UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Uma Arquitetura Distribuída de Hardware e Software para Controle de um Robô Móvel Autônomo

Ricardo de Sousa Britto

Orientador: Prof. Dr. Adelardo Adelino Dantas de Medeiros

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Computação da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Número de ordem PPgEE: M208
Natal, RN, Janeiro de 2008

Divisão de Serviços Técnicos

Catalogação da publicação na fonte. UFRN / Biblioteca Central Zila Mamede

Britto, Ricardo de Sousa.
Uma arquitetura distribuída de hardware e software para controle de um robô móvel autônomo - Natal, RN, 2008
84 p.

Orientador: Adelardo Adelino Dantas de Medeiros

Dissertação (mestrado) - Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica.

1. Arquitetura de computadores - Dissertação. 2. Arquitetura de controle - Dissertação. 3. Barramento CAN - Dissertação. 4. Paradigma híbrido deliberativo-relativo - Dissertação. I. Medeiros, Adelardo Adelino Dantas de. II. Universidade Federal do Rio Grande do Norte. III. Título.

RN/UF/BCZM CDU 681.3.02(043.3)

Aos meus pais, por todo suporte,
amor e carinho recebido durante
minha vida e principalmente durante
os dois anos passados no
desenvolvimento deste trabalho.

Agradecimentos

Ao meu orientador e ao meu co-orientador, professores Adelardo e Pablo, por toda a orientação recebida para o desenvolvimento deste trabalho. Em vocês não tenho apenas orientadores, mas também amigos, que espero ficarem para o resto da vida.

A todos meus amigos da colônia piauiense em Natal, que muito ajudaram em todos os momentos vividos nesses dois anos.

A todos os amigos do LAR, que foram de fundamental importância.

Aos demais colegas de pós-graduação, pelas críticas e sugestões.

À minha família pelo apoio durante esta jornada.

À CAPES, pelo apoio financeiro.

Resumo

Neste trabalho é apresentada uma arquitetura de hardware e software para controle do robô móvel autônomo Kapeck. O hardware do robô Kapeck é composto por um conjunto de sensores e atuadores organizados em um barramento de comunicação CAN. Dois computadores embarcados e oito placas microcontroladas foram utilizadas no sistema. Um dos computadores foi utilizado para o sistema de visão, devido à grande necessidade de processamento deste tipo de sistema. O outro computador foi utilizado para coordenar e acessar o barramento CAN e realizar as outras atividades do robô. Placas microcontroladas foram utilizadas nos sensores e atuadores. O robô possui esta configuração distribuída para um bom desempenho em tempo-real, onde os tempos de resposta e a previsibilidade temporal do sistema são importantes. Foi seguido o paradigma híbrido deliberativo-reativo para desenvolver a arquitetura proposta, devido à necessidade de aliar o comportamento reativo da rede de sensores-atuadores com as atividades deliberativas necessárias para realizar tarefas mais complexas.

Palavras-chave: Arquitetura de Controle, Barramento CAN, Paradigma Híbrido Deliberativo - Reativo.

Abstract

In this work, we present a hardware-software architecture for controlling the autonomous mobile robot Kapeck. The hardware of the robot is composed of a set of sensors and actuators organized in a CAN bus. Two embedded computers and eigth microcontroller-based boards are used in the system. One of the computers hosts the vision system, due to the significant processing needs of this kind of system. The other computer is used to coordinate and access the CAN bus and to accomplish the other activities of the robot. The microcontroller-based boards are used with the sensors and actuators. The robot has this distributed configuration in order to exhibit a good real-time behavior, where the response time and the temporal predictability of the system is important. We adopted the hybrid deliberative-reactive paradigm in the proposed architecture to conciliate the reactive behavior of the sensors-actuators net and the deliberative activities required to accomplish more complex tasks.

Keywords: Control Architecture, CAN Bus, Hybrid Deliberative-Reactive Paradigm.

Sumário

Sumário i

Lista de Figuras iii

Lista de Tabelas v

1 Introdução 1
- 1.1 Caracterização do Problema . . . 2
- 1.2 Motivação . . . 3
- 1.3 Objetivos . . . 3
- 1.4 Organização do Texto . . . 3

2 Arquiteturas de Controle de Sistemas Robóticos 5
- 2.1 Paradigmas de Arquiteturas de Controle . . . 5
  - 2.1.1 Paradigma Hierárquico . . . 5
  - 2.1.2 Paradigma Reativo . . . 6
  - 2.1.3 Paradigma Híbrido Deliberativo/Reativo . . . 7
- 2.2 Exemplos de Arquiteturas de Controle . . . 9
  - 2.2.1 NASREM . . . 9
  - 2.2.2 AuRa . . . 9
  - 2.2.3 DAMN . . . 11
  - 2.2.4 COHBRA . . . 11
  - 2.2.5 Arquitetura de Tangirala . . . 13
  - 2.2.6 Arquitetura de Liu . . . 13
- 2.3 Conclusões . . . 15

3 Tecnologias Utilizadas em Arquiteturas 17
- 3.1 Processamento Distribuído e Processamento Centralizado . . . 17
  - 3.1.1 Processamento Centralizado . . . 17
  - 3.1.2 Processamento Distribuído . . . 17
- 3.2 Protocolos de Comunicação . . . 20
  - 3.2.1 Protocolo Ethernet . . . 20
  - 3.2.2 Protocolo CAN . . . 21
- 3.3 Sistemas Operacionais . . . 23
- 3.4 Tecnologias para Implementação de Software . . . 24
- 3.5 Conclusões . . . 24

4 Arquitetura Proposta **27**
4.1 Arquitetura de Hardware 27
4.2 Arquitetura de Software 31
4.2.1 Blackboards 31
4.2.2 Periodic 33
4.2.3 Robot 34
4.3 Conclusões 36

5 Organização de Controle Proposta **41**
5.1 Módulos da Organização de Controle 42
5.1.1 Localizer 42
5.1.2 Cartographer 42
5.1.3 actionPl 43
5.1.4 pathPl 45
5.1.5 trajExec 45
5.1.6 posCon 45
5.1.7 visionSys 47
5.2 Exemplos de Execução da Organização de Controle 47
5.2.1 Cenário 1 47
5.2.2 Cenário 2 49
5.2.3 Cenário 3 51
5.3 Conclusões 54

6 Resultados **55**
6.1 Descrição do Protótipo 55
6.2 Experimento 55
6.2.1 Execução da Tarefa 56
6.2.2 Resultados Adicionais 57
6.3 Conclusões 61

7 Conclusões **63**

Referências bibliográficas **65**

Lista de Figuras

2.1 Arquitetura NASREM 10
2.2 Arquitetura AuRa 10
2.3 Arquitetura DAMN 11
2.4 Arquitetura COHBRA 12
2.5 Arquitetura de Tangirala 13
2.6 Arquitetura de Liu 14

3.1 Processamento Centralizado 18
3.2 Sistema Multiprocessado Utilizando mais de uma CPU em Plataforma Única 18
3.3 Sistema Multiprocessado Utilizando uma CPU por Plataforma 20
3.4 Sistema Multiprocessado Utilizando mais de uma CPU por Plataforma . . 20

4.1 Robô Kapeck 27
4.2 Arquitetura de Hardware 29
4.3 Diagrama de blocos do funcionamento da placa de controle de motor . . . 30
4.4 Diagrama de blocos do funcionamento da placa de controle de sonar . . . 30
4.5 Utilização da Classe bBoard 31
4.6 Utilização de bBoardRe, bBoardAtt e bBoardServer 32
4.7 Arquitetura do Mecanismo Robot 35

5.1 Organização de Controle 41
5.2 Ligações do Módulo Localizer 42
5.3 Ligações do Módulo Cartographer 43
5.4 Um Exemplo de Mapa Exportado pelo Cartographer 44
5.5 Ligações dos Módulos actionPl e pathPl 44
5.6 Ligações do Módulo trajExec 46
5.7 Ligações do Módulo posCon 46
5.8 Ligações do visionSys 47
5.9 Cenário 1 48
5.10 Grade de Execução dos Módulos de Software em Função do Tempo . . . 50
5.11 Cenário 2 50
5.12 Grade de Execução dos Módulos de Software em Função do Tempo . . . 52
5.13 Cenário 3 52
5.14 Grade de Execução dos Módulos de Software em Função do Tempo . . . 54

6.1 Cenário 2 56

6.2 Grade de Execução dos Módulos de Software em Função do Tempo . . . 57
6.3 Tempos de Execução do Módulo posCon 58
6.4 Tempos de Comunicação com o bBoardServer 59
6.5 Tempos de Leitura dos Encoders 60
6.6 Tempos de Acionamento dos Motores 61

Lista de Tabelas

4.1 Métodos da Classe bBoard 32
4.2 Métodos da Classe bBoardAtt 32
4.3 Métodos da Classe bBoardRe 33
4.4 Métodos da Classe periodic 34
4.5 Construtor e Destrutor da Classe Robot 36
4.6 Métodos da Classe robot para Acionamento dos Motores das Rodas do Kapeck 37
4.7 Métodos da Classe robot para Leitura dos Encoders das Rodas do Kapeck 37
4.8 Métodos da Classe robot para Acionamento dos Motores da Cabeça Estéreo Kapeck 38
4.9 Métodos da Classe robot para Leitura dos Encoders da Cabeça Estéreo do Kapeck 39
4.10 Métodos da Classe robot para Leitura dos Sonares do Kapeck 40

6.1 Média e Desvio Padrão dos Tempos de execução do Módulo posCon . . . 58
6.2 Média e Desvio Padrão dos Tempos de Comunicação com o bBoardServer 59
6.3 Média e Desvio Padrão dos Tempos de Leitura dos Encoders 60
6.4 Média e Desvio Padrão dos Tempos de Acionamento dos Motores 61

Capítulo 1

Introdução

A robótica móvel é uma área de pesquisa que lida com o controle de veículos autônomos ou semiautônomos. O que diferencia a robótica móvel de outras áreas de pesquisa em robótica, tal como a robótica de manipuladores, é a sua ênfase nos problemas relacionados com a operação em ambientes complexos de larga escala, que se modificam dinamicamente, pois são compostos de obstáculos estáticos e móveis. Para operar neste tipo de ambiente, o robô deve ser capaz de adquirir e utilizar conhecimento sobre o ambiente; estimar uma posição dentro deste ambiente; possuir a habilidade de reconhecer obstáculos e responder em tempo-real às situações que possam ocorrer neste ambiente. Além disso, todas estas funcionalidades devem operar em conjunto. As tarefas de perceber, se localizar e se mover por um ambiente são problemas fundamentais no estudo dos robôs móveis autônomos [Dudek & Jenkin 2000].

O projeto de uma arquitetura de controle visa habilitar um robô móvel autônomo a operar em seu ambiente utilizando-se de seus recursos físicos e computacionais. Uma arquitetura de controle deve assegurar o cumprimento das tarefas do robô de maneira estável e robusta. Flexibilidade e modularidade para lidar com vários tipos de tarefa, bem como a possibilidade de alteração ou adição de novos módulos de hardware e software são características levadas em conta no desenvolvimento de uma arquitetura. Um bom desempenho em tempo-real, de forma que o robô consiga reagir aos eventos do ambiente em tempos aceitáveis também deve ser contemplado por uma arquitetura. Desta forma, pode-se citar as seguintes características como fundamentais a uma arquitetura de controle para sistemas robóticos autônomos [Medeiros 1998]:

- **Reatividade**: O sistema robótico deve ser reativo para conseguir reagir a mudanças súbitas do ambiente e também ser capaz de levar em conta eventos externos, de modo a executar de forma correta, segura e eficiente suas tarefas.
- **Comportamento Inteligente**: As reações do sistema robótico aos estímulos externos devem ser guiadas pelos objetivos de sua tarefa principal.
- **Integração de Múltiplos Sensores**: A limitação de precisão, confiabilidade e aplicabilidade dos sensores individuais pode ser compensada pela integração de vários sensores complementares.
- **Resolução de Múltiplos Objetivos**: No caso de sistemas robóticos autônomos ou semiautônomos, situações que requerem ações concorrentes conflitantes são inevitáveis. Uma arquitetura de controle deve prover meios para a execução desses

múltiplos objetivos.

- **Robustez**: O robô deve manipular leituras imperfeitas, eventos inesperados e falhas súbitas.
- **Confiabilidade**: É a habilidade de operar sem falhas ou degradação de desempenho em um certo período de tempo.
- **Modularidade**: É a habilidade de dividir todo o controle do sistema robótico em módulos que podem ser separada e incrementalmente desenvolvidos, implantados e mantidos.
- **Programabilidade**: Um sistema robótico utilizável deve ser capaz de realizar múltiplas tarefas que sejam descritas em um certo nível de abstração.
- **Flexibilidade**: Sistemas robóticos experimentais requerem mudanças contínuas durante a fase de desenvolvimento. Conseqüentemente, estruturas de controle flexíveis são requeridas de modo a permitir o desenvolvimento de uma arquitetura de controle.
- **Capacidade de Expansão**: Um longo tempo é necessário para desenvolver, construir e testar os componentes individuais de um sistema robótico. Conseqüentemente, uma arquitetura expansível é desejável de modo a ser possível a construção do sistema incrementalmente.
- **Adaptabilidade**: Como o estado do mundo muda rápida e imprevisivelmente, o sistema robótico deve ser capaz de se adaptar ao ambiente, chaveando entre várias estratégias de controle diferentes.
- **Raciocínio Global**: Um agente tomador de decisões global, responsável pelo entendimento total de uma situação, é necessário para se corrigir erros introduzidos por interpretações errôneas de dados sensoriais e também para fundir as informações parciais existentes.

Muitas são as arquiteturas propostas na literatura, porém não existe um paradigma definitivo que atenda a todas as funcionalidades requeridas por um sistema robótico autônomo [Medeiros 1998]. Neste contexto, um paradigma deve ser representativo e prover meios de se organizar o controle do sistema robótico, bem como prover um modelo de desenvolvimento.

Uma arquitetura de controle definida adequadamente é essencial para o desenvolvimento de robôs móveis complexos. Ela deve englobar diferentes módulos de hardware e software. Também deve determinar como os módulos de hardware e software devem ser implantados, assim como formalizar o relacionamento entre esses módulos. Uma instanciação de uma arquitetura de controle é conhecida como organização de controle.

1.1 Caracterização do Problema

A inexistência de uma arquitetura de controle para o robô Kapeck, do Laboratório de Robótica (LAR) do Departamento de Engenharia de Computação e Automação (DCA) da Universidade Federal do Rio Grande do Norte (UFRN) torna difícil o uso deste robô como plataforma para desenvolvimento de pesquisas no âmbito da robótica.

1.2 Motivação

O desenvolvimento de uma arquitetura de controle para sistemas robóticos autônomos tem se mostrado um grande desafio para a comunidade científica até os dias atuais. Diferentes abordagens para o projeto de arquiteturas de controle vêm sendo utilizadas em diversas áreas de pesquisa. Por muitos anos os pesquisadores têm construído arquiteturas de controle que apresentam um comportamento inteligente, mas normalmente não no mundo real e somente em ambientes controlados.

Existem diversas aplicações possíveis para os sistemas robóticos autônomos. No transporte, vigilância, inspeção, limpeza de casas, exploração espacial, auxílio a deficientes físicos, entre outros. No entanto, eles ainda não causaram muito impacto em aplicações domésticas ou industriais, principalmente devido a falta de uma arquitetura de controle robusta, confiável e flexível que permitiria que estes robôs operassem em ambientes dinâmicos, pouco estruturados, e habitados por seres humanos.

O desenvolvimento de uma arquitetura de hardware e software para controle do robô Kapeck, de modo a tornar este robô um ambiente propício para desenvolvimento de pesquisas em robótica constitui-se na maior motivação deste trabalho.

1.3 Objetivos

Este trabalho possui como objetivo geral a definição de uma arquitetura de hardware e software de baixo custo para controle de um robô móvel autônomo, mais especificamente para o robô Kapeck, do LAR do DCA-UFRN.

Os objetivos específicos deste trabalho são:

- O desenvolvimento de uma base de software que permita a utilização do Kapeck como plataforma para pesquisas futuras.
- A implementação de uma organização de controle para demonstração da arquitetura aqui proposta.

1.4 Organização do Texto

Este trabalho encontra-se estruturado da seguinte maneira:

- **Capítulo 2 -** São apresentados os principais paradigmas utilizados como base para implementação de arquiteturas de controle para sistemas robóticos, bem como exemplos de arquiteturas existentes na literatura.
- **Capítulo 3 -** Apresenta as abordagens mais utilizadas para a implementação dos módulos de hardware e software de uma arquitetura de controle para robôs.
- **Capítulo 4 -** Apresenta a arquitetura de hardware e software proposta neste trabalho, por meio da qual será implementada uma organização de controle, visando prover mobilidade e autonomia para o robô Kapeck.
- **Capítulo 5 -** Apresenta a organização de controle desenvolvida neste trabalho, a qual visa demonstrar o uso dos recursos da arquitetura proposta.

- **Capítulo 6 -** Apresenta os resultados adquiridos através da implementação de um protótipo da organização de controle que utiliza a arquitetura proposta neste trabalho.
- **Capítulo 7 -** Apresenta as conclusões provindas deste trabalho, assim como caminhos a serem seguidos nos trabalhos futuros.

Capítulo 2

Arquiteturas de Controle de Sistemas Robóticos

Este capítulo apresenta os principais paradigmas utilizados como base para especificação e implementação de arquiteturas de controle para sistemas robóticos móveis autônomos, bem como exemplos de arquiteturas existentes na literatura.

2.1 Paradigmas de Arquiteturas de Controle

Existem atualmente três paradigmas principais para desenvolvimento de uma arquitetura de controle de sistemas robóticos móveis. Eles são chamados respectivamente de paradigma hierárquico, paradigma reativo e paradigma híbrido deliberativo-reativo [Murphy 2000]. Esses paradigmas podem ser descritos utilizando duas abordagens:

- **Relação entre as Primitivas Sentir, Planejar e Agir** - As funções de um robô podem ser divididas em três categorias gerais. Se uma função está adquirindo informações de sensores e produzindo uma saída utilizável por outras funções, esta função é colocada na categoria **Sentir**. Se uma função está adquirindo informações (de sensores ou de seu próprio conhecimento de como o mundo funciona) e produzindo tarefas para o robô executar, esta função é colocada na categoria **Planejar**. Funções que produzem comandos para atuadores são colocadas na categoria **Agir**.
- **Modo como Dados Sensoriais são Processados e Distribuídos no Sistema** - Em alguns paradigmas, informação sensorial é restrita a ser utilizada de maneira específica ou dedicada para cada função de um robô. Nesse caso, o processamento é local para cada função. Outros paradigmas esperam toda a informação sensorial para processar toda a informação em um modelo global de mundo e depois distribuir subconjuntos do modelo para outras funções se necessário.

2.1.1 Paradigma Hierárquico

O paradigma hierárquico funciona de forma seqüencial e organizada. Primeiramente, o robô sente o mundo e constrói um mapa global do mundo; depois, o robô planeja todas as diretivas necessárias para alcançar o objetivo; finalmente, o robô age para realizar a primeira diretiva; depois de ter realizado a seqüência sentir, planejar, agir, ele começa

o ciclo novamente; o robô sente a conseqüência de suas atuações, replaneja as diretivas (mesmo que as diretivas não tenham sido modificadas) e age.

O paradigma hierárquico é monolítico, onde todo o processo de relação do robô com o mundo é seqüencial. Todas as observações dos sensores são fundidas em uma única estrutura global de dados, que o planejador acessa. Esta estrutura é geralmente chamada de modelo de mundo. Este modelo de mundo, como pode ser visto em Murphy (2000), contém:

- Uma representação do ambiente onde o robô está operando, adquirida previamente.
- Informação sensorial.
- Outros conhecimentos cognitivos adicionais que podem ser necessários para realizar uma tarefa.

Dentre as arquiteturas baseadas no paradigma hierárquico, a mais significativa é a arquitetura NASREM (*NASA Standard Reference Model for Telerobot Control System Architectures*) [Albus et al. 1989].

Vantagens e Desvantagens do Paradigma Hierárquico

Pode-se afirmar que a principal vantagem do paradigma hierárquico é a precisão oferecida pelo modelo global de mundo no que se refere à localização do robô, bem como a possibilidade de execução de tarefas mais complexas, que necessitem de planejamento. Como desvantagens, pode-se citar o grande gasto computacional envolvido na atualização do modelo de mundo, além da rigidez envolvida na seqüência sentir, planejar, agir. Os algoritmos das porções sentir e planejar são mais lentos que os da porção agir. Além do mais, as atividades da porção sentir do paradigma são totalmente descorrelacionadas das atividades da porção agir. Como não se pode fazer essas atividades paralelamente, o desempenho do sistema robótico é prejudicado e não robusto a eventos dinâmicos do ambiente, como o aparecimento inesperado de um obstáculo a frente do robô.

2.1.2 Paradigma Reativo

Esse paradigma se baseia no comportamento animal, que organiza a inteligência em camadas verticais. Todo sistema reativo é formado por comportamentos, embora o significado de um comportamento possa ser levemente diferenciado em cada arquitetura reativa. Comportamentos podem ser executados concorrentemente ou seqüencialmente [Murphy 2000].

Utilizando decomposição vertical, uma arquitetura reativa inicia-se com comportamentos primitivos de sobrevivência e desenvolve novas camadas de comportamento. As novas camadas geradas podem reutilizar as camadas mais baixas, inibi-las ou mesmo criar novos caminhos paralelos. O acesso aos sensores e aos atuadores é independente para cada camada. Devido a isso, se uma camada de nível mais alto parar de funcionar as camadas de nível mais baixo continuam operando. O paradigma reativo desconsidera o componente **planejar** do trio sentir-planejar-agir.

A percepção em arquiteturas reativas é local a cada comportamento. Um comportamento não sabe o que o outro está fazendo ou percebendo. Essa solução é oposta ao modelo de mundo utilizado nas arquiteturas hierárquicas.

Dentre as arquiteturas existentes na literatura que utilizam o paradigma reativo, as mais significativas são a arquitetura de Subsunção [Brooks 1986] e a arquitetura baseada em Campos Potenciais [Murphy 2000].

Vantagens e Desvantagens do Paradigma Reativo

O paradigma reativo possui como vantagem a velocidade de execução do sistema robótico, além de robustez a eventos dinâmicos do ambiente. A maior velocidade de execução é decorrente do acoplamento existente entre as porções sentir e agir deste paradigma, além da possível execução paralela de diferentes comportamentos. A grande desvantagem é a inexistência de planejamento, que limita a aplicação desse paradigma a situações que podem ser resolvidas apenas com comportamentos reflexivos, não permitindo a execução de tarefas mais complexas.

2.1.3 Paradigma Híbrido Deliberativo/Reativo

A organização de uma arquitetura híbrida deliberativa/reativa pode ser descrita na forma **planejar, sentir - agir**, ou seja, primeiro planejar, depois sentir e agir em um único passo. O robô primeiro planeja como cumprir uma missão (usando um modelo de mundo) ou uma tarefa, e então dispara diferentes comportamentos (sentir-agir) pertencentes a um conjunto pré-definido para executar o plano. Os comportamentos são executados até que o plano se complete. Após isso, o planejador gera um novo conjunto de comportamentos e o procedimento recomeça.

Existem diversas arquiteturas híbridas. Segundo Murphy (2000), essas arquiteturas possuem um conjunto de componentes básicos. Os módulos normalmente presentes nessas arquiteturas são:

- **Seqüenciador**: É o módulo que gera o conjunto de comportamentos necessários à realização de uma subtarefa, assim como a seqüência e condições de ativação desses comportamentos.
- **Gerenciador de recursos**: Aloca recursos para os comportamentos.
- **Cartógrafo**: Esse módulo é responsável por criar, armazenar e manter um mapa, além de conter os métodos de acesso a esse mapa. O cartógrafo normalmente contém um modelo de mundo global e representação de conhecimento, mesmo que esses não sejam um mapa.
- **Planejador de missão**: Ele interage com o ser humano, operacionaliza comandos para o robô e constrói um plano de missão.
- **Monitoração de desempenho e resolução de problemas**: É o módulo que permite ao robô verificar se está fazendo progresso ou não.

As arquiteturas híbridas herdam das arquiteturas reativas a utilização de comportamentos no seu módulo de percepção. Em conseqüência, o problema da coordenação e

ou combinação de comportamentos também é herdada. Além das abordagens por subsunção e por campos potenciais existem outras abordagens que resolvem os problemas de coordenação e ou combinação dos comportamentos; dentre elas pode-se citar: votação [Roisemblatt & Thorpe 1995], lógica nebulosa [Konolige & Myers 1998], filtragem [Murphy & Mali 1997] e fusão de sinais de controle [Freire et al. 2004].

Geralmente, as várias arquiteturas híbridas existentes são classificadas dentro de três categorias. Essas categorias são chamadas respectivamente de arquiteturas gerenciais, arquiteturas de hierarquia de estados e arquiteturas orientadas a modelos.

As **arquiteturas gerenciais** têm como principal característica a subdivisão da porção deliberativa em subcamadas. No topo fica a camada que faz o planejamento de alto nível. Esta passa o plano de ação às camadas de nível mais baixo, que refinam este plano e repassam funções para os comportamentos da parte reativa da arquitetura. As camadas de nível mais alto podem supervisionar as camadas de nível inferior, podendo passar instruções, porém só podem modificar a camada diretamente inferior a elas. Caso ocorram problemas na execução de uma tarefa, uma camada pode tentar resolver localmente o problema, podendo pedir ajuda à camada superior caso não consiga resolver o problema.

Arquiteturas de hierarquia de estados organizam as atividades deliberativas e reativas de acordo com o conhecimento que as atividades possuem do estado do robô e também através do escopo de tempo (passado, presente e futuro). Comportamentos reativos não possuem conhecimento do estado do robô e só funcionam no presente. Atividades deliberativas são divididas entre as que requerem conhecimento sobre o estado passado do robô (em que seqüência de comandos ele está executando) e as que requerem conhecimento sobre o estado futuro (missão e planejamento de caminho). Dessa forma, as arquiteturas de hierarquia de estados possuem três camadas, sendo que uma camada mais alta pode supervisionar uma camada diretamente inferior a ela.

Nas **arquiteturas orientadas a modelo**, os comportamentos podem acessar porções de um modelo de mundo global além de possuírem sua própria percepção específica. Esse modelo de mundo global funciona como um sensor virtual, provendo percepção a alguns comportamentos. A utilização desse sensor virtual assemelha-se ao modelo monolítico de mundo global das arquiteturas hierárquicas puras.

Alguns exemplos de arquiteturas híbridas são as arquiteturas AuRa [Arkin et al. 1987], SFX [Murphy & Arkin 1992], PyramidNet [Roisenberg et al. 2004], 3T ou 3-Tiered [Bonasso et al. 1997], Saphira [Konolige & Myers 1998], TCA [Simmons 1994], arquitetura de Park et al. (1999), SHA [Kim, Im, Shin, Yi & Lee 2003], arquitetura de Tangirala et al. (2005), COHBRA [Heinen 2002], MEBCA [Chen & Luo 1998], DAMN [Rosenblatt 1997], arquitetura de Kim, Chung, Kim & Lee (2003), arquitetura de Liu et al. (2006), arquitetura de Ardizzone et al. (2006), VOMAS [Hsu & Liu 2007], arquitetura de Hentout et al. (2007) e arquitetura LAAS [Alami et al. 1993].

Vantagens e Desvantagens do Paradigma Híbrido Deliberativo/Reativo

A grande vantagem desse paradigma reside na existência de planejamento aliado ao acoplamento entre as porções sentir e agir. Isso significa que a porção planejar, que é mais lenta, pode ser executada paralelamente às porções sentir e agir. Além do mais, a existência de um modelo de mundo global aumenta a precisão referente a localização do

robô, sendo que esse modelo de mundo pode também ser utilizado pela porção reativa do paradigma. Como desvantagens, pode-se citar o maior gasto computacional inerente a utilização de um modelo global de mundo, além do problema de sincronismo inerente à execução em paralelo das porções planejar, sentir e agir.

2.2 Exemplos de Arquiteturas de Controle

Serão apresentadas nesta seção uma descrição simplificada de algumas arquiteturas de controle existentes na literatura. É importante salientar que o escopo e os objetivos destas arquiteturas de controle são bastantes distintas umas das outras. Desta forma, estas arquiteturas não podem ser comparadas como diferentes soluções para um mesmo problema.

2.2.1 NASREM

NASREM [Albus et al. 1989] é uma arquitetura hierárquica definida em níveis. Em cada nível são identificados 4 elementos de inteligência:

- **Geração de Comportamentos** - Seleciona os objetivos, planeja e executa tarefas. As tarefas são recursivamente decompostas em sub-tarefas.
- **Modelo de Mundo** - Contém uma estimativa do estado do ambiente. O modelo de mundo também contém capacidades de simulação e gera expectativas e previsões.
- **Processamento Sensorial** - Processa as entradas sensoriais e as previsões do modelo de mundo, atualizando o modelo de mundo de acordo.
- **Julgamento de Valor** - Calcula os custos e riscos e avalia tanto os estados observados como os estados previstos pelos planos hipotéticos.

Esta arquitetura (Figura 2.1) foi utilizado em um grande número de veículos teleoperados ou semiautônomos em missões espaciais e subaquáticas.

2.2.2 AuRa

A arquitetura AuRa (*Autonomous Robot Architecture*) [Arkin et al. 1987] utiliza uma abordagem provinda da teoria de *schemas* como base. *Schemas* são as primitivas que servem como blocos de construção para as atividades motoras e de percepção. Eles podem ser definidos recursivamente e são independentes de implementação. Na figura 2.2, pode ser visto um esquema representativo desta arquitetura.

Esta arquitetura possui 5 subsistemas principais:

- **Percepção** - Obtém e filtra todos os dados sensoriais.
- **Cartográfico** - Conhecimento sobre o ambiente.
- **Planejamento** - Um planejador hierárquico e um controlador utilizando um *schema* motor.
- **Motor** - Interface para com o robô que será controlado.

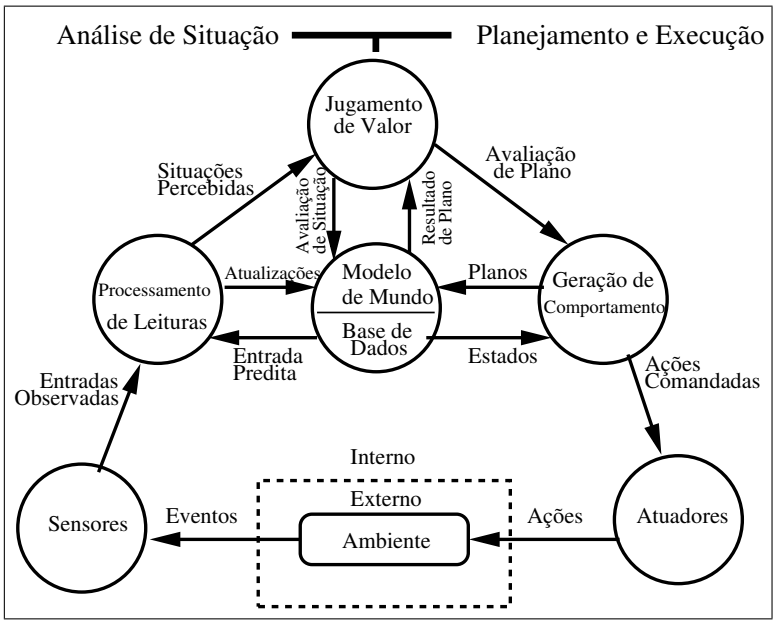

Figura 2.1: Arquitetura NASREM

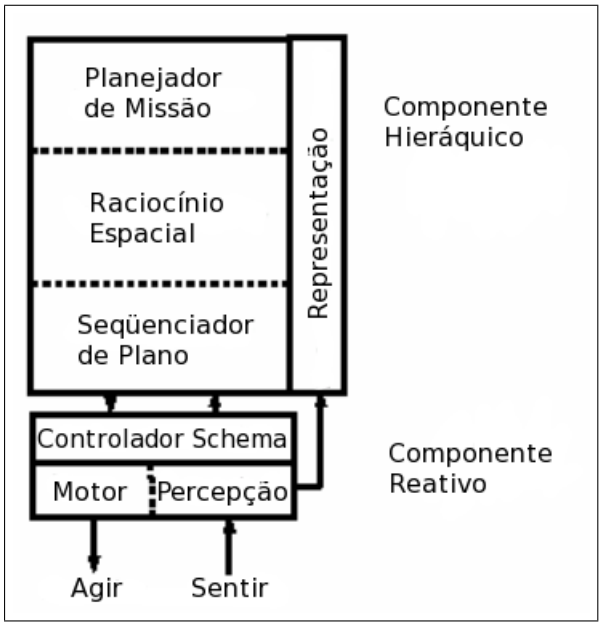

Figura 2.2: Arquitetura AuRa

- **Homeostático** - Monitora as condições internas para o replanejamento dinâmico.

O subsistema cartográfico consiste em um modelo *a priori* do ambiente armazenado em uma memória de longa duração, conhecimentos sobre o ambiente obtidos dinamicamente, e um modelo de incerteza espacial.

O subsistema de planejamento possui um componente deliberativo e um componente reativo. O componente deliberativo é um planejador hierárquico com 3 níveis de hierarquia (Planejador de Missões, Raciocínio Espacial, Seqüenciador de Planos). O componente reativo é um controlador baseado em um *schema* motor.

AuRA define uma arquitetura híbrida que incorpora propriedades tanto de sistemas reativos, deliberativos como hierárquicos. Modularidade, robustez e o aprendizado são mencionados como vantagens desta arquitetura e dos sistemas baseados em *schemas* em geral.

2.2.3 DAMN

A arquitetura DAMN (*Distributed Architecture for Mobile Navigation*) [Rosenblatt 1997] é formada por um árbitro central e um certo número de módulos distribuídos. Os módulos são processos independentes que podem representar comportamentos reativos ou deliberativos. O árbitro recebe votos a favor ou contra determinados comandos de cada módulo e decide a seqüência de ações dependendo dos votos recebidos. O árbitro executa uma fusão de comandos para selecionar a ação mais apropriada. Na figura 2.3, pode ser visto um esquema representativo desta arquitetura.

Esta arquitetura foi aplicada com sucesso em um sistema que dirigia um carro de forma autônoma em uma estrada. DAMN é uma arquitetura híbrida com uma representação de conhecimento distribuída. A estrutura é distribuída com um árbitro central.

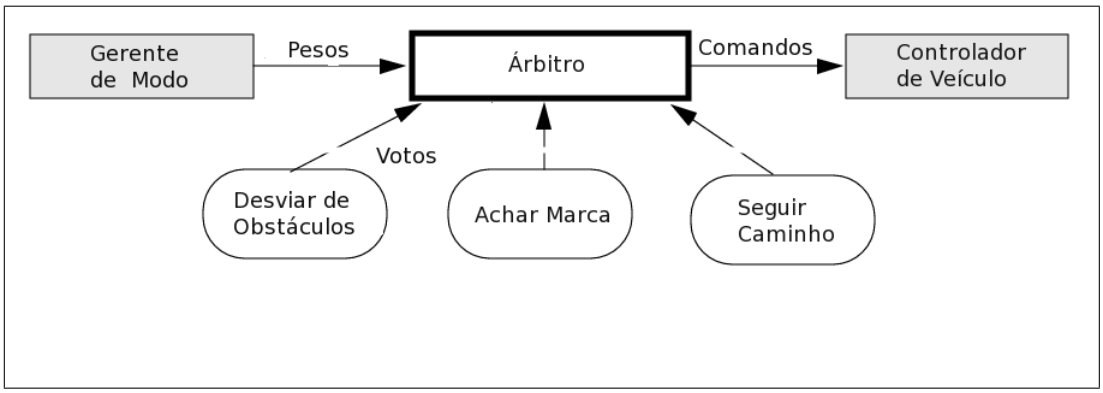

Figura 2.3: Arquitetura DAMN

2.2.4 COHBRA

A arquitetura COHBRA (Controle Híbrido de Robôs Autônomos) [Heinen 2002] separa o controle do robô móvel em 3 camadas:

- **Camada Vital** - Responsável pelo controle reativo.
- **Camada Funcional** - Responsável pelo controle de execução dos planos gerados pela camada deliberativa.
- **Camada Deliberativa** - Responsável por gerar planos a serem utilizados para realizar tarefas mais complexas.

Na figura 2.4, pode ser visto um esquema representativo desta arquitetura.

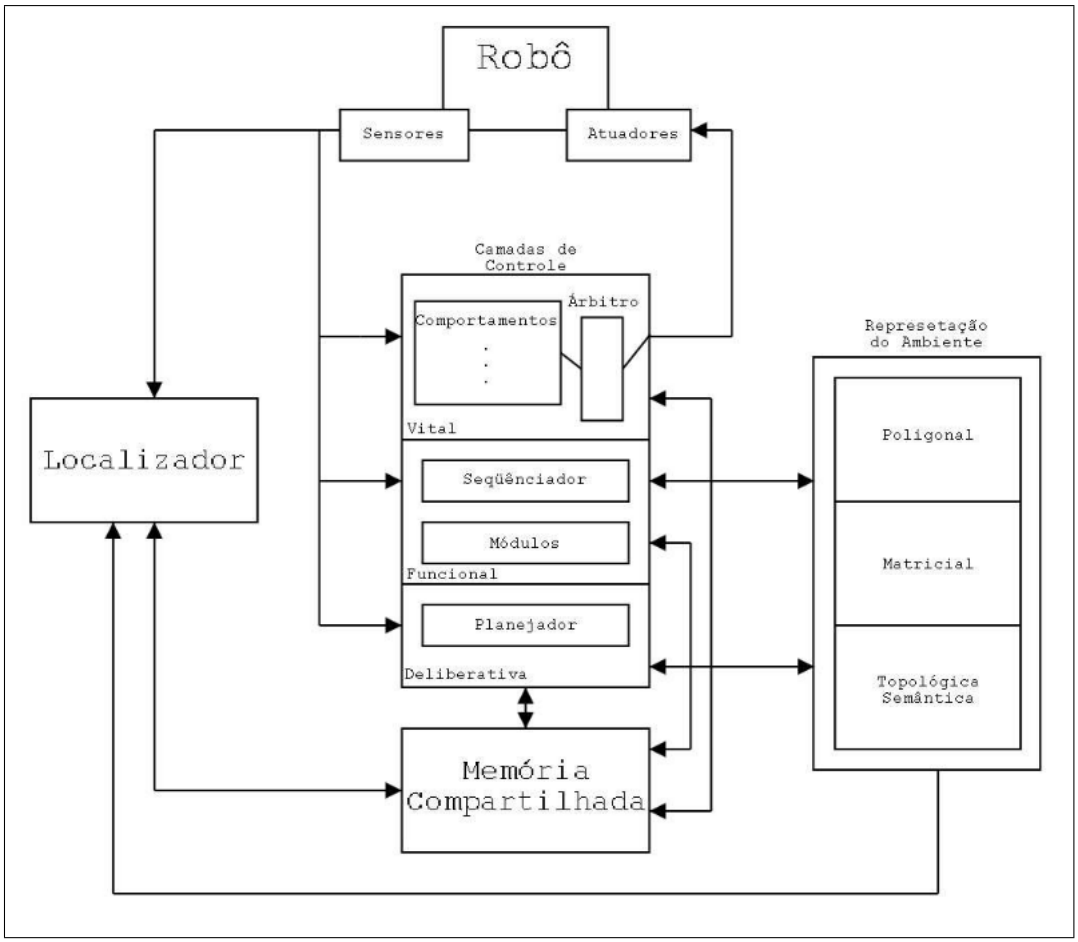

Figura 2.4: Arquitetura COHBRA

Para fornecer uma base sólida para a execução das tarefas desempenhadas pelas camadas de controle, foi integrado na arquitetura um módulo localizador. Este módulo localizador deve fornecer uma estimativa da posição do robô em relação a um mapa utilizando os dados sensoriais.

Para permitir a comunicação entre os vários componentes da arquitetura de controle é disponibilizada uma área de memória compartilhada. Através do uso deste depósito central de informações, os diversos módulos podem trocar informações vitais para o funcionamento do robô móvel autônomo.

2.2.5 Arquitetura de Tangirala

A arquitetura de Tangirala et al. (2005), foi desenvolvida para ser utilizada em AUVs (*Autonomous Underwater Vehicles*). Esta arquitetura híbrida é orientada a modelos (Subseção 2.1.3) e foi desenvolvida com um sistema semi-automático de verificação de segurança e desempenho, além dos requerimentos usuais em uma arquitetura de controle, tais como um bom desempenho em tempo-real e integridade dos dados compartilhados pelos módulos de software da arquitetura. Na figura 2.5, pode ser visto um esquema representativo desta arquitetura.

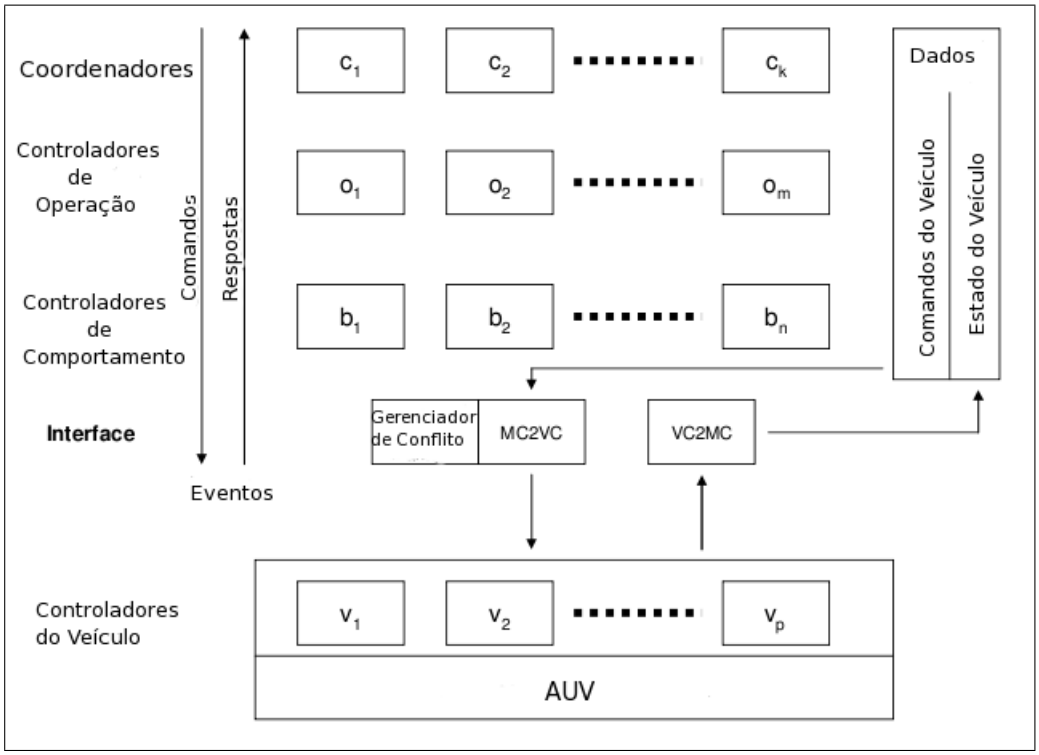

Figura 2.5: Arquitetura de Tangirala

Nesta arquitetura, o controle das tarefas foi dividido em dois níveis chamados respectivamente de Controle de Veículo e Controle de Missão.

A idéia básica desta arquitetura é hierarquicamente decompor as missões de um AUV em seqüências de operações, operações em seqüências de comportamentos e comportamentos em seqüências de manobras a serem efetuadas pelo AUV. Depois de feita esta decomposição, cada nível coordena o nível a baixo dele, de forma a realizar alguma tarefa corrente.

2.2.6 Arquitetura de Liu

Esta arquitetura híbrida [Liu et al. 2006] foi desenvolvida para ser utilizada em robôs-peixe autônomos, de forma a torná-los aptos a navegar em ambientes desconhecidos e

dinâmicos. Ela é composta por 3 camadas:

- Camada Cognitiva
- Camada Comportamental
- Camada de Padrão de Nado

Na figura 2.6, pode ser visto um esquema representativo desta arquitetura.

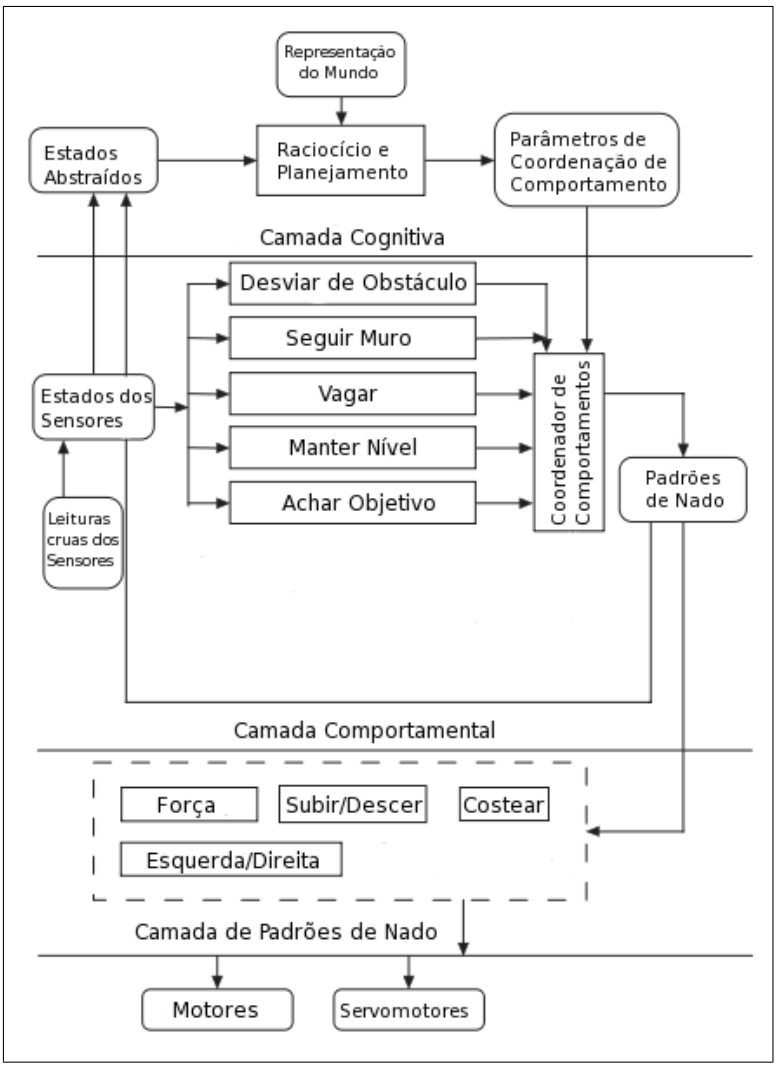

Figura 2.6: Arquitetura de Liu

Padrões de nado são as descrições dos movimentos básicos do robô-peixe. Esses padrões consistem em uma série de funções de movimento das juntas da cauda do robô-peixe que são utilizadas de forma conjunta para alcançar algum posição objetivo. A camada de padrão de nado está no nível mais baixo da arquitetura. Ela consiste de todos os padrões de nado possíveis do robô para propósito de controle básico.

A camada comportamental fica localizada no meio da arquitetura. Ela é responsável pelo controle reativo do robô. Ela primeiramente extrai os dados brutos provindos

dos sensores, utilizando uma função *fuzzy*, e insere as informações em cada comportamento existente na arquitetura. As respostas de cada comportamento são coordenadas juntamente por uma função de coordenação de comportamentos, que é definida pelos parâmetros de saída da camada cognitiva. A arquitetura de Subsunção [Brooks 1986] foi utilizada no desenvolvimento desta camada por causa da flexibilidade desta abordagem.

A camada cognitiva é a de mais alto nível desta arquitetura. Ela extrai o estado do robô a partir da camada comportamental e depois faz o raciocínio e planejamento baseado em tarefas. Finalmente ela toma a decisão de como coordenar os comportamentos na camada comportamental, de modo a realizar uma dada tarefa.

2.3 Conclusões

Neste capítulo foi apresentada uma revisão bibliográfica dos fundamentos referentes à especificação de uma arquitetura de controle para robôs móveis autônomos. Foram apresentados os principais paradigmas, bem como exemplos de arquiteturas de controle existentes na literatura. Após esta revisão bibliográfica, constatou-se que o paradigma híbrido deliberativo/reativo é o que mais se adequa ao robô Kapeck, uma vez que em tal robô existe a necessidade de aliar o comportamento reativo da rede de sensores-atuadores existente nele com as atividades deliberativas necessárias para realizar tarefas mais complexas.

Capítulo 3

Tecnologias Utilizadas em Arquiteturas

Existem diversas tecnologias e abordagens que devem ser levadas em consideração na implementação de uma arquitetura de hardware e software para controle de um robô móvel. Uma arquitetura deste tipo especifica o ambiente e fornece os meios para o desenvolvimento de diversas aplicações em uma plataforma robótica. Este capítulo apresenta as técnicas e abordagens mais utilizadas para a implementação dos módulos de hardware e software de uma arquitetura de controle para robôs.

3.1 Processamento Distribuído e Processamento Centralizado

Existe duas abordagens utilizadas para organização dos processadores de um sistema. Elas são chamadas de **Processamento Centralizado** e **Processamento Distribuído**.

3.1.1 Processamento Centralizado

Sistemas de processamento centralizado (Sistemas Monoprocessados) concentram toda a computação em apenas uma CPU (Unidade Central de Processamento). Em tais sistemas, a execução paralela real de processos não é possível, porém consegue-se atingir um pseudo-paralelismo através do chaveamento de processos efetivado pelos sistemas operacionais [Tanenbaum 2006].

Estes sistemas possuem como vantagem a simplicidade de implementação e manutenção. Como desvantagens pode-se citar a impossibilidade de utilizar paralelismo real, bem como a inoperância de todo o sistema caso a unidade de processamento pare de funcionar. Um exemplo desta abordagem pode ser visto na Figura 3.1.

3.1.2 Processamento Distribuído

Os sistemas de processamento distribuído (Sistemas Multiprocessados) se caracterizam pela existência de duas ou mais CPUs. A existência de múltiplos processadores em um sistema permite a execução de processos em paralelismo real. Esses sistemas podem executar processos independentes em processadores diferentes, bem como podem utilizar os vários processadores para melhorarem o tempo de execução de um único processo.

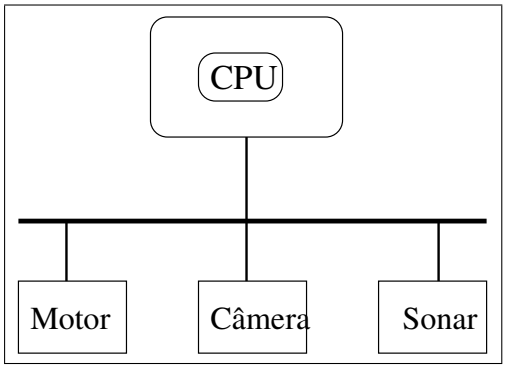

Figura 3.1: Processamento Centralizado

Existem três formas de distribuir estas CPUs por um sistema:

- Colocá-las em uma mesma plataforma (Computador) (Sistema Multiprocessado utilizando mais de uma CPU em Plataforma Única).
- Colocá-las em plataformas distintas, cada CPU possuindo memória distinta (Sistema Multiprocessado utilizando uma CPU por Plataforma).
- Uma combinação das duas abordagens descritas acima (Sistema Multiprocessado utilizando mais de uma CPU por Plataforma).

Sistema Multiprocessado utilizando mais de uma CPU em Plataforma Única

Nesta abordagem (Figura 3.2), as unidades de processamento podem se comunicar por meio de uma memória compartilhada ou por meio de troca de mensagens. Em ambas as formas, a memória física pode ser implementada de forma unitária ou distribuída.

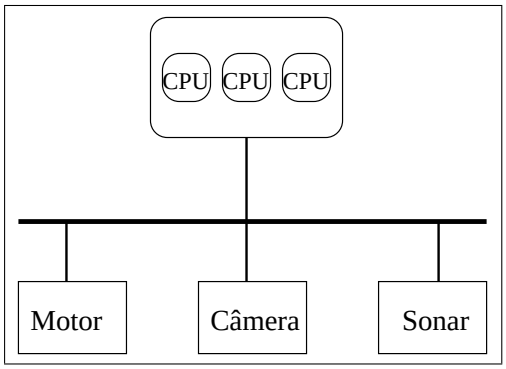

Figura 3.2: Sistema Multiprocessado Utilizando mais de uma CPU em Plataforma Única

Em sistemas de memória compartilhada, o próprio sistema oferece, de forma implícita, a comunicação entre os processadores, permitindo que todos os processadores vejam os dados escritos por qualquer processador, através do uso de uma área de memória com

espaço de endereçamento único. Tal característica pode levar a problemas de consistência dos dados conhecidos por cada processador. Existem várias abordagens utilizadas para contornar o problema da consistência de dados; dentre as mais relevantes estão o protocolo de Coerência de Cache e o protocolo MESI [Carter 2003].

Em contraste com a abordagem de memória compartilhada, os sistemas multiprocessados que utilizam troca de mensagens como meio de comunicação interagem através de mensagens explícitas. Para enviar uma mensagem, um processador executa uma operação explícita `SEND(dados,destinos)`, geralmente uma chamada de procedimento, que instrui o hardware a enviar os dados especificados para o processador destino. Depois, o processador destino executa uma operação `RECEIVE(buffer)` para copiar os dados enviados para o `buffer` especificado, tornando-os disponível para uso. Se o processador responsável por enviar os dados não tiver executado a operação `SEND` antes que a operação `RECEIVE` seja executada, a operação `RECEIVE` espera que a operação `SEND` seja completada, forçando a ordem das operações `SEND-RECEIVE`. Em sistemas de troca de mensagens, cada processador tem sua área de memória com seus respectivos espaços de endereçamento, sendo que os processadores não podem ler ou escrever diretamente dados contidos fora de seu espaço de endereçamento.

A implementação de aplicações que utilizam esta abordagem acaba não sendo trivial devido à necessidade de mecanismos para comunicação dos processadores. Por outro lado, o ganho de desempenho observado em relação a abordagem de processamento centralizada é bastante relevante.

Sistema Multiprocessado utilizando uma CPU por Plataforma

Nesta abordagem (Figura 3.3), as CPUs são distribuídas em plataformas diferentes, de modo que para comunicá-las deve-se lançar mão de alguma estrutura de rede, tais como redes TCP/IP (*Transmission Control Protocol/Internet Protocol*) sobre Ethernet. A necessidade de uma comunicação via rede gera um gargalo que pode vir a degradar o desempenho de um sistema que utiliza esta abordagem, além de aumentar a complexidade de implementação de aplicações para tais sistemas.

A distribuição de CPUs em plataformas diferentes gera um ganho de desempenho para o sistema em comparação a sistemas monoprocessados, além de maior robustez, uma vez que problemas em uma das plataformas do sistema não acarretam necessariamente em parada de todo o sistema.

Exemplos de robôs que utilizam esta abordagem pode ser visto em Ly et al. (2004) e em Matsui et al. (2005).

Sistema Multiprocessado utilizando mais de uma CPU por Plataforma

Esta abordagem (Figura 3.4), possui as vantagens e desvantagens existentes nas duas primeiras abordagens, sendo um híbrido delas. Porém, a complexidade de implementação de aplicações, bem como o custo financeiro desta abordagem, são maiores.

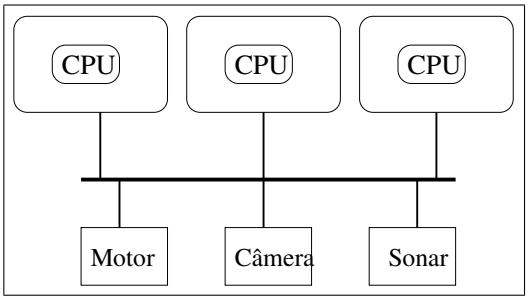

Figura 3.3: Sistema Multiprocessado Utilizando uma CPU por Plataforma

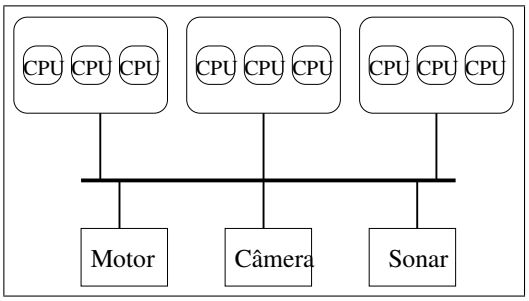

Figura 3.4: Sistema Multiprocessado Utilizando mais de uma CPU por Plataforma

3.2 Protocolos de Comunicação

A implementação de uma arquitetura de controle para um robô que possui processamento distribuído, além de possuir vários sensores e atuadores, exige a utilização de protocolos de comunicação que permitam a interação entre os vários processadores e dispositivos do sistema. Duas tecnologias utilizadas para este fim são os protocolos Ethernet e CAN.

3.2.1 Protocolo Ethernet

O protocolo Ethernet (padrão IEEE 802.3) [Tanenbaum 2003*a*] surgiu em 1972 nos laboratórios da Xerox. Ethernet é a tecnologia mais utilizada na implementação de redes de computadores, principalmente nas redes que utilizam arquitetura TCP/IP. Utilizando como referência o modelo OSI, este protocolo equivaleria à camada de enlace.

Inicialmente, este protocolo foi desenvolvido para atingir velocidades máximas de 10 Mbits/s. Posteriormente foram feitos melhoramentos no protocolo, visando aumento de velocidade. As atualizações do Ethernet foram: IEEE 802.3u (Fast Ethernet - velocidade máxima de 100 Mbits/s); 802.3z e 802.3ab (Gigabit Ethernet - velocidade máxima de 1000 Mbits/s).

Ethernet usa como método de acesso ao meio o protocolo CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) [Tanenbaum 2003*a*]. Esse método gera menos *overhead* na rede, porém é responsável pelo comportamento não determinístico do

Ethernet, o que significa que não é possível determinar em que períodos de tempo uma mensagem atinge seu destino.

Exemplos de arquiteturas que utilizam o protocolo Ethernet podem ser vistos em Ly et al. (2004) e em Atta-Konadu et al. (2005).

Vantagens e Desvantagens do Ethernet

Este protocolo possui como vantagens o baixo custo e a facilidade de implementação, além de alta velocidade de tráfego. Como desvantagens pode-se citar a inexistência de priorização de mensagens bem como o não determinismo deste protocolo. O não determinismo do ethernet dificulta a utilização dele em sistemas *hard real-time* [Farines et al. 2000].

3.2.2 Protocolo CAN

O protocolo CAN (*Controler Area Network*) [Bosch 1991] foi desenvolvido nos laboratórios da multinacional Bosch para ser utilizado na indústria automobilística, porém ganhou popularidade em outras áreas. Esse protocolo é bastante adequado para sistemas que utilizam muitos sensores e atuadores e possuem restrições temporais. CAN pode funcionar com múltiplos mestres ou em modo mestre-escravo. Esse protocolo pode atingir velocidades de 1 Mbit/s em distâncias de até 30 metros. A distância máxima permitida de um nó a um mestre é de 5 quilômetros, onde a taxa de transmissão máxima alcançada é de 10 Kbits/s.

O protocolo CAN possui nativamente um mecanismo que controla o acesso simultâneo ao barramento, através do protocolo CSMA/CA (*Carrier Sense - Multiple Access with Collision Avoidance*) [Bosch 1991]. Isto significa que sempre que ocorrer de duas ou mais mensagens serem enviadas ao mesmo tempo, a de mais alta prioridade terá o acesso ao meio físico assegurado e prosseguirá a transmissão.

Estrutura do Protocolo

Com o objetivo de se obter transparência no projeto e flexibilidade de implementação, o protocolo CAN foi dividido em duas camadas, obedecendo o modelo OSI/ISO: Camada Física e Camada de Enlace. Por sua vez, a Camada de Enlace foi dividido em duas subcamadas, chamadas de LLC (*Logical Link Control*) e MAC (Medium Acces Control).

A Camada Física trata de aspectos de temporização, codificação e sincronização de bits. A subcamada MAC representa o núcleo do protocolo, recebendo mensagens de/para a subcamada LLC, sendo responsável pela divisão das mensagens em quadros (*framing*), arbitragem, reconhecimento, detecção e sinalização de erros, sendo gerenciada por uma entidade chamada *Fault Confinement*, que é responsável pela distinção entre falhas permanentes e falhas temporárias. A subcamada LLC está relacionada com a filtragem de mensagens, notificação de *overload* e o gerenciamento de recuperação.

Transmissão de Mensagens

As informações transitam em um barramento CAN na forma de mensagens de formato fixo e de tamanho variável, porém limitado. Estas mensagens possuem um identificador que define a prioridade delas no acesso ao barramento e também pode identificar o conteúdo da mensagem. Um nó de uma rede CAN não conhece informações acerca da configuração dos outros nós (Ex: endereço dos nós). Tal característica gera conseqüências importantes:

- **Flexibilidade do Sistema** - Novos nós podem ser adicionados ao sistema, sem a necessidade de mudança de configuração dos nós já existentes.
- **Roteamento de Mensagens** - Através do identificador de uma mensagem, cada nó decide se processa ou não tal mensagem.
- **Multicast** - Por conseqüência da filtragem de mensagens, todos os nós da rede podem processar a mesma mensagem ao mesmo tempo.
- **Consistência dos Dados** - O CAN garante a recepção de uma mensagem ou por todos os nós da rede ou por nenhum. Isto ocorre porque todos os nós recebem a mensagem ao mesmo tempo. Se um deles acusar algum erro de recepção, a transmissão para, evitando que os outros nós recebam uma mensagem corrompida.

Cada dispositivo conectado a um barramento CAN deve transmitir na mesma velocidade definida para o barramento. Em sistemas que envolvam mais de um barramento CAN pode-se configurar velocidades distintas para cada barramento.

Para garantir o máximo de confiabilidade na transmissão de dados, o protocolo CAN implementa métodos robustos de detecção/sinalização de erros. Com estes métodos e o mecanismo *Fault Confinement* é possível detectar e diferenciar falhas temporárias e falhas permanentes. Desta forma, se em um nó é detectado um erro permanente, ele é desligado do barramento.

O número de conexões em um barramento CAN é teoricamente ilimitada. Na prática, este número é limitado pelos tempos de atraso e cargas elétricas no barramento. O meio físico pode ser implementado de várias formas. Geralmente implementa-se um barramento CAN por meio de par trançado, mas também pode-se implementá-lo através de fibra ótica e rádio freqüência.

Protocolos de Alto Nível

O padrão do protocolo CAN define apenas a Camada Física e a de Enlace de dados. Isto significa que este protocolo especifica como transportar pequenos pacotes de dados em um meio comum de comunicação. Visto isso, existem linhas de pesquisa que tratam do desenvolvimento de protocolos de alto nível, que forneçam melhoramentos ao CAN, tais como controle do fluxo de dados, envio de mensagens com tamanho superior a 8 bytes, endereços dos nós, entre outros. Existem diversos protocolos deste tipo, tais como CANOpen [Etschberger & Schlegel 2007], DeviceNet [Seixas 2007], CAN Kingdom [Cia 2007] e SCoCAN [Coronel et al. 2005].

Vantagens e Desvantagens do Protocolo CAN

Seu baixo custo e flexibilidade de implementação, aliado a um desempenho robusto e um sofisticado mecanismo de análise de erros fazem do protocolo CAN um solução bastante utilizada para a construção de robôs. Devido ao seu funcionamento determinístico, CAN é bastante utilizado em sistemas que possuem características de tempo-real. Como desvantagens pode se citar o tamanho reduzido dos pacotes utilizados neste protocolo, bem como menores velocidades e menor alcance se comparado com outros protocolos, tais como o ethernet.

Exemplos de aplicações que utilizam CAN podem ser vistas em Coronel et al. (2005) e em Shen et al. (2003). Algumas arquiteturas utilizam em conjunto os protocolos CAN e Ethernet em sua implementação, como a arquitetura desenvolvida por Ly et al. (2004).

3.3 Sistemas Operacionais

Em sistemas robóticos, bem como em outros sistemas que utilizam recursos computacionais, a utilização de um sistema operacional (SO) [Tanenbaum 2003*b*] se torna necessária, uma vez que ele é responsável por fazer a ligação entre o hardware e os módulos de software do sistema. Para sistemas com restrições temporais, a utilização de um sistema operacional que possua comportamento determinístico é normalmente cogitada. Exemplos de SOs de tempo-real: QNX, VxWorks, Linux, RT-Linux e Xenomai [Farines et al. 2000].

QNX e VxWorks foram desenvolvidos desde o início como sistemas operacionais de tempo-real, pois os mesmos garantem a previsibilidade de execução dos processos e o cumprimento das restrições temporais impostas. São distribuições pagas e também possuem uma curva de aprendizado maior para quem deseja desenvolver aplicações de tempo-real para os mesmos.

Linux é um sistema operacional tradicional o qual não foi concebido como de tempo-real no seu desenvolvimento. Desta forma, o Linux não possui o mesmo comportamento determinístico do QNX e VXWorks, não sendo adequado para sistemas que possuam sérias restrições de tempo e onde essas restrições de tempo devem ser obedecidas de forma a não ocorrerem acidentes (sistemas *hard real-time* [Farines et al. 2000]). Por outro lado, para sistemas com restrições temporais mais brandas, o Linux se torna uma opção viável, principalmente levando em consideração a curva de desenvolvimento mais rápida.

RT-Linux e Xenomai são módulos que transformam o Linux Standard em um sistema operacional de tempo-real. Funcionam como um *middleware* entre o hardware de um sistema e o próprio Linux, controlando todas as interrupções. O Linux funciona como um processo de menor prioridade quando se utiliza um desses módulos. Desta forma, processos que deverão funcionar em tempo-real devem ter maior prioridade e serão tratados diretamente pelos módulos. Processos normais serão repassados ao Linux e tratados por ele. A utilização desses módulos é indicada para sistemas *hard real-time*. Porém, é valido citar que a complexidade de implementação inerente à utilização desses módulos é maior se comparada à implementação de aplicações similares para o Linux Standard.

3.4 Tecnologias para Implementação de Software

No desenvolvimento de qualquer software, atributos como portabilidade, reusabilidade, modularidade, facilidade de manutenção e robustez são desejáveis. Na implementação de uma arquitetura de software de um robô não é diferente. Porém, existem atributos adicionais que devem ser levados em consideração, tais como execução paralela de partes do software além da existência ou não de comportamento de tempo-real (restrições temporais). Muitas abordagens são utilizadas para se ter esses atributos, tais como o emprego de linguagens de programação orientadas a objeto, uso de *threads*, uso de *sockets*, uso de arquiteturas cliente-servidor [Berger et al. 1997], uso de *blackboards*, entre outras.

As linguagens `C++` e Java são orientadas a objeto e possuem uma vasta comunidade de usuários. Java possui muitos recursos que aumentam a produtividade no desenvolvimento do software, porém não é tão flexível quanto `C++`, que entre outras características, possui recursos que permitem acesso direto ao hardware do sistema.

A paralelização dos módulos de software de uma arquitetura pode ser alcançada através do uso de *threads*. A existência de tais estruturas em uma arquitetura funcionando de forma paralela pode gerar problemas de inconsistência, que devem ser contornados através do uso de semáforos [Tanenbaum 2003*b*] e protocolos de prioridade [Farines et al. 2000].

Outro recurso bastante utilizado na implementação de uma arquitetura de software são os *blackboards*. Um *Blackboard* consiste em uma área de memória compartilhada, onde vários processos ou threads podem ler ou escrever dados, funcionando como meio de comunicação entre esses processos e threads. Alguns exemplos de trabalhos que utilizam *blackboards* são vistos em Brzykcy et al. (2001), Liscano et al. (1995), Fayek (1993) e Occello & Thomas (1992).

3.5 Conclusões

Neste capítulo foi apresentada uma revisão bibliográfica dos fundamentos técnicos referentes à implementação de uma arquitetura de controle para robôs móveis autônomos. Foram apresentadas algumas abordagens e tecnologias utilizadas na implementação dos módulos de hardware e software de uma arquitetura.

A arquitetura a ser implementada utilizará a abordagem processamento distribuído. Tal abordagem possui o melhor custo-benefício e melhor se adequa ao sistema proposto, onde se deseja ter processamento paralelo para aumentar o desempenho do sistema robótico. As plataformas irão se comunicar por meio dos protocolos Ethernet e CAN. O protocolo Ethernet será o meio de comunicação entre os computadores embarcados do sistema e o protocolo CAN será utilizado para conectar os sensores e atuadores aos computadores embarcados.

Utilizar-se-á o sistema operacional Linux. Uma vez que o processamento será bastante distribuído na arquitetura e o sistema robótico é do tipo *soft real-time*, a utilização de um sistema operacional de tempo-real não possuiria um bom custo-benefício, pois o ganho de desempenho seria menor que o tempo gasto para se implementar os módulos de software.

Os módulos de software da arquitetura serão implementados em `C++`, com o uso de *threads*, *sockets* e *blackboards*. A linguagem de programação `C++` foi escolhida devido à sua flexibilidade, além de possuir todas as características necessárias para implementar os atributos descritos na subseção 3.4. O uso de *threads* é justificado pela necessidade da execução em paralelo dos módulos da arquitetura. Os *sockets* serão utilizados para se promover a comunicação de módulos de software que estejam em plataformas distintas. Os *blackboards* funcionarão como meio de interação para as *threads* e módulos de software implementados em processos independentes.

Capítulo 4

Arquitetura Proposta

Uma arquitetura de controle para robôs é implementada através de módulos de hardware e software. Esses módulos interagem entre si e podem funcionar de forma síncrona ou assíncrona. Este capítulo apresenta a arquitetura de hardware e software proposta neste trabalho, por meio da qual será implementada uma organização de controle, visando prover mobilidade e autonomia para o robô Kapeck.

4.1 Arquitetura de Hardware

O Kapeck é um robô multi-tarefa não-holonômico que se locomove através de rodas com acionamento diferencial. Ele possui dois manipuladores com 5 graus de liberdade cada, uma cabeça estéreo com duas câmeras e um colar com 8 sonares. O robô pode ser visto na Figura 4.1.

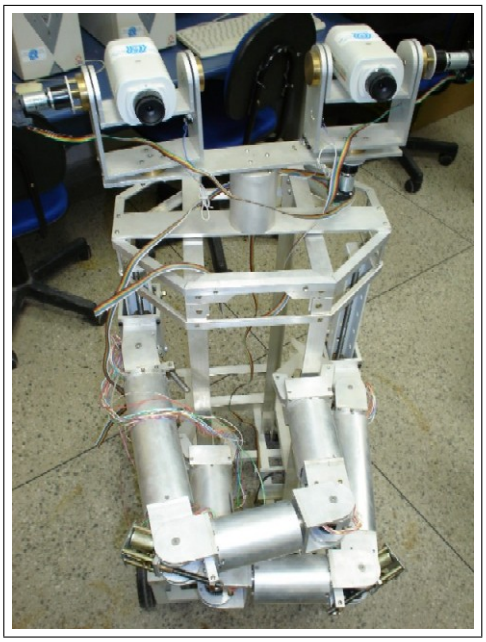

Figura 4.1: Robô Kapeck

O sistema de locomoção é composto por duas rodas independentes com acionamento diferencial, com um motor CC cada, e uma roda solta para dar equilíbrio à plataforma. Cada roda motorizada possui um encoder acoplado, que gera 1024 pulsos por revolução.

Os manipuladores possuem cinco juntas e uma garra, com um motor CC acoplado a cada uma delas. Em cada manipulador, quatro juntas possuem potenciômetros acoplados, uma junta (junta do ombro) possui um encoder, que gera 1024 pulsos por revolução, e um fim de curso. A garra não possui realimentação para acionamento do motor. No estágio atual do projeto, os módulos referentes aos manipuladores do robô Kapeck ainda não foram incluídos na arquitetura, mas a característica modular da arquitetura aqui proposta facilita esta inclusão no futuro.

A cabeça estéreo possui cinco motores CC. Cada motor possui um encoder, que gera 1024 pulsos por revolução, e um fim de curso acoplado. A visão é efetuada por duas câmeras CCD Gradiente Color SC-60. Essas câmeras possuem resolução máxima de 640X480 pixels e captura 30 quadros por segundo.

Todos os sensores e atuadores do robô, com exceção das câmeras, foram organizados em um barramento CAN. O protocolo CAN (*Controler Area Network*) [Bosch 1991] é bastante adequado para sistemas que utilizam muitos sensores e atuadores e possuem restrições temporais.

Devido ao baixo período de amostragem e ao curto período de tempo necessário para acionar sensores e atuadores, a existência de paralelismo real na execução dos processos do robô se torna necessária. Sendo assim, a arquitetura de hardware do Kapeck (Figura 4.2) foi desenvolvida de forma a prover processamento distribuído, através dos 10 processadores existentes no robô. Essa arquitetura é modular, o que significa que a inclusão de novos módulos de hardware, bem com a de módulos de software, é possível e de fácil execução.

A arquitetura de hardware é composta por um computador comum, dois computadores embarcados e oito placas microcontroladas. Os computadores embarcados e as placas microcontroladas localizam-se no robô. Eles concentram funções distintas, fornecendo um ambiente propício à execução em paralelo dos módulos de software.

O computador `monitor` (Figura 4.2) localiza-se fora do robô, comunicando-se com esse através de uma rede wireless do tipo IEEE 802.11b. Ele servirá para monitorar e interagir com o robô, contendo o módulo de interação com o usuário.

O computador `kapeck1` (Figura 4.2) abriga todo o software responsável pela deliberação da arquitetura de controle, assim como o controle de alto nível dos sensores e atuadores. Ele é responsável por fazer a interface com o `monitor`, passando a ele informações do sistema e recebendo e executando possíveis comandos.

O computador `kapeck2` (Figura 4.2) abriga o sistema de visão. O sistema de visão foi colocado em um computador a parte devido à sua grande exigência computacional. Esse computador é conectado ao `kapeck1` através de uma rede ethernet TCP/IP ponto-a-ponto (cabo crossover).

As placas microcontroladas (Figura 4.2) são responsáveis pelo controle de baixo nível dos sensores e atuadores da arquitetura. Dessa forma, existe um ganho de desempenho do sistema devido ao processamento dedicado dessas placas, liberando em parte os computadores embarcados para outras tarefas. Essas placas são conectadas ao computador

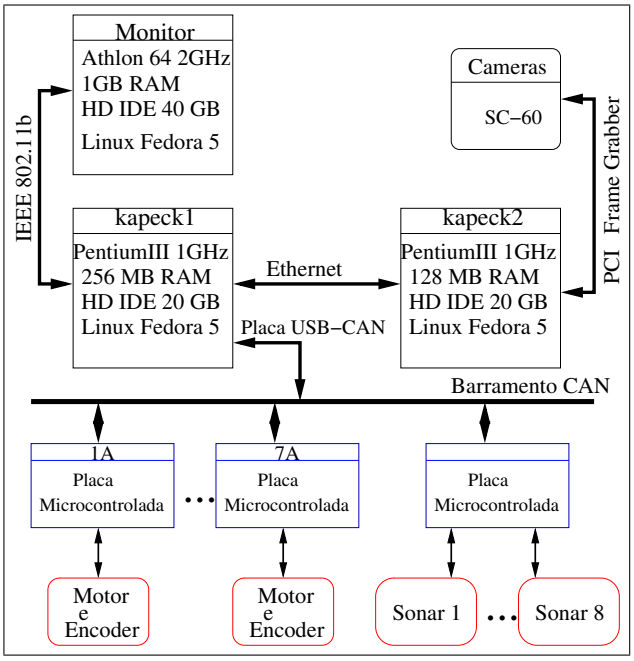

Figura 4.2: Arquitetura de Hardware

`kapeck1` por meio de uma rede CAN funcionando em modo mestre-escravo. Isso significa que as placas só utilizam o barramento se ordenadas pelo mestre da rede, contido no `kapeck1`. Todas as placas microcontroladas possuem o microprocessador de 8 bits PIC 18F258. Este microprocessador foi escolhido devido ao seu suporte ao protocolo CAN, além de dispor de gerador de sinais PWM (Modulação em Largura de Pulso), temporizadores e portas de entrada e saída digitais, necessários às aplicações. Dois tipos de placas foram desenvolvidas, chamadas de placa de controle de motor e placa de controle de sonar.

A placa de controle de motor (Figura 4.3) é responsável por gerar o sinal PWM para um determinado motor através de um algoritmo de controle PID, recebendo do controlador de alto nível as referências de posição ou velocidade. Esta placa também possuirá a função de adquirir as medições de encoder. Caso a placa seja acoplada a algum motor da cabeça estéreo, também receberá o sinal de uma chave de fim de curso. Os dados medidos são retornados para o módulo de percepção, localizado no `kapeck1`, apenas quando esse módulo os requisita assincronamente. As medições de encoder também são utilizadas localmente pelo algoritmo de controle PID. O controlador embarcado dessa placa possui um período de amostragem de 10 ms.

A placa de controle de sonar é responsável pelo funcionamento dos sonares do robô Kapeck. Ela coleta assincronamente informações dos sonares quando o módulo de percepção requisita tais informações. A placa retorna para o módulo de percepção uma medida da distância ao obstáculo mais próximo.

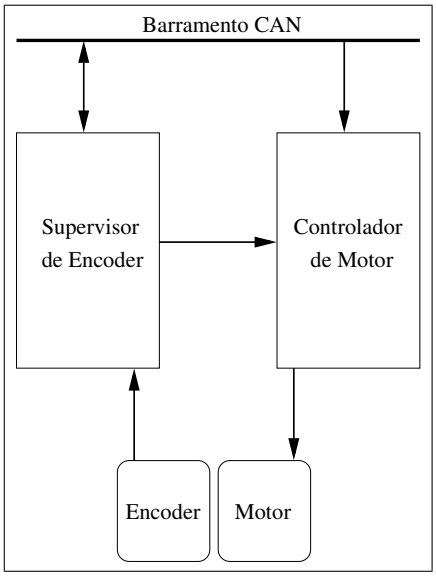

Figura 4.3: Diagrama de blocos do funcionamento da placa de controle de motor

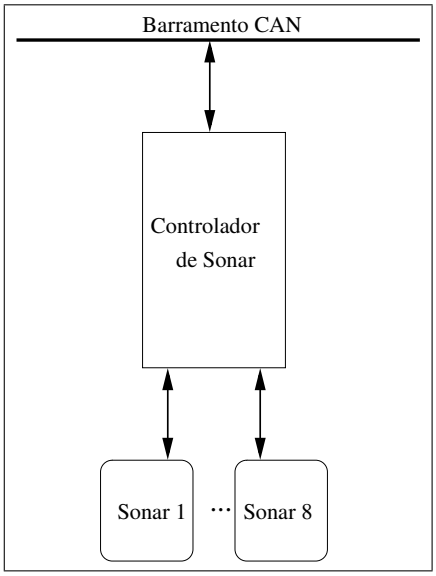

Figura 4.4: Diagrama de blocos do funcionamento da placa de controle de sonar

4.2 Arquitetura de Software

De modo a aproveitar o hardware distribuído do robô Kapeck, os módulos de software do robô devem ser capazes de funcionarem em paralelo. Além do mais, os módulos devem funcionar de forma temporalmente previsível. Para tornar isto possível, este trabalho propõe algumas ferramentas que tornam possível a execução em paralelo e temporalmente previsível dos módulos de software. Essas ferramentas são chamadas de *blackboards*, *robot* e *periodic*.

4.2.1 Blackboards

Um *blackboard* consiste em uma área de memória compartilhada que permite a interação entre os módulos de software. Este trabalho propõe a implementação desse mecanismo através de três classes `C++` chamadas de `bBoard`, `bBoardAtt`, `bBoardRe` e do módulo de software `bBoardServer`.

A classe `bBoard` (Figura 4.5) é utilizada para instanciar objetos que criam e destroem áreas de memória compartilhada e gerenciam seu acesso em exclusão mútua via semáforos [Tanenbaum 2003*b*]. Ela é utilizada pelos módulos de inicialização do sistema, responsáveis por iniciar todos os outros módulos de software e criar todos os *blackboards* e semáforos a serem utilizados para troca de informações dentro do sistema. Todo *blackboard*, quando é criado, recebe uma chave de identificação (ID) única. Na Tabela 4.1 é feita uma descrição dos métodos desta classe.

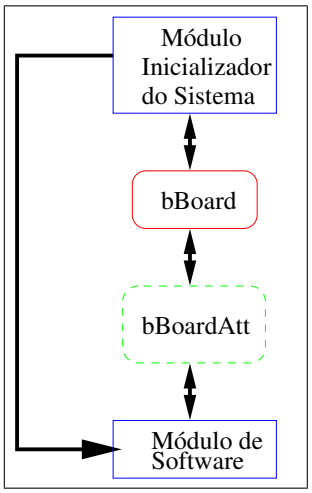

Figura 4.5: Utilização da Classe bBoard

A classe `bBoardAtt` (Figura 4.6) é utilizada para se acoplar a *blackboards* já criados. Fornece as primitivas de leitura e escrita no *blackboard*. Ao se instanciar um objeto dessa classe passa-se como parâmetro o ID do *blackboard* ao qual se deseja acoplar. Na Tabela 4.2 é feita uma descrição dos métodos desta classe.

A classe `bBoardRe` (Figura 4.6) tem as mesmas funções e funciona de forma semelhante à classe `bBoardAtt`, porém se acoplando a um *blackboard* que foi criado em outra

Tabela 4.1: Métodos da Classe bBoard

Método	Valores de Entrada	Valores de Saída	Descrição
`bBoard`	`int key`	`void`	Construtor da classe `bBoard`
`~bBoard`	`void`	`void`	Destrutor da classe `bBoard`

Tabela 4.2: Métodos da Classe bBoardAtt

Método	Valores de Entrada	Valores de Saída	Descrição
`bBoardAtt`	`int key`	`void`	Construtor da classe `bBoardAtt`
`~bBoardAtt`	`void`	`void`	Destrutor da classe `bBoardAtt`
`write`	`T` (Tipo de dado do *blackboard*)	`void`	Método para escrita em um *blackboard*
`read`	`void`	`T` (Tipo de dado do *blackboard*)	Método para leitura em um *blackboard*

máquina. Isso é feito através de uma conexão por *socket tcp* com o `bBoardServer` do outro computador. Ao se instanciar um objeto desta classe, passam-se como parâmetros o IP do outro computador e a ID do *blackboard* ao qual se deseja acoplar. A utilização de *blackboards* locais ou remotos é transparente para o programa cliente, pois ambas as classes `bBoardAtt` e `bBoardRe` oferecem as mesmas primitivas de leitura e escrita. Na Tabela 4.3 é feita uma descrição dos métodos desta classe.

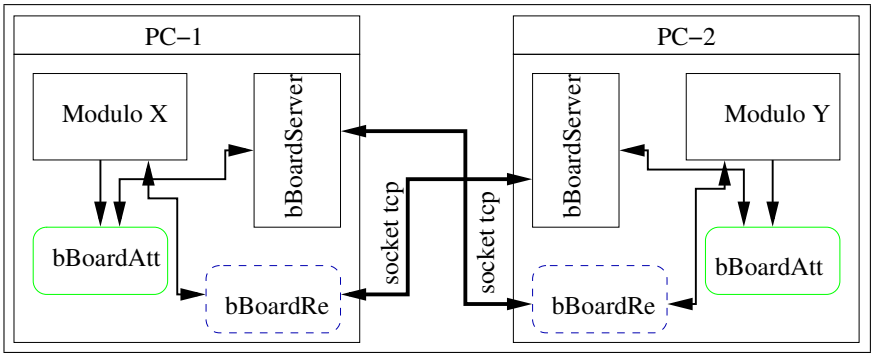

Figura 4.6: Utilização de bBoardRe, bBoardAtt e bBoardServer

O `bBoardServer` (Figura 4.6) é um módulo de software implementado em ambos os computadores embarcados. Ele efetua operações de leitura e escrita em *blackboards* locais em nome de clientes localizados em computadores remotos. Uma instância da classe `bBoardAtt` comunica com o `bBoardServer` do computador onde reside o *blackboard* a ela associado.

Tabela 4.3: Métodos da Classe bBoardRe

Método	Valores de Entrada	Valores de Saída	Descrição
`bBoardRe`	`const char* ip, int key, uint16_t porta`	`void`	Construtor da classe `bBoardRe`
`bBoardRe`	`const char* ip, int key`	`void`	Construtor da classe `bBoardRe` que utliza porta padrão
`~bBoardRe`	`void`	`void`	Destrutor da classe `bBoardRe`
`write`	`T` (Tipo de dado do *blackboard*)	`void`	Método para escrita em um *blackboard* remoto
`read`	`void`	`T` (Tipo de dado do *blackboard*)	Método para leitura em um *blackboard* remoto

Trabalhos Relacionados

Um exemplo de arquitetura que utiliza *blackboard* pode ser visto na arquitetura COHBRA [Heinen 2002]. Na arquitetura COHBRA é utilizado apenas um *blackboard* central onde os módulos de software compartilham informações, diferente da abordagem distribuída apresentada neste trabalho.

4.2.2 Periodic

O Linux Standard não é um sistema operacional de tempo-real. Desta forma, os processos e threads não são executados de forma temporalmente previsível. A ferramenta `Periodic`, implementado na forma de uma classe `C++`, fornece primitivas que melhoram a previsibilidade temporal na execução dos processos que o utilizam.

Esta ferramenta possui dois papéis importantes:

- Garantir que a tarefa vai ficar bloqueada apenas o tempo especificado;
- Sincronizar a grade de tempo da tarefa com a grade de tempo do Linux.

Para que a tarefa seja bloqueada pelo tempo determinado por suas especificações temporais, o valor de intervalo do `sleep` (instrução fornecida pelo próprio Linux) é calculado a cada iteração. Baseando o cálculo no valor do período e no instante de tempo de término da instância da tarefa, pode-se variar o valor do intervalo de `sleep` e obter a freqüência de execução desejada, respeitando as demais especificações temporais.

Para tentar sincronizar a grade de tempo de uma tarefa com a do Linux, utiliza-se a própria chamada `sleep`, como por exemplo `sleep`(1), antes do inicio do laço de instruções. Dessa forma, a tarefa periódica só começará a ser executada após o término do

`sleep`, que ocorrerá um segundo depois mais o tempo até o próximo tick, onde o escalonador saberá que a tarefa não deve mais ficar dormente. Assim, a primeira tomada de tempo e a construção da grade de tempo da tarefa ocorrerão logo após um tick, e as grades estarão praticamente sincronizadas. Outro fator importante a se considerar é que para se aproveitar esse sincronismo de grades o período de uma tarefa deverá ser um tempo múltiplo de 10 milisegundos.

Na Tabela 4.4 é feita uma descrição dos métodos desta classe.

Tabela 4.4: Métodos da Classe periodic

Método	Valores de Entrada	Valores de Saída	Descrição
`periodic`	`void`	`void`	Construtor da classe `periodic`
`~periodic`	`void`	`void`	Destrutor da classe `periodic`
`setStart`	`void`	`long`	Método que captura o momento de início de tarefa
`setSleep`	`long time`, `long t0`	`long`	Método que determina o tempo de dormência da tarefa
`agora`	`long t`	`long`	Método que determina o tempo decorrido

4.2.3 Robot

Como já discutido anteriormente neste trabalho, modularização e facilidade de manutenção são características necessárias a uma arquitetura de controle adequadamente definida. Esta arquitetura tenta atender a estas características por meio da ferramenta *robot*. Esta ferramenta (Figura 4.7) torna transparente aos módulos de software as características do hardware do robô. A existência desta ferramenta facilita alterações do hardware do robô, pois torna a implementação dos módulos de software desacoplada dos módulos de hardware, uma vez que os módulos de software operam sobre um barramento CAN lógico, que encapsula o barramento CAN real. Este mecanismo é implementado através da classe `robot` e do módulo de software `busArbiter`.

A classe `robot` é utilizada para se interagir com os sensores e atuadores do robô, tal como acionar um motor ou requisitar uma leitura de encoder. Esta classe se comunica com o módulo `busArbiter` via *socket unix*. O módulo `busArbiter` recebe as requisições dos módulos de software que necessitam de acesso a rede de sensores-atuadores, executa as requisições e envia os dados aos módulos requisitantes.

Além de atender aos requisitos de modularidade e facilidade de manuntenção, este mecanismo foi desenvolvido para resolver um problema inerente à arquitetura de con-

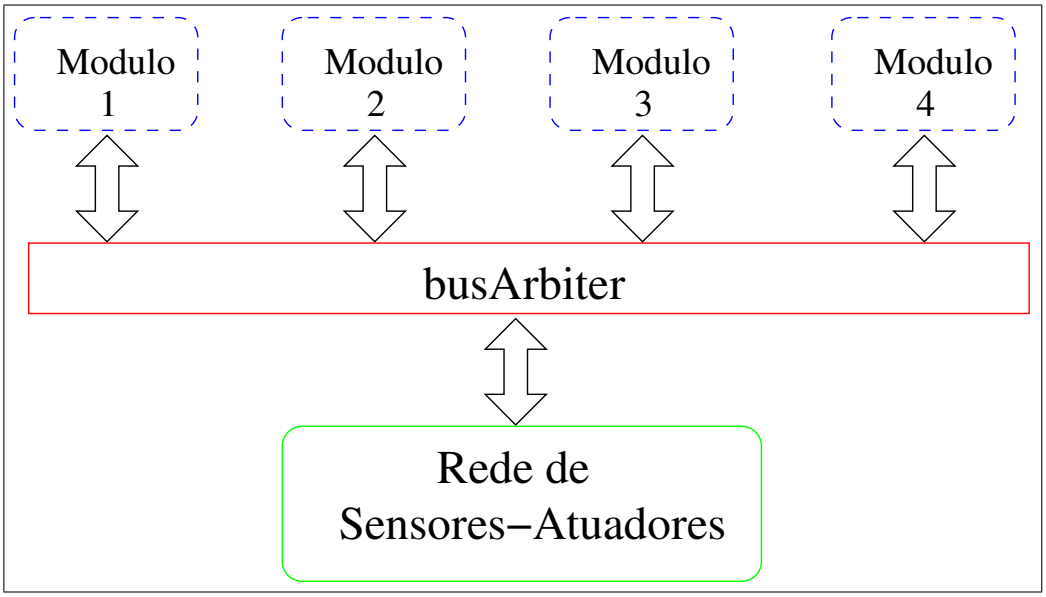

Figura 4.7: Arquitetura do Mecanismo Robot

trole proposta neste trabalho. O protocolo CAN possui nativamente um mecanismo que controla o acesso simultâneo ao barramento, através do protocolo CSMA/CA (*Carrier Sense - Multiple Access with Collision Avoidance*) [Bosch 1991]. Porém, na arquitetura de controle aqui apresentada, diferentemente da maioria das aplicações que utilizam barramento CAN, existe um nó da rede CAN (computador `kapeck1`) onde mais de um processo concorre pelo acesso ao barramento. Nesta situação, a existência de um mecanismo que garanta a exclusão mútua no acesso ao barramento é necessária, visto que a inexistência de tal mecanismo pode ocasionar colisão de mensagens antes mesmo destas chegarem ao barramento CAN. A exclusão mútua do barramento é garantida configurando o barramento CAN em modo mestre-escravo e utilizando um semáforo mutex no módulo `busArbiter`. Todos os métodos implementados pela classe `robot` internamente requisitam o acesso ao barramento CAN real. Por meio dos métodos da classe, os módulos que necessitarem enviar ou receber informações de algum sensor ou atuador do barramento devem mandar uma requisição ao `busArbiter`. Se algum módulo já estiver utilizando o barramento, o módulo requisitante será bloqueado até que o `busArbiter` conceda o acesso ao barramento para o mesmo. Uma vez que o acesso foi concedido, o módulo requisitante deve executar as ações desejadas e em seguida sinalizar ao `busArbiter` que o barramento está livre para ser utilizado por outros módulos.

É importante salientar que esta ferramenta poderia ser utilizada com outros protocolos de rede, sem a necessidade de modificações muito amplas.

Trabalhos Relacionados

Um exemplo de arquitetura que utiliza o barramento CAN é a arquitetura presente no trabalho de Coronel et al. (2005). Nesse trabalho é utilizado um protocolo de comunica-

ção de alto nível, acima do protocolo CAN, chamado de SCoCAN (*Shared Channel on CAN*) para controlar o acesso ao barramento. O protocolo SCoCAN segue uma abordagem baseada no protocolo TDMA (*Time Division Multiple Access*), onde *slots* de tempo são alocados em um ciclo básico do barramento para as mensagens de acordo com suas prioridades. Além disso, SCoCAN utiliza um esquema híbrido entre TTP (*Time Triggered Protocol*) puro e o protocolo ETP (*Event Triggered Protocol*) nativo de CAN. Um aspecto importante deste protocolo é o isolamento temporal de ambos os tráfegos. Uma implementação clássica utiliza um ciclo básico do barramento contendo *slots* de tempo para mensagens disparadas por tempo (TT) e para mensagens disparadas por evento (EV). SCoCAN utiliza uma abordagem diferente, onde além de prover os *slots* de tempo clássicos, ele provê maior flexibilidade ao ciclo básico do barramento, permitindo que *slots* de tempo para tráfego TT possam ser transformados em *slots* para tráfego EV.

Os métodos existentes na classe `robot` são descritos nas Tabelas 4.5, 4.6, 4.7, 4.8, 4.9 e 4.10.

4.3 Conclusões

Neste capítulo foi apresentada a arquitetura de hardware e software proposta neste trabalho. Essa arquitetura é distribuída e modular, o que propicia um ambiente ideal para a implementação de módulos de software funcionando em paralelo. A inclusão de novos módulos de hardware e software também é facilitada devido à modularidade da arquitetura. A previsibilidade temporal na execução dos processos e threads do sistema robótico é melhorada por meio da utilização do mecanismo `periodic`.

Tabela 4.5: Construtor e Destrutor da Classe Robot

Método	Valores de Entrada	Valores de Saída	Descrição
`robot`	`int speed, bool arb_status`	`void`	Construtor da classe `robot`
`robot`	`void`	`void`	Construtor da classe `robot`
`~robot`	`void`	`void`	Destrutor da classe `robot`

Tabela 4.6: Métodos da Classe robot para Acionamento dos Motores das Rodas do Kapeck

Método	Valores de Entrada	Valores de Saída	Descrição
`onMotores`	`VELOCIDADES referencias`	`void`	Aciona os motores das rodas do robô
`onMotorD`	`double w`	`void`	Aciona o motor da roda direita do robô
`onMotorE`	`double w`	`void`	Aciona o motor da roda esquerda do robô

Tabela 4.7: Métodos da Classe robot para Leitura dos Encoders das Rodas do Kapeck

Método	Valores de Entrada	Valores de Saída	Descrição
`readEncs`	`void`	`ENCODERS leituras`	Lê os valores retornados pelos encoders das rodas do robô
`readEncD`	`void`	`double leitura`	Lê o valor retornado pelo encoder da roda direita do robô
`readEncE`	`void`	`double leitura`	Lê o valor retornado pelo encoder da roda esquerda do robô

Tabela 4.8: Métodos da Classe robot para Acionamento dos Motores da Cabeça Estéreo Kapeck

Método	Valores de Entrada	Valores de Saída	Descrição
onMotoresC	VELOCIDADES_C referencias	void	Aciona os motores da cabeça estéreo
onMotorCentral	double w	void	Aciona o motor central da cabeça estéreo
onMotorPanD	double w	void	Aciona o motor que faz movimento de guinada da câmera direita da cabeça estéreo
onMotorPanE	double w	void	Aciona o motor que faz movimento de guinada da câmera esquerda da cabeça estéreo
onMotorTiltD	double w	void	Aciona o motor que faz movimento de ataque da câmera direita da cabeça estéreo
onMotorTiltE	double w	void	Aciona o motor que faz movimento de ataque da câmera esquerda da cabeça estéreo

Tabela 4.9: Métodos da Classe robot para Leitura dos Encoders da Cabeça Estéreo do Kapeck

Método	Valores de Entrada	Valores de Saída	Descrição
`readEncsC`	`void`	`ENCODERS_C leituras`	Lê os valores retornados pelos encoders da cabeça estéreo
`readEncCentral`	`void`	`double leitura`	Lê o valor retornado pelo encoder do motor central da cabeça estéreo
`readEncPanD`	`void`	`double leitura`	Lê o valor retornado pelo encoder do motor que faz movimento de guinada da câmera direita da cabeça estéreo
`readEncPanE`	`void`	`double leitura`	Lê o valor retornado pelo encoder do motor que faz movimento de guinada da câmera esquerda da cabeça estéreo
`readEncTiltD`	`void`	`double leitura`	Lê o valor retornado pelo encoder do motor que faz movimento de ataque da câmera direita da cabeça estéreo
`readEncTiltE`	`void`	`double leitura`	Lê o valor retornado pelo encoder do motor que faz movimento de ataque da câmera esquerda da cabeça estéreo

Tabela 4.10: Métodos da Classe robot para Leitura dos Sonares do Kapeck

Método	Valores de Entrada	Valores de Saída	Descrição
`onSonares`	`void`	`void`	Método para acionar os sonares do robô
`onSonaresT`	`t time`	`void`	Método que faz com que as placas controladoras fiquem acionando os sonares de acordo com o valor de tempo desejado
`readSonares`	`void`	`SONARES leituras`	Método para ler os valores retornados pelos sonares
`readSonar`	`ID sonar`	`double leitura`	Método para ler o valor retornado pelo sonar desejado

Capítulo 5

Organização de Controle Proposta

A organização de controle desenvolvida para o robô Kapeck (Figura 5.1) baseia-se no paradigma híbrido deliberativo-reativo [Murphy 2000] devido à necessidade de aliar o comportamento reativo da rede de sensores e atuadores do robô com as atividades deliberativas necessárias para realizar tarefas mais complexas. Desta forma, a organização de controle apresentada é composta pelos módulos de software `Localizer`, `Cartographer`, `actionPl`, `pathPl`, `trajExec`, `posCon` e `visionSys`.

É importante salientar que os módulos desta organização de controle podem ser ativados ou desativados de acordo com a complexidade da tarefa a ser realizada pelo robô.

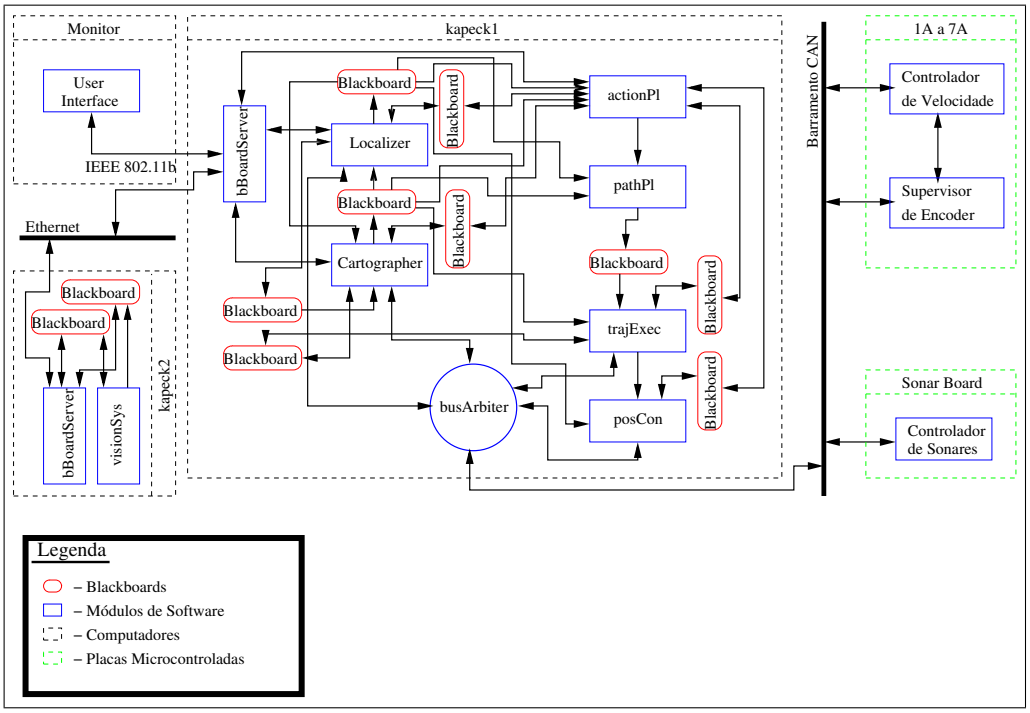

Figura 5.1: Organização de Controle

5.1 Módulos da Organização de Controle

Nesta seção serão apresentados os módulos existentes na organização de controle desenvolvida para o robô Kapeck, apresentada na Figura 5.1.

5.1.1 Localizer

O módulo `Localizer` (Figura 5.2) possui o papel de estimar a pose real do robô no ambiente real a partir das entradas sensoriais recebidas dos encoders e da representação interna (mapa) do ambiente. Para localizar o robô no mapa exportado pelo módulo `Cartographer`, este módulo requisita às placas microcontroladas as leituras dos encoders das rodas do robô e utiliza estes dados para calcular a odometria do robô. A estimativa da pose é armazenada em um *blackboard* para que todos os outros módulos da arquitetura de controle possam ter acesso. Uma vez que esta estimativa é constantemente atualizada, os componentes que a utilizarem devem monitorar suas alteração constantemente para que não sejam utilizadas estimativas desatualizadas.

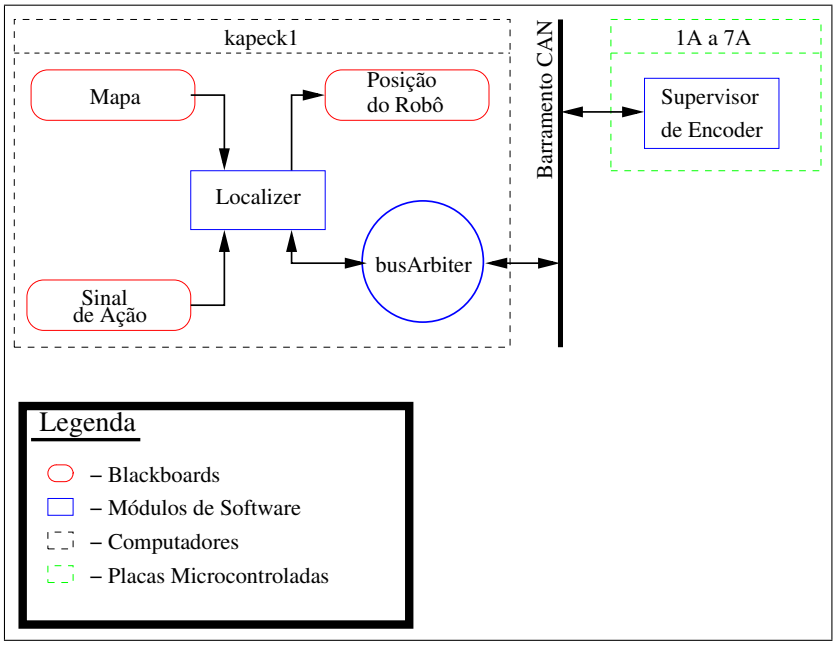

Figura 5.2: Ligações do Módulo Localizer

5.1.2 Cartographer

O módulo `Cartographer` (Figura 5.3) mapeia o ambiente utilizando a posição do robô exportada pelo módulo `Localizer` e leituras dos sonares, requisitadas diretamente pelo módulo `Cartographer`. Esse módulo pode também conter um mapa conhecido *a priori*. O mapa gerado pelo `Cartographer` ou conhecido *a priori* é exportado em um *blackboard*.

As técnicas de mapeamento podem ser divididas em duas categorias: métricas e topológicas. As técnicas métricas geralmente são implementados utilizando-se matrizes de ocupação que dividem a área a ser mapeada em unidades menores. Cada uma destas unidades possui um atributo que indica a probabilidade daquele espaço estar ou não ocupado. As técnicas topológicas representam o ambiente como uma coleção de pontos de referência interconectados. Neste trabalho, o mapa é topológico com informações métricas, implementado na forma de um grafo. Na Figura 5.4 pode ser visto um exemplo de mapa exportado pelo `Cartographer`. Neste mapa exemplo, as portas que ligam os ambientes são representadas pelas arestas do grafo e os ambientes são representados pelos nós.

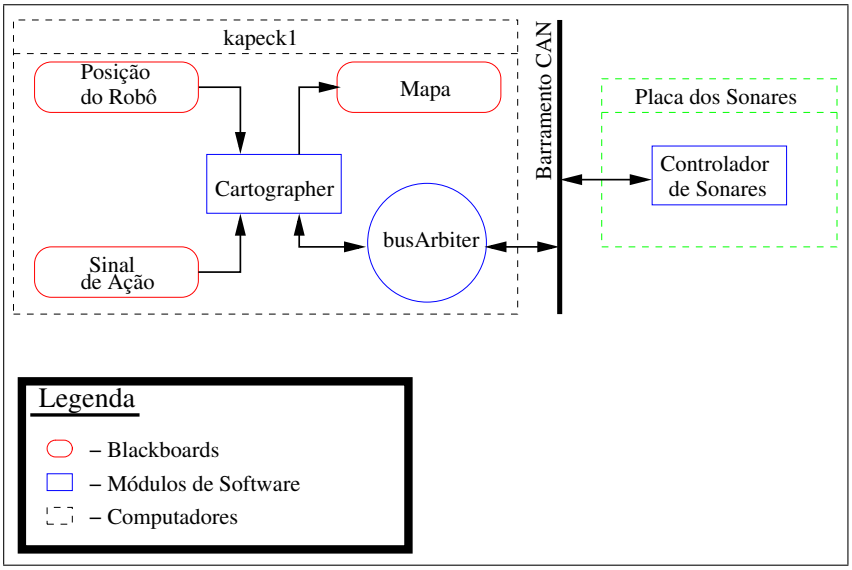

Figura 5.3: Ligações do Módulo Cartographer

5.1.3 actionPl

O módulo `actionPl` (Figura 5.5) é responsável por determinar as ações necessárias pelo robô para realizar uma dada tarefa. Ele delibera em um nível mais alto que os outros módulos. Para determinar as ações necessárias para se alcançar um objetivo ele se utiliza de: a posição do robô no mundo, fornecida pelo módulo `Localizer`; o mapa fornecido pelo módulo `Cartographer`; informações fornecidas pelo módulo `visionSys`.

Uma ação constitui em um objetivo local que deve ser realizado de forma que uma dada tarefa seja concluída. As ações a serem executadas pelo robô devem ser configuradas de acordo com a tarefa a ser realizada. Exemplos de ações possíveis são **ir para centro de um ambiente**, **ir para um ambiente não explorado**, **parar o robô**, entre outras.

O `actionPl` informa aos outros módulos sobre uma ação que deve ser executada através de sinais de ação. Esses sinais, que variam de acordo com a ação disparada, possuem etiquetas de tempo. Os sinais são enviados para *blackboards*. Cada módulo possui um blackboard de onde periodicamente lê esses sinais. Cada módulo possui comportamentos

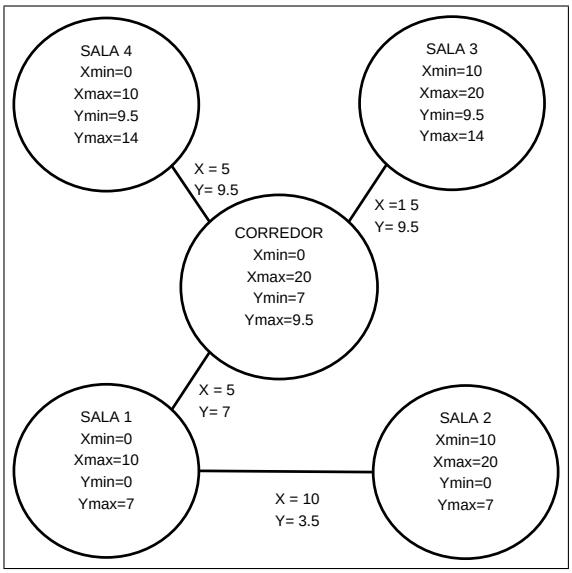

Figura 5.4: Um Exemplo de Mapa Exportado pelo Cartographer

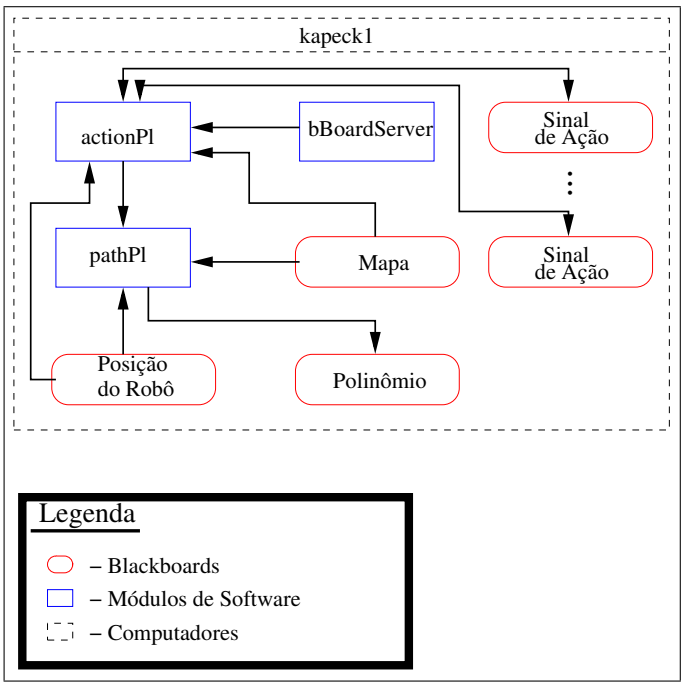

Figura 5.5: Ligações dos Módulos actionPl e pathPl

diferentes para cada tipo de ação. A etiqueta de tempo vinculada a cada sinal é utilizada de modo a permitir aos módulos saber se um dado sinal é mais novo que o sinal da ação atual. Desta forma, os módulos podem desprezar o comportamento que estejam executando e mudar para um novo comportamento coerente com a nova ação disparada pelo `actionPl`.

5.1.4 pathPl

O módulo `pathPl` (Figura 5.5) possui o papel de gerar um conjunto de pontos que levem o robô para o destino passado sincronamente pelo módulo `actionPl`. Tais pontos devem respeitar as restrições cinemáticas do robô Kapeck. Uma das possíveis abordagens para a geração dos pontos é utilizar polinômios paramétricos cúbicos. Esta abordagem foi utilizada pois um polinômio paramétrico de terceiro grau possui graus de liberdade suficientes para gerar um caminho executável por um robô com restrições não-holonômicas [Pedrosa et al. 2003*a*]. Os polinômios ou sequências de polinômios são gerados de acordo com a ação disparada pelo módulo `actionPl`. Os polinômios ou seqüência deles são exportados em um *blackboard*.

5.1.5 trajExec

O módulo `trajExec` (Figura 5.6) possui a função de executar a trajetória proveniente do caminho gerado pelo polinômio paramétrico cúbico, gerado pelo módulo `pathPl`. Para isto, ele utiliza um método que amostra o caminho não-holonômico fornecido pelo polinômio, levando em consideração a máxima distância que o robô pode percorrer entre passos consecutivos de amostragem do módulo de controle de posição (`posCon`) [Pedrosa et al. 2003*b*]. Desta forma, são geradas referências que melhoram a controlabilidade do robô. As referências são passadas sincronamente ao módulo `posCon`.

O `trajExec` também possui internamente um módulo desviador de obstáculos. O desviador de obstáculos possui o papel de evitar que o robô colida com algum obstáculo. Para isso, ele monitora as leituras dos sonares, por meio do *blackboard* onde o módulo `Cartographer` exporta tais leituras. Uma vez que um obstáculo é encontrado, uma posição de referência segura é gerada. No caso de o robô receber uma tarefa que não utilize o módulo `Cartographer`, o módulo `trajExec` pode ser configurado para requisitar diretamente à placa controladora dos sonares as leituras dos sonares.

5.1.6 posCon

O `posCon` (Figura 5.7) é um módulo reativo que possui a função de gerar os percentuais de velocidade para as rodas do robô, por meio de um controlador PID, de modo a levar o robô para uma posição gerada pelo módulo `trajExec`. Esses percentuais são enviados via CAN para as placas de controle de motor das rodas, fornecendo a referência para o controle embarcado nessas placas. Esse módulo lê também os dados exportados pelo `Localizer`, de modo a averiguar se o robô alcançou ou não a posição desejada.

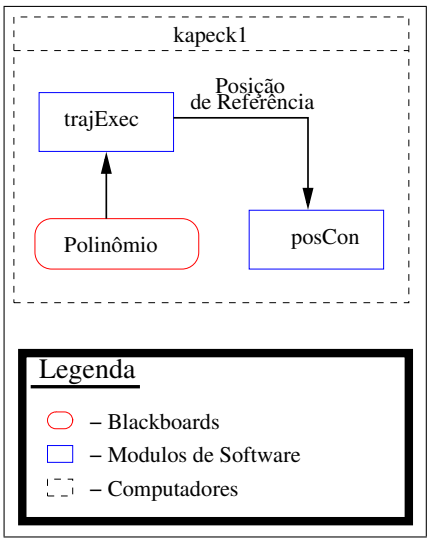

Figura 5.6: Ligações do Módulo trajExec

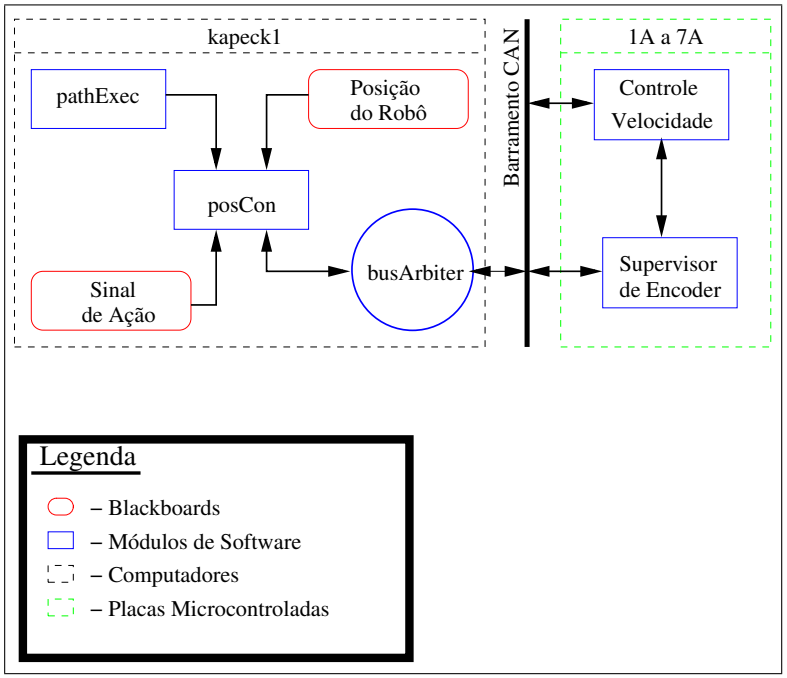

Figura 5.7: Ligações do Módulo posCon

5.1.7 visionSys

O módulo `visionSys` (Figura 5.8) possui o papel de processar as imagens capturadas pelas câmeras. Dependendo da tarefa que esteja sendo executada no robô, o sistema de visão pode identificar uma marca e determinar sua posição, bem como pode identificar marcas para corrigir a localização do robô.

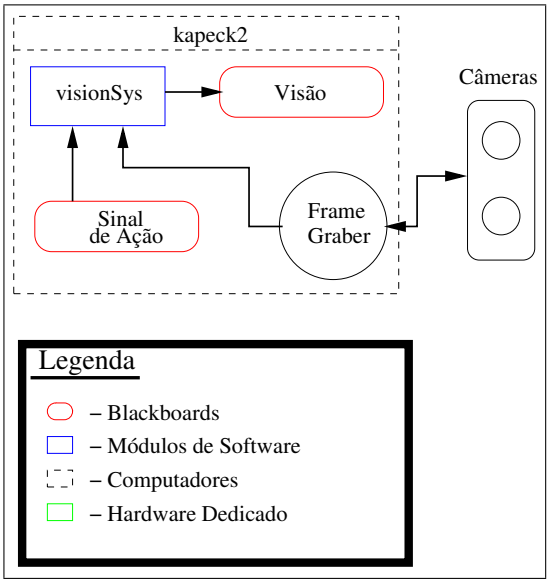

Figura 5.8: Ligações do visionSys

5.2 Exemplos de Execução da Organização de Controle

Para demonstrar a execução da organização de controle proposta, serão apresentados nesta seção alguns cenários hipotéticos, onde o robô possui uma tarefa a ser executada em um dado ambiente.

5.2.1 Cenário 1

Descrição do Cenário

Neste cenário, a tarefa do robô é vagar por um prédio, mapeado *a priori*, até que o robô tenha passado por todos os ambientes do prédio. Na Figura 5.9 é apresentado o cenário 1.

As ações configuradas no módulo `actionPl` são:

- **irCentro -** Ir para centro de um ambiente não-percorrido;
- **irAmbiente -** Ir para um outro ambiente ainda não-percorrido;
- **pararRobô -** Parar o robô.

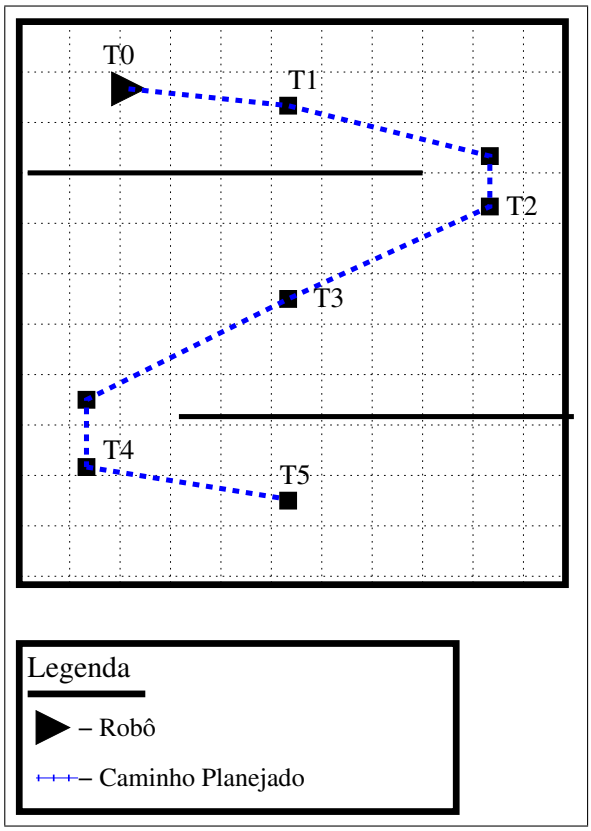

Figura 5.9: Cenário 1

Os módulos `Cartographer` e `visionSys` não são inicializados, uma vez que não são necessários para a realização da tarefa.

Execução da Tarefa

No momento T0, o módulo `actionPl` dispara uma ação do tipo **irCentro**, uma vez que o ambiente onde o robô se encontra ainda não foi explorado. Conseqüentemente, o módulo `pathPl` gera um caminho até o centro do ambiente e o exporta em um *blackboard*. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`. Nos momentos T2 e T4, o robô tem o mesmo comportamento.

No momento T1, o módulo `actionPl` dispara uma ação do tipo **irAmbiente**, uma vez que o ambiente onde o robô se encontra já foi explorado. Conseqüentemente, o módulo `pathPl` gera um caminho até o próximo ambiente não-explorado e o exporta em um *blackboard*. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`. No momento T3, o robô tem o mesmo comportamento.

No momento T5, o módulo `actionPl` dispara uma ação do tipo **pararRobô**, uma vez que todos os ambientes já foram mapeados. Desta forma, o robô e todos os seus módulos do robô são parados.

O módulo *Localizer* funciona de acordo com a forma descrita anteriormente, onde o mesmo requisita as leituras dos encoders das rodas do robô para poder estimar a pose do robô dentro do mapa conhecido *a priori*.

Na Figura 5.10 é apresentada a grade de execução dos módulos em função do tempo, expondo o que cada módulo faz em cada momento de acordo com a ação dispara pelo módulo `actionPl`.

5.2.2 Cenário 2

Descrição do Cenário

Neste cenário, a tarefa do robô é encontrar uma marca no piso de um prédio, mapeado *a priori*. O robô deve explorar todos os ambientes do prédio até encontrar a marca. Caso a marca não seja encontrada após a exploração do último ambiente, o robô deve parar. O módulo `visionSys` possui o papel de capturar e processar as imagens de modo a encontrar a marca. Quando ele identifica a marca, a posição da marca juntamente com uma sinalização para o módulo `actionPl` são exportadas em um *blackboard*. Na Figura 5.11 é apresentado o cenário 2.

As ações configuradas no módulo `actionPl` são:

- **irCentro -** Ir para centro de um ambiente não-explorado;
- **irAmbiente -** Ir para um outro ambiente ainda não-explorado;
- **irMarca -** Ir para a posição da marca;
- **pararRobô -** Parar o robô.

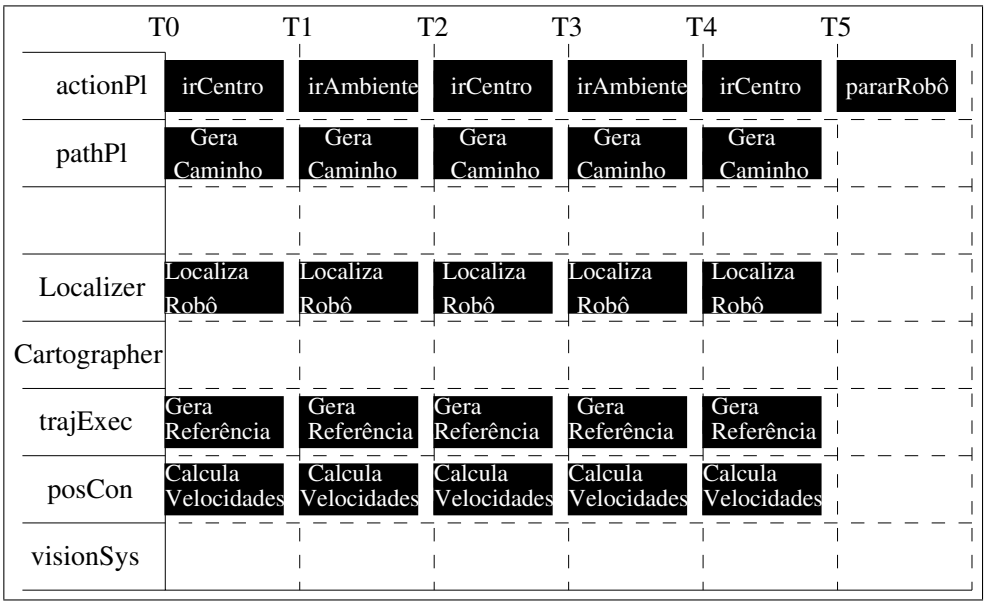

Figura 5.10: Grade de Execução dos Módulos de Software em Função do Tempo

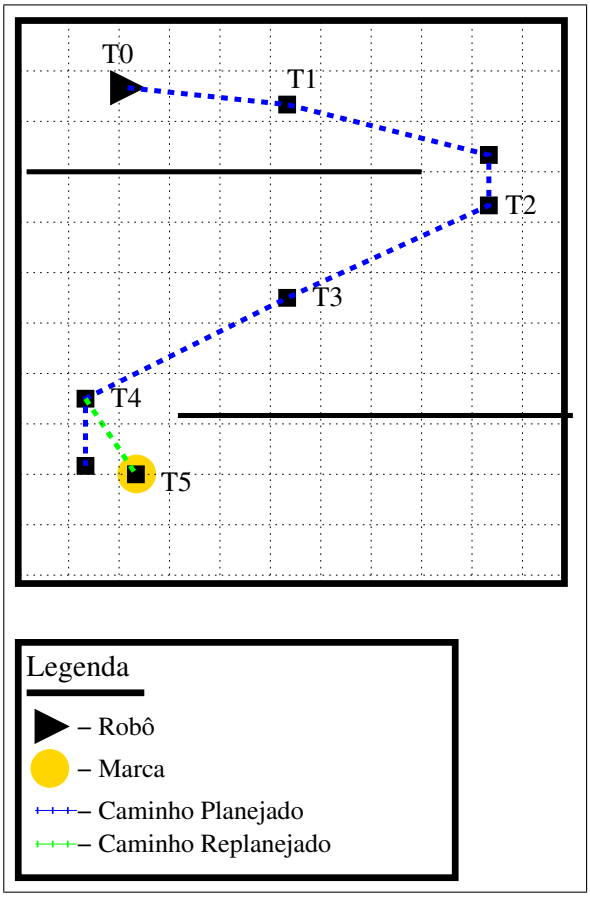

Figura 5.11: Cenário 2

O módulo `Cartographer` está desativado neste cenário, uma vez que ele não é necessário para a realização da tarefa.

Execução da Tarefa

No momento T0, o módulo `actionPl` dispara uma ação do tipo **irCentro**, uma vez que o ambiente onde o robô se encontra ainda não é explorado. Conseqüentemente, o módulo `pathPl` gera um caminho até o centro do ambiente e o exporta em um *blackboard*. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`. No momento T2, o robô tem o mesmo comportamento.

No momento T1, o módulo `actionPl` dispara uma ação do tipo irAmbiente, uma vez que o ambiente onde o robô se encontra já foi explorado. Conseqüentemente, o módulo `pathPl` gera um caminho até o próximo ambiente não-explorado e o exporta em um *blackboard*. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`. No momento T3, o robô tem o mesmo comportamento.

No momento T4, o módulo `actionPl` dispara uma ação do tipo **irMarca**. Esta ação é disparada em função do encontro da marca pelo módulo `visionSys`. Devido a isso, o resto da caminho gerado no momento T3 é descartado e o `actionPl` solicita ao módulo `pathPl` a geração de um novo caminho, que leve o robô até a marca encontrada. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`.

No momento T5, o módulo `actionPl` dispara uma ação do tipo **pararRobô**, uma vez que todos os ambientes já foram mapeados. Desta forma, o robô e todos os seus módulos do robô são parados.

O módulo *Localizer* funciona de acordo com a forma descrita anteriormente, onde o mesmo requisita as leituras dos encoders das rodas do robô para poder estimar a pose do robô dentro do mapa conhecido *a priori*.

Na Figura 5.12 é apresentada a grade de execução dos módulos em função do tempo, expondo o que cada módulo faz em cada momento de acordo com a ação dispara pelo módulo `actionPl`.

5.2.3 Cenário 3

Descrição do Cenário

Neste cenário, a tarefa do robô é mapear metricamente os ambientes de um prédio, utilizando para isso um mapa topológico conhecido *a priori* que indica a ligação dos ambientes através de marcas colocadas nos centros dos mesmos. Quando o sistema de visão identifica uma marca, o módulo `actionPl` deve mandar o robô ir para a posição da marca. Uma vez na posição da marca, o `actionPl` deve sinalizar para o módulo `Cartographer` ativar os sonares do robô de modo a capturar leituras para a confecção do mapa métrico. Após encontrar a última marca, o robô deve efetuar a captura das leituras dos sonares para o último ambiente e após isso parar. Na Figura 5.13 é apresentado o cenário 3.

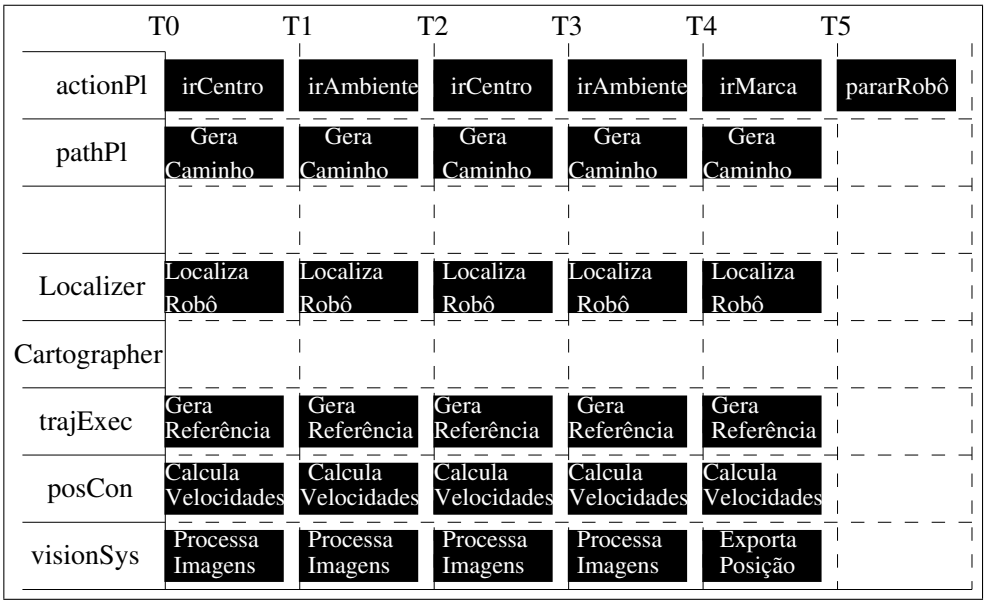

Figura 5.12: Grade de Execução dos Módulos de Software em Função do Tempo

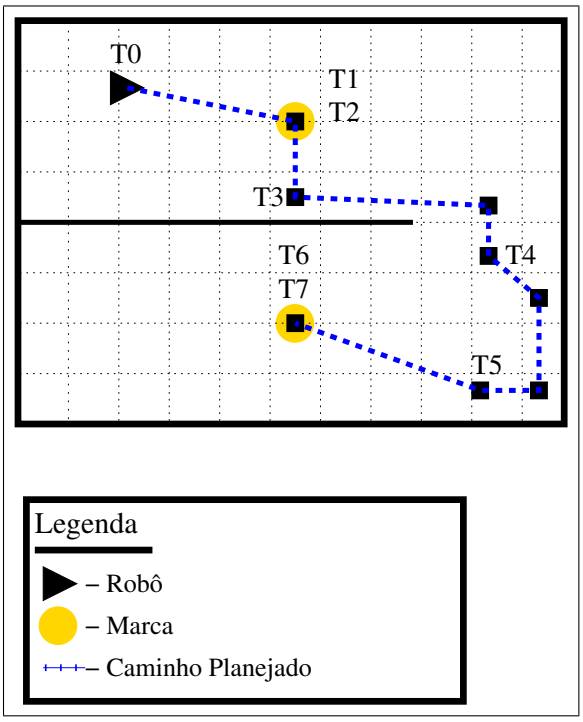

Figura 5.13: Cenário 3

As ações configuradas no módulo `actionPl` são:

- **expMarca -** Explorar ambiente não percorrido até encontrar marca;
- **irMarca -** Ir para a posição da marca;
- **mapAmb -** Mapear ambiente;
- **expPass -** Explorar ambiente até encontrar passagem;
- **irAmb -** Ir para próximo ambiente;
- **pararRobô -** Parar robô.

Execução da Tarefa

No momento T0, o módulo `actionPl` dispara uma ação do tipo **irMarca**, uma vez que neste momento o módulo `visionSys` já encontrou a marca. Conseqüentemente, o módulo `pathPl` gera um caminho até a marca, utilizando as coordenadas repassadas pelo módulo `visionSys`. Os outros módulos, que estão executando em paralelo, realizam o necessário para levar o robô até a posição da marca. No momento T5, o robô tem o mesmo comportamento.

Quando o robô chega na marca, no momento T1, o módulo `actionPl` dispara uma ação do tipo **mapAmb**, sinalizando para o módulo `Cartographer` ativar os sonares do robô de modo a capturar leituras para a confecção do mapa métrico, uma vez que o ambiente onde o robô se encontra já foi explorado. Neste momento, o módulo `pathPl` fica inativo, pois o robô deve se manter estático durante a coleta dos dados dos sonares. Os outros módulos, que estão executando em paralelo, realizam o necessário para manter o robô estático. No momento em que o módulo `Cartographer` termina a coleta, o mesmo sinaliza para o módulo `actionPl`, que deve analisar a próxima ação a ser tomada. No momento T6, o robô tem este mesmo comportamento.

No momento T2, o módulo `actionPl` dispara uma ação do tipo **expPass**. Esta ação é disparada de modo a fazer o robô explorar o ambiente em busca de uma passagem para o próximo ambiente. Essa passagem deve ser identificada pelo módulo `visionSys`. O módulo `pathPl` gera um caminho que faça o robô vagar dentro do ambiente corrente seguindo as paredes do mesmo. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`.

No momento T3, o módulo `actionPl` dispara uma ação do tipo **irAmb**, uma vez que o módulo `visionSys` encontrou uma passagem e sinalizou para o módulo `actionPl`. O módulo `pathPl` gera um caminho que leva o robô ao próximo ambiente e exporta este caminho em um *blackboard*. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`.

No momento T4, o módulo `actionPl` dispara uma ação do tipo **expMarca**. Esta ação é disparada de modo a fazer o robô explorar o ambiente em busca da marca que representa o centro do ambiente. Essa marca deve ser identificada pelo módulo `visionSys`. O módulo `pathPl` gera um caminho que faça o robô vagar dentro do ambiente corrente seguindo as paredes do mesmo. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`.

No momento T7, o módulo `actionPl` dispara uma ação do tipo **pararRobô**, uma vez que todos os ambientes já foram mapeados. Desta forma, o robô e todos os seus módulos

do robô são parados.

O módulo *Localizer* funciona de acordo com a forma descrita anteriormente, onde o mesmo requisita as leituras dos encoders das rodas do robô para poder estimar a pose do robô dentro do mapa conhecido *a priori*.

Na Figura 5.14 é apresentada a grade de execução dos módulos em função do tempo, expondo o que cada módulo faz em cada momento de acordo com a ação dispara pelo módulo `actionPl`.

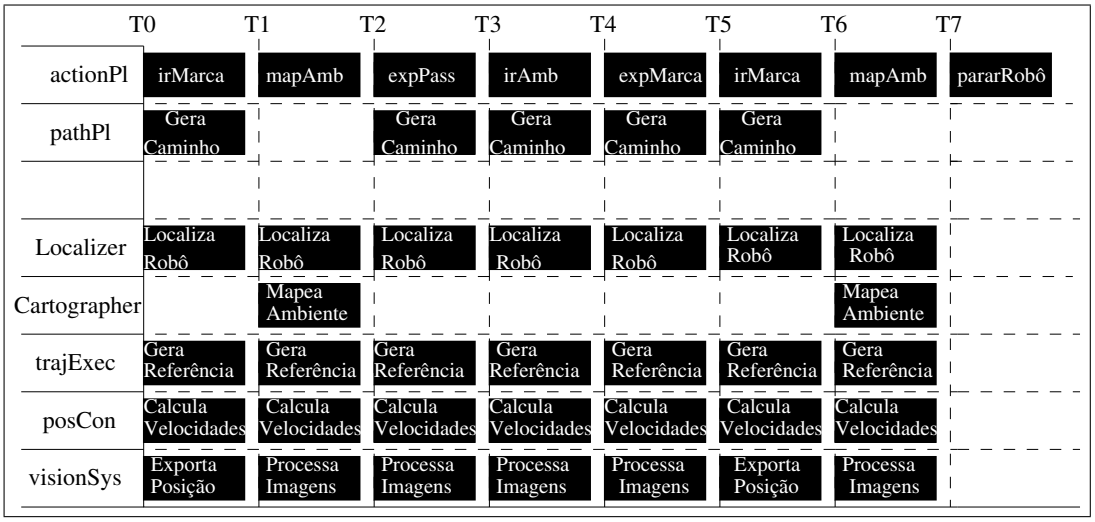

Figura 5.14: Grade de Execução dos Módulos de Software em Função do Tempo

5.3 Conclusões

Neste capítulo foi apresentada a organização de controle proposta neste trabalho, a qual utiliza os recursos e abordagens definidos na arquitetura de hardware e software apresentada no Capítulo 4. Essa organização de controle é híbrida, conseguindo aliar o comportamento reativo da rede de sensores e atuadores com as atividades deliberativas necessárias para realizar tarefas mais complexas.

Foram apresentados alguns cenários hipotéticos de modo a apresentar possíveis configurações da organização de controle. As configurações da organização de controle variam de acordo com a tarefa a ser realizada, onde as ações do módulo `actionPl` e a ativação ou desativação de alguns módulos de software variam de acordo com a complexidade da tarefa.

Capítulo 6

Resultados

Este capítulo apresenta os resultados adquiridos por meio da implementação de um protótipo da organização de controle proposta no capítulo 5.

6.1 Descrição do Protótipo

O protótipo implementado utilizou todas as ferramentas de software desenvolvidas neste trabalho e apresentadas no capítulo 4. Da organização de controle proposta no capítulo 5, foram implementados os seguintes módulos de software: `Localizer`, `Cartographer`, `actionPl`, `pathPl`, `trajExec`, `posCon` e `visionSys`. O protótipo foi executado no robô Kapeck. O funcionamento do módulo `visionSys` foi simulado, mas procurou-se incluir nessa simulação os tempos necessários para captura e processamento das imagens em uma situação real, de modo a tornar a simulação mais realística.

6.2 Experimento

No experimento realizado, a tarefa do robô é encontrar uma marca no piso de um prédio, mapeado *a priori*. O robô deve percorrer todos os ambientes do prédio até encontrar a marca. Caso a marca não seja encontrada após a exploração do último ambiente, o robô deve parar. O módulo `visionSys` possui o papel de capturar e processar as imagens de modo a encontrar a marca. Quando ele identifica a marca, a posição da marca juntamente com uma sinalização para o módulo `actionPl` são exportadas em um *blackboard*. O período de amostragem do módulo `posCon` escolhido empiricamente foi de 100 ms. Tal período é alto para um módulo de software relacionado com o controle de posição do robô, porém este período de amostragem é compensado pelo controle de velocidade embarcado nas placas de acionamento de motor, o qual possui período de amostragem de 10 ms.

O mapa conhecido *a priori* possui as dimensões de 10,46 m de largura por 4,62 m de comprimento. Além disso, ele foi dividido em 3 ambientes, chamados respectivamente de **corredor** (largura de 8,46 m e comprimento de 2,0 m), **sala 01** (largura de 5,98 m e comprimento de 2,60 m) e **sala 02** (largura de 2,0 m e comprimento de 1,2 m). A pose inicial (x; y; θ) do robô,com as coordenadas em metros e a orientação em radianos, é

(0,96; 0,83; 0). Na Figura 6.1 é apresentado o mapa juntamente com o caminho planejado e o caminho executado pelo robô.

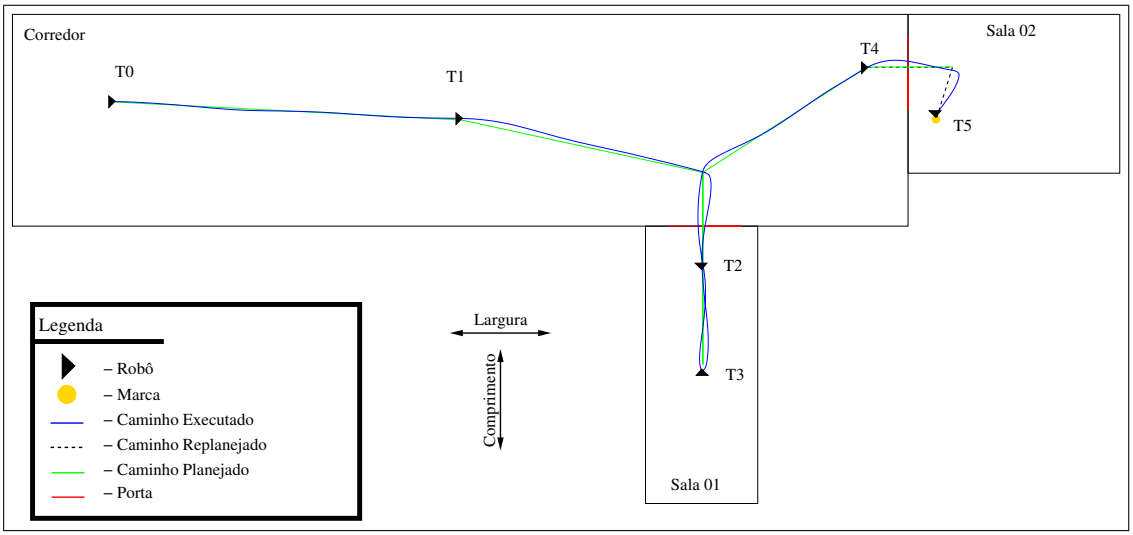

Figura 6.1: Cenário 2

As ações configuradas no módulo `actionPl` foram:

- **irCentro -** Ir para centro de um ambiente não-percorrido;
- **irAmbiente -** Ir para um outro ambiente ainda não-percorrido;
- **irMarca -** Ir para a posição da marca;
- **pararRobô -** Parar o robô.

O módulo `Cartographer` está desativado neste cenário, uma vez que ele não é necessário para a realização da tarefa.

6.2.1 Execução da Tarefa

No momento T0, o módulo `actionPl` dispara uma ação do tipo **irCentro**, uma vez que o ambiente onde o robô se encontra ainda não foi explorado. Conseqüentemente, o módulo `pathPl` gera um caminho até o centro do ambiente e o exporta em um *blackboard*. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`. No momento T2, o robô tem o mesmo comportamento.

No momento T1, o módulo `actionPl` dispara uma ação do tipo **irAmbiente**, uma vez que o ambiente onde o robô se encontra já foi explorado. Conseqëentemente, o módulo `pathPl` gera um caminho até o próximo ambiente não-explorado e o exporta em um *blackboard*. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`. No momento T3, o robô tem o mesmo comportamento.

No momento T4, o módulo `actionPl` dispara uma ação do tipo **irMarca**. Esta ação é disparada em função do encontro da marca pelo módulo `visionSys`. Devido a isso, o

resto da caminho gerado no momento T3 é descartado e o `actionPl` solicita ao módulo `pathPl` a geração de um novo caminho, que leve o robô até a marca encontrada. O módulo `trajExec` lê o caminho gerado por `pathPl` e gera as referências a serem executadas pelo módulo `posCon`.

No momento T5, o módulo `actionPl` dispara uma ação do tipo **pararRobô**, uma vez que a marca foi encontrada e alcançada. Desta forma, todos os módulos do robô são parados, bem como o próprio robô.

O módulo *Localizer* funciona de acordo com a forma descrita na seção anterior, onde ele requisita as leituras dos encoders das rodas do robô para poder estimar a posição do robô dentro do mapa conhecido *a priori*.

Na Figura 6.2 é apresentada a grade de execução dos módulos em função do tempo, expondo o que cada módulo faz em cada momento de acordo com a ação dispara pelo módulo `actionPl`.

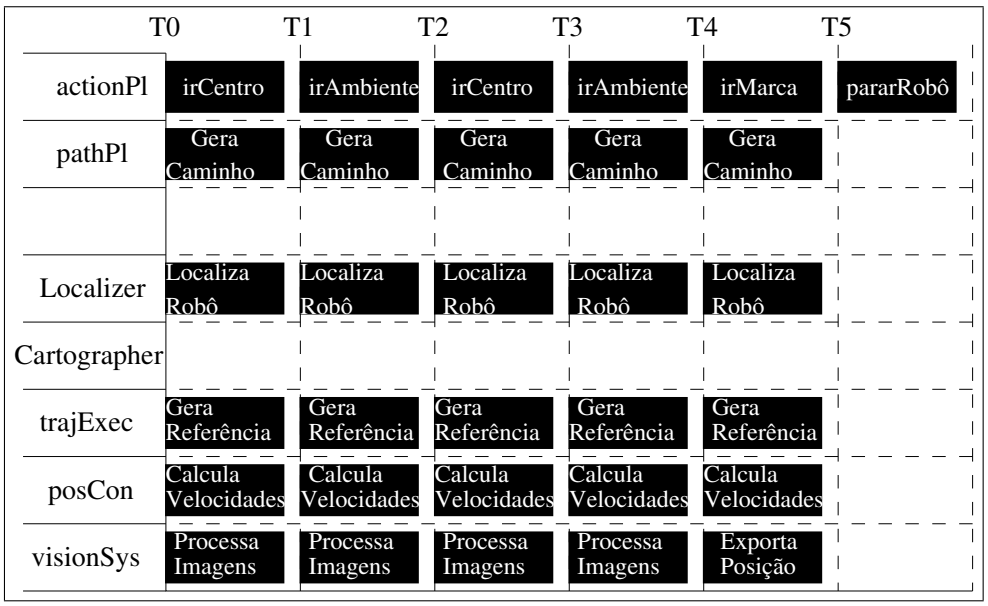

Figura 6.2: Grade de Execução dos Módulos de Software em Função do Tempo

6.2.2 Resultados Adicionais

Além dos resultados já apresentados, também foram coletados os tempos de execução do módulo `posCon` (Figura 6.3), de modo a analisar o papel da ferramenta `periodic`; os tempos de leitura dos encoders (Figura 6.5) e os tempos de acionamento dos motores (Figura 6.6), para analisar o desempenho do da ferramenta `robot`; os tempos de acesso ao módulo `bBoardServer` (Figura 6.4), de modo a analisar o desempenho do `bBoardRe`.

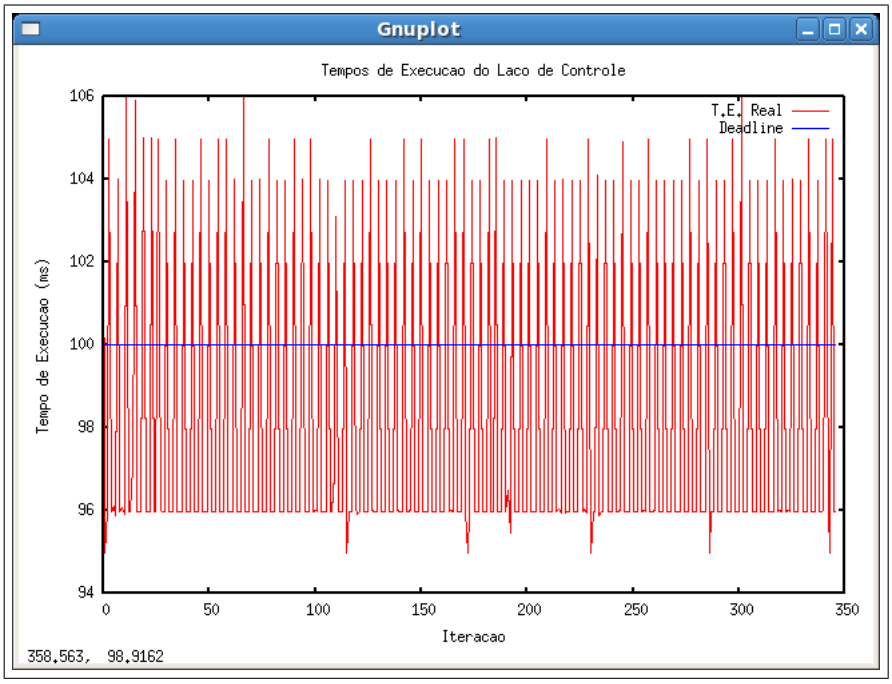

Figura 6.3: Tempos de Execução do Módulo posCon

Resultados da utilização de Periodic

Como pode ser visualizado na Figura 6.3 e na Tabela 6.1 e na Tabela, a ferramenta `periodic` permitiu a execução do módulo `posCon` respeitando o deadline imposto na maioria das vezes. Mesmo nas vezes que o deadline não foi respeitado, o tempo adicional foi muito pequena se comparado com o período de amostragem desse módulo.

Levando-se em consideração que o Linux Standard, o qual é utilizado no robô, não é otimizado para aplicações *real-times* e que o robô não consiste em um sistema *hard real-time*, o desempenho do sistema com o uso da ferramenta proposta mostra-se satisfatório.

Tabela 6.1: Média e Desvio Padrão dos Tempos de execução do Módulo posCon

Média	Desvio Padrão
98,253	3,856

Resultados da utilização de bBoardRe

Como pode ser visto na Figura 6.4 e na Tabela 6.2, os tempos de acesso ao `bBoardRe` utilizado pelo módulo `actionPl` são relativamente altos (40 ms), porém, como esses dados são acessados esporadicamente e são necessários em operações mais lentas, não prejudicam o desempenho do sistema robótico.

Pode-se afirmar que a utilização de *blackboards* remotos por meio do `bBoardRe` é indicada para módulos de software que tenham funcionamento em períodos de amostragem maiores.

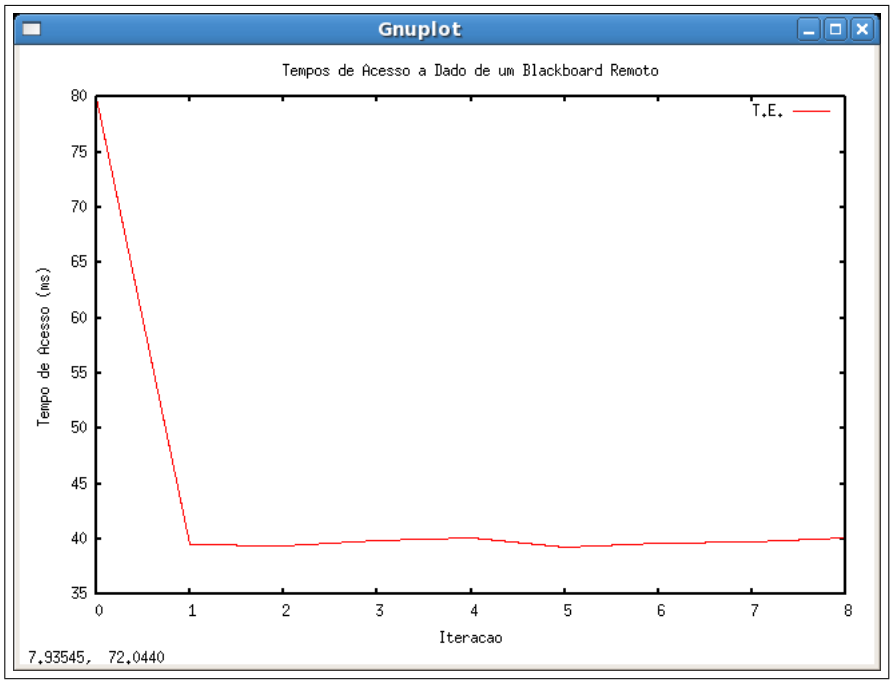

Figura 6.4: Tempos de Comunicação com o bBoardServer

Tabela 6.2: Média e Desvio Padrão dos Tempos de Comunicação com o bBoardServer

Média	Desvio Padrão
13,385	44,185

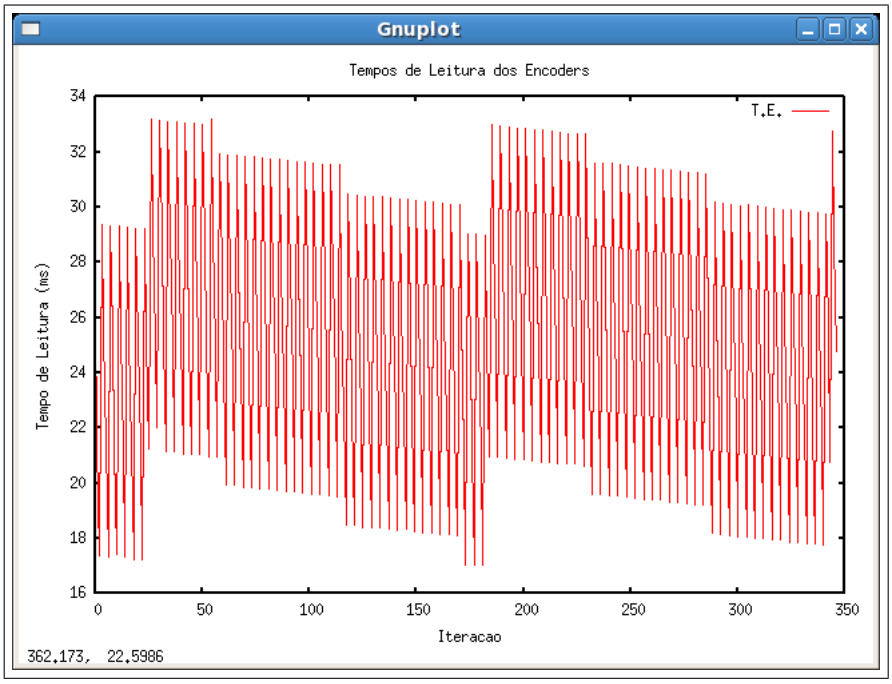

Figura 6.5: Tempos de Leitura dos Encoders

Resultados da utilização de Robot para leitura de Encoders

Como pode ser visto na Figura 6.5 e na Tabela 6.3, os tempos de leitura dos dois encoders do robô foram feitos em um tempo relativamente alto se comparado ao período de amostragem do módulo que necessitou de tais leituras (`posCon`). Acredita-se que tais tempos são decorrentes da utilização do barramento CAN em modo mestre-escravo, uma vez que a leitura de um encoder consiste em uma escrita no barramento e na espera pela resposta da placa microcontrolada. Como o módulo `Localizer` realiza tal operação duplamente, o tempo necessário para adquirir as leituras é duplicado.

É importante salientar que mesmo os tempos de leitura dos encoders sendo maiores que o desejado, o sistema robótico não teve seu funcionamento prejudicado, pois o controle embarcado nas placas de acionamento de motores compensam o atraso encorporado à odometria calculada em alto nível pelo `Localizer`.

Tabela 6.3: Média e Desvio Padrão dos Tempos de Leitura dos Encoders

Média	Desvio Padrão
25,652	4,68

Resultados da utilização de Robot para Acionamento de Motores

Como pode ser visto na Figura 6.6 e na Tabela 6.4, o acionamento dos motores foi feito em um taxa de tempo satisfatória. O tempo necessário para acionar um motor é bem

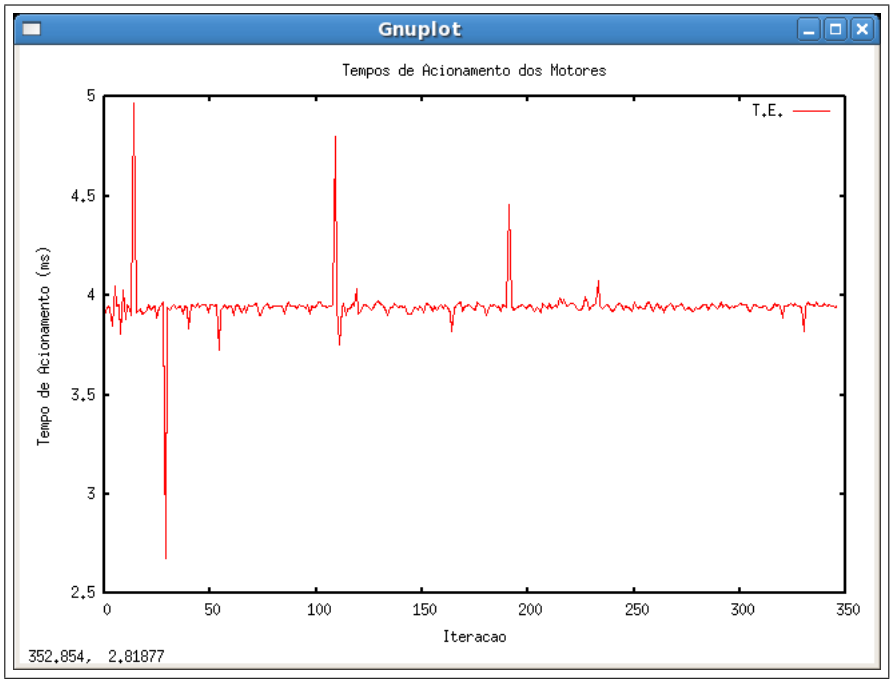

Figura 6.6: Tempos de Acionamento dos Motores

menor que o tempo necessário para ler um encoder em função desta operação consistir em apenas uma escrita no barramento CAN, sem ocorrer uma espera por uma operação de leitura bloqueante como na operação de leitura de encoder.

Tabela 6.4: Média e Desvio Padrão dos Tempos de Acionamento dos Motores

Média	Desvio Padrão
3,933	0,168

6.3 Conclusões

Neste capítulo foram apresentados os resultados experimentais obtidos a partir da execução de um protótipo da organização de controle apresentada no Capítulo 4, a qual utiliza a arquitetura proposta neste trabalho. Através do experimento realizado pôde-se demonstrar:

- A utilização da abordagem de interação entre módulos de software por meio de *blackboards* distribuídos.
- A utilização da representação simbólica do barramento CAN (ferramenta `robot`), bem como o funcionamento do controle ao acesso ao barramento CAN por meio do árbitro de barramento.
- A utilização da ferramenta `periodic`.

Capítulo 7

Conclusões

Este trabalho apresentou uma arquitetura de hardware e software para controle de um robô móvel. Também foi apresentada uma organização de controle híbrida, que utiliza todos os recursos fornecidos pela arquitetura de hardware e software.

No que se refere a arquitetura de hardware, foi definida uma arquitetura distribuída, que utiliza 2 computadores embarcados para processamentos de alto nível e 8 placas microcontroladas que desoneram computacionalmente os computadores embarcados, sendo essas placas responsáveis pelo controle de velocidade dos motores, pela leitura dos encoders e pelo acionamento e leitura dos sonares do robô.

A arquitetura de software proposta consiste em um conjunto de ferramentas que permitem a utilização do hardware distribuído do robô em sua plenitude. Foram definidas 3 ferramentas:

- ***Blackboards -*** Áreas de memória compartilhada que permitem a interação entre vários módulos de software distribuídos;
- ***Robot -*** Ferramenta que permite a implementação dos módulos de software desacoplada diretamente do hardware do robô, uma vez que os módulos de software utilizam uma representação lógica da rede de sensores e atuadores do robô;
- ***Periodic -*** Ferramenta que permite execução dos módulos de software do robô de uma forma mais temporalmente previsível.

A organização de controle proposta é híbrida e possui os requisitos necessários para aliar a reatividade da rede de sensores à capacidade de deliberação necessária para a realização de tarefas de relativa complexidade.

As principais contribuições desse trabalho foram:

- Ferramentas de software que permitem a programação de módulos de software distribuídos. Além disso, a implementação desse módulos de software é desacoplada do hardware do robô, o que se mostra muito útil no tocante a manuntenção e alteração dos módulos de software e hardware do sistema robótico.
- Uma organização de controle que permite a realização de tarefas autônomas por um robô móvel com rodas.

Como pontos que podem ser melhorados neste trabalho, pode-se citar:

- A gerência da rede de sensores implementada é muito simples.
- A implementação dos módulos de software da organização de controle proposta é muito simples.
- O método de comunicação entre o módulo `actionPl` e os outros módulos da organização de controle não é o mais eficiente. A utilização de sinais unix para sinalizar aos outros módulos qual ação deve ser executada se mostra uma abordagem promissora.

Além dos pontos a serem melhorados citados, a partir da execução do protótipo implementado pôde-se constatar que existem alguns pontos de estrangulamento no sistema robótico. Esses pontos de estrangulamento acontecem em subsistemas onde existe um grande tráfego de informação em um curto período de tempo (gerência da rede de sensores e atuadores) e também em subsistemas onde existe um tráfego mais esparso de informações, porém este tráfego exige muito poder computacional do sistema robótico (sistema de visão computacional).

Estudos preliminares indicam que a utilização de hardware reconfigurável e/ou dedicado nesses subsistemas pode desonerar computacionalmente as unidades de processamento principais do sistema robótico, melhorando o desempenho global do sistema.

No tocante aos caminhos a serem seguidos em trabalhos futuros, pode-se citar:

- Uma investigação sobre métodos mais elaborados de gerência de redes de sensores e posterior implementação;
- Melhoramento da implementação dos módulos de software da organização de controle;
- Modelagem e verificação formal da arquitetura já desenvolvida, provavelmente com redes de petri temporizadas;
- Realização de análise de desempenho, análise temporal e de tolerância a falhas da arquitetura;
- Realização de estudos mais profundos para se averiguar como será utilização de hardware reconfigurável ou dedicado na arquitetura.

Referências Bibliográficas

Alami, R., R. Chantila & B. Espiau (1993), Designing an inteligent control archtecture for autonomous mobile robots, *em* 'Proceedings of ICAR'93', Tokyo, Japão, pp. 435–440.

Albus, J. S., H. G. McCain & R. Lumia (1989), NASA/NBS standard reference model for tele-robot control system architecture (NASREM), Relatório técnico, National Institute of Standards and Tecnology.

Ardizzone, E., A. Chella, I. Macaluso & D. Peri (2006), A lightweight software architecture for robot navigation and visual logging through environmental landmarks recognition, *em* 'Proceedings of ICPP'06', Ohio,USA.

Arkin, R. C., E. M. Riseman & A. Hansen (1987), AuRa: An architecture for vision-based robot navagation, *em* 'DARPA Image Understanding Workshop', Los Angeles, USA, pp. 413–417.

Atta-Konadu, Rodney, Sherman Y. T. Lang, Chris Zhang & Peter Orban (2005), Design of a robot control architecture, *em* 'Proceedings of the IEEE International Conference on Mechatronics and Automation', Canada.

Berger, Matthias Oliver, Olaf Kubitz, René Dumoulin & Robert Posielek (1997), A modular, layered client-server control architecture for autonomous mobile robots, *em* 'Proceedings of ISIE'97', Guimarães, Portugal, pp. 697–701.

Bonasso, R. P., J. Firby, E. Gat, D. Kortenkamp, D. Miller & M. Slack (1997), 'A proven three-tired architecture for programming autonomous robots', *Journal of Experimental and Theoretical Artificial Inteligence* **9**(2).

Bosch, R. (1991), Bosch CAN specification, Relatório técnico, Bosch Gmbh.

Brooks, R. A. (1986), 'A robust layered control system for a mobile robot', *IEEE Journal Robotics and Automation* **9**(2), 14–23.

Brzykcy, Grazyna, Jacek Martinek, Adam Meissner & Skrzypczynski (2001), Multi-agent blackboard architecture for a mobile robot, *em* 'Proceedings of IROS'01', Muai, USA, pp. 2364–2369.

Carter, Nicholas (2003), *Arquitetura de Computadores*, Bookman.

Chen, Tse M. & Ren C. Luo (1998), Integrated multi-behavior mobile robot navigation using decentralized control, *em* 'Proceedings of IROS'98', Canada, pp. 564–569.

Cia (2007), CAN in automation - cia, Página na internet, CAN in Automation - Cia.
URL: *http://www.can-cia.org/*

Coronel, J. O., G. Benet, J. E. Simó, P. Pérez & M. Albero (2005), CAN-based control architecture using de SCoCAN communication protocol, *em* 'Proceedings of ETFA'05', Catania, Italy.

Dudek, G. & M. Jenkin (2000), *Computational Principles of Mobile Robotics*, Cambridge University Press, UK.

Etschberger, K. & C. Schlegel (2007), CANopen-based distributed intelligent automation, Página na internet, CAN in Automation - Cia.
URL: *http://www.canopensolutions.com/english/articles/ar_1_e.shtml*

Farines, Jean-Marie, Joni da Silva Fraga & Rômulo Silva de Oliveira (2000), *Sistemas de Tempo Real*.

Fayek, R.E.and Liscano, R.and Karam G.M. (1993), A system architecture for a mobile robot based on activities and a blackboard control unit, *em* 'Proceedings of ICRA'93', Ottawa, Canada, pp. 267–274.

Freire, Eduardo, Teodiano Bastos-Filho, Mário Sarcinelli-Filho & Ricardo Carelli (2004), 'A new mobile robot control approach via fusion of control signals', *IEEE Transactions on Systems, Man, and Cybernetics* .

Heinen, Farlei José (2002), Sistema de controle híbrido para robôs móveis autônomos, Dissertação de mestrado, Centro de Ciências Exatas e Tecnológicas, UNISINOS, São Leopoldo, RS.

Hentout, A., B. Bouzouia & Z. Toukal (2007), Behaviour-based architecture for piloting mobile manipulator robots, *em* 'Proceedings of ISIE'07', Vigo,Spain, pp. 2095–2100.

Hsu, Harry Chia-Hung & Alan Liu (2007), 'A flexible architecture for navigation control of a mobile robot', *IEEE Transactions on Systems, Man, and Cybernetics* **37**(3), 310–318.

Kim, Gunhee, Woojin Chung, Munsang Kim & Chongwon Lee (2003), Tripodal schematic design of the control architecture for the service robot PSR, *em* 'Proceedings of ICRA'03', Taipei,Taiwan, pp. 2792–2797.

Kim, Jin-Oh, Chang-Jun Im, Hyun-Jong Shin, Keon Y. Yi & Ho G. Lee (2003), A new task-based control architecture for personal robots, *em* 'Proceedings of IROS'03', Las Vegas, USA, pp. 1481–1486.

Konolige, K. & K. Myers (1998), *Artificial Intelligence and Mobile Robots*, MIT Press, Cambridge, Massachusetts 02142.

Liscano, Ramiro, Allan Manz & Elizabeth R. Stuck (1995), 'Using a blackboard to integrate multiple activities and achieve strategic reasoning for mobile-robot navigation', *Intelligent Robotic System* pp. 381–384.

Liu, Jindong, Huosheng Hu & Dongbing Gu (2006), A hybrid control architecture for autonomous robotic fish, *em* 'Proceedings of IROS'06', China, pp. 312–317.

Ly, D. N., T. Asfour & R. Dillmann (2004), A modular and embedded control architecture for humanoid robots, *em* 'Proceedings of IROS'04', Sendai, Japão, pp. 2775–2780.

Matsui, Toshihiro, Hirohisa Hirukawa, Yutaka Ishikawa, Nobuyuki Yamasaki, Satoshi Kagami, Fumio Kanehiro, Hajime Saito & Tetsuya Inamura (2005), Distributed real-time processing for humanoid robots, *em* 'Proceedings of RTCSA'05', Hong Kong, China.

Medeiros, Adelardo A. D. (1998), 'A survey of control architectures for autonomous mobile robots', *Journal of Brazilian Computer Society* .

Murphy, R. & A. Mali (1997), 'Lessons learned in integrating sensin into autonomous mobile robot architecture', *Journal of Experimental and Theoretical Artificial Inteligence* **9**(2), 191–209.

Murphy, Robin R. (2000), *Introduction to AI robotics*, 2ª edição, MIT Press, Cambridge, Massachusetts 02142.

Murphy, Robin R. & R. C. Arkin (1992), SFX: An architecture for action-oriented sensor fusion, *em* 'Proceedings of IROS'92', Raleigh, pp. 1079–1086.

Occello, Michel & Marie-Claude Thomas (1992), A distributed blackboards methodology for designing robotic control softwares, *em* 'Proceedings of 1992 International Conferenence on System Engineering', Nice, France, pp. 147–150.

Park, Jung-Min, Insup Song, Young-Jo Cho & Sang-Rok Oh (1999), A hybrid control architecture using a reactive sequencing strategy for mobile robot navigation, *em* 'Proceedings of IROS'99', Kyongju, Korea, pp. 1279–1284.

Pedrosa, Diogo, Adelardo A. D. Medeiros & Pablo J. Alsina (2003*a*), Point-to-point paths generation for wheeled mobile robots, *em* 'Proceedings of ICRA'03', Taipei, Taiwan, pp. 3752–3757.

Pedrosa, Diogo, Adelardo A. D. Medeiros & Pablo J. Alsina (2003*b*), Um método de geração de trajetória para robôs não-holonômicos com acionamento diferencial, *em* 'Proceedings of SBAI'03', Bauru, Brazil, pp. 840–845.

Roisemblatt, J. K. & C. E. Thorpe (1995), Combining multiple goals in a behavior-based archtecture, *em* 'Proceedings of IROS'95', Pittsburg, pp. 136–141.

Roisenberg, M., J. M. Barreto, F. de Almeida Silva, R. C. Vieira & D. K. Coelho (2004), Pyramidnet: A modular and hierarchical neural network architecture for behavior-based robotics, *em* 'Proceedings of ISRA'04', Querétero, México, pp. 32–37.

Rosenblatt, Julio (1997), DAMN: A Distributed Architecture for Mobile Navigation, Tese de doutorado, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Seixas, Constantino Filho (2007), DeviceNet, Página na internet, UFMG.
URL: *www.cpdee.ufmg.br/~seixas/PaginaSDA/Download/DownloadFiles/R2_DeviceNet.pdf*

Shen, Jinxiang, Jiangping Liu, Hui Zhang, Lingyun Zhou & Jianbing Tang (2003), Distributed control system for modular self-reconfigurable robots, *em* 'Proceedings of the 2003 IEEE International Conference on Robotics,Intelligent Systems and Signal Processing', Changsha, China, pp. 933–936.

Simmons, G.Reid (1994), 'Structured control for autonomous robots', *IEEE Transactions on Robotics and Automation* .

Tanenbaum, Andrew S. (2003*a*), *Redes de Computadores*, 4ª edição, Campus.

Tanenbaum, Andrew S. (2003*b*), *Sistemas Operacionais*, 2ª edição, Prentice-Hall.

Tanenbaum, Andrew S. (2006), *Organização Estruturada de Computadores*, 5ª edição, Pretice-Hall.

Tangirala, S., S. Kumar, S. Bhattacharyya, M. O'Connor & L. E. Holloway (2005), Hybrid-model based hierarchical mission control architecture for autonomous underwater vehicles, *em* '2005 American Control Conference', Portland,USA.