1}{2} \cdot (t_e'^2 + t_1^2 + t_4^2) + t_1 \cdot (t_1 + 2.t_2)] \\
\frac{1}{K} &= \frac{\delta_0}{\theta''_e.t_1} \cdot [(t_1 + t_6).t''_e - \frac{1}{2} \cdot (t_e''^2 + t_1^2 + t_6^2) - 2.t_1 \cdot (t_4 - t_2)] \\
&\dots
\end{aligned} \tag{5.6}$$

O gráfico apresentado na Figura 5.33 mostra os dados coletados em um ensaio de *zig-zag* realizado com o modelo em escala de navio. A partir destes dados, obtiveram-se os seguintes valores médios para as constantes do sistema:

$$\overline{K} = 0,196 \tag{5.7}$$

$$\overline{T} = 0,440 \tag{5.8}$$

Figura 5.33: Ângulo de rumo e leme do modelo em escala de navio em ensaio de *zig-zag*.

5.2.2 Ensaio com Piloto Automático

De posse da relação aproximada para a manobrabilidade do modelo ITTC em escala reduzida é possível à elaboração de um piloto automático (controle de rumo) para o veículo. Adotou-se um controlador PI (Proporcional + Integral), dado pela forma geral:

$$G_c = \frac{K_p \cdot s + K_i}{s} \quad (5.9)$$

onde

K_p : ganho proporcional;

K_i : ganho integral.

A Figura 5.34 representa a malha de controle, onde ψ_d é o rumo desejado e ψ rumo real, negligenciando-se a dinâmica do sensor de rumo (bússola eletrônica).

Figura 5.34: Diagrama de blocos representando modelo do veículo mais controlador PI.

A Tabela 5.8 apresenta três pares de ganhos utilizados para o controlador PI durante os ensaios com o piloto automático.

Tabela 5.8: Ganhos utilizados para o controlador PI.

Par de Ganhos	K_p	K_i
(a)	12,82	52,44
(b)	11,22	40,15
(c)	10,20	33,18

Partiu-se então para a implementação do piloto automático no modelo em escala de navio. Foram então realizados os primeiros ensaios neste trabalho visando a coleta de dados para os estudos de identificação dinâmica do casco ITTC.

Nestes ensaios, o veículo utilizou o piloto automático para a manutenção do rumo enquanto navegava em velocidade constante e eram adquiridos os dados de movimento medidos pela unidade inercial. As Figuras 5.35, 5.36, 5.37 e 5.38 contém fotos do veículos mostrando a configuração de equipamentos utilizados nestes ensaios.

O melhor desempenho do piloto automático deu-se com o veículo navegando de encontro às ondas, como era esperado. Os resultados apresentados a seguir correspondem a esta condição.

Quatro ensaios são analisados: três deles com o veículo à velocidade de 1,4 m/s (um ensaio para cada par de ganhos do controlador apresentados na Tabela 5.8) e um com velocidade de 2 m/s. Os dados experimentais obtidos foram organizados da seguinte forma: inicialmente são apresentadas informações iniciais a respeito do ensaio, seguidas de uma tabela contendo

Figura 5.35: Visão superior do modelo em escala de navio.

Figura 5.36: Detalhe mostrando bússola eletrônica TCM2, equipamento *wireless* LAN e unidade inercial VG600AA.

Figura 5.37: Detalhe mostrando motores de propulsão, servo *drives*, PC104 (centro) e unidade inercial VG600AA (à direita).

Figura 5.38: Detalhe mostrando conjunto de baterias (esquerda) utilizadas, servo *drive* (direita) e mecanismo para posicionamento dos lemes.

um resumo dos resultados obtidos e uma breve análise dos mesmos. Os gráficos de dados experimentais são apresentados no final de cada análise de resultados, sendo os gráficos de movimento do veículo mostrados apenas em um dos ensaios à velocidade de 1,4 m/s e no ensaio a 2 m/s.

Ensaio I

Informações iniciais a respeito do ensaio:

Par de ganhos utilizado no controlador PI: a (ver Tabela 5.8);

Velocidade do veículo: 1,4 m/s;

Rotação de referência para propulsão: 1500 rpm.

Rumo de referência: 328,90°.

Tabela 5.9: Resumo dos resultados do Ensaio I.

Variável	Média	Desvio Médio
Rotação no Motor Esquerdo	1490 rpm	15 rpm
Rotação no Motor Direito	1520 rpm	27 rpm
Corrente Instantânea no Motor Esquerdo	4,17 A	1,04 A
Corrente Instantânea no Motor Direito	2,58 A	1,22 A
Rumo	328,99°	1,25°
Ângulo de Leme	-2,01°	17,77°
Deslocamento Angular em <i>Roll</i>	1,49°	1,71°
Deslocamento Angular em <i>Pitch</i>	-3,90°	0,78°
Movimento de Avanço $\overline{A_x}$	-0,0502 g	0,0243 g
Movimento Lateral $\overline{A_y}$	0,0192 g	0,0554 g
Movimento de Arfagem $\overline{A_z}$	-0,9907 g	0,1071 g
$sen(\overline{\theta} - \theta_d)$	-0,0446	-
$cos(\overline{\theta} - \theta_d)$	-0,9990	-
Velocidade Angular ωx	0,5027 graus/s	7,05 graus/s
Velocidade Angular ωy	-0,2125 graus/s	2,77 graus/s
Velocidade Angular ωz	0,1583 graus/s	4,50 graus/s

Das médias de rotação dos motores de propulsão observa-se que ambos apresentaram erro inferior a 1,4%, apesar de consumirem em média valores bem distintos de corrente. Tal discrepância sugere tanto a presença de intensidades de atrito distintas nos eixos dos propulsores

quanto uma degradação desigual dos motores em função da utilização dos mesmos fora das especificações do fabricante. A segunda hipótese parece preponderante uma vez que, em testes realizados em laboratório, não se observou discrepância tão significativa dos valores de corrente nos motores funcionando em vazio, com mesma referência de rotação.

Com relação ao piloto automático, apesar do veículo ter mantido o rumo durante todo o ensaio sem grandes oscilações observáveis através do gráfico apresentado na Figura 5.39, constata-se grande sensibilidade do esforço de controle (ver Figura 5.43 e valor de desvio médio do ângulo de leme na Tabela 5.9). Isto pôde ser explicado (infelizmente apenas após a realização de todos os ensaios) pela descoberta de um erro de projeto no controlador implementado. Constatou-se que foi negligenciado o integrador do diagrama de blocos da Figura 5.34, o que ocasionou uma alocação de pólos errônea (presença de um pólo no semi-plano positivo, próximo de zero). A manutenção do rumo do veículo, mesmo diante do erro constatado, é atribuída às condições a que foram realizados os ensaios, com o veículo já iniciando a manobra no rumo desejado.

O valor médio do ângulo de leme diferente de zero (ver Tabela 5.9), que se observa em todos os ensaios realizados (ver Tabelas 5.10, 5.11 e 5.12), pode ser explicado tanto pela encontro do veículo com as ondas quanto pela diferença de rotação do propulsores.

Comparando-se os resultados obtidos para as velocidades angulares deste ensaio (ver gráficos das Figuras 5.45, 5.46 e 5.47) com os resultados obtidos nos ensaios de repouso e de vibrações, é possível constatar o efeito da grande sensibilidade do controlador, em especial quando analisa-se os valores de desvios para ω_z e também para ω_x que está acoplado com ω_z . O aumento do valor de desvio para ω_y é esperado em função do encontro das ondas com o veículo.

Quanto aos resultados obtidos para os movimentos de avanço A_x e arfagem A_z (gráficos das Figuras 5.48 e 5.50), tem-se que os mesmos condizem com os valores obtidos respectivamente para $\sin(\bar{\theta} - \theta_d)$ e $\cos(\bar{\theta} - \theta_d)$, apresentando diferenças dentro da faixa de erros das medidas dos sensores (8.5 mg para as medidas de movimento da unidade inercial e 0.2° para o deslocamento angular em *Pitch* da bússola eletrônica).

Observou-se uma ligeira elevação do ângulo de *Pitch*, que pode ser constatada através do gráfico da Figura 5.44, em relação ao obtido com a embarcação em águas calmas (ver gráfico da Figura 5.26). Este fato sugere uma tendência que condiz com o efeito hidrodinâmico esperado para a embarcação em movimento (a proa do navio tende a se elevar com o movimento), apesar

da elevação observada ser pouco maior que o erro de $0,2^\circ$ atribuída à bússola eletrônica por seu fabricante.

Ensaio II

Informações iniciais a respeito do ensaio:

Par de ganhos utilizado no controlador PI: b (ver Tabela 5.8);

Velocidade do veículo: 1,4 m/s;

Rotação de referência para propulsão: 1500 rpm.

Rumo de referência: $309,1^\circ$.

Neste ensaio os valores médios de rotação e corrente nos motores de propulsão ficaram bem próximos dos valores verificados no ensaio anterior. A manutenção do rumo também foi conseguida com ligeira melhora nas oscilações, melhoria esta que refletiu nos valores de desvios médios das velocidades angulares.

Com relação ao esforço de controle, observou-se ainda grande sensibilidade, porém com um intervalo de deslocamento do leme mais condizente com a embarcação real (entre -30° e 30°).

A diferença observada entre o valor médio do movimento de avanço e $\sin(\bar{\theta} - \theta_d)$, assim como do valor médio do movimento de arfagem e $\cos(\bar{\theta} - \theta_d)$ continuam dentro da faixa de erro das medidas dos sensores.

A tendência de elevação do ângulo de *Pitch* também foi observada neste ensaio.

Ensaio III

Informações iniciais a respeito do ensaio:

Par de ganhos utilizado no controlador PI: c (ver Tabela 5.8);

Velocidade do veículo: 1,4 m/s;

Rotação de referência para propulsão: 1500 rpm.

Rumo de referência: $301,6^\circ$.

Figura 5.39: Ângulo de rumo em Ensaio I.

Figura 5.40: Rotação dos motores de propulsão em Ensaio I

Figura 5.41: Corrente instantânea em motor de propulsão em Ensaio I.

Figura 5.42: Corrente instantânea em motor de propulsão em Ensaio I.

Figura 5.43: Esforço de controle em Ensaio I

Figura 5.44: Deslocamentos angulares em Ensaio I

Figura 5.45: Movimento em *Roll* ω_x em Ensaio I.

Figura 5.46: Movimento em *Pitch* ω_y em Ensaio I.

Figura 5.47: Movimento em $Yaw \omega_z$ em Ensaio I.

Figura 5.48: Movimento de avanço A_x em Ensaio I.

Figura 5.49: Movimento lateral A_y em Ensaio I.

Figura 5.50: Movimento de arfagem A_z em Ensaio I.

Tabela 5.10: Resumo dos resultados do Ensaio II.

Variável	Média	Desvio Médio
Rotação no Motor Esquerdo	1496 rpm	15 rpm
Rotação no Motor Direito	1539 rpm	25 rpm
Corrente Instantânea no Motor Esquerdo	4,30 A	0,89 A
Corrente Instantânea no Motor Direito	2,21 A	1,21 A
Rumo	309,4°	0,67°
Ângulo de Leme	-4,47°	9,06°
Deslocamento Angular em <i>Roll</i>	1,30°	1,28°
Deslocamento Angular em <i>Pitch</i>	-3,83°	0,47°
Movimento de Avanço $\overline{A_x}$	-0,0498 g	0,0222 g
Movimento Lateral $\overline{A_y}$	0,0165 g	0,0506 g
Movimento de Arfagem $\overline{A_z}$	-0,9952 g	0,0870 g
$\sin(\bar{\theta} - \theta_d)$	-0,0434	-
$\cos(\bar{\theta} - \theta_d)$	-0,9990	-
Velocidade Angular ω_x	0,2663 graus/s	4,01 graus/s
Velocidade Angular ω_y	-0,1797 graus/s	1,67 graus/s
Velocidade Angular ω_z	0,1500 graus/s	1,79 graus/s

Este ensaio utilizando o último par de ganhos da Tabela 5.8 foi o que apresentou, em relação ao três pares de ganhos utilizados, os melhores resultados para o controle de rumo do veículo. Tal fato pode ser comprovado pela comparação dos valores de desvio médio das velocidades angulares ω_z nas Tabelas 5.9, 5.10 e 5.11.

Ensaio IV

Informações iniciais a respeito do ensaio:

Par de ganhos utilizado no controlador PI: b (ver Tabela 5.8);

Tabela 5.11: Resumo dos resultados do Ensaio III.

Variável	Média	Desvio Médio
Rotação no Motor Esquerdo	1501 rpm	16 rpm
Rotação no Motor Direito	1530 rpm	26 rpm
Corrente Instantânea no Motor Esquerdo	4,05 A	1,02 A
Corrente Instantânea no Motor Direito	2,09 A	1,26 A
Rumo	301,61°	0,45°
Ângulo de Leme	-4,45°	5,39°
Deslocamento Angular em <i>Roll</i>	1,18°	0,97°
Deslocamento Angular em <i>Pitch</i>	-3,78°	0,55°
Movimento de Avanço $\overline{A_x}$	-0,0532 g	0,0233 g
Movimento Lateral $\overline{A_y}$	0,0195 g	0,0495 g
Movimento de Arfagem $\overline{A_z}$	-1,002 g	0,1001 g
$\text{sen}(\overline{\theta} - \theta_d)$	-0,0478	-
$\text{cos}(\overline{\theta} - \theta_d)$	-0,9989	-
Velocidade Angular ωx	0,4470 graus/s	4,10 graus/s
Velocidade Angular ωy	-0,2334 graus/s	2,58 graus/s
Velocidade Angular ωz	0,1746 graus/s	1,30 graus/s

Velocidade do veículo: 2 m/s;

Rotação de referência para propulsão: 2000 rpm.

Rumo de referência: 321,70°.

Neste ensaio, apesar do conhecimento das limitações de torque dos motores de propulsão utilizados, aumentou-se a referência de rotação adotada para os mesmos. Isto elevou a velocidade do modelo em escala de navio, propiciando a coleta de dados em velocidade superior (2 m/s) a alcançada nos ensaios anteriores (1,4 m/s). Como esperado, os motores não alcançaram a rotação desejada e também apresentaram rotações bem diferentes entre si. Este fato dificultou ainda mais as condições para o controle de rumo pelo piloto automático, que mesmo assim não

Tabela 5.12: Resumo dos resultados do Ensaio IV.

Variável	Média	Desvio Médio
Rotação no Motor Esquerdo	1426 rpm	20 rpm
Rotação no Motor Direito	1949 rpm	29 rpm
Corrente Instantânea no Motor Esquerdo	3,70 A	1,37 A
Corrente Instantânea no Motor Direito	4,62 A	0,87 A
Rumo	321,50°	1,19°
Ângulo de Leme	-0,86°	13,55°
Deslocamento Angular em <i>Roll</i>	1,42°	2,15°
Deslocamento Angular em <i>Pitch</i>	-3,88°	0,74°
Movimento de Avanço $\overline{A_x}$	-0,0458 g	0,0265 g
Movimento Lateral $\overline{A_y}$	0,0273 g	0,0542 g
Movimento de Arfagem $\overline{A_z}$	-1,0139 g	0,0890 g
$\text{sen}(\overline{\theta} - \theta_d)$	-0,0496	-
$\text{cos}(\overline{\theta} - \theta_d)$	-0,9987	-
Velocidade Angular ωx	0,0871 graus/s	6,82 graus/s
Velocidade Angular ωy	-0,0610 graus/s	2,09 graus/s
Velocidade Angular ωz	0,1819 graus/s	3,16 graus/s

apresentou oscilações muito grandes (ver Figura 5.51).

Tais oscilações ainda ficaram abaixo das observadas no Ensaio I, como pode ser constatado pelos valores de desvio médio para as velocidades angulares (ver gráficos das Figuras 5.57, 5.58 e 5.59) apresentados na Tabela 5.12.

As medidas fornecidas pela unidade inercial e bússola eletrônica continuam condizentes entre si e a tendência de elevação do deslocamento angular em *Pitch* (ver gráfico da Figura 5.56) não apresentou grande mudança em relação à observada nos ensaios anteriores, apesar da elevação da velocidade de deslocamento do veículo.

Figura 5.51: Ângulo de rumo em Ensaio IV.

Figura 5.52: Rotação dos motores de propulsão em Ensaio IV.

Figura 5.53: Corrente instantânea em motor de propulsão em Ensaio IV.

Figura 5.54: Corrente instantânea em motor de propulsão em Ensaio IV.

Figura 5.55: Esforço de controle em Ensaio IV.

Figura 5.56: Deslocamentos angulares em Ensaio IV.

Figura 5.57: Movimento em *Roll* ω_x em Ensaio IV.

Figura 5.58: Movimento em *Pitch* ω_y em Ensaio IV.

Figura 5.59: Movimento em $Yaw \omega_z$ em Ensaio IV.

Figura 5.60: Movimento de avanço A_x em Ensaio IV.

Figura 5.61: Movimento lateral A_y em Ensaio IV.

Figura 5.62: Movimento de arfagem A_z em Ensaio IV.

5.2.3 Ensaios de Navegação

Estes foram os últimos ensaios realizados neste trabalho visando a validação da arquitetura de controle proposta no capítulo 4.

O objetivo específico destes ensaios foi demonstrar que a estrutura de *software* implementada é capaz de adquirir, em tempo real, as informações do equipamento DGPS (sistema de posicionamento global com correções diferenciais).

O funcionamento do sistema GPS se baseia no princípio da triangulação, segundo o qual o observador conhece a posição de um conjunto de satélites em relação a um referencial inercial e a sua posição em relação a este conjunto, e obtém sua própria posição no sistema de referência.

Figura 5.63: Sistema GPS.

A Figura 5.63 apresenta os parâmetros básicos utilizados pelo GPS na determinação da posição do usuário. Definindo-se $\vec{R}_u$ com sendo a posição do usuário, $\vec{r}_i$ posição do i-ésimo satélite, $\vec{\rho}_i$ posição do usuário em relação ao i-ésimo satélite e admitindo-se $\vec{R}_u = (X_u, Y_u, Z_u)$, tem-se a seguinte relação:

$$2.(X_u - X_i) + 2.(Y_u - Y_i) + 2.(Z_u - Z_i) = 2.\vec{\rho}_i \quad (5.10)$$

Cada satélite i transmite sua posição (X_i, Y_i, Z_i) e o instante de transmissão T_0 .

O usuário possui um receptor que mede os intervalos de tempo de propagação Δt decorridos a partir da transmissão do sinal pelo i-ésimo satélite:

$$\Delta t_i = T_i - T_0 \quad (5.11)$$

Considerando uma perfeita sincronização dos relógios do receptor GPS e do i -ésimo satélite, e desprezando os efeitos de distorção da ionosfera, efeitos relativísticos e outros, têm-se:

$$\rho_i = c \cdot \Delta t_i \quad (5.12)$$

sendo c a velocidade da luz.

Se há desvios de sincronização dos relógios, tem-se:

$$\rho_i = p_i - b_u \quad (5.13)$$

sendo p_i a pseudo-distância do usuário ao i -ésimo satélite e b_u o erro correspondente ao desvio dos relógios.

Assim, é necessário dados de quatro satélites observados simultaneamente para se obter um sistema de quatro equações, e determinar X_u , Y_u , Z_u e b_u . Neste ponto é importante ressaltar que, dependendo da geometria relativa dos satélites, o sistema de equações pode não ter solução. Além disso, se mais de quatro satélites são observados simultaneamente, existe um conjunto de quatro que fornece a solução com menor erro, e cabe ao receptor do usuário determiná-lo (Paz, 1997).

O princípio de funcionamento do GPS Diferencial (DGPS), consiste em compensar os erros do sistema através de correções nas pseudodistâncias, coordenadas de posição ou outras variáveis de interesse. Para isto, além do receptor GPS móvel, isto é, que se locomove pelos pontos cujas coordenadas se deseja determinar, utiliza-se um outro receptor GPS, chamado de referência, que permanece fixo num ponto cuja posição é bem conhecida. Esse receptor de referência, utilizando os sinais que recebe dos satélites, determina a sua posição sujeita a todos os erros presentes no sistema GPS tais como: atrasos de propagação dos sinais na ionosfera, atrasos por refração dos sinais na troposfera, desvios nos relógios dos satélites, desvios no relógio do receptor, erros devidos a trajetórias múltiplas dos sinais, diluição da precisão, erros devidos a disponibilidade seletiva, dentre outros (Parkinson e Spilker Jr., 1996). Comparando-a com a sua posição real, previamente conhecida, ele pode determinar, a cada instante, o erro a que está sujeito o sinal

enviado por cada satélite que ele avista. Dessa forma, o receptor de referência pode corrigir os erros dos sinais captados pelo receptor móvel, assumindo que esses erros sejam os mesmos.

Os ensaios com o equipamento DGPS deste trabalho consistiram da execução de uma manobra pré-programada (no caso o giro) pelo modelo ITTC, enquanto eram adquiridas as informações de posicionamento do mesmo através do receptor GPS embarcado. As correções diferenciais do receptor GPS de referência, previamente instalado e configurado em laboratório, eram enviadas para o *notebook* do operador através de uma rádio UHF, que então as retransmitiam para o PC104 embarcado no veículo por meio da rede *wireless* LAN utilizada em princípio para monitoramento.

As Figuras 5.65 e 5.66 apresentam a configuração de *hardware* utilizada nestes ensaios.

Os dados de posicionamento fornecidos pelo receptor GPS móvel são referenciados ao sistema de coordenadas ECEF (*Earth-Centered, Earth-Fixed*), cuja origem é o centro da terra (sistema de coordenadas xyz da Figura 5.64). No caso dos ensaios realizados neste trabalho, desejou-se referenciar a posição do veículo em relação a um sistema de coordenadas ENU (*East-North-Up*) posicionado nas coordenadas da antena do receptor GPS de referência (sistema de coordenadas $x^1y^1z^1$ da Figura 5.64).

Figura 5.64: Sistema de coordenadas ECEF e ENU.

Faz-se então necessário a rotação e translação dos dados para o sistema de coordenadas que se deseja utilizar. Isto é feito através da equação:

Figura 5.65: Vista superior o modelo em escala de navio.

Figura 5.66: Detalhe mostrando antena e receptor GPS, PC104 e motores de propulsão.

$$\begin{pmatrix} x^1 \\ y^1 \\ z^1 \end{pmatrix} = \begin{pmatrix} -\text{sen}\lambda & \cos\lambda & 0 \\ -\text{sen}\varphi.\cos\lambda & -\text{sen}\varphi.\text{sen}\lambda & \cos\varphi \\ \cos\varphi.\cos\lambda & \cos\varphi.\text{sen}\lambda & \text{sen}\varphi \end{pmatrix} \cdot \begin{pmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{pmatrix} \quad (5.14)$$

onde

φ : latitude da origem do novo sistema de coordenadas no ECEF;

λ : longitude da origem do novo sistema de coordenadas no ECEF;

(x_0, y_0, z_0) : coordenadas da antena do receptor GPS de referência no ECEF;

A seguir são apresentados os resultados experimentais obtidos nos ensaios com o modelo ITTC utilizando o equipamento DGPS. Tais resultados foram organizados da seguinte forma: inicialmente são apresentadas informações iniciais a respeito do ensaio, seguido dos gráficos de dados obtidos, finalizando-se a seção com uma breve análise dos resultados. Para melhor visualização gráfica dos resultados, efetuaram-se as medições em relação à posição de início de cada manobra realizada, com um sistema de coordenadas paralelo ao posicionado na antena do receptor GPS de referência. Assim, admitiu-se que as latitudes e longitudes destes dois pontos (posição da antena de referência e da origem da manobra) são praticamente as mesmas, não sendo efetuada a multiplicação dos dados pela matriz de rotação entre os dois sistemas de coordenadas.

Ensaio 1

Informações iniciais a respeito do ensaio:

Manobra padrão: *giro*;

Velocidade do veículo: 1,8 m/s;

Rotação de referência para propulsão: 1500 rpm;

Ângulo de leme: 20°;

Ângulo de entrada na manobra em relação às ondas: $\cong 90^\circ$.

Ensaio 2

Informações iniciais a respeito do ensaio:

Manobra padrão: *giro*;

Velocidade do veículo: 2 m/s;

Rotação de referência para propulsão: 2000 rpm;

Ângulo de leme: 20°;

Ângulo de entrada na manobra em relação às ondas: $\cong 90^\circ$.

Ensaio 3

Informações iniciais a respeito do ensaio:

Manobra padrão: *giro*;

Velocidade do veículo: 2 m/s;

Rotação de referência para propulsão: 2000 rpm;

Ângulo de leme: 10°;

Ângulo de entrada na manobra em relação às ondas: $\cong 90^\circ$.

Em todos os três ensaios realizados com o equipamento DGPS foi adquirido os dados de posicionamento do veículo com sucesso, a uma taxa de 5Hz, sem nenhuma interrupção provocada por perda dos sinais dos satélites.

Como é possível observar nos gráficos apresentados nas Figuras 5.67, 5.69 e 5.70, o veículo descreveu uma trajetória elíptica em vez de circular. Atribui-se esse desvio à presença de ventos e ondas durante os ensaios, podendo-se inclusive estimar o deslocamento das ondas através dos gráficos apresentados. Por exemplo, no gráfico da Figura 5.67 estima-se um deslocamento das ondas mais acentuado no sentido positivo do eixo x, juntamente com ligeiro deslocamento no sentido negativo do eixo y.

Já no gráfico da Figura 5.68 observa-se um problema ocorrido apenas nestes últimos ensaios que foi a presença de ruído nas informações da bússola eletrônica. Como a única alteração

realizada na configuração de equipamentos utilizada nestes ensaios foi a substituição da unidade inercial pelo receptor e antena GPS, a maior suspeita para causa de tais ruídos é a presença da antena GPS colocada nas proximidades da bússola eletrônica.

Figura 5.67: Posição X e Y do veículo em Ensaio 1.

Figura 5.68: Ângulo de rumo em Ensaio 1.

Figura 5.69: Posição X e Y do veículo em Ensaio 2.

Figura 5.70: Posição X e Y do veículo em Ensaio 3.

Capítulo 6

Conclusões e Sugestões para Trabalhos Futuros

Em seu capítulo 2, o presente trabalho tratou das arquiteturas de controle para robôs móveis autônomos, apresentando para isto uma breve visão geral dos diversos paradigmas que estiveram presentes no desenvolvimento de robôs móveis autônomos, em particular, dos veículos autônomos submarinos. Através deste estudo, constatou-se uma tendência, no que tange às implementações de arquitetura de controle destinadas a robôs móveis autônomos na atualidade, para a utilização de abordagens híbridas buscando com isto as vantagens associadas a este tipo de arquitetura, tais como uma representação bem definida da estrutura e, em especial, uma boa flexibilidade.

A partir da constatação acima, foi proposta uma arquitetura estruturada em uma comunidade de agentes, cada qual com responsabilidades específicas dentro da arquitetura, mas que cooperam entre si através de troca de mensagens. Como isto, a flexibilidade atribuída às arquiteturas híbridas é refletida nesta proposta de arquitetura por meio da fácil inclusão de novos agentes à estrutura (acréscimo de novos sensores ou atuadores ao veículo, por exemplo), assim como pela fácil substituição dos componentes do veículo, que se apóia na generalidade com que são desenvolvidos os agentes da arquitetura de controle. Tal flexibilidade é particularmente vantajosa e bem vinda em implementação de aplicações robóticas em geral, onde se tem constantemente a necessidade de substituição de componentes de *hardware* e *software*, assim como a utilização de um mesmo componente em diferentes aplicações.

A desvantagem das arquiteturas híbridas, apontada pelas referências bibliográficas (Valavanis et al., 1997; Ridao et al., 1999), é também um tema fecundo para trabalhos futuros. A verificação formal de sistemas que utilizem abordagens híbridas, considerado de difícil realização, constitui um assunto que necessita maiores desenvolvimentos.

O estudo dos sistemas de tempo real, a identificação de suas principais características, em especial daquelas que os diferenciam dos demais sistemas em geral, assim como das implicações destas características específicas em implementações de arquiteturas de controle destinadas a robôs móveis autônomos foram tratados no capítulo 3. Foram apresentadas no referido capítulo quatro características: a resposta temporal, que constitui a característica que melhor diferencia os sistemas de tempo real e que se refere à capacidade de fornecer resultados não somente corretos mas também no tempo apropriado; a simultaneidade, que representa a capacidade do sistema de receber diversas entradas e responder a cada uma delas em determinados limites de tempo; a previsibilidade, que é a capacidade de se conhecer o comportamento futuro de um sistema através de suas especificações de projeto; e, finalmente, a dependabilidade, que indica um requisito geral de confiança de um sistema.

Analizando a abordagem de arquitetura proposta neste trabalho em relação às características mencionadas acima, tem-se que a estrutura multi-agentes escolhida já pressupõe a simultaneidade entre os diferentes elementos da estrutura. Isto traz também implicações diretas na resposta temporal do sistema, um vez que se divide os requisitos temporais do mesmo entre os diversos módulos que compõem a arquitetura, módulos estes que executam de maneira paralela através de multiprogramação e processamento paralelo em processadores distintos.

Já a previsibilidade e a dependabilidade representam dois temas a serem mais bem explorados em trabalho futuros. Por um lado a utilização de metodologias de desenvolvimento que contemplem análises mais aprofundadas dos requisitos temporais de um sistema podem gerar melhores abordagens para a previsibilidade. Por outro lado, a busca de sistema mais robustos que contemple a tolerância a falhas, assim como o desenvolvimento de técnicas de detecção e recuperação de erros propiciam campo de estudo para a abordagem da dependabilidade.

O capítulo 4 tratou da modelagem baseada em objetos, buscando aliar os conceitos de sistemas multi-agentes presentes na proposta de arquitetura apresentada no capítulo 2, com a utilização de metodologias de desenvolvimento orientado a objetos no projeto e implementação dessa arquitetura. Procurou-se, então, apresentar uma modelagem da arquitetura proposta

através da UML, Linguagem de Modelagem Unificada.

A escolha da UML deveu-se à constatação de que a mesma figura como um padrão de fato no desenvolvimento de sistemas orientados a objetos, possuindo inclusive proposta de extensão de sua base conceitual destinadas especificamente para a aplicação em sistemas de tempo real (Douglass, 2000) e sistemas multi-agentes (Odell et al., 2000; Bauer, 2001).

Dentro do conjunto de diagrama presente na base conceitual da UML, utilizaram-se três grupos de diagramas para a representação de aspectos distintas da arquitetura modelada. Inicialmente a modelagem apresentou os aspectos estruturais da estrutura, os quais foram representados em diagramas de classes, devidamente adaptados para a ilustração da noção de agentes presente na idéia central da estrutura.

Através de diagramas de gráfico de estados e de atividades foi representado o comportamento interno de cada agente da arquitetura, sendo possível à ilustração das características de tempo real do sistema.

O último grupo de diagramas utilizados na modelagem representou as interações entre os agentes da arquitetura fazendo uso de diagramas de colaboração e de seqüência. Estes últimos diagramas, dentre os demais utilizados na modelagem aqui realizada, são os mais indicados para, em trabalhos futuras, realizar-se as análises de previsibilidade anteriormente sugeridas neste capítulo.

Dentre as vantagens comumente atribuídas à utilização da UML no desenvolvimento de sistemas de *software*, as que mais marcadamente foram sentidas no processo de projeto e implementação realizados neste trabalho são: a possibilidade de se enxergar o sistema sob diferentes aspectos, antecipando-se problemas a serem enfrentados e minimizando a reconstrução de código tão comum na implementação de *software*; melhor organização do código implementado, que facilitou a manipulação e documentação das diversas bibliotecas e executáveis que compõem a arquitetura desenvolvida.

O capítulo 5 apresentou as implementações realizadas neste trabalho para a validação da arquitetura proposta, sendo descritos as estruturas de *hardware* e *software* utilizadas, os ensaios realizados e a análise dos resultados experimentais obtidos.

Dentro das limitações do *hardware* escolhido (torque dos motores, taxa de amostragem máxima dos sensores, além dos erros de suas medidas), as análise dos resultados experimentais foram condizentes com as expectativas iniciais, o que confere, além da validação do arcabouço

de *hardware* e *software* implementado, possibilidades promissoras de desenvolvimentos futuros para a arquitetura controle proposta em áreas como fusão sensorial, guiagem de robôs móveis autônomos, dentre outros.

Como sugestões para trabalhos futuras visando melhorar a arquitetura implementada pode-se citar:

- O piloto automático implementado apresentou grande sensibilidade em seu esforço de controle. Faz-se necessário aperfeiçoar o controlador e realizar mais ensaios para a melhor sintonia do mesmo. Tais atividades não foram realizadas em virtude da escassez de tempo para a execução de novos ensaios;
- Os dados de velocidade angular fornecidos pela unidade inercial VG600AA no ensaio em repouso apresentaram erros superiores aos informados pelo fabricante, sendo desejável a realização de mais ensaios com este equipamento em separado visando uma aferição mais precisa de seus erros;
- O ensaio de vibrações foi realizado em bancada e apenas para um valor de rotação dos propulsores. É interessante para análises futuras a realização de mais ensaios, desta vez com o modelo ITTC na água, em *bollard-pull* (preso por um cabo, evitando seu deslocamento longitudinal), para diversas rotações dos propulsores;
- Pela análise inicial, observou-se que as vibrações mencionadas no item anterior influenciam nas medidas fornecidas pela unidade inercial. Faz-se necessário o aperfeiçoamento das linhas de eixo dos propulsores, assim como a utilização de algum tipo de amortecimento nos mancais de escora dos motores buscando minimizar tal influência;
- Nos ensaios de navegação constatou-se a presença de ruídos nos dados de deslocamento angular da bússola eletrônica, ruídos estes atribuídos à antena GPS fixada nas proximidades desta bússola. É necessário o rearranjo dos equipamentos embarcados de forma a afastar este sensor de fontes de interferência eletromagnéticas (motores, *drives*, antenas, etc);
- Em função da limitação dos motores de propulsão, não foi possível a realização de ensaios com o modelo ITTC navegando com velocidade superior a 2 m/s. É necessário o

aperfeiçoamento do sistema de propulsão de forma que seja possível repetir os ensaios para uma gama maior de velocidades, o que possibilitaria melhores análises tanto para os estudos de dinâmica do modelo ITTC, como para demais aperfeiçoamentos da estrutura de *hardware* e *software* implementada.

Referências Bibliográficas

- Amat, J., Monferrer, A., Batlle, J. e Cufi, X. (1999). GARBI: a low-cost underwater vehicle, *Microprocessors and Microsystems* **23**(2): 61–63.
- Audsley, N. e Burns, A. (1990). Real-time system scheduling, *Technical Report YCS 134*, Department of Computer Science, University of York, York, UK.
- Awad, M., Kuusela, J. e Ziegler, J. (1996). *Object-Oriented Technology for Real-Time Systems*, Prentice Hall, NJ.
- Badr, S. M., Byrnes, R. B., Brutzman, D. P. e Nelson, M. L. (1992). Real-time systems, *Technical Report NPSCS-92-004*, Naval Postgraduate School, Monterey, California.
- Bauer, B. (2001). UML class diagrams revisited in the context of agent-based systems, *em* M. Wooldridge, P. Ciancarini e G. Weiss (eds), *Proceedings of Agent-Oriented Software Engineering (AOSE 2001)*, número 2222, FIPA, Springer-Verlag, Montreal, Canada, pp. 1–8.
- URL:** <http://www.auml.org/auml/working/>
- Bauer, B., Müller, J. P. e Odell, J. (2001). Agent UML: A formalism specifying multiagent interaction, *em* P. Ciancarini e M. Wooldridge (eds), *Agent-Oriented Software Engineering*, Springer-Verlag, Berlin, Germany, pp. 91–103.
- Bellingham, J. G. e Consi, T. R. (1990). State configured layered control, *Proceedings of the IARP 1st Workshop on: Mobile Robots for Subsea Environments*, número 23-26, All IARP, IARP, Monterey, CA, pp. 75–80.
- Booch, G. (1993). *Object-Oriented Analysis and Design with Applications*, 2a. edição, Addison-Wesley Publishing Company, Redwood City, Califórnia.

- Booch, G., Rumbaugh, J. e Jacobson, I. (1999). *The Unified Modeling Language User Guide*, The Addison-Wesley object technology series, Addison-Wesley Publishing Company, Reading Mass.
- Corrêa, E. e Friedrich, L. F. (1997). Padronização POSIX e sistemas operacionais de tempo-real: Uma análise comparativa, *III Congreso Argentino de Ciencias de la Computación - CACIC'97*, RedUNCI, La Plata, Argentine.
- De Barros, E. A. (1994). *A Cooperative Control System and Its Application to the Collision Avoidance Guidance of Autonomous Underwater Vehicles*, PhD thesis, University of Tokyo, Tokyo, Japan.
- De Barros, E. A. e Ferreira, G. A. N. (2000). Plataformas de testes aplicadas ao desenvolvimento de robôs móveis oceânicos, *Anais do XIII Congresso Brasileiro de Automática*, Anais do CBA, UFSC, SBA, Florianópolis/SC, Brasil, pp. 2144–2149.
- De Barros, E. A. e Ferreira, G. A. N. (2001). Test bed development for research on performance of high speed vessels, em M. Blanke, M. M. A. Pourzanjani e Z. Z. Vukic (eds), *Proceedings of 5th IFAC Conference on Manoeuvring and Control of Marine Crafts*, número 23-25 em *All IFAC Conferences*, Aalborg University, Aalborg University, Aalborg, Denmark, pp. 239–244.
- De Barros, E. A., Maeda, H. e Miyajima, S. (1995). Application of a cooperative control system to the collision avoidance guidance of AUVs, *Journal of The Society of Naval Architects of Japan* **178**: 688–697.
- De Hollanda, G. M. (1993). Normas e terminologia na área da confiabilidade, *Anais do V Simpósio de Computadores Tolerantes a Falhas*, São José dos Campos, SP, Brasil.
- DIN 44 300 A2: Informationsverarbeitung* (1985). Deutsches Institut für Normung e.V., Berlin-Cologne.
- Douglass, B. P. (2000). *Real-Time UML: Developing Efficient Objects for Embedded Systems*, The Addison-Wesley object technology series, 2a. edição, Addison-Wesley Publishing Company, NJ.

- Farines, J. M., Fraga, J. S. e De Oliveira, R. S. (2000). *Sistemas de Tempo Real*, Vol. 1, Escola de Computação 2000, IME/USP, São Paulo, SP.
- Fikes, R. E., Hart, P. E. e Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* **2**(3-4): 189–208.
- Franklin, S. e Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents, *em* J. P. Müller, M. J. Wooldridge e N. R. Jennings (eds), *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, Budapest, Hungary.
- URL:** <http://citeseer.nj.nec.com/franklin96is.html>
- Halang, W. A. (1992). Real-time systems: Another perspective, *The Journal of Systems and Software* pp. 101–108.
- Hall, W. D. e Adams, M. B. (1992). Autonomous vehicles software taxonomy, *Proc. of the IEEE Symposium on Autonomous Underwater Vehicle Technology*, The OES of the IEEE, IEEE, pp. 49–64.
- Heimerdinger, W. L. e Weinstock, C. B. (1992). A conceptual framework for system fault tolerance, *Technical Report CMU/SEI-92-TR-33*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- Henrickson, M. F. e Dzielski, J. E. (1995). Control system architecture for an underwater autonomous testbed, *Conference Proceedings OCEANS' 95*, Vol. 2, The MTS of IEEE, IEEE, pp. 1011–1018.
- Hoogeboom, B. e Halang, W. A. (1992). The concept of time in the specification of real time systems, *em* K. M. Kavi (ed.), *Real-Time Systems: abstractions, languagens, and design methodologies*, IEEE Computer Society Press, Los Alamitos, CA, pp. 19–38.
- Jacobson, I., Christerson, M., Jonsson, P. e Overgaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Publishing Company, Wokingharn, Inglaterra.
- Kirner, T. G. (1997). Quality requirements for real-time safety-critical systems, *Control Eng. Practice* **5**(7): 965–973.

- Kolnick, F. (1998). *The QNX 4 Real-time Operating System*, 1a. edição, Basis Computer Systems Inc., Canada.
- Kurki-Suonio, R. (1994). Real time: Further misconceptions (or half-truths), *IEEE Computer* **27**(6): 71–76.
- Kwak, S. e Thornton Jr, F. P. B. (1994). A concurrent, object-oriented implementation for tactical level of the rational behavior model software architecture for UUV control, *Proc. of the IEEE Symposium on Autonomous Underwater Vehicles Technology*, número 19-20, The OES of the IEEE, IEEE, Cambridge, Massachusetts, USA, pp. 54–60.
- Löffler, M. S., Chitrakaran, V. K. e Dawson, D. M. (2001). Design and implementation of the robotic platform, *Proc. of the IEEE Conference on Control Applications*, IEEE Control Systems Society (CSS), Mexico City, Mexico, pp. 357–362.
- López-Orozco, J. A., De la Cruz, J. M., Domínguez, E., Besada, E. e Polo, O. R. (1998). An open sensing architecture to autonomous mobile robots, *Proceedings of the 1998 IEEE International Symposium on Intelligent Control*, IEEE, Gaithersburg, MD, USA, pp. 610–615.
- Madsen, H. O., Christensen, P. e Lauridsen, K. (2000). Securing the operational reliability of an autonomous mini-submarine, *Reliability Engineering and System Safety* **68**(1): 7–16.
- Marco, D. B., Healey, A. J. e McGhee, R. B. (1996). Autonomous underwater vehicles: Hybrid control of mission and motion, *Journal of Autonomous Robots* **3**: 169–186.
- Matijevic, J. (1997). Sojourner - The Mars Pathfinder Microrover Flight Experiment, *Technical Report 2315*, Jet Propulsion Laboratory - California Institute of Technology.
- Müller, J. P. (1996). *The design of intelligent agents: a layered approach*, Vol. 1177 of *Lecture notes in artificial intelligence*, Springer-Verlag, Berlin, Germany.
- Muir, P. F. e Horner, J. W. (1998). Mechatronic objects for real-time control software development, em P. E. Muir, Patrick F.; Orban (ed.), *Proc. of the SPIE: Sensors and Controls for Intelligent Machining, Agile Manufacturing, and Mechatronics*, número 3518, SPIE: The International Society for Optical Engineering, Proceedings of the SPIE, Boston, USA, pp. 252–265.

- Nomoto, K., Taguchi, K., Honda, K. e Hirono, S. (1957). On the steering qualities of ships, *International Shipbuilding Progress* 4(35): 354–370.
- Odell, J., Parunak, H. e Bauer, B. (2000). Extending UML for agents, *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence*, AOIS, Austin, TX, pp. 3–17.
URL: <http://citeseer.nj.nec.com/odell00extending.html>
- O'Reilly, T. C. (2000). An object oriented approach to AUV software development, *Proc. of the Unmanned Underwater Vehicle Showcase 2000*, Southampton, UK.
- Panzieri, F. e Davoli, R. (1993). Real time systems: A tutorial, *Technical Report UBLCS-93-22*, University of Bologna, Bologna, Italy.
- Parkinson, B. W. e Spilker Jr., J. J. (1996). *The global positioning system: theory and applications*, Vol. I, American Institute of Aeronautics and Astronautics, Washington, DC.
- Paz, S. M. (1997). *Uma ferramenta para desenvolvimento de equipamentos que utilizem um receptor do sistema de posicionamento global (gps)*, Dissertação de Mestrado, EPUSP, São Paulo.
- Piveropoulos, M. (2000). *Requirements Engineering for Hard Real-Time Systems*, PhD thesis, The University of York, York, England.
- Pronk, C. e Tercero, C. (1994). Methods and techniques for object-oriented analysis and design, *Technical Report 94-107*, Delft University of Technology, Delft, Netherlands.
- QNX (1997). *QNX Operating System - System Architecture (For QNX 4.24)*, 2a. edição.
- Ramamritham, K. e Stankovic, J. A. (1994). Scheduling algorithms and operating systems support for real-time systems, *Proceeding of the IEEE*, número 82(1), IEEE, pp. 55–67.
- Reid, S. e Valley, H. (1996). Operational use of the Pioneer unmanned aerial vehicle (UAV), *SPIE Proceedings: Airborne Reconnaissance XX*, Vol. 2829, SPIE, SPIE, Denver, CO, USA, pp. 123–130.

- Ridao, P., Batlle, J., Amat, J. e Roberts, G. N. (1999). Recent trends in control architectures for autonomous underwater vehicles, *International Journal of Systems Science* **30**(9): 1033–1056.
- Rodseth, O. J. (1991). Object oriented software system for AUV control, *Engineering Applications of Artificial Intelligence* **4**(4): 269–277.
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. e Lorensen, W. (1991). *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ.
- Rumbaugh, J., Jacobson, I. e Booch, G. (1999). *The Unified Modeling Language Reference Manual*, The Addison-Wesley object technology series, Addison-Wesley Publishing Company, Reading Mass.
- Russel, S. e Norvig, P. (1995). *Artificial intelligence: a modern approach*, 1a. edição, Prentice-Hall, Inc., New Jersey, USA.
- Selic, B. (1999). Turning clockwise: Using UML in the real-time domain, *Communications of the ACM* **42**(10): 46–54.
- Selic, B., Gullekson, G. e Ward, P. T. (1994). *Real-Time Objected-Oriented Modeling*, John Wiley, New York.
- Singh, A. D. e Murugesan, S. (1990). Fault-tolerant systems, *IEEE Computer* **23**(7): 15–17.
- Smith, S. M. (1994). An approach to intelligent distributed control for autonomous underwater vehicles, *Proceedings of the IEEE Symposium on Autonomous Underwater Vehicles Technology*, Vol. 19-20, The OES of the IEEE, IEEE, Cambridge, Massachusetts, USA, pp. 105–111.
- Stankovic, J. A. (1992). Real-time computing, *Byte* **17**(8): 155–160.
- Stankovic, J. A., Spuri, M., Di Natale, M. e Buttazzo, G. (1995). Implications of classical scheduling results for real-time systems, *IEEE Computer* **28**(6): 16–25.
- Valavanis, K. P., Gracanin, D., Matijasevic, M., Kolluru, R. e Demetriou, G. A. (1997). Control architectures for autonomous underwater vehicles, *IEEE Control Systems Magazine* **17**(6): 48–64.

- Veríssimo, P. e De Lemos, R. (1989). Confiança no funcionamento: Proposta para uma terminologia em português, *Technical Report RT/48-89*, INESC, Lisboa, Portugal.
- Wang, H. H. e Rock, S. M. (1996). OTTER: The design and development of an intelligent underwater robot, *Journal of Autonomous Robots* **3**(2-3): 297–320.
- Wooldridge, M. (1999). Intelligent agents, em G. Weiss (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, Cambridge, MA, USA, pp. 27–78.
- URL:** citeseer.nj.nec.com/wooldridge95intelligent.html
- Yoerger, D. R., Bradley, A. M., Cormier, M. H., Ryan, W. B. F. e Walden, B. B. (1999). High resolution mapping of a fast spreading mid ocean ridge with the autonomous benthic explorer, *Proceedings of the 11th International Symposium on Unmanned Untethered Submersible Technology*, Durham, NH.
- Zheng, X., Jackson, E. e Kao, M. (1990). Object-oriented software architecture for mission configurable robots, *Proc. of the IARP 1st Workshop on: Mobile Robots for Subsea Environments*, All IARP, IARP, pp. 63–73.