BRENO CARNEIRO PINHEIRO

SISTEMA DE CONTROLE TEMPO REAL EMBARCADO PARA AUTOMAÇÃO DE MANOBRA DE ESTACIONAMENTO

FLORIANÓPOLIS
2009

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

SISTEMA DE CONTROLE TEMPO REAL EMBARCADO PARA AUTOMAÇÃO DE MANOBRA DE ESTACIONAMENTO

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Automação e Sistemas.

Breno Carneiro Pinheiro

Florianópolis, Março de 2009.

SISTEMA DE CONTROLE TEMPO REAL EMBARCADO PARA AUTOMAÇÃO DE MANOBRA DE ESTACIONAMENTO

BRENO CARNEIRO PINHEIRO

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia de Automação e Sistemas, Área de Concentração em *Controle, Automação e Informática Industrial*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina.'

Prof. Leandro Buss Becker, Dr.
Orientador

Prof. Julio Elias Normey Rico, Dr.
Co-Orientador

Eugênio de Bona Castelan Neto, Ph.D.
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Banca Examinadora:

Leandro Buss Becker, Dr.
Presidente

Julio Elias Normey-Rico, Dr.
Co-Orientador

Antônio Augusto M. Fröhlich, Dr.

Jean Marie Alexandre Farines, Dr.

Ubirajara Franco Moreno, Dr.

Aos meus adoráveis pais Tarcizo e Lúcia, meus irmãos Délio, Tássio e Líssia, e minha amada esposa Kadydja.

AGRADECIMENTOS

Agradeço a toda minha família por todo apoio e motivação que me deram em todos os momentos em minha vida e, em particular, durante esse trabalho. Agradeço em especial aos meus pais, pela ajuda e empenho em todos os sentidos, ao meu irmão mais velho, Délio, pelo seu grande suporte.
Agradeço ao meu Orientador, Prof. Leandro Buss Becker, por sua paciência, serenidade e dedicação e por sempre estar disponível a contribuir com o projeto, indicando caminhos e propondo soluções em favor do desenvolvimento técnico e humano de todos.
Agradeço aos amigos da equipe SIAMES (Daniel Martins, Luiz e Guilherme), sem os quais a execução desse trabalho tornar-se-ia muito mais difícil e muito menos gratificante. Agradeço também aos amigos Marcelo Sobral, pela amizade, e por pertinentes ajudas e Ronaldo Aparecido Silva pelas valiosas dicas.
Agradeço a todos os professores do Departamento de Automação e Sistemas (DAS), o meu muito obrigado por todos os ensinamentos e encaminhamentos. Aos membros dessa banca examinadora, que contribuíram na revisão deste trabalho, colaborando com sugestões, o meu agradecimento.

Agradeço imensamente a minha amada esposa, Kadydja Fonseca, a quem devo muito por sua paciência, dedicação, amor e por ter suportado os momentos mais difíceis sempre firme ao meu lado.
Por fim, agradeço ao Programa de Pós-Graduação em Automação e Sistemas, por toda sua estrutura e oportunidade de desenvolver meu trabalho na UFSC e ao Programa de Desenvolvimento Tecnológico Avançado (PDTA) representado pelo seu coordenador, prof. Wu Feng Chung, pela confiança e total apoio.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia de Automação e Sistemas.

SISTEMA DE CONTROLE TEMPO REAL EMBARCADO PARA AUTOMAÇÃO DE MANOBRA DE ESTACIONAMENTO

BRENO CARNEIRO PINHEIRO

Março/2009

Orientador: Leandro Buss Becker, Dr
Co-Orientador: Julio Elias Normey Rico, Dr
Área de Concentração: Controle, Automação e Informática Industrial
Palavras-chave: Sistemas Embarcados, Sistemas Operacionais, Protocolos de Rede, Controle Distribuído
Número de Páginas: 126

Esta dissertação apresenta uma plataforma *hardware* e *software* adequada ao desenvolvimento de aplicações de controle tempo real embarcado. Além da plataforma em si, são apresentadas e discutidas as principais etapas de desenvolvimento de um SCTRE, onde são ressaltados os modelos computacionais que representam sub-partes do sistema a ser projetado e as ferramentas utilizadas em todo o processo. O trabalho também faz um levantamento sobre alguns sistemas operacionais de tempo real, protocolos de redes e outras ferramentas que podem auxiliar o desenvolvimento de aplicações de controle embarcado.

Nesse sentido, desenvolveu-se um estudo de caso para o controle de seguimento de trajetória aplicado à manobras de estacionamento visando o estudo detalhado de cada etapa apresentada e as principais características das ferramentas escolhidas. O projeto consiste então de um sistema de estacionamento autônomo, onde são realizados o controle de velocidade e direção do veículo projetados e analisados em simulações computacionais.

Para a realização de testes do sistema, visando a uma aplicação real, instrumentou-se um veículo protótipo de modo a ser possível observar detalhes construtivos da aplicação e validar os modelos e projetos de controle desenvolvidos. Por fim, foram realizadas filmagens que constatam a operacionalidade do sistema de estacionamento.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Automation and Systems Engineering.

EMBEDDED REAL TIME CONTROL SYSTEM FOR THE AUTOMATION OF PARKING MANEUVERS

BRENO CARNEIRO PINHEIRO

March/2009

Advisor: Leandro Buss Becker, Dr
Co-Advisor: Julio Elias Normey Rico, Dr
Area of Concentration: Control, Automation and Industrial Computing
Key words: Embedded System, Operating Systems, Networks Protocols, Distributed Control
Number of Pages: 126

This dissertation presents a platform *(*hardware) and *(*software) appropriate to the development of applications of embedded real-time control systems. Besides the platform itself, are presented and discussed the main steps for developing a SCTRE, where they emphasized the computational models that represent sub-parts of the system being designed and the tools used in the process. The work also makes a survey on some real-time operating systems, networking protocols and other tools that can assist in the development of applications of control board.

In this sense, has developed a case study to control applied to trajectory tracking of maneuvering for parking to the detailed study of each stage and presented the main features of the tools chosen. The project is then an autonomous parking system, where they are made to control speed and direction of the vehicle designed and tested in computer simulations.

For testing the system, to a real application, an instrument is a prototype vehicle to be possible to see construction details of the application and validate the models and control of projects developed. Finally, we note that the operational footage of the parking system.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas

ADEOS	*Adaptative Domain Environment for Operating System*
AOSD	*Application Oriented System Design*
ARM	*Advanced Risc Machine*
CAN	*Controller Area Network*
CISC	*Complex Instruction Set Computer*
CPBM	*Controle Preditivo Baseado em Modelo*
CRC	*Cyclic Redundant Check*
CSMA/AMP	*Carrier Sense Multiple Access/Arbitration by Message Priority*
ELDK	*Embedded Linux Development Kit*
eCOS	*Embedded Configurable Configurable Operating System*
ECU	*Electronic Control Unit*
EPOS	*Embedded Parallel Operating System*
ES	*Engenharia de Software*
FPU	*Float Point Unit*
GPL	*General Public License*
HAL	*Hardware Abstraction Layer*
ISO	*International Standards Organization*
ISR	*Interrupt Service Routine*
LIN	*Local Interconnect Network*
MBPC	*Model Based Predictive Control*
MoC	*Model of Computation*
MMU	*Memory Management Unit*
NFS	*Network File System*
OO	*Object Oriented*
POWERPC	*Power Optimization With Enhanced RISC - Performance Computing*
SCTRE	Sistemas de Controle Tempo Real Embarcados
RISC	*Reduce Instruction Set Computer*
RTAI	*Real Time Application Interface*
RTOS	*Real Time Operating System*
SE	Sistemas Embarcados
SIAMES	Sistema Automático de Manobras de Estacionamento

SO	Sistemas Operacionais
TTCAN	*Time-Triggered CAN*
UML	*Unified Modeling Language*

Capítulo 1

Introdução

1.1 Motivação

O avanço tecnológico na área de semicondutores vem permitindo que um número crescente de funcionalidades sejam cada vez mais inseridas em uma única pastilha de silício. Os processadores ganharam maior capacidade de processamento, as memórias armazenam maior quantidade de informações e o fluxo de dados entre as unidades de *hardware* através dos barramentos está cada vez mais rápido. Devido a esse avanço, unidades de processamento (microprocessadores ou microcontroladores) e de armazenamento, combinadas com equipamentos eletromecânicos, passaram a desempenhar funções específicas dentro do contexto das aplicações industriais, por exemplo. Esse agrupamento de dispositivos, presentes numa grande variedade de sistemas como os automotivos, os robóticos, os aeronáuticos e os de telecomunicações, são denominados Sistemas Embarcados (*Embedded Systems*) (MASSA, 2006).

Constituído geralmente de uma infra-estrutura distribuída e heterogênea, a essência do desenvolvimento de tais sistemas é, além implementação do conjunto de funcionalidades relativas à aplicação, o atendimento dos requisitos não-funcionais como desempenho, custo, consumo de energia e requisitos temporais (SANGIOVANNI-VINCENTELLI; MARTIN, 2003). O desenvolvimento de um sistema embarcado (SE) caracteriza-se por apresentar algumas restrições que não são encontradas nos sistemas computacionais de propósito geral. Dentre elas estão:

- Processador: as limitações quanto a capacidade de processamento;
- Memória: o espaço reduzido de memória utilizada para armazenar o programa executável e manipular dados.

- Consumo de Energia: a necessidade de redução da potência ($mW/$MIPS) consumida durante operação.
- Tempo de vida (*lifetime*): as exigências quanto ao tempo esperado para que o sistema esteja em uso e suas possibilidades de expansão.
- Confiabilidade: o elevado grau de confiabilidade exigido pelas aplicação embarcadas.

Uma classe especial de sistemas embarcados distingui-se do restante devido aos requisitos temporais de resposta a eventos externos (LI, 2003), sendo classificada como Sistemas Tempo-Real Embarcado (*Realtime Embedded Systems*). Os sistemas de controle, foco desse trabalho, representam uma importante classe das aplicações computacionais embarcadas, que apresentam exigências quanto ao tempo de computação e instantes de ativação das tarefas.

Na área automotiva, certamente um dos domínios mais relevantes e economicamente atrativo para o desenvolvimento de tais sistemas, as ECU's (*Electronic Control Unit*) são responsáveis por inúmeras tarefas ligado ao controle de subsistemas como, por exemplo, o controle de tração, frenagem e estabilidade do veículo, o que torna o automóvel mais fácil de manusear e mais seguro para dirigir. Uma área de aplicação dos sistemas tempo real embarcado em automóveis que vem despertando interesse de muitos pesquisadores é a automatização de sistemas de transporte em rodovias e a navegação de veículos comerciais autonomamente. Nesses sistemas, a navegação do veículo é realizada considerando o planejamento e a geração de uma trajetória. É necessário que o sistema possa informar a sua posição dentro de um ambiente, bem como os dados do próprio ambiente, possibilitando o veículo seguir uma trajetória definida. Nesse sentido, os sistemas de controle, os sistemas embarcados e os sistemas tempo real, juntamente com seus desafios são unidos para a execução das tarefas desejadas. As questões como atrasos na comunicação, período de amostragem, latência de atuação podem ser decisivos no comportamento adequado desses sistemas.

Para exemplificar e discutir o problema em questão, desenvolveu-se uma aplicação de controle para realização da manobra de estacionamento de veículos de modo autônomo. O projeto do sistema embarcado aproveita componentes de software já produzidos em projetos anteriores de seguimento de trajetória em veículos móveis. Para validar a proposta elaborada, integrou-se o sistema desenvolvido num veículo instrumentado de acordo com os requisitos do projeto. O projeto SIAMES (Sistema Autônomo de Manobras de Estacionamento) pretende desenvolver uma aplicação embarcada que execute a manobra de estacionamento com o mínimo de interferência do motorista, mas que

possa interagir com o mesmo recebendo e enviando informações. Após serem realizadas algumas adaptações, essa proposta pode vir a ser implantada em veículos comerciais, facilitando a vida do usuário e estreitando as relações entre o meio industrial e o acadêmico.

1.2 Objetivo

Esta dissertação tem por objetivo projetar e desenvolver um Sistema de Controle Tempo Real Embarcado (SCTRE) para automatizar manobras de estacionamento em veículo automotivos. Tendo em vista o grande *gap* entre os profissionais da computação e os projetistas de sistemas de controle, devido o conteúdo multidisciplinar desse tipo de aplicação, também discute-se todo o seu ciclo de desenvolvimento. São apresentados os modelos de alto nível usados para representar o problema de forma abstrada, o projeto dos controladores, a integração desses modelos e simulação do sistema totalmente integrado além da escolha de uma plataforma (*HW/SW*) de execução adequada aos requisitos de desempenho.

Nesse sentido, o trabalho apresenta um ambiente de desenvolvimento integrado que auxilia as várias etapas de projeto, pois ressaltam as características do sistema desde a análise inicial até detalhes da implementação. Nesse ambiente, procura-se elaborar modelos independentes da plataforma de execução que expressem as especificações de projeto e o comportamento global do sistema. Assim, todo o comportamento pode ser analisado através de simulações, do modo mais próximo ao real. Além disso, visando às etapas de implementação e elaboração de códigos, destacam-se ferramentas que permitem a obtenção de informações a respeito da demanda computacional da solução proposta.

Ressalta-se ainda um conjunto de componentes de *HW/SW* utilizados para projetar uma plataforma embarcada otimizada para atender os requisitos das aplicações de controle tempo real. Desse modo, considerando principalmente as restrições econômicas a que o desenvolvimento desses SE's está sujeito, lança-se mão de ferramentas que auxiliam o rápido desenvolvimento de aplicações com restrições temporais, através de recursos que permitem o acesso aos dispositivos de *hardware* de modo fácil e previsível. Dentre as características desejáveis para esta plataforma destacam-se: flexibilidade, modularidade, reusabilidade e facilidade de uso. Esses aspectos garantem futuras expansões do sistema sem grandes alterações na estrutura já montada.

Como resultado, tem-se uma aplicação de controle tempo real, onde são discriminadas as etapas de desenvolvimento, a elaboração e a simulação dos modelos computacionais que representam

detalhadamente o sistema e seu comportamento através de um ambiente integrado, e a utilização da plataforma projetada para a execução da solução proposta. Procura-se assim, face aos principais desafios comuns ao sistemas de controle tempo real, elaborar, analisar e utilizar uma proposta de desenvolvimento utilizando componentes (*HW/SW*) que garantam o funcionamento desejado do sistema projetado.

1.3 Estrutura da Dissertação

A organização do restante deste trabalho dá-se da seguinte forma:

- O segundo capítulo apresenta alguns dos principais sistemas operacionais disponíveis atualmente, bem como tecnologias de rede que podem ser utilizados no desenvolvimento de aplicações embarcadas em sistemas automotivos. Outras soluções que auxiliam o desenvolvimento de aplicações de controle são apresentadas como alternativas de projeto.

- O terceiro capítulo apresenta o ciclo de desenvolvimento do projeto de um sistema de controle. Ressalta-se as vantagens do controle multinível e algumas estratégias de controle que pode ser utilizadas para o controle dos subsistemas do veículo e o controle de trajetória.

- O quarto capítulo apresenta a etapas de desenvolvimento de um sistema de controle embarcado. São discutidos os modelo computacionais comumente utilizados e as ferramentas que auxiliam na modelagem e simulação do comportamento global do sistema.

- O quinto capítulo apresenta a plataforma proposta para desenvolvimento de um SE. Destacam-se todas as soluções de *HW/SW* e de rede utilizadas, discutindo suas principais vantagens.

- O sexto capítulo apresenta-se o projeto SIAMES como um estudo de caso, onde realiza-se uma modelagem do sistema em todos os níveis hierárquicos, utiliza-se a plataforma proposta e o projeto dos controladores de velocidade e trajetória. Ao fim, são mostrados os resultados e análise de desempenho do sistema.

- O sétimo capítulo apresenta as conclusões e algumas propostas de trabalhos futuros relacionados ao tema, que podem ser utilizadas a partir dos conhecimentos adquiridos durante a execução do SIAMES.

Capítulo 2

Ferramentas para Projeto e Desenvolvimento de Sistemas Embarcados

2.1 Introdução

Neste capítulo são apresentados alguns ambientes integrados de desenvolvimento para sistemas de controle tempo real embarcado, que auxiliam o projeto durante as etapas de simulação e validação até a geração de código para a plataforma alvo. Além disso, são apresentados alguns dos sistemas operacionais (SOs) que podem ser utilizados em aplicações embarcadas, com ênfase àqueles com suporte de tempo real, devido a sua grande aplicabilidade nos projetos de controle. Alguns protocolos de rede de comunicação adequados às aplicações de controle distribuído também são discutidos, dada sua importância em sistemas com arquitetura distribuída.

2.2 Sistemas Embarcados em Aplicações Automotivas

Por muitos anos, os veículos eram constituídos basicamente de estruturas mecânicas ao contrário do cenário atual, onde se estima que 30% do custo desses produtos esteja concentrado nos componentes eletrônicos instalados. O Mercedes Benz S-Class, por exemplo, possui em torno de 60 ECU's interligadas, responsáveis por diversas funções no automóvel (JOHANSSON; TÖRNGREN; NIELSEN, 2005).

Dado o alto nível de integração e elevada complexidade no desenvolvimento de um SCTRE, é fundamental a escolha correta das ferramentas e linguagens, juntamente com o suporte de uma plataforma adequada. Deseja-se portanto que desenvolvedor tenha disponível:

- **Linguagens:** Linguagens que expressem o comportamento de cada sub-parte do sistema através dos modelos computacionais.
- **Ferramentas:** Ferramentas amigáveis que auxiliem a elaboração desses modelos e permita a integração dos mesmos para a simulação completa;
- **Componentes (*HW/SW*):** Plataformas que permitam um rápido desenvolvimento de aplicações embarcadas com garantias temporais;

Em muitos casos, uma estrutura embarcada, constituída de uma unidade central interligada via um barramento de comunicação a outras de menor capacidade de processamento, aquisitam sinais do ambiente, processam e atuam nos subsistemas do veículo ou auxiliam o motorista sobre quais medidas devem ser tomadas.

Considerando o desenvolvimento desses sistemas, nas próximas secções são apresentados componentes importantes de um SE. Dentre eles está o uso de sistemas operacionais.

2.3 Ambientes Integrados de Desenvolvimento de Aplicações de Controle Embarcado

Segundo (VERBAUWHEDE; SCHAUMONT, 2005), processo de desenvolvimento dos sistemas embarcados pode ser realizado seguindo duas vertentes :

- Vertical: cobre os métodos em *software* de redução do consumo de energia e ocupação de memória ou refinamentos para operações em ponto fixo.
- Horizontal: cobre as várias alternativas de plataformas, incluindo arquiteturas de *hardware* como *Application-Specific Integrated Circuits* (ASIC), *Domain-Specific Processors*, *Digital Signal Processors* (DSP), *Embedded Cores*, *Programmable Processors*, *System-on-chip* (SOC) e as opções de sistemas operacionais.

Considerando todos os desafios que envolvem o projeto e desenvolvimento de um SE, procura-se utilizar ambientes e ferramentas que viabilizem o desenvolvimento confiável desses sistemas. As secções subseqüentes tratam dos ambientes integrados para desenvolvimento de SE's existentes atualmente que constituem soluções para um rápido desenvolvimento, ou seja, a partir da especificação e projeto, passa-se para a execução numa plataforma alvo. Por outro lado, existem as soluções que não estão diretamente ligadas a um ambiente de desenvolvimento, que, apesar de poderem apresentar um tempo de desenvolvimento maior, permitem mais flexibilidade na utilização de ferramentas e composição dos componentes.

2.3.1 Plataformas Prontas

RT-Druid Code Generator

O RT-Druid *Code Generator* desenvolvido pela *Evidence* é uma ferramenta integrada de desenvolvimento de sistemas embarcados aplicáveis principalmente aos sistemas de controle. É uma ferramenta aberta e expansível, que consiste de um gerador de código utilizando sintaxe OIL (*OSEK Implementation Language*). O projeto visa ao desenvolvimento de um ambiente de desenvolvimento que permite o projeto de controle e uso de atuadores sem a necessidade de escrever explicitamente um código.

A principal idéia é, a partir de um projeto em blocos realizado com o Scilab/Scicos, que é um CACSD *Computer-Aided Control System Design*, obter um código gerado automaticamente, que pode ser executado em plataformas de *hardware* como: dsPIC, AVR, ARM7, Altera Nios II. Nessas arquiteturas, dois sistemas operacionais de tempo-real podem ser utilizados: o sistema operacional ERIKA (EVIDENCE, 2008a) ou o Linux/Xenomai (XENOMAI, 2007). A Figura 2.1 mostra como é realizado o projeto.

É possível desenvolver soluções que utilizam interface CAN com a utilização de blocos disponíveis no Scicos. Um exemplo de projeto realizado nesse ambiente e integrado ao RT-Druid, a ser executado por uma plataforma composta pelo sistema operacional Erika e o processador dsPIC, é mostrado na Figura 2.2.

Essa ferramenta apresenta-se em contínuo desenvolvimento visando cobrir outras arquiteturas e inserindo novos blocos funcionais ao Scilab/Scicos.

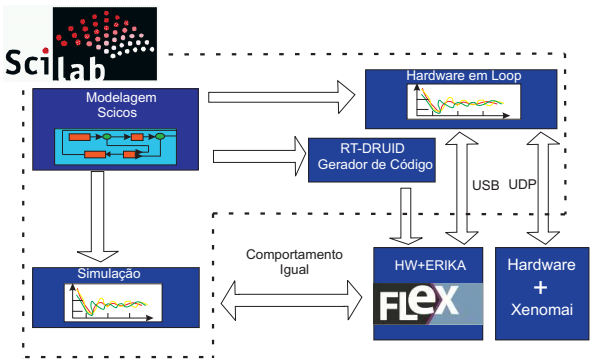


Figura 2.1: Ambiente de Desenvolvimento Scilab/Scicos com RT-Druid Code Generator (EVIDENCE, 2008b)

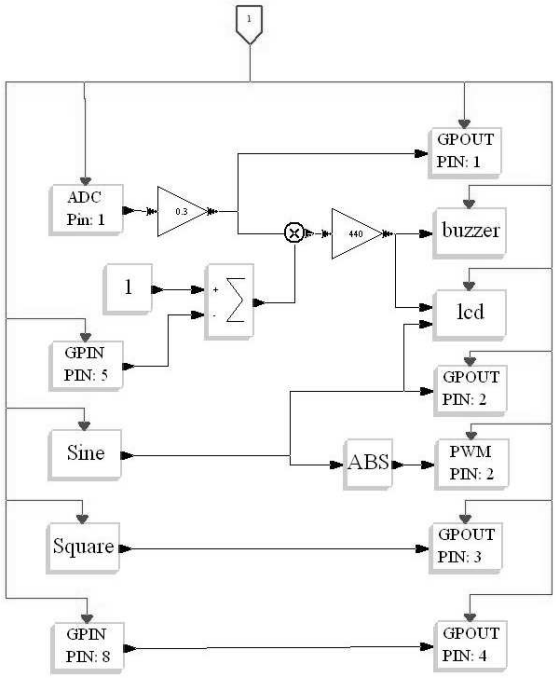


Figura 2.2: Aplicação em diagrama de blocos feito em Scicos a ser executado pela plataforma FLEX (ERIKA + dsPIC)

RTAI-Lab

O RTAI-LAB (BUCHER; MANNORI; NETTER, 2008) é uma conjunto de ferramentas específicas para desenvolvimento de sistemas de controle tempo-real. Os diversos componentes constituintes são: o RTAI, o Comedi, o RTAI-lib e o XRTAI-lab.

O RTAI e o Comedi foram discutidos anteriormente, os outros componentes desta ferramenta são detalhado a seguir:

- RTAI-Lib: é um conjunto de blocos para o Scicos que possibilitam o projeto de controle

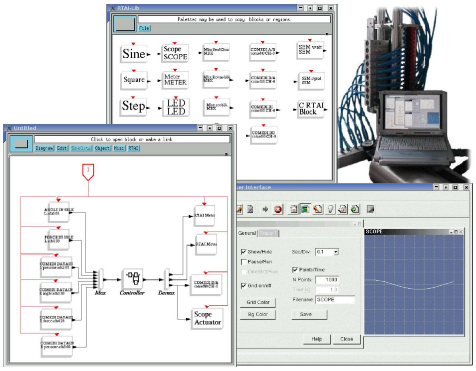


Figura 2.3: Blocos do RTAI-Lab adicionados ao Scilab/Scicos e visualizadores de sinal (BUCHER; MANNORI; NETTER, 2008)

em blocos com sensores e atuadores. Ele provê uma interface para o RTAI e para placas de aquisição de dados, ou seja, aplicações em espaço de *kernel* podem trocar dados com os blocos do Scicos via, por exemplo, *mailboxes* ou *shared memory* e então mostrar os resultados da aplicação durante a execução do sistema. Os blocos em Scicos do RTAI-Lab também permitem geração automática de executáveis para sistemas com suporte de tempo real.

- XRTAILAB: consiste de um osciloscópio que pode ser conectado aos executáveis gerados para aplicações tempo real. Dentre suas maiores vantagens destaca-se a possibilidade de visualizar sinais ou alterar parâmetros da aplicação enquanto o sistema está em execução.

A Figura 2.3 mostra esse conjunto de componentes. Essa ferramenta está disponível sem custo algum, bem como vários exemplos de aplicações. Em analogia a ferramentas comerciais, pode-se fazer a seguinte associação:

- Scilab/Scicos → Matlab/Simulink; [1]
- RTAI-Lib → Real-Time Workshop; [2]
- XRTAI-Lab → LabView; [3]

MicroAutoBox

O MicroAutoBox da dSpace (DSPACE, 2008) é uma plataforma comercial de desenvolvimento voltada para sistemas de controle automotivo. Com essa plataforma é possível desenvolver, testar

[1] Matlab e Simulink são marcas registradas da Mathworks
[2] Real-Time Workshop uma marca registrada da Mathworks
[3] LabView é uma marca registrada da National Instruments Corp.

e otimizar funções de controle que são executadas nas ECU's dispostas no veículo, ou seja, o MicroAutoBox é perfeito para elaboração de protótipos. Além disso, a ferramenta possui interfaces CAN, LIN e FlexRay.

A estrutura interna de *hardware* consiste de um PowerPC IBM 750FX com entradas e saídas analógicas e digitais e vários conectores. A Figura2.4 mostra o MicroAutoBox com suas entradas e saídas.

Figura 2.4: MicroAutoBox da dSpace (DSPACE, 2008)

O MicroAutoBox trabalha integrado com o Simulink, onde é realizado o projeto utilizando os blocos funcionais dessa ferramenta. A partir de então, é utilizado um gerador de código automático e carregado na plataforma.

2.3.2 Plataformas Personalizados

Seguindo o desenvolvimento do sistema na vertente horizontal (VERBAUWHEDE; SCHAUMONT, 2005), pode-se decidir por arquiteturas personalizadas, onde tanto *hardware* como *software* podem ser escolhidos ou mesmo projetados de acordo com as necessidades específicas de cada aplicação.

Nesse tipo de solução estão presentes os circuitos integrados dedicados a uma aplicação específica (ASIC's), estruturas de *hardware* programável como FPGA's e CPLD's ou ainda sistemas que integram várias estruturas eletrônicas num único circuito integrado (*System on Chip*).

No entanto, é comum adotar soluções híbridas, com unidades centrais de processamento compostas pelos processadores conhecidos como COTS (*Commercial Off-The-Shelf*). A Figura 2.5 mostra o exemplo de uma solução para um sistema de freios do veículo e controle de estabilidade da Feescale, onde estão presentes circuitos dedicados ao sensoriamento e atuação nas rodas do veículo.

Visando atender principalmente a indústria automotiva, alguns fabricantes de microcontroladores como a Atmel e a Microchip já incorporam controladoras CAN em seus produtos como o AT90CAN

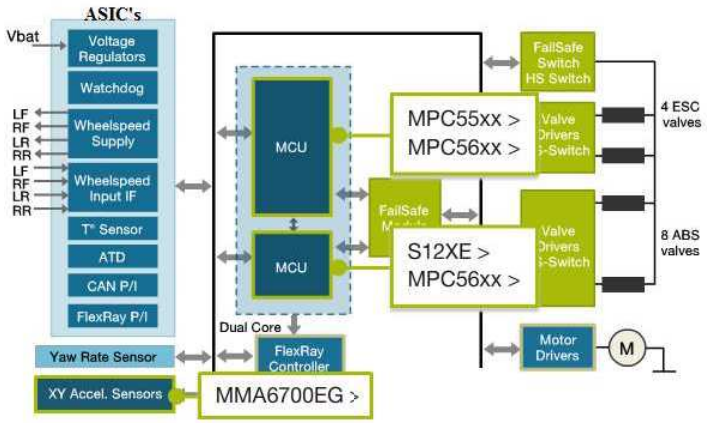


Figura 2.5: Sistema de Freios e Controle de Estabilidade do Veículo (FREESCALE, 2008)

e o PIC18C658 respectivamente. Essa, então, passa ser uma opção para o processamento do controle local dos subsistemas de veículos, onde a principal vantagem está na flexibilidade que esses dispositivos apresentam, pois são facilmente reprogramáveis e amplamente utilizados em diversas aplicações.

Além do *hardware* propriamente dito, pode-se optar por utilizar um sistema operacional ou desenvolver o próprio *firmware*, utilizando as plataformas de desenvolvimento proprietário como o *Code Composer* da Texas Instruments ou *Code Warriors* da Freescale. Para aplicações onde SO's serão utilizados, o fator desempenho é crucial, uma vez que essa camada de *software*, apesar de gerenciar o sistema, passa a compartilhar os recursos de *hardware* juntamente com a aplicação.

2.4 Sistemas Operacionais

Os sistemas computacionais podem ser vistos como uma estrutura multinível, onde, entre cada nível, existe um "interpretador" que abstrai ou facilita o acesso às funcionalidades da camada imediatamente abaixo (Tanenbaum, 2000). A Figura 2.6 mostra essa organização, onde o SO oferece serviços como: acesso facilitado ao *hardware*, abstração dos detalhes da arquitetura e alocação coordenada dos recursos disponíveis entre as tarefas em execução.

Apesar de muitas vezes não estar claro quais as fronteiras entre *hardware* e *software*, um sistema operacional consiste de uma camada de *software* entre a aplicação de alto nível e as camada de baixo nível, que desempenha algumas funções importantes como:

- Tratamento de Interrupções (*Interrupt Handling*);

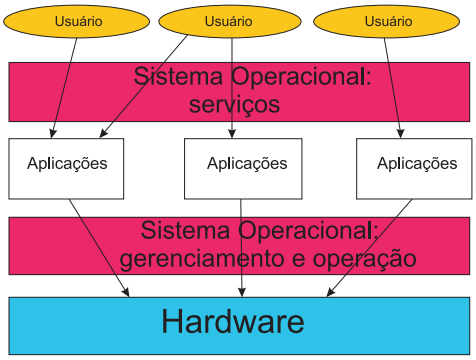


Figura 2.6: Estrutura em Camadas dos Sistemas Computacionais (GAGNE, 2004)

- Chaveamento de contexto entre tarefas (*Context Switching*);
- Gerenciamento de memória;
- Sistema de arquivos;

Considerando o ambiente multitarefas como o das aplicações embarcadas, essas funcionalidades citadas são extremamente úteis, com destaque ao escalonador de tarefas, cujo intuito é decidir qual a próxima tarefa que irá ocupar o processador. O escalonador é geralmente implementado como um *Interrupt Service Routine* (ISR) das interrupções do timer. Assim, a saída de uma tarefa e a entrada de outra requer pelo menos três estágios: salvar o contexto da tarefa corrente, executar o código do escalonador e carregar o contexto da próxima tarefa a ser executada. Apesar de favorecer o desenvolvimento de aplicações multitarefas, o processo de escalonamento é uma importante fonte de atrasos (*jitter*) a que as tarefas estão sujeitas.

Para os sistemas embarcados, algumas características presentes num SO podem ser vantajosas (ÅRZÉN; CERVIN, 2005; HENRIKSSON, 2006):

- Possibilidade de incorporar *device drivers* ao sistema e seus tratadores de interrupção;
- Possibilidade de medição do tempo de execução das tarefas, detecção de *overruns* e perda de *deadlines*;
- Possibilidade de alteração de parâmetros das tarefas *on-line*: prioridades, períodos de ativação entre outros.

A redução no tempo de projeto, custo de mão de obra e conseqüentemente o "*time to market*", faz então da escolha correta do SO umas das etapas de projeto mais relevantes. Sob essa ótica, outro fator

importante é quanto o tipo de sistema a ser utilizado, se um sistema aberto ou comercial. Apesar de não haver uma resposta exata para isso, mesmo entre os desenvolvedores, cada caso pode ser analisado separadamente. Alguns fatores como: documentação detalhada, suporte técnico eficiente, pequeno *footprint*, defeitos e portabilidade são considerados melhores cobertos por versões comerciais de sistemas. Porém, outras questões como facilidade de melhorias ou adaptação no sistema, código aberto e, portanto, maiores garantias contra a descontinuidade do software e o baixo custo são favorecidos pelos sistemas abertos e potencialmente relevantes na decisão final.

Dentre os sistemas operacionais, destacam-se aqueles que procuram fornecer garantias temporais em durante sua execução, denominados sistemas operacionais de tempo real (SOTR). Esse tipo de SO possui peculiaridades que o distingue dos sistemas operacionais de propósito geral (SOPG).

Devido a sua presença em situações críticas no que tange à segurança (*safety critical*), os sistemas embarcados precisam, além de fornecer o serviço desejado, fazê-lo em um determinado tempo. Sendo assim, em um SOTR, o tratamento às interrupções de tempo (*time interrupt handling*) é fundamental para o comportamento previsível do sistema, pois dela dependem (RAMMIG, 1999):

- Verificação de possíveis perdas de *deadlines*;
- Instante de ativação das tarefas periódicas;
- Monitoramento da integridade do sistema;

São portanto os aspectos temporais do comportamento do sistema que o definem como um sistema de tempo real, ou seja, a garantia *a priori* do desempenho temporal de um SO é um requisito fundamental para sua utilização nas aplicações de tempo-real (KOPETZ, 1998).

Algumas métricas são importantes para análise antes do desempenho de um SOTR. Questões como tamanho de memória ocupada (*footprint*), baixa sobrecarga *overhead* associado às trocas de contexto entre tarefas, rápidas respostas à ocorrências de interrupções externas, uso limitado ou nulo de memória virtual, suporte às restrições temporais como escalonamento baseado em prioridades e relógios (*clocks*) de tempo real (MASSA, 2006). Enquanto os SOPG's procuram alocar recursos de maneira justa entre as aplicações presentes no sistema, os SOTR's visam a garantir que tais alocações sejam realizadas de modo a atender todos os requisitos temporais da aplicação.

2.4.1 Sistemas Operacionais Não Baseados em Linux

Nessa secção são apresentados alguns SO's, com destaque àqueles que atualmente podem ser utilizados nos processadores escolhidos no sistema desenvolvido nesse trabalho.

EPOS O sistema EPOS (*Embedded Parallel Operating System*) é um *framework* baseado em componentes para a geração de sistemas de suporte de execução a aplicações de computação dedicada. O projeto é baseado na metodologia de Projeto de Sistema Orientado à Aplicação (AOSD)(FRöHLICH, 2001), que permite o desenvolvimento de aplicações independentes da plataforma. Essa metodologia guia o processo de engenharia de domínio na direção de famílias de componentes, onde as dependências do cenário de execução são fatoradas como aspectos, e as relações externas são capturadas em um *framework* de componentes. O AOSD portanto visa à automatização do processo de desenvolvimento de sistemas computacionais dedicados através de ferramentas de análise de aplicação que permitem a geração do suporte computacional que agrega apenas os recursos necessários e suficientes para suportar a aplicação-alvo. A utilização do AOSD permite, portanto, que componentes altamente reutilizáveis e adaptáveis sejam facilmente combinados na geração de uma instância do sistema.

As famílias de abstrações do EPOS representam abstrações tradicionais de sistema operacional, e implementam serviços como gerência de memória e processos, coordenação entre processos, temporização e comunicação. Cada família de abstrações é composta por um conjunto de componentes de SO com funcionalidades semelhantes entre si. Todas essas abstrações são modeladas e implementadas independentemente do cenário de execução. As unidades de *hardware* dependentes da arquitetura são abstraídas como mediadores de *hardware*, que exportam, através de suas interfaces independentes de plataforma, as funcionalidades exigidas pelas abstrações (WANNER, 2006).

Uma instância do sistema consiste de uma plataforma de hardware e o correspondente sistema de suporte de tempo de execução: abstrações, mediadores de hardware, adaptadores de cenário e aspectos. A Figura 6.1 mostra uma visão do projeto de sistemas orientado à aplicação, onde observa-se que as diferenças entre componentes em uma mesma família são exploradas através das hierarquias de classe, e uma "interface inflada" exporta a família como se fosse um "super" componente, que implementa as funcionalidades atribuídas à família (FRöHLICH, 2001).

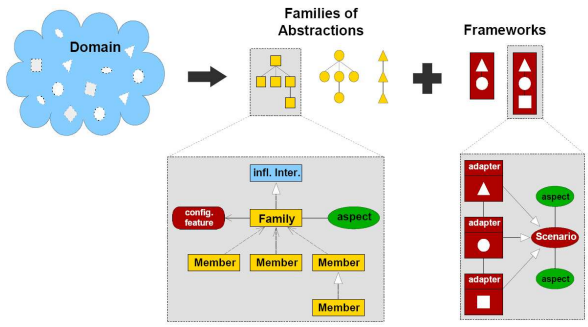


Figura 2.7: Visão Geral do Projeto de Sistemas Orientado à Aplicação (FRöHLICH, 2001)

FreeRTOS O FreeRTOS (FREERTOS, 2008) é um *kernel* de tempo real com suporte a multiprogramação, que pode ser utilizado em aplicações comerciais que utilizam diversas arquiteturas(AVR, PowerPC, MIPS), sendo atualmente mantido por uma comunidade de usuários. Dentre as principais funcionalidades fornecidas pela API do FreeRTOS estão (FREERTOS, 2008):

- Gerenciamento de Tarefas (*tasks*):
  - O número de tarefas é limitado pelo tamanho da RAM disponível;
  - *Kernel* interrompível (*preemptive*);
  - Atribuição de prioridades às tarefas e possibilidade de alterar parâmetros em tempo de execução;
- Mecanimos de comunicação entre tarefas (*Inter-task Communication*):
  - Filas (*Queue*);
  - Semáforos (binários ou de contagem) e mutexes com herança de prioridades;
- Escalonador de tarefas:
  - *Round-Robin* preemptável, ou seja, tarefas de menor prioridade serão interrompidas caso haja uma tarefa de maior prioridade pronta para executar.

O FreeRTOS caracteriza-se por ser de fácil compreensão e uso, podendo ser integrado a ferramentas de desenvolvimento e depuração. Devido ao seu alto desempenho e tamanho reduzido, permite utilização de processadores mais baratos, eliminando, por exemplo, uso de memória externa. As aplicações desenvolvidas sobre o FreeRTOS são organizadas em tarefas (*tasks*), ideal para

aplicação embarcadas. A organização do *kernel* e seu código escrito em C facilitam a compreensão e adição de novas características ao sistema ou alterações de algumas funcionalidades.

ERIKA Enterprise ERIKA Enterprise (EVIDENCE, 2008a) é um típico sistema operacional de tempo real para sistemas incorporados para pequenos sistemas embarcados. O ERIKA é o primeiro sistema comercial com pequeno *footprint* que suporta mecanismos avançados de programação para arquiteturas multiprocessadas.

O sistema está disponível para uma ampla variedade de processadores (8, 16 e 32 bit) e é totalmente configurável em termos de serviços que devem ser incluídas na imagem final e em número de objetos (tarefas, recursos, eventos entre outros). O ERIKA Enterprise pode ser utilizado com sucesso em todas as situações em que fatores como desempenho, dimensões reduzidas, e efetiva utilização do *hardware* são necessários.

Dentre suas principais características destacam-se:

- Utilizável em microcontroladores de 8bits e arquiteturas multiprocessadas;
- Algoritmos de escalonamento estático e dinâmico como *Earliest Deadline First*
- Rápidas trocas de contexto;
- Ferramentas de desenvolvimento para arquiteturas multiprocessadas;

2.4.2 Kernel do Linux para Sistemas Embarcados

O suporte de um SO é fundamental para o processo de desenvolvimento de um sistema embarcado. Se por um lado os sistemas operacionais proprietários (ex. Windows) fornecem vantagens como padronização e evolução conjunta com as novas tecnologias de *chips*, por outro, os sistemas de código aberto (ex. Linux) facilitam a adaptação do SO às exigências da aplicação. Nesse sentido, o Linux surgiu como uma grande solução para os desenvolvedores, pois é um SO de código aberto, amplamente difundido, que facilita o uso dos recursos de *hardware* do sistema, pois provê abstrações de alto nível como processos, *sockets*, sistema de arquivos, grande variedade de *device drivers* e documentação extensiva. No entanto, o Linux utilizado para sistemas embarcados apresenta algumas diferenças em relação ao utilizado em computadores convencionais como: binários para determinado processador/plataforma, ausência de ambiente de janelas, ferramentas de

desenvolvimento (analisadores de memória e depuradores) e ferramentas de desenvolvimento voltadas para outras plataformas (*cross-tools*) (YAGHMOUR, 2003).

Apesar de sua grande versatilidade, o *kernel* padrão do linux não pode ser classificado como um sistema determinístico, o que o torna inadequado para aplicações com restrições temporais. Algumas de suas limitações para esses tipos de aplicações são (BARBALACE A. LUCHETTA; TALIERCIO, 2008):

- *Dynamic priorities*: As prioridades dos processos variam ao longo do tempo, o que é ideal para prover acesso justo ao processador a todos os processos. No entanto, poder impedir que um processo urgente acesse o processador assim que necessitar;
- *Paging*: Mecanismos de paginação podem introduzir atrasos inexperados a menos que a página esteja carregada em memória;
- *MMU remaping*: Remapeamento das entradas da tabela de paginação pode impedir trocas de contexto rápidas;
- *Coarse-grained synchronization*: Devido ao *kernel* não-interrompível (*preemptivo*), o sistema pode demorar a responder aos eventos, devido a alguma operação que esteja sendo realizada pelo *kernel*.

No entanto, o sistema Linux vem sendo utilizado em várias aplicações cujas restrições temporais são críticas, devido principalmente às extensões adicionadas ao sistema *patch* ou alterações no próprio código do *kernel*. A partir do *kernel* 2.6 foi possível atribuir prioridades fixas a um determinado conjunto de tarefas e proteger segmentos não interrompíveis com *spin locks* no lugar de desabilitar as interrupções. Desse modo, o Linux é considerado atualmente um sistema operacional *soft real-time* com operações básicas para manuseio do tempo. Através de algumas extensões, adiciona-se ao linux a possibilidade de definir *tasks* para as quais se garante acesso à CPU e melhorias quanto ao tempo de resposta do sistema. Algumas dessas modificações são tratadas a seguir.

RT_Preempt O *Realtime Preemption patch* (Preempt_RT) desenvolvido por Ingo Molnar juntamente com a camada de gerenciamento de eventos de *clock* com alta resolução transformam o Linux em um *kernel* totalmente *preemptivo* (FU; SCHWEBEL, 2006) . As principais modificações realizadas são:

- As primitivas de *locking* em *kernel* (*spinlocks*) tornaram-se "preemptivas" com a utilização de *rtmutexes*;
- Secções críticas protegidas por *spinlock_t* e *rwlock_t* tornaram-se "preemptivas". Secções não "preemptivas" em *kernel* ainda são possíveis com *raw_spinlock_t*.
- Implementação da herança de prioridades para *spinlocks* e semáforos em *kernel*.
- Conversão dos tratadores de interrupção em *threads* "preemptivas" em espaço de *kernel* com uma estrutura semelhante aos processos em espaço de usuário.
- Conversão da API de tempo usual do Linux em uma infra-estrutura separada visando a temporizadores em *kernel* com alta resolução, possibilitando maior resolução também em espaço de usuário (POSIX).

ADEOS O *Adaptative Domain Environment for Operating System* (ADEOS) (ADEOS, 2004) é um *nanokernel Hardware Abstraction Layer* (HAL) que opera entre a camada de *hardware* do computador e o sistema operacional que roda sobre ele. O ADEOS fornece um ambiente adaptativo que pode ser utilizado para compartilhar os recursos de *hardware* entre múltiplos sistema operacionais ou entre múltiplas instâncias de um mesmo sistema operacional. Assim, para garantir acesso confiável e justo aos recursos de *hardware*, o ADEOS controla alguns comandos de *hardware* enviados pelos SO's. O ADEOS distingui-se dos outros *nanokernels*, pois não representa uma camada de baixo nível para um *kernel* externo.

Cada SO é alocado em um domínio, onde possui controle total e consiste de um espaço de endereçamento privado, memória virtual, sistema de arquivos, etc. A operação básica é a implementação de uma fila *queue* de sinais, onde a cada instante em que é enviado um sinal, os diferentes SO's, que estão rodando na máquina, são notificados seguindo uma ordem e devem decidir sobre a aceitação, descarte ou ainda a não propagação desse sinal. Caso o sinal não seja aceito e possa ser propagado, um outro SO será notificado. Essa cadeia é mostrada na Figura 2.8.

RTAI O Linux é um sistema operacional multitarefa *multitasking*, o qual fornece um escalonamento justo e não-preemptivo entre processos criados e destruídos dinamicamente. Em aplicações de tempo-real é fundamental que o tempo de resposta dos processos sejam garantidos ou que o comportamento do sistema seja previsível. O Linux, no entanto, não consegue atender a essas

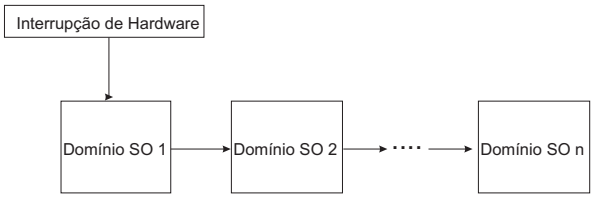


Figura 2.8: Cadeia de interrupção do ADEOS (ADEOS, 2004)

especificações. No entanto, recentemente, algumas abordagens, utilizadas para prover serviços de tempo real, têm ganhado destaque, entre elas o RTAI.

No intuito de utilizar o Linux em aplicações de tempo-real, pesquisadores do Departamento de Engenharia Aeroespacial da Escola Politécnica de Milão (DIAPM) projetaram uma camada de abstração de hardware real-time (RTHAL), onde estão montadas as API's de tempo real. Entretanto, pesquisas realizadas na época mostraram que o kernel do linux2.0.25 (1996) não estaria "maduro" suficiente para suportar tal implementação. Após estudos aprofundados sobre o *kernel* e o *hardware*, foram realizadas modificações necessárias para fornecer um escalonamento "preemptivo" e determinístico, sendo lançada a versão linux2.2.x (1998) que passasse a suportar tais conceitos (SAROLAHTI, 2001),(LINEO, 2000).

O RTAI (RTAI, 2006) provê garantias e escalonamento "*hard real-time*"mantendo todas as características e serviços do Linux padrão. Além disso, fornece suporte para UP e SMP - com possibilidade de atribuir *tasks* e IRQ's para CPU's específicas- ampla variedade de arquiteturas, escalonadores em *one-shot mode* e *periodic mode*, compartilhamento de memória, suporte a FPU , mecanismos de sincronização *inter-tasks*, semáforos, mutexes, filas, RPC's, mailboxes e serviços de tempo real a aplicações rodando em espaço de usuário. A arquitetura básica do RTAI é mostrado na Figura 2.9.

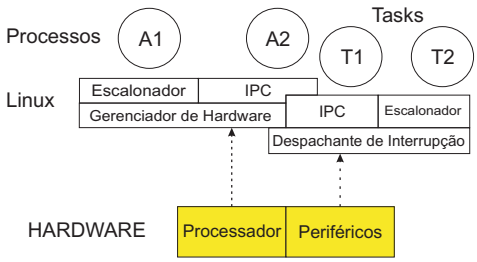


Figura 2.9: Arquitetura básica do RTAI (RTAI, 2006)

Como pode ser visto, existem interrupções originárias dos periféricos e do processador, sendo essas, como sinais de erro (*division error*), tratadas pelo kernel padrão do Linux, e aquelas

provenientes dos periféricos, tratadas pelo RTAI *Interrupt Dispatcher*. Assim, caberá ao *dispatcher* decidir sobre o tratamento ou não, num determinado instante, dessas interrupções (BRUYNINCKX; K.U.LEUVEN, 2000).

O RTAI, portanto, altera a estrutura do sistema padrão ao adicionar um *Hardware Abstraction Layer* (HAL) contendo uma estrutura de ponteiros para vetores de interrupção e funções de habilitação ou desabilitação de interrupções. Toda essa estrutura é implementada modificando-se apenas 20 linhas do código existente e adicionando 50 linhas ao novo código(SAROLAHTI, 2001).

Comedi O Comedi *Control and Measurement Device Interface* (SCHLEEF; HESS; BRUYNINCKX, 2007) é um projeto de *software* livre para desenvolvimento de *device drivers* além de bibliotecas para diversas formas de aquisição de dados: leitura e escrita de sinais analógicos, leitura e escrita de entradas/saídas digitais, geração de pulsos, determinação de freqüência, leitura encoders entre outros. O Comedi trabalha juntamente com o *kernel* padrão do Linux e também com suas extensões em tempo real, o RTAI e o RTLinux/GPL.

Xenomai O Xenomai *Real-Time Framework for Linux* (XENOMAI, 2007) é um *framewok* de tempo-real que trabalha em cooperação com o kernel do linux visando a um amplo suporte *hard real-time* a aplicações em espaço de usuário. Assim, o Xenomai adiciona uma camada de abstração ao sistema Linux, o qual adiciona comandos para lançamento e execução de tarefas em tempo real (*Xenomai* tasks), de modo a garantir que o linux não exerça nenhuma interferência nas tarefas de tempo real. Para tanto, todo o SO linux é lançado como uma tarefa de baixa prioridade por um micro-kernel de tempo real, o que garante que todas as tarefas de tempo real, lançadas pelo micro-kernel, nunca são interrompidas pelo Linux. De fato, o Linux só é executado quando nenhuma tarefa de tempo-real, com alta prioridade, está pronta para executar. O Xenomai depende do *nanokernel* de tempo real ADEOS para gerir interrupções em tempo real (*Ipipe: interrupt pipeline*). Ambos ADEOS e Xenomai funcionam como partes do *kernel* do Linux conforme Figura 3.8.

O Xenomai visa primeiramente auxiliar desenvolvedores, que trabalham em um ambiente de tempo-real tradicional, migrarem para um ambiente GNU/Linux da maneira menos impactante possível, sem a necessidade de reescrever o código inteiramente, ou seja, *real-time API* para qualquer linux (GERUM, 2004).

Ao contrário do seu antecessor RTAI, as tarefas de tempo real podem ser lançadas em espaço

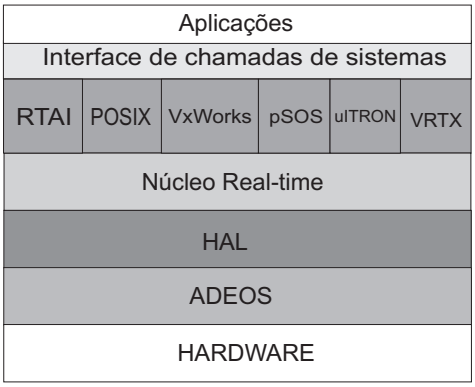


Figura 2.10: Visão em camadas da estrutura do Xenomai (XENOMAI, 2007)

de usuário, bem como a partir de um módulo do kernel. Normalmente, essas tarefas são executadas em modo primário, onde sua prioridade é maior do que as tarefas do linux. No entanto, caso seja necessário efetuar uma chamada à API do linux, isto é, uma chamada de sistema *system call* como, por exemplo, **write** ou **read**, a tarefa passa a ser executada em modo secundário, onde, nesse caso, outras tarefas em espaço ou domínio do linux podem tomar prioridade daquela.

RTDM O *Real-Time Driver Model* (RTDM) (KISZKA, 2007) é um projeto de unificação de interfaces de desenvolvimento de *device drivers* e aplicações associadas em tempo real para ambiente Linux. O RTDM age assim como mediador entre a aplicação que requer um serviço de certo dispositivo e o *driver* do dispositivo.

A API do RTDM está integrada ao Xenomai, permitindo que o atendimento de interrupções geradas pelos dispositivos de *hardware* cujos *drivers* foram escritos com essa API sejam atendidas seguindo as políticas do Xenomai.

Semelhanças e Diferenças entre o Xenomai e o RTAI O Xenomai e o RTAI apresentam muitos conceitos em comum, uma vez que são originários do mesmo projeto. Ambos procuram adicionar componentes que funcionam em conjunção com o Linux, de modo a permitir a criação de tarefas de tempo real para as quais é garantida a execução em um tempo determinístico assim que as mesmas estiverem prontas para rodar.

Conforme mencionado anteriormente, a camada de *hardware* é compartilhada entre o Linux e os novos componentes através do ADEOS, que propaga as notificações de interrupção aos componentes (Linux-RTAI ou Linux-Xenomai) que estão orginazados como um *pipeline*. Tanto o Xenomai como o RTAI estão na cabeça do *pipe* recebendo, portanto, as notificações de interrupção primeiramente.

No entanto, esses dois sistemas apresentam algumas diferenças quanto a sua organização, conforme mostrado na figura 3.9 (BARBALACE A. LUCHETTA; TALIERCIO,).

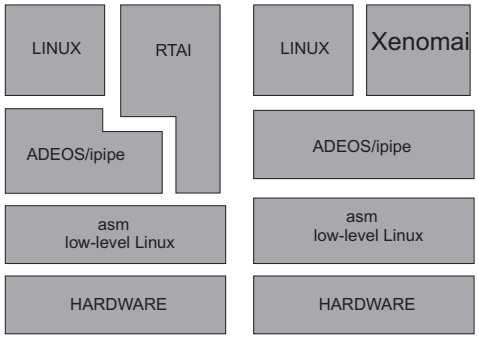


Figura 2.11: Estrutua em Camadas - RTAI e Xenomai (BARBALACE A. LUCHETTA; TALIERCIO,)

Diferentemente do Xenomai, o RTAI intercepta algumas interrupções, deixando aquelas que não interferem no escalonamento das tarefas de tempo-real para serem notificadas ao Linux pelo ADEOS. Essa estratégia procura melhorar o desempenho do sistema, evitando o *overhead* associado ao tratamento de interrupção realizado pelo ADEOS. O principal objetivo do RTAI portanto é reduzir a latência ao menor nível tecnicamente possível, enquanto o Xenomai procura favorecer aspectos como portabilidade e mantenabilidade. Ambos sistemas são portáveis a arquiteturas como PowerPC e ARM.

Quando comparado a outros sistemas baseados em Linux como o VxWorks, tanto o RTAI quanto o Xenomai apresentam melhores desempenhos, principalmente em redes de comunicação conforme análise realizada com protocolo de rede *hard real-time* RTNet comparado ao protocolo VxWorks IP (BARBALACE A. LUCHETTA; TALIERCIO, 2008).

2.5 Redes de Comunicação Usadas em Automóveis

2.5.1 CAN - Controller Area Network

O CAN (BOSCH, 1991) é de um protocolo de comunicação serial desenvolvido pela Bosch em meados de 1980 objetivando a transmissão de dados em automóveis. Sua especificação descreve uma pilha de rede objetivando uma específica aplicação: sistemas de controle tempo real via rede. Apesar de ter sido projetado para interligar os dispositivos eletrônicos responsáveis pelo controle e monitoramento das diversas partes que compõem um veículo, as redes CAN também vêm sendo utilizadas em sistemas de controle industrial, em automação predial e em outros sistemas

específicos como aparelhos médicos, simuladores de vôo e telescópios (PARET, 2007). Os sistemas de automotivos modernos são altamente distribuídos caracterizando-se por apresentar restrições temporais rígidas e por necessitar de garantias no que tange à confiabilidade e segurança do sistema.

A tecnologia CAN oferece elevada confiabilidade na transmissão de dados, devido aos avançados mecanismos de detecção de erros, incluindo mensagens de sinalização de erros e uma rápida recuperação após a ocorrência de distorções na comunicação. Além disso, as redes CAN possuem um acesso ao meio determinístico, o que permite estimar o limite máximo do tempo de resposta das mensagens e conseqüentemente verificar se o conjunto de mensagens da aplicação será escalonável, ou seja, se é transmitido sem a perda de *deadlines*. O protocolo é controlado por eventos *event triggered*, sendo que, para o envio de mensagens pela rede, é necessário o atendimento de algumas regras: o envio de mensagens só pode ser iniciado quando o meio físico está ocioso (*idle state*); na ocorrência de transmissões simultâneas, prevalecerá a mensagem de maior prioridade (menor *ID*). A vantagem de sistemas *event-triggered* é a habilidade de reagir rapidamente a eventos assíncronos, ou seja, que não são esperados. Além disso, os sistemas *event-triggered* possuem maior flexibilidade, pois permitem adaptações a uma eventual demanda da rede sem a necessidade de se redesenhar o sistema inteiro.

Dentre as características desse protocolo que favorecem sua utilização em aplicações de tempo real estão: elevadas taxas de transmissão se comparado a outros protocolos tradicionais (ex. RS232 e RS485), reduzido tamanho dos pacotes de dados, método de acesso ao meio governado pelo CSMA/AMP (*Carrier Sense Multiple Access / Arbitration by Message Priority*) com mecanismo de arbitragem no caso de transmissões simultâneas, técnicas modernas para detecção e correção de erros e confinamento de nodos defeituosos.

A ISO (*International Standards Organization*), através dos padrões 11519-2 e 11898, especifica as duas camadas de rede definidas pelo CAN: camada física e camada de enlace. A Figura 3.10 mostra a relação entre as camadas da especificação e o que realmente existe implementado.

Camada Física A camada física especifica o modo como o sinal é transmitido, ou seja, suas características elétricas, a codificação (Non-Return-to-Zero-NRZ), o sincronismo e os meios físicos suportados. O cabeamento normalmente utilizado para transmissão de dados é um par de fios de cobre ou uma fibra óptica, com comprimento variando de 0,2m a 10.000m e 120 Ohms em ambas as terminações. As taxas de transmissão variam de 100K até 1M bits-por-segundo (bps) dependendo do

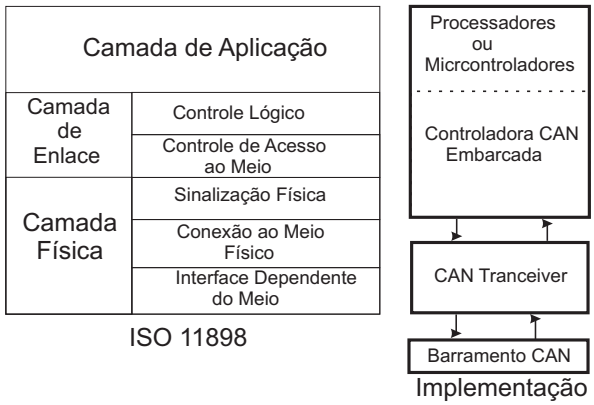


Figura 2.12: A especificação ISO 11898 para redes CAN (LAN, 2008)

comprimento do barramento. Para comprimentos de até 30m a taxa de transmissão é de 1Mbps, no entanto, para comprimentos próximos de 500m essa taxa cai para 125Kbps.

Os dados (bits) são enviados em dois estados: dominante (nível lógico 0) e recessivo (nível lógico 1). O mecanismo de arbitragem é quem lida com os problemas de conflito no barramento. Sempre que o barramento CAN está livre, qualquer unidade pode começar a transmitir uma mensagem. Possíveis conflitos, oriundos do fato de mais de uma unidade começar a transmitir simultaneamente, são resolvidos pela arbitragem de bits usando o identificador de cada unidade. Durante a fase de arbitragem, cada unidade em questão transmite seu identificador e o compara com o nível monitorado no barramento (*bus level*). Se estes níveis são iguais, a unidade continua a transmitir caso contrário, uma unidade ao detectar um nível dominante no barramento, enquanto ela tentava transmitir um bit recessivo, levá-la-á a aborta a transmissão e se torna uma receptora. A atuação desse mecanismo se estende a todas as unidades ligadas ao sistema, restando apenas uma unidade transmitindo no barramento.

Camada de Enlace Existem quatro tipos de mensagens definidas no protocolo: pacote de dados, pacote de requisição, pacote de erro e pacote de sobrecarga do nodos. O formato do pacote de dados CAN é mostrado na Figura 2.13 em seus dois padrões, normal (2.0A) e estendido (2.0B).

2.5.2 TTCAN - *Time-Triggered* CAN

As transmissões de pacotes numa rede CAN são decorrência de uma ação num determinado nodos ou de uma requisição de informação de um nodos para outro. Assim, como visto, a rede CAN é disparada por eventos (*event-triggered*).

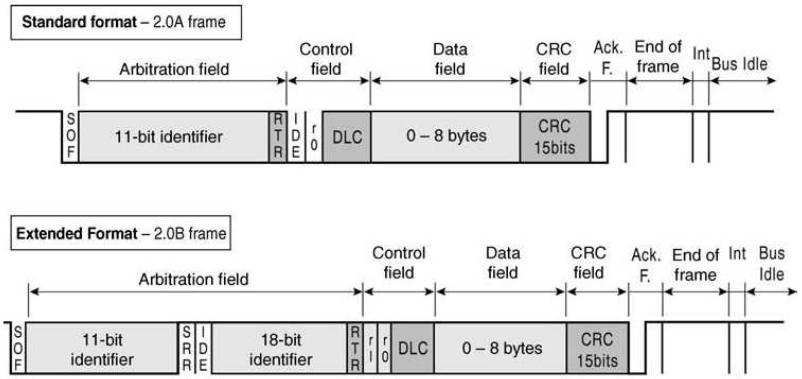


Figura 2.13: Frame de dados CAN em seus dois padrões (PARET, 2007)

Conforme mencionado na secção 2.5.1, o acesso ao meio de uma mensagem na rede CAN é realizado através de um mecanismo de arbitragem, que depende exclusivamente do campo de identificação de cada pacote. Desse modo, a transferência das mensagens é probabilística, uma vez que a latência na transmissão de determinada mensagem depende do surgimento de outras mensagens e seus identificadores em outros nodos da rede.

No entanto, algumas aplicações necessitam que o envio de mensagens seja disparado em instantes precisos, de modo que tenha-se certeza da transmissão e recepção dessas mensagens. Nesse caso, as mensagens são em enviadas em intervalos de tempo (*time slot*) caracterizando a rede como disparada por tempo ou *time-triggered* (PARET, 2007).

Devido a necessidade de um comunicação determinística, ou seja, que haja garantias de que determinada mensagem estará presente na rede com uma freqüência conhecida, desenvolveu-se uma extensão do protocolo CAN conhecida como TTCAN (FüHRER et al., 2000).

Apesar de não haver nenhuma alteração na camadas física e enlace da rede CAN originária, o TTCAN requer a implementação em *hardware* de um relógio global utilizado para sincronizar o envio de mensagens. A sincronização portanto ocorre através do envio de uma mensagem periódica de referência que todos os nodos TTCAN reconhecem e utilizam para sincronizar seus relógios. Os nodos são configurados para saber quando mandar suas mensagens depois de terem recebido a mensagem de referência e o intervalo de transmissão deve ser um múltiplo desse intervalo de referência. O nodos que utilizam o protocolo CAN tradicional (ou nodos TTCAN baseados em

eventos) podem estar presentes numa rede TTCAN onde, nesse caso, competem pelo acesso às "janelas" de tempo entre as mensagens de referência.

A desvantagem das configurações *time-triggered* é a falta de flexibilidade e a necessidade de um processo de construção mais sofisticado onde todos os processos e as especificações temporais devem conhecidas *a priori* ou não será possível uma implementação eficiente. Além disso, a comunicação e a atuação das unidades de controle devem ser sincronizadas durante a operação de modo a garantir que as especificações temporais sejam respeitadas.

2.5.3 LIN

O protocolo LIN (*Local Interconnet Network*) (LIN, 1999) é essencialmente destinado a apoiar o controle de elementos mecatrônicos encontrados em sistemas distribuídos para veículos, mas pode ser aplicado em muitos outros campos. Essa rede foi desenvolvida para ser utilizada em aplicações onde o custo é crítico e as taxas de transmissão de dados são baixas, sendo muitas vezes utilizada como um complemento à rede CAN nas aplicações automotivas. Nesse caso a rede LIN pode ser aplicada ao acionamento de equipamentos como: vidro elétrico, trava elétrica, iluminação, controle de clima, sensor de chuva, regulagem de assentos, entre outras.

Essa tecnologia de rede é baseada no conceito de subredes contendo somente um "mestre" e um número finito de "escravos". Desse modo, o tráfego de dados no barramento é inicializado e controlado exclusivamente pela tarefa(*task*) mestre da rede, que é a única entidade que decide quais são quadros (*frames*) a serem transferidos pelo barramento, e quando serão transmitidos. O sistema de comunicação é mostrado Figura 3.6 (PARET, 2007).

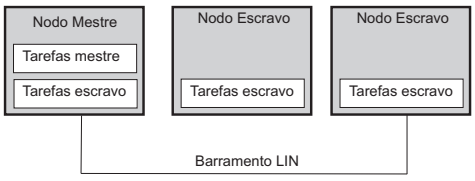


Figura 2.14: Rede de comunicação LIN (PARET, 2007)

Essa regra de funcionamento garante a previsibilidade do sistema de comunicação nas redes LIN, uma vez que a latência de pior caso é conhecida, o que torna seu uso muito vantajoso em aplicações de tempo real.

2.5.4 FlexRay

O protocolo FlexRay (FLEXRAY, 2008) consiste de um infraestrutura para aplicações de controle em veículos que exigem elevadas taxas de transmissão. Essa tecnologia procura atender necessidades não atendidas em outros casos como:

- **CAN:** velocidade inferior às exigidas por um grande número de aplicações e a falta de redundância e um completo determinismo na comunicação.
- **TTCAN:** além das baixas taxas de transmissão, não possuem canais de comunicação redundantes e tolerância a falhas nos relógio globais.

O FlexRay, no entanto, não substitui o CAN mas opera como um adicional ao mesmo. O objetivo é prover comunicação para sistemas de controle e monitoramento oferecendo:

- Elevadas taxas de transmissão (10 $Mbit/s$) com a largura de banda 20 vezes maior do que o barramento CAN;
- Comportamento determinístico com redundância (dois canais de comunicação enviando o mesmo dado em paralelo).
- Capacidade para ser utilizado em soluções *X-by-Wire*, sendo mais adaptado às futuras aplicações da indústria automotiva.

O protocolo de comunicação FlexRay inclui especificações para as transmissões de alta velocidade e definições de interfaces de *hardware* e *software* entre os componentes de um nodos FlexRay. Esses nodos podem estar ligado a um ou a dois barramentos, onde, nesse último caso, não é possível utilizar o mesmo controlador para acessar os dois meios de comunicação.

2.6 Considerações Finais

O desenvolvimento dos chamados SE's exige o conhecimento de diversas áreas de conhecimento e são diretamente influenciados por questões econômicas. Nesse capítulo apresentou-se alguns ambientes que procuram facilitar as etapas de desenvolvimento de modo rápido e confiável.

Além das soluções integradas, opções de projeto personalizado também apresentam vantagens principalmente quanto a flexibilidade. Em todos os casos, muitas vezes opta-se pela utilização de um

sistema operacional e essa escolha mostra-se muitas vezes difícil. Além das diversas opções existentes comerciais ou não, requer-se um estudo sobre o impacto do uso do sistema sobre o desempenho da aplicação.

Os sistemas operacionais comerciais desenvolvem-se rapidamente, acompanhando a evolução de processadores e de *chips* em geral. Além disso, suporte e documentação são outros pontos fortes que pesam a favor desse tipo de sistema. Entretanto, o Linux vem ganhando espaço devido principalmente a sua modularidade, eficiência e ampla utilização no meio acadêmico e entre usuários em geral. Através das extensões discutidas nesse capítulo, o *kernel* do Linux tornou-se opção para em sistemas *hard real-time*, mesmo naqueles que exigem um suporte de rede de tempo-real.

Protocolos de rede como RTNet ou soluções em rede CAN como o RTCAN, fazem do Linux uma um excelente opção quanto ao custo/benefício para desenvolvimento de sistemas embarcados, em especial os sistemas de controle distribuído.

Capítulo 3

Projeto do Sistema de Controle do Veículo

3.1 Introdução

Este capitulo apresenta a aplicação de um sistema de controle multinível baseado em PID-MPC para as manobras de estacionamento de um veiculo, onde também é avaliado a aplicabilidade de tais técnicas ao controle de determinados processos em sistemas digitais. Primeiramente revisam-se os conceitos e características de alguns controladores como PID e MPC, assim como a metodologia de projeto do sistema de controle. Por fim, apresenta-se a aplicação de tais técnicas ao sistema de manobra de estacionamento, em que são detalhadas as fases de projeto realizadas e os resultados obtidos.

3.2 Controle de Processos

O controle automático de processos tem desempenhado um papel fundamental no avanço da engenharia e da ciência. Além da extrema importância em sistemas de automotivos e robóticos, o controle automático tem se tornado fundamental e parte integrante dos modernos processos industriais e de produção. Com os avanços no controle automático, na teoria e na prática, vêm-se produzindo meios para otimizar o desempenho de sistemas dinâmicos.

Os sistemas de controle tratam basicamente da manutenção de grandezas como temperatura e pressão em valores operacionais desejados ou da condução de uma determinada variável a

determinados valores. Uma forma de atingir esse objetivo é medir a saída do sistema, compará-la com um valor pré-definido e, então, decidir o que fazer para diminuir qualquer desvio. A Figura 3.1 mostra etapas intermediárias entre a medição do sinal de saída do processo e a atuação no mesmo.

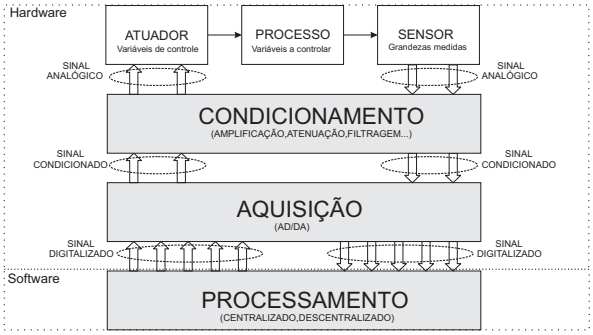


Figura 3.1: Camadas de um sistema de controle clássico

Essas grandezas são denominadas variáveis controladas, ou seja, variáveis que se deseja manter num determinado valor. Para que esse valor seja alcançado, a manipulação de uma outra variável, denominada variável manipulada, é realizada, ou seja, a atuação nessas variáveis permite que a variável controlada seja mantida próxima ou igual a um ponto de operação *setpoint*. Qualquer evento que atue no sistema de modo a retirá-lo do seu ponto de operação desejado é denominado distúrbio.

No topo da Figura 3.1 destaca-se o modelo da planta ou processo a ser controlado. A saída ou estado desse processo é alterado de acordo com o sinal enviado pelo atuador e aquisitado pelos sensores acoplados. Dentre os principais desafios encontrados nessa camada estão os erros de modelagem, saturação e sensibilidade dos dispositivos.

O valor ou a magnitude dessas grandezas mencionadas são coletadas por sensores como ultra som, termopar, infravermelho(IR), *encoders* (diferencial ou absoluto), *strain gate*, resistores *shunt* etc. Os sinais provenientes desses dispositivos são traduzidos para valores de corrente ou tensão (contínuo, alternado ou pulsado) para então serem aquisitados pelas unidades de processamento.

Entretanto, em muitos casos, é necessário um tratamento ou condicionamento desses sinais, de modo a garantir que seus valores possam ser representados corretamente de maneira digital. Esse condicionamento é realizado geralmente com a utilização amplificadores, atenuadores ou filtros *anti-aliasing*. Assim, dependendo do comportamento dinâmico do processo a ser controlado, determina-se uma taxa de amostragem adequada a ser utilizada. Essa é a variável mais importante a ser considerada na implementação do controlador, pois o intervalo de tempo entre a aquisição, o processamento e a atuação não deverá superar o período ou intervalo de tempo até uma nova aquisição.

Objetivo O objetivo do controle automático é a manutenção ou condução de uma variável controlada a partir de uma variável manipulada até um ponto de operação desejado, independente de perturbações.

A partir da amostragem do sinal de saída do processo, é possível identificar o estado em que o mesmo se encontra e assim calcular qual ação deve ser realizada sobre ele. Portanto, uma vez definida a taxa de amostragem a ser empregada, pode-se utilizar o intervalo de tempo entre as amostras como o *deadline* da tarefa de controle responsável por atuar num determinado processo. Em sistemas de controle descentralizados, o problema é agravado, pois adiciona-se ao tempo de computação dessa ação de controle o intervalo de tempo para transmissão de mensagens entre as unidades de sensoriamento, processamento e atuação. Devido a essas características temporais, o sistema de controle automático distribuído deve ser realizado por tarefas de tempo real e escalonadas adequadamente em um sistema determinístico.

3.2.1 Estratégias de Controle

Controle com Realimentação

O controle de realimentação ou malha de controle realimentada responde por praticamente 80% de todas as estratégias de controle utilizadas atualmente. Essa estratégia objetiva manter a variável controlada num ponto de operação ao medir o sinal de saída e compará-lo com uma referência. Desse modo, não há preocupação com qual distúrbio entra no processo, mas em compensá-lo a fim de diminuir seus efeitos. A desvantagem desse tipo de controle reside no fato de somente ser possível compensar um distúrbio no sistema quando a variável controlada desviou de seu ponto de operação, ou seja, o distúrbio precisa propagar-se por todo o processo antes que a ação de controle possa ser iniciada (SMITH; CORRIPIO, 2006).

Controle com Pré-Alimentação

O controle com pré-alimentação *feedfoward control* é uma estratégia de controle bastante comum na indústria de processos, devido principalmente a sua simplicidade. Em situações onde o controle com realimentação não fornece um desempenho satisfatório, técnicas avançadas de controle com pré-alimentação podem ser então utilizadas em conjunto. O objetivo dessa estratégia é medir as perturbações e compensá-las, antes que a variável controlada seja desviada do ponto de operação

(MELLICHAMP, 2004).

A Figura 3.2 mostra a estrutura do controle com pré-alimentação, onde o efeito do sinal de perturbação é medido e o controlador (FFC), com essa informação, deve gerar uma saída que resulte em um efeito igual e contrário ao dessa perturbação.

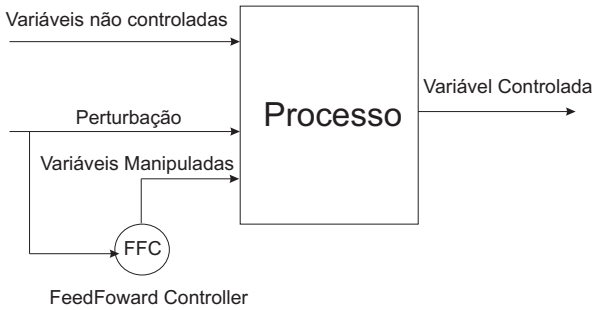


Figura 3.2: Estrutura do controle com pré-alimentação (MELLICHAMP, 2004)

3.2.2 Etapas de Desenvolvimento

Um procedimento sistemático no desenvolvimento dos sistemas de controle evita que surpresas inesperadas venham a ocorrer em etapas de finais de implementação. Algumas etapas a serem seguidas são detalhadas a seguir (NORMEY-RICO, 2005):

1. Definição do Problema

   (a) Levantamentos das Especificações: etapa onde estabelece-se as necessidades e o desempenho desejado para o sistema em malha fechada. As especificações devem apresentar certo grau de flexibilidade de modo a permitir existência de uma solução.

   (b) Modelagem do Sistema: a modelagem consiste de uma representação matemática através de equações diferenciais, equações a diferenças ou relações entrada/saída do sistema a ser controlado. Métodos analíticos ou técnicas de identificação de sistemas podem ser utilizados nessa etapa.

   (c) Obtenção de uma representação matemática: a partir da modelagem, realiza-se a representação do sistema através de funções de transferência ou em espaço de estados. Essas representações permitem uma análise compacta de sistema com múltiplas entradas e saídas.

2. Análise da Solução

(a) Adoção de uma estrutura de controle: nesse ponto, adota-se uma estratégia de controle, visando ao atendimento das especificações.

(b) Análise de existência de solução: nessa etapa, avalia-se o conjunto de especificações e a estrutura de controle proposta quanto à solução do problema.

(c) Se necessário, reformulação do problema: em situações onde a estrutura de controle proposta não atende satisfatoriamente, faz-se necessário uma reestruturação do controlador ou, em último caso, o relaxamento de algumas restrições de operação do sistema.

3. Síntese da Solução

(a) Síntese do compensador: obtenção da representação com valores numéricos do compensadores e controladores projetados, baseando-se no comportamento dinâmico do processo e nas especificações acertadas.

(b) Simulação e Validação: etapa fundamental, onde os valores obtidos anteriormente são validados ou refutados com auxílio de ferramenta computacional como Matlab/Simulink[1] ou Scilab/Scicos.

(c) Ajustes empíricos: corresponde ao ajuste fino da solução proposta para adequar-se perfeitamente ao sistema real.

3.2.3 Controle Multinível

Em plantas industriais ou mesmo em sistemas automotivos e aviônicos é extremamente vantajoso que parte do processamento da estratégia de controle seja realizada localmente, ou seja, próximo dos sensores e atuadores do sistema a ser controlado. Em uma estrutura de controle multinível, várias unidades menores de processamento aquisitam sinais, processam informações e atuam nos processos, sob supervisão ou seguindo uma estratégia definida em unidades maiores e centralizadas. Dentre as vantagens dessa estrutura destacam-se:

- Diminuição de interferências: com parte do processamento global centralizado, reduz-se a probabilidade de interferência em sinais provenientes de sensores e sinais enviados aos atuadores, o que eleva a confiabilidade do sistema.

[1] Matlab/Simulink são marcas registradas da MathWorks

- Adaptabilidade: alterações nos subsistemas que exijam mudanças na implementação de controladores ou alterações em *hardware* podem ser realizadas sem impacto em toda a estrutura de controle ou necessidade de reprojeto.

- Manutenção: a manutenção do sistema fica extremamente facilitada, uma vez que a localização e o confinamento de erros é mais fácil em uma estrutura multinível.

- Custo: além de facilitar a manutenção, uma estrutura de controle em diferentes níveis reduz o custo com a cablagem ou eventuais trocas de componentes de *hardware* nas unidades de controle locais pois, em geral, são mais baratos.

Duas técnicas de controle que podem ser utilizadas nesses diferentes níveis de controle, que apresentam diferentes propósitos, são o Controle Preditivo para um nível mais alto, onde é realizado o planejamento e supervisão de todo o funcionamento do sistema e o controle PID para o controle localizado, direcionado a cada subsistema envolvido. As seções seguintes detalham essas duas abordagens que serão usadas neste trabalho para o controle do carro durante as manobras de estacionamento.

3.3 Controladores PID

Os controladores PID são os mais empregados na indústria atualmente (cerca de 50% dos controladores industriais utilizam esquema PID ou PID modificado). A utilidade dos controles PID está na sua aplicabilidade geral à maioria dos sistemas de controle. Quando o modelo matemático da planta não é conhecido e, portanto, métodos de projeto analítico não podem ser utilizados, controles PID se mostram os mais úteis, apresentando resultados satisfatórios, embora em algumas situações eles podem não proporcionar um controle ótimo (BAZANELLA; J. M. GOMES da SILVA Jr., 2003).

Esses controladores combinam as ações proporcional, integral e derivativa para gerar um sinal de controle. O objetivo é aproveitar as características particulares de cada uma destas ações a fim de se obter uma melhora significativa do comportamento transitório e em regime permanente da planta ou sistema a ser controlado (ÅSTRöM; HäNGGLUND, 1995). O sinal de controle gerado pelo controlador PID ideal é dado por:

$$u(t) = K_c[e(t) + \frac{1}{T_i}\int_0^t e(t)dt + T_d\frac{d\,e(t)}{dt}] \tag{3.1}$$

onde $e(t)$ é o erro entre a referência (*setpoint*) e o sinal de saída a ser controlado $y(t)$:

$$e(t) = r(t) - y(t) \tag{3.2}$$

Os parâmetros K_c, T_i e T_d são denominados ganho proporcional, tempo integral e tempo derivativo. Essa configuração é conhecida como acadêmica ou não interativa, uma vez que o ajuste de uma das ações não afeta as outras. Na prática, pelas restrições de ordem física dos atuadores ou por questões de segurança, não é possível a aplicação de sinais de controle com amplitudes ilimitadas. Estabelece-se assim um limite máximo u_{max} e um limite mínimo u_{min} para a variável de controle. O sinal de controle pode ser visto como uma função do erro como visto na equação 3.3:

$$u(t) = Kf(e(t)) = Kw(t) \tag{3.3}$$

Para que o comportamento do controlador PID seja dado exatamente por 3.1, ou seja, linear, o sinal $w(t)$ deve pertencer ao intervalo $[w_1, w_2]$ onde $w_1 = u_{min}/K$ e $w_2 = u_{max}/K$. A largura do intervalo $[w_1, w_2]$ é definido como banda proporcional, P_b (BAZANELLA; J. M. GOMES da SILVA Jr., 2003):

$$P_b = w_2 - w_1$$

Considerando os limites do sinal de controle $u(t)$, tem-se:

$$u(t) = u_{max} \quad se \quad w_2(t) > w_{2max} \tag{3.4}$$

$$u(t) = Kw(t) \quad se \quad w_{min} \geq w(t) \leq w_{max} \tag{3.5}$$

$$u(t) = u_{min} \quad se \quad w_1(t) < w_{1min} \tag{3.6}$$

Assim, para valores de $Kw_2(t) > u_{max}$ ou $Kw_2(t) < u_{min}$ ocorre uma saturação de controle e, neste caso, o comportamento do controlador torna-se não linear.

3.3.1 Ação Proporcional

O controle proporcional $u(t) = Ke(t)$ é uma das formas mais simples de controle. Neste tipo de ação, o sinal de controle aplicado a cada instante à planta é proporcional à amplitude do valor do sinal de erro. Assim se, em um dado instante, o valor da saída do processo é menor (maior) que o valor da referência, i.e. $e(t) > 0$ ($e(t) < 0$), o controle a ser aplicado será positivo (negativo) e proporcional ao módulo de $e(t)$. Dentre as características básicas desse tipo de controle, destacam-se:

- Maiores valores do ganho K_c permite obter menores valores do erro em regime permanente.
- Devido ao limites mostrados em 3.4, 3.5 e 3.6, a ação de controle limita-se a faixa da banda proporcional P_b.
- Trata-se de um controlador instantâneo e sem memória, dependendo exclusivamente do valor do erro $e(t)$.
- Aplicável com resultados satisfatórios em sistemas com respostas bem amortecidas em malha aberta.

3.3.2 Ação Integral

O controle integral consiste em aplicar um sinal de controle $u(t)$ proportional à integral do sinal $e(t)$:

$$u(t) = \frac{1}{T_i} \int e(t)dt$$

onde T_i também é conhecido como reset-time.

A introdução dessa ação de controle justifica-se quando as características estáticas do sistema em malha fechada não atendem as especificações, definidas geralmente por erros nulos em regime permanente para entradas ou perturbações do tipo degrau. O controle integral permitirá, no sistema em malha fechada, obter-se o seguimento de uma referência com erro nulo em regime permanente, pois garante a aplicação ao processo de um sinal de constante de forma a ter-se $r(t) = y(t)$, i.e. $e(t) = 0$ (ÅSTRöM; HäNGGLUND, 1995).

A função de transferência da ação integral é dada por:

$$G_c(s) = \frac{u(s)}{e(s)} = \frac{1}{sT_i}$$

Assim sendo, sob o ponto de vista matemático, a ação integral introduz um pólo na origem na nova função de transferência em malha aberta $G_c(s)G(s)$. Devido a isso, quando utilizado isoladamente, a ação integradora piora as características dinâmicas do sistema. No entanto, sua utilização juntamente com o controle proporcional introduz um zero no sistema em malha fechada que melhora o comportamento transitório. A utilização de uma ação integral possibilitará a rejeição assintótica de perturbações de carga do tipo $\frac{1}{s}$.

3.3.3 Ação Derivativa

A ação derivativa é principalmente aplicada para corrigir a resposta transitória do sistema. Esta ação corresponde a aplicação de um sinal de controle proporcional a derivada do sinal de erro:

$$u(t) = T_d \frac{d\, e(t)}{dt}$$

Um vez que a derivada reflete a inclinação da curva de erro, a ação derivativa pode ser vista como um predição linear do mesmo, sendo predominante nos instantes onde ocorrem variações rápidas, ou seja, o controle procura prever o comportamento do sistema e tenta minimizá-lo (ÅSTRöM; HäNGGLUND, 1995). A função de transferência desta ação é dada por:

$$G_c(s) = \frac{u(s)}{e(s)} = T_d s$$

Observa-se que a função de transferência é uma função não própria, o que resulta num sistema sensível aos ruídos de alta freqüência. Além disso, a implementação analógica de um derivador puro é fisicamente impossível. Assim, na prática, ela é implementada com um filtro passa baixas.

A função de transferência torna-se então:

$$G_c(s) = \frac{u(t)}{e(t)} = T_d \frac{sp}{s+p}$$

3.3.4 Vantagens do PID

Os controladores PID apresentam grandes vantagens, que justificam seu vasto uso na indústria principalmente devido aos seus métodos de análise de estabilidade e robustez serem bastante consolidados e sua independência quanto ao uso preciso do modelo da planta. Além disso, questões como simplicidade, confiabilidade e rapidez no processamento adequam o uso dessa técnica para o controle localizado, direcionado a cada subsistema.

Para um desempenho satisfatório, é extremamente importante o ajuste (*tunning*) correto dos parâmetros do controlador. Existem vários métodos na literatura para ajuste de PID bastante utilizados na indústria. Dentre os mais conhecidos estão os *Ziegler-Nichols* (1942) e o de *Cohen e Coon* (1952), que estimam valores de partida para os 3 parâmetros do controlador, além dos métodos analíticos, como o Lugar das Raízes, para determinação da localização de todos os pólos da malha realimentada a partir do conhecimento da localização de todos os pólos e zeros da malha aberta (HOUPIS, 1981). Outra vantagem é o fato de serem flexíveis, ou seja, outros esquemas, diferentes do apresentado na equação3.1, podem ser utilizados como: PI+D, PID com dois graus de liberdade, entre outros.

Em muitas aplicações, a ação derivativa não é utilizada, sendo as ações proporcional e integral as mais adequadas em processos com dinâmicas de primeira ordem (ÅSTRöM; HäNGGLUND, 1995). Em processos com dinâmicas de segunda ordem, onde a ação proporcional sofre limitação quanto sua ação, justifica-se o uso da ação derivativa para melhorar a resposta transitória do sistema.

3.3.5 Limitações do PID

Em processos industriais, é comum a presença de atrasos consideráveis. O desempenho dos controladores PID são severamente degradados caso o processo tenha um atraso grande em relação a sua constante de tempo ou em casos onde a resposta do sistema seja muito rápida. Além disso, o PID mostra-se ineficiente quando atua em t predizendo um erro em $t + T_d$ que é muito diferente do $e(t + T_d)$ real (HOUPIS, 1981).

Em sistemas de ordem maior do que 2, o desempenho do controle pode ser melhorado com o uso de técnicas mais sofisticadas. Além disso, em processos com presença de pólos oscilatórios não amortecidos, o PID não apresenta um comportamento satisfatório.

3.4 Controle Preditivo

Originário de fins da década de 70 e primeiramente implementado em ambiente industrial, o Controle Preditivo Baseado em Modelo *Model Predictive Control*(MPC) é considerado atualmente uma das tecnologias de controle avançado mais utilizadas em processos industriais(QIN; BADGWELL, 1997). O interesse acadêmico, no entanto, cresceu apenas no final da década de 80 impulsionado principalmente por dois *workshops* organizados pela Shell (PARET; RAMARKER, 1990).

O MPC descreve uma série de técnicas de controle que se utilizam de um modelo do processo para estimar seu comportamento futuro e, assim, obter a lei de controle. Os valores de entrada do processo são obtidos através da minimização de uma função objetivo (custo), ou seja, da solução *on-line* de um problema de otimização construído com base no modelo e nas medições do processo. A Figura 3.7 mostra a estrutura típica do MPC:

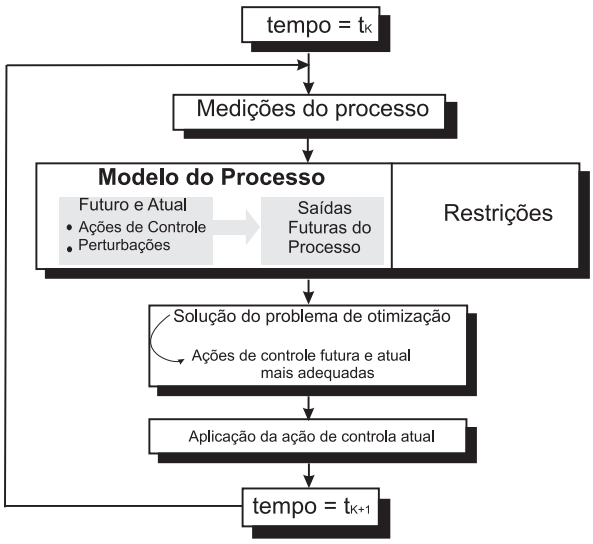


Figura 3.3: Estrutura Básica do MPC (NIKOLAOU, 2001)

Observa-se que o projeto do MPC pode apresentar-se bastante complexo devido a ação de controle ser determinada pelo resultado da solução de um problema de otimização *on-line*. No entanto, essa técnica de controle avançada tem mostrado melhorias quanto ao desempenho, flexibilidade, agilidade e manutenção (NIKOLAOU, 2001), além de ser eficiente em problemas de controle multivariável com restrições (MORARI; LEE, 1999), introduzir o controle *feed forward* para predição de perturbações e pode ser aplicado a sistemas com atrasos de transporte.

De acordo com (CAMACHO; BORDONS, 2007), (ABU-AYYAD; DUBAY, 2007), as idéias básicas comuns a todos os tipos de controle preditivo são:

- Uso explícito de um modelo para predição de saídas do processo em instantes de tempo futuros;
- Cálculo de uma seqüencia de controle que minimize uma função objetivo;
- Estratégia deslizante de modo que a cada instante o horizonte é movido em direção ao futuro, de forma que o primeiro sinal de controle da seqüência é aplicado a cada passo;

Para processos sem restrições, o método apresenta uma solução analítica para o problema de otimização, onde o número de variáveis independentes para o cálculo de otimização é reduzido, assumindo $u(t+j|t) = u(t+N_u-1|t), j = m....,p-1$ ou $\delta u(t+j-1) = 0, j > m$ (CAMACHO; BORDONS, 2007).

Nos casos que apresentam algum grau de restrição como, por exemplo, nos limites do sinal de controle ou de saída do processo, conforme equações 3.7, 3.8 e 3.9, a minimização da torna-se muito mais complexa necessitando de métodos iterativos de otimização.

$$u_{min} \leq u(t) \leq u_{max} \quad \forall t \tag{3.7}$$

$$du_{min} \leq u(t) - u(t-1) \leq du_{max} \quad \forall t \tag{3.8}$$

$$y_{min} \leq y(t) \leq y_{max} \quad \forall t \tag{3.9}$$

Existem diferentes métodos que utilizam a estratégia do MPC, diferenciando-se basicamente no modelo utilizado para descrever a planta em questão. Em geral, a maioria dos processos estáveis e instáveis podem ser expressos utilizando modelos autoregressivos como ARX, ARIMAX ou CARIMA. Em particular, o método *Generalized Predictive Control* (GPC) foi utilizado no contexto deste trabalho.

3.4.1 Generalized Predictive Control (GPC)

Considerado um dos mais populares métodos de controle preditivo na indústria e no meio acadêmico, o GPC apresenta um bom desempenho e um certo grau de robustez. Esse método vem sendo aplicado com sucesso em muitos processos industriais(CLARKE, 1988). A idéia básica é o cálculo de uma seqüencia de sinais de controle de modo a minimizar uma função custo multiestágio definida em um horizonte de predição. Na ausência de restrições, o método apresenta uma solução analítica para o problema de controle, além de de ser aplicável a planta instáveis e de

fase não-mínima(CAMACHO; BORDONS, 2007). Para a formalização do GPC, considere uma planta dada pelo seguinte modelo discreto, linearizado em um ponto de operação particular:

$$A(z^{-1})y(t) = z^d B(z^{-1})u(t-1) + C(z^{-1})\frac{e(t)}{\Delta} \qquad \Delta = 1 - z^{-1} \tag{3.10}$$

onde A, B, C são polinômios, $u(t)$ e $y(t)$ são, respectivamente, a entrada e a saída do sistema, d o atraso(*dead time*), e $e(t)$ é um ruído branco. Esse modelo é conhecido como *Controller Auto-Regressive Moving-Average* (CARIMA).

O objetivo do GPC portanto é aplicar uma seqüência controle $(u(t), u(t+1), u(t+2)...)$ que minimize um função custo $(J(N_1, N_2, N_u))$ da forma descrita em 3.11, levando a saída do sistema o mais próximo possível da referência futura.

$$J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} \delta(j)[\hat{y}(t+j|t) - w(t+j)]^2 + \sum_{j=1}^{N_u} \lambda(j)[\Delta u(t+j-1)]^2 \tag{3.11}$$

onde $\hat{y}(t+j|t)$ é a j-ésima saída futura ótima a partir do instante t, $N_1 N_2$ são os horizontes mínimo e máximo de predição, N_u é o horizonte do sinal de controle, δ_j e λ_j são as seqüências de ponderação, usualmente escolhidas constantes ou exponenciais crescentes, e $w(t+j)$ é a trajetória de referência futura.

Considerando um sistema com *dead time* de d períodos de amostragem, a saída do sistema será influenciada pelo sinal $u(t)$ depois de um período $d+1$.Usando este modelo é possível calcular as saídas futuras da planta ou predições num horizonte de tempo N. O vetor de predições $(\hat{y}(t+d+1|t), \hat{y}(t+d+2|t)..., \hat{y}(t+d+N|t))$ pode ser reescrito como:

$$\mathbf{y} = \mathbf{Gu} + \mathbf{f} \tag{3.12}$$

onde y e u são matrizes de ordem Nx1 e representam, respectivamente, o vetor de predições e o de controle futuros, **G** é a matriz dinâmica do sistema, **Gu** representa a resposta forçada e **f** a resposta livre do sistema, que depende apenas de valores passados:

$$y = \begin{pmatrix} \hat{y}(t+d+1|t) \\ \hat{y}(t+d+2|t) \\ . \\ . \\ . \\ \hat{y}(t+d+N|t) \end{pmatrix} \quad u = \begin{pmatrix} \Delta u(t) \\ \Delta u(t+1) \\ . \\ . \\ . \\ \Delta u(t+N-1) \end{pmatrix} \quad \mathbf{G} = \begin{pmatrix} g_0 & 0 & ... & 0 \\ g_0 & g_1 & ... & 0 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ g_{N-1} & g_{N-2} & ... & g_0 \end{pmatrix} \tag{3.13}$$

Pode-se mostrar que a seqüência de sinais de controle que minimizam a função custo

$$J(N_1, N_2, N_u),$$

desde que não haja restrições no controle, é dada por:

$$u = K(w - \mathbf{f}) \qquad K = (\mathbf{G}^T\mathbf{G} + \lambda I)^{-1}\mathbf{G}^T \tag{3.14}$$

O método consiste em recalcular a seqüência de sinais de controle a cada iteração e aplicar no sistema apenas o primeiro elemento de **u**. A solução do problema envolve inversão de matrizes NxN, o que demanda muito esforço computacional. No entanto, assumindo que as ações de controle serão constantes após $N_u(\Delta u(t+j-1) = 0)$ para processos sem restrições, esse esforço é substancialmente reduzido. A Figura 3.4 mostra a estratégia do GPC:

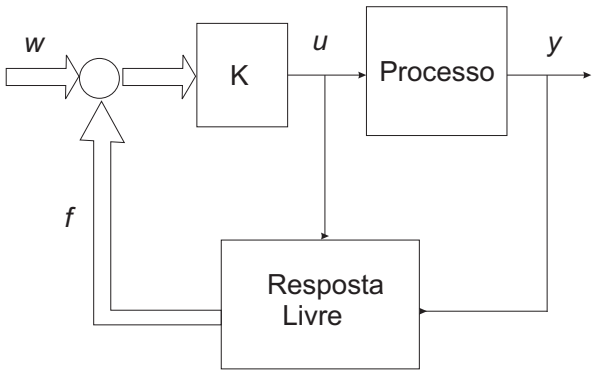


Figura 3.4: Lei de Controle do GPC (CAMACHO; BORDONS, 2007)

O controle preditivo, portanto, volta-se principalmente para o comportamento futuro, através do modelo interno embutido, realizando um constante replanejamento do controle necessário conforme

as perturbações na saída desejada.

Analisando a estrutura de controle multinível, a técnica de controle preditivo desempenharia um papel de supervisão, sendo executada em uma unidade central de processamento. No caso de controle de trajetória, o GPC mostra-se adequado pois considera o comportamento futuro do sistema, antecipando a ação corretiva.

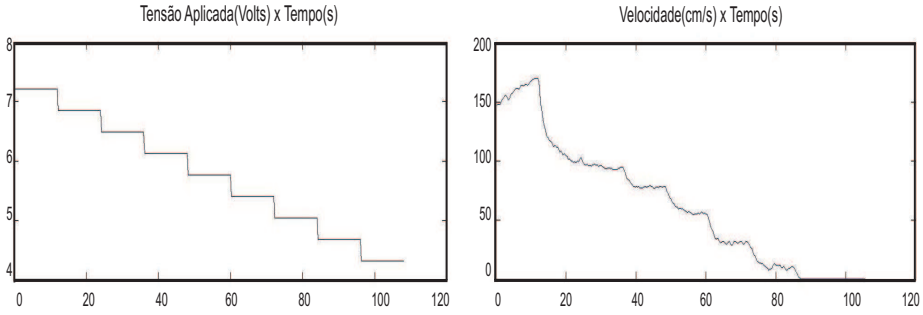
3.5 Projeto do Sistema de Controle

O sistema de estacionamento envolve o projeto de controle dos subsistemas do veículo como o controle de velocidade e o controle de trajetória. Essa seção inicia com o levantamento dos modelos correspondentes aos motores que compõem o veículo e a modelagem do movimento a ser realizado pelo mesmo para seguir a trajetória de estacionamento, que também deverá ser equacionada. Após isso, projeta-se os controladores que serão responsáveis por garantir o comportamento desejado durante a manobra.

3.5.1 Controle de Velocidade

Modelo do Motor

Inicialmente levantou-se o modelo da planta responsável pela tração do veículo, que corresponde a um motor de corrente contínua, e o funcionamento de sensores e atuadores. Os gráficos da Figura 3.5 mostram os resultados obtidos com o ensaio realizado no motor.



(a) Gráfico de Tensão x Tempo (b) Gráfico de Velocidade x Tempo

Figura 3.5: Gráficos de Velocidade do Motor e Tensão Aplicada versus Tempo

Com base nos resultados dos ensaios, foi possível criar um modelo matemático que representasse o sistema de maneira aproximada. A variável mais importante nesse caso, a constante de tempo do

motor, com exceção de alguns pontos específicos, varia em torno de 2.1 segundos, a qual passou a ser a constante de tempo utilizada. Observa-se, no entanto, que o ganho estático é bastante variado, optando-se assim por utilizar um modelo com ganho estático variável.

Projeto do Controlador de Velocidade Nesse caso optou-se por um controlador Proporcional-Integrador (PI) para garantir o seguimento de referência da velocidade, devido a sua comprovada eficácia nesse tipo de aplicação e simplicidade de implementação. Utilizou-se um filtro de primeira ordem na entrada do controlador para evitar mudanças abruptas no sinal de controle. O controlador foi projetado para que houvesse um cancelamento entre o zero do controlador e o pólo do modelo. Além disso, devido ao ganho estático variável da planta, utilizou-se uma forma de controle adaptativo onde o ganho do controlador também é variável. Assim, considerando que a variação do ganho na planta pode ser representado por uma função $f(x)$, o ganho do controlador deve variar de acordo com $\frac{1}{f(x)}$. A Figura 3.6 mostra o diagrama em blocos referente a cada ao modelo da planta, projeto do controlador e os ganhos variáveis de ambos.

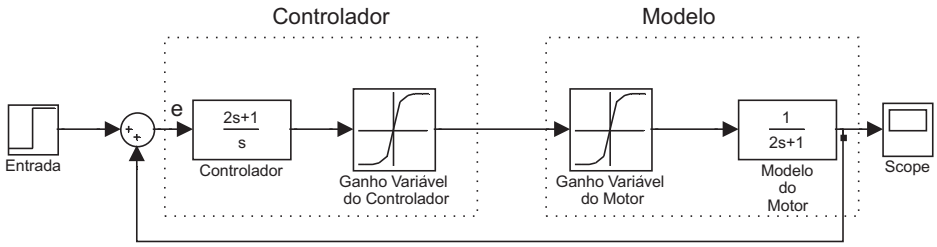


Figura 3.6: Blocos do sistema de controle em malha fechada

Simulação Através das ferramentas de simulação, como o Simulink, é possível validar o projeto do controlador PI para manter a velocidade do motor numa velocidade desejada. A Figura 3.7 mostra o resultado do controlador PI quando atuando no motor do veículo para diferentes pontos de referência.

3.5.2 Controle de Trajetória

Modelo do Veículo

A abordagem utilizada neste trabalho considera o modelo cinemático do veículo baseado em coordenadas locais apresentado em (GOMES, 2006), onde assume-se que os incrementos do ângulo de orientação θ do veículo são pequenos a cada amostra. A partir dessas suposições, chega-se ao modelo

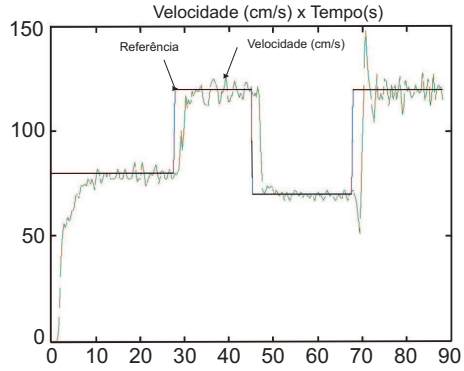


Figura 3.7: Simulação do controle PI de velocidade

cinemático linearizado para o veículo.

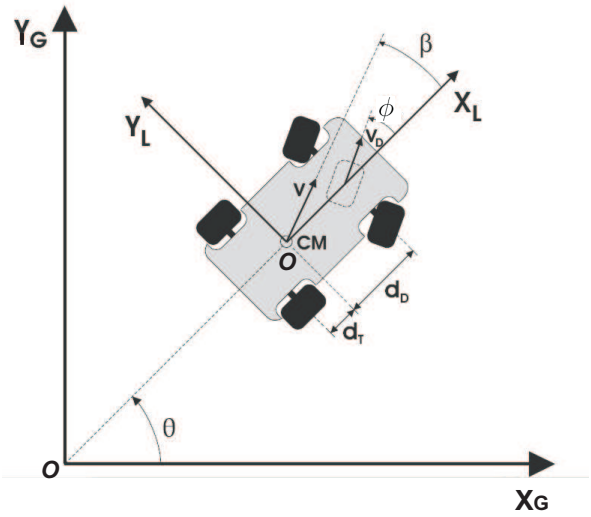


Figura 3.8: Saídas do sistema comparadas com suas referências

O veículo considerado é do tipo direcional, com dois pares de rodas, sendo o par traseiro fixo (tração) e alinhado com o veículo e o par dianteiro móvel (direção), podendo girar sobre o eixo vertical centrado em cada roda. Para simplificar a modelagem, os pares de rodas dianteiras e traseiras são representadas por apenas uma roda na frente e outra atrás na metade de cada eixo mecânico (MURRAY; SASTRY, 1993). Assim, assume-se que não ocorre escorregamento e, portanto, a velocidade do veículo no centro das duas rodas de cada eixo mecânico é sempre tangente à orientação do mesmo (BARRAQUAND; LATOMBE, 1989). Além disso, uma vez que as rodas dianteiras e traseiras possuem a mesma dimensão, pode-se considerar que o módulo da velocidade em ambas é igual. Com isso, as restrições do sistema permitem que as rodas rolem e girem, mas não deslizem.

A partir da Figura 3.8, é definido que o ponto de guiamento está localizado no centro do eixo dianteiro e o centro de rotação está localizado no centro de massa (CM). O modelo cinemático utilizado é o descrito em (MURRAY; SASTRY, 1993):

$$\begin{cases} \dot{x}(t) = v_D(t)\cos\theta(t) \\ \dot{y}(t) = v_D(t)\sin\theta(t) \\ \dot{\theta}(t) = \frac{v_D(t)}{d_D}\sin\phi(t) \end{cases} \tag{3.15}$$

onde x e y são as coordenadas cartesianas globais do CM do veículo, v_D é a velocidade, θ e ϕ são, respectivamente, o ângulo que o eixo longitudinal do veículo faz com o eixo Ox e o ângulo da direção em relação a esse mesmo eixo longitudinal, d_D é a distância entre o cm e o eixo dianteiro.

A obtenção do modelo cinemático em coordenadas locais do veículo direcional apresentado em (GOMES, 2006) parte da noção que o sistema de coordenadas x_L, y_L é fixo e localizado no centro de massa do veículo e que o eixo Ox_L é paralelo ao comprimento do veículo.

A partir desse modelo em coordenadas locais obtém-se o modelo cinemático linearizado, assumindo que os incrementos dos ângulos de orientação do veículo, $\theta(t)$, e de direção da roda dianteira, $\phi(t)$, são pequenos a cada amostra. Após expandir através de Séries de Taylor as funções $\cos\phi(t)$ e $\sin\phi(t)$, e truncando a expressão no primeiro termo ($\cos\phi(t) \cong 1$ e $\sin\phi(t) \cong \phi(t)$), chega-se ao modelo linearizado do sistema. Após a discretização das expressões obtidas com um período de amostragem T_s, tem-se:

$$\begin{cases} x_L(k+1) = x_L(k) + v_D(k)\cdot T_s \\ y_L(k+1) = y_L(k) + v_D(k)\cdot T_s \cdot \phi(k) \\ \theta(k+1) = \theta(k) + \frac{v_D(k)}{d_D}\cdot\phi(k) \end{cases} \tag{3.16}$$

que representam, portanto, os deslocamentos das coordenadas locais do veículo (X_L,Y_L) me relação às coordenadas globais (X_L,Y_L).

A Trajetória de Estacionamento A trajetória de estacionamento a ser executada pelo veículo foi desenvolvida com base nos trabalhos de (HOYLE, 2003) e (HERMANN, 2003). Como pode ser visto na Figura 3.9, esta trajetória em forma de S é obtida posicionando-se corretamente duas circunferências. Considerando este tipo de trajetória, a vaga mínima para que ocorra o estacionamento sem choques

com outros veículos é dada pela equação 3.17 (HERMANN, 2003).

$$L = \sqrt{2 \times R \times Wcar + (Lcar - cm)^2} \tag{3.17}$$

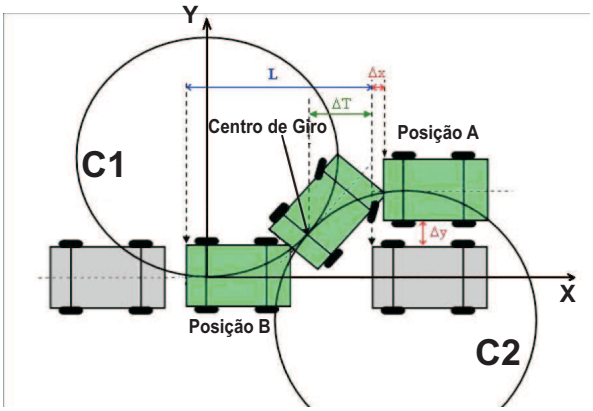


Figura 3.9: Trajetória de Estacionamento

onde $Wcar$, $Lcar$ são, respectivamente, a largura e o comprimento do veículo, CM é a posição do centro de massa do veículo considerando a traseira deste como referência, L é o tamanho mínimo da vaga e R é o raio da circunferência. O valor de R depende do ângulo de esterçamento das rodas, ou seja, quanto maior a deflexão possível das rodas, menor será o raio e, conseqüentemente, menor será a vaga necessária para a realização do estacionamento.

O posicionamento da primeira circunferência é automático. As coordenadas do seu centro, $C_{1,x}$ e $C_{1,y}$, devem ser tais que $C_{1,x} = x_{cm,PB}$, onde $x_{cm,PB}$ é a abscissa do CM na Posição B (posição final de estacionamento), e sua ordenada seja tal que $C_{1,y} = R$. O posicionamento da segunda circunferência está relacionada com a Posição A (posição inicial do veículo), que por sua vez depende de Δy e Δx. Δy é a distância entre a lateral do veículo na Posição A e a lateral do veículo estacionado, Δx é a distância horizontal entre as traseiras desses mesmos dois veículos, conforme mostrado na Figura 3.9. Não se pode especificar valores para Δx e Δy simultaneamente, pois a mudança de um implica a modificação do outro. Dessa forma, optou-se em manter Δy constante e variar Δx, pois, para um veículo na Posição A, é muito mais simples modificar Δx, bastando andar para frente ou de ré.

Assim, com Δy especificado, pode-se calcular as coordenadas do centro da segunda circunferência, $C_{2,x}$ e $C_{2,y}$. $C_{2,y}$ tem valor igual à ordenada do CM na Posição A menos o valor de R ($C_{2,y} = y_{cm,PA} - R$). $C_{2,x}$ é calculado através da relação de Pitágoras 3.18 para o triângulo

retângulo, mostrado na Figura 3.10.

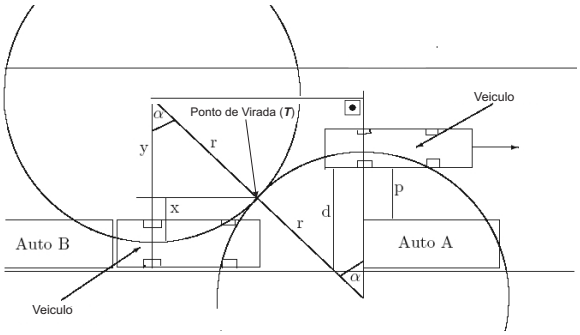


Figura 3.10: Cálculo da Trajetória de Estacionamento

$$R^2 = (C_{1,y} - C_{2,y})^2 + (C_{2,x} - C_{1,x})^2 \tag{3.18a}$$

$$C_{2,x} = \sqrt{R^2 - (C_{1,y} - C_{2,y})^2} + C_{1,x} \tag{3.18b}$$

Com $C_{2,x}$, determina-se Δx e as coordenadas do CM na Posição A:

$$\Delta x = C_{2,x} - C_{1,x} - L \tag{3.19}$$

Assim, com as coordenadas dos centros das circunferências encontrados, tem-se a trajetória de estacionamento especificada. Por fim, necessita-se encontrar o ponto de virada T, ou seja, coordenada onde as circunferências se tocam e onde ocorre a mudança no ângulo da direção do veículo. Podemos calcular o ângulo alfa, indicado na figura 3.10, através da Equação 3.20

$$\alpha = \arctan(\frac{C_{2,x} - C_{1,x}}{C_{1,y} - C_{2,y}}) \tag{3.20}$$

que está relacionado com as coordenas do ponto de virada do veículo conforme a Equação 3.21.

$$T_y = C_{2,y} + R \times \cos(\alpha) \tag{3.21a}$$

$$T_x = C_{2,x} - R \times \sin(\alpha) \tag{3.21b}$$

Simulações em Matlab foram realizadas para validar o algoritmo de geração de trajetória de

estacionamento. O resultado é mostrado na Figura 3.11.

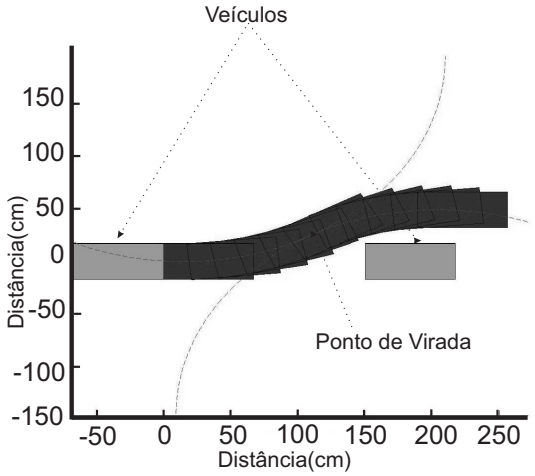


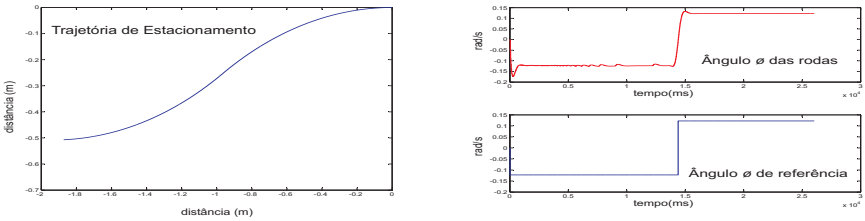
Figura 3.11: Validação do algoritmo de estacionamento

Projeto do Controle de Trajetória Nesta etapa foi utilizado o Controle Preditivo Generalizado (*Generalized Predictive Control*), pois a partir do modelo da planta, prediz-se o comportamento futuro do sistema e compara-se essa informação com o estado atual do sistema real e com a referência fornecida. Esses dados são então repassados a um algoritmo otimizador, que irá calcular a próxima ação de controle, de modo que os erros futuros sejam minimizados.

Para o cálculo da trajetória de referência do controlador, utiliza-se o próprio modelo não-linear do veículo. Desta maneira cria-se uma referência que possa ser seguida pelo veículo perfeitamente, considerando a ausência de pertubações. Utilizando a equação 3.15 iterativamente e especificando corretamente o valor de ϕ, obtemos a trajetória de referência. Para percorrer o segundo arco de circunferência, ϕ deve ter seu sinal trocado exatamente no ponto T. Assim, as coordenadas de T são utilizadas no algoritmo de geração de referência para especificar o ponto da mudança do sinal de ϕ.

Simulação Os gráficos mostrados na Figura 3.12 mostram os resultados obtidos em simulação, utilizando o GPC como técnica de controle para seguimento de trajetória.

Devido à predição dos valores de referência, observa-se na Figura 3.12b uma pequena constante de tempo na resposta das rodas dianteiras do veículo. Para ressaltar essa vantagem do GPC, foram realizadas simulações com PID, mostrado na Figura 3.13. Observa-se uma constante de tempo maior para o ângulo ϕ das rodas quando utilizado o PI.



(a) Trajetória S a ser executada (b) Ângulo ϕ das rodas dianteiras

Figura 3.12: Resultados de simulação com o GPC para o controle de trajetória

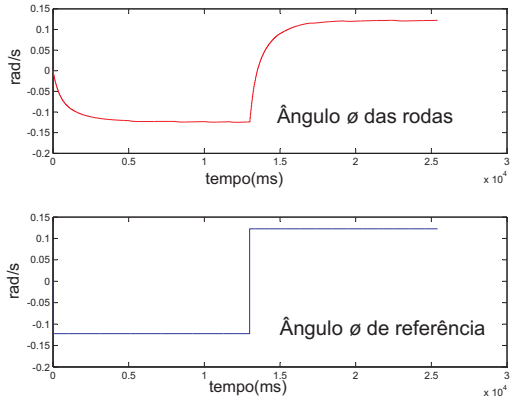


Figura 3.13: Ângulo ϕ das rodas dianteiras com PI

3.6 Considerações Finais

Apresentou-se nesse capítulo algumas estapas do desenvolvimento de um sistema de controle a ser executado em uma plataforma embarcada. Algumas estratégias e técnicas de controle como PID e GPC foram abordadas, ressaltando suas vantagens, desvantagens e onde melhor aplicam-se. Outras estratégias de controle consideradas avançadas foram apresentadas principalmente devido a sua aplicabilidade em processos industriais.

Discutiu-se também a estrutura de controle multinível, onde unidades de processamento de baixo custo, próximas dos sensores e atuadores, executam algoritmos de controle mais simples e rápidos. Para um nível superior, uma unidade central com maior capacidade de processamento executa técnicas mais adequada ao controle do comportamento geral do sistema como, por exemplo, a supervisão e controle do seguimento de trajetória de um veículo autônomo ou um robô.

Por fim, apresentou-se os desafios e as principais soluções que vêm sendo desenvolvidas para minimizar os efeitos no desempenho do controle devido a problemas de amostragem de sinal e atrasos na rede em sistemas distribuídos.

Capítulo 4

Metodologia de Projeto de Sistemas de Controle Tempo Real Embarcados

4.1 Introdução

Esse capítulo discute as etapas de projeto, bem como linguagens e ferramentas aplicáveis ao desenvolvimento de sistemas de controle tempo real embarcado. Após destacar e discutir os principais desafios inerentes a esse tipo de sistema, o que o tornam peculiar e carente de uma abordagem específica de desenvolvimento, o capítulo apresenta as etapas fundamentais a serem cobertas durante o processo de desenvolvimento proposto.

4.2 Sistemas de Controle Tempo Real Embarcado

O desenvolvimento de um sistema de controle tempo real embarcado envolve diversas áreas como eletrônica, teoria de controle e computação. Com isso o projeto desses sistemas é visto como um *co-design*, que engloba o conjunto *hardware*, *software* e processos (plantas).

As dificuldades encontradas nesse *co-design* dependem de vários fatores, conforme destacado em (ÅRZÉN; CERVIN, 2005):

- Sistemas Operacionais: Quais escalonadores são suportados? Qual a precisão nas medições de tempo?
- Algoritmos de escalonamento: Os algoritmos são baseados em tempo, eventos, prioridades ou

deadlines?

- Método de Síntese dos Controladores: Qual o critério a ser utilizado, projeto no domínio contínuo e depois discretizado, projeto diretamente no domínio discreto?
- Características do Tempo de Execução dos Algoritmos de Controle: Os algoritmos apresentam um tempo de execução de pior caso previsível? Qual a amplitude da variação no tempo de execução entre amostras?
- Otimização *on-line/off-line*: Quais as informações disponíveis *off-line* para projeto dos controladores? Essas informações podem ser medidas *on-line*? Há possibilidade do surgimento de novas tarefas e assim a necessidade de re-otimização do sistema durante operação?
- Protocolo de Rede: É possível calcular a latência de pior caso e probabilidade de perdas de pacotes?

Vários esforços vêm sendo realizados para inserir as variáveis supracitadas no projeto dos controladores. No que tange aos problemas relacionados aos atrasos via rede, expostos anteriormente, pode-se ilustrar isso de acordo com a Figura 4.1, onde t_{ca} e t_{sc} representam o atraso presente na comunicação entre o controlador e o atuador, e entre o sensor e o controlador respectivamente.

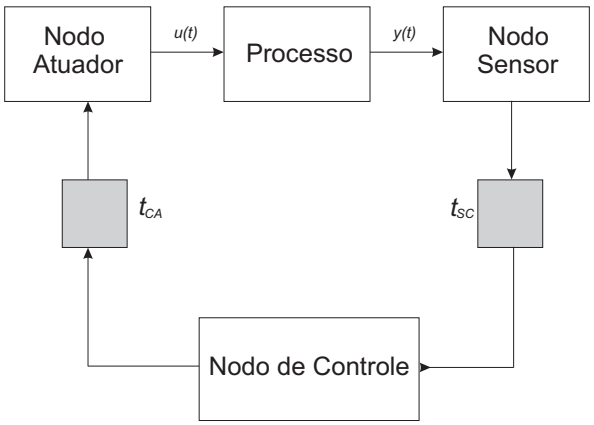


Figura 4.1: Sistema de controle distribuído com os atrasos de comunicação

Em geral, nos sistemas de controle distribuído (descentralizado), o atraso (*jitter*) (Δ) pode ser visto como sendo composto por uma parcela constante (L), que surge devido às características da planta e o tempo mínimo para transmissão da mensagem e por uma parcela variável ($V(t)$), que é conseqüência do compartilhamento do barramento ($\Delta = L + V(t)$). A partir deste modelo de atraso podem-se modelar atrasos periódicos, aleatórios ou constantes.

O atraso nos sistemas via rede é geralmente variante no tempo, o que exige a utilização de métricas de robustez do projeto de controle que contemplem esta variação. Diante disso, novas propostas para o desenvolvimento desse tipo de sistema vêm sendo lançadas. Em (SANTOS, 2006) discute-se o desenvolvimento de sistemas de controle via rede usando o conceito de margem de *jitter*. A margem de jitter ($\hat{J}m(L)$) é definida como o maior valor para o qual a estabilidade de malha fechada é garantida para qualquer atraso $\Delta \in [L, L + \hat{J}m(L))]$. Ou seja a margem de jitter ($\hat{J}m(L)$) é um parâmetro obtido a partir do conjunto planta-controlador-atraso constante, enquanto o jitter (J), por sua vez, é uma medida da amplitude da variação do atraso do sistema sob análise e pode ser medido ou estimado por meio de simulações ou pela via analítica. Uma forma simples de se obter o valor do jitter (J) é a partir da diferença entre o maior ($R = max(\Delta)$) e o menor (L) intervalo de tempo entre os instantes de medição e a atuação, onde:

$$J = R - L \tag{4.1}$$

A margem de jitter baseia-se no critério de estabilidade para sistemas com atraso variável, assim, ela pode ser utilizada para estabelecer os *hard deadlines*, que garantem a estabilidade do sistema em malha fechada.

Uma outra abordagem útil aos sistemas de controle computacional é apresentado em (CERVIN et al., 2004), (PEREZ U. F. MORENO, 2006) e (LINCOLN, 2002). Nesse caso, são apresentados métodos de projeto que procuram compensar a degradação causada por possíveis irregularidades na amostragem do sinal de saída. Uma solução apresentada faz uso de *time-stamps* enviados pelo atuador ao controlador, de modo que esse possa saber quando o sinal de controle está sendo aplicado à planta. Os compensadores são projetados baseados na medida da diferença de tempo entre o momento real de aplicação do sinal e o momento idealmente esperado.

As soluções apresentadas acima referem-se a modificações no projeto de controle, de modo a inserir variáveis que refletem os atrasos na comunicação ou aqueles entre a amostragem e a aplicação do sinal de controle (*sampling jitter*). Outras estratégias procuram ajustar dinamicamente a utilização do processador pelas tarefas de controle de modo que todos os requisitos de desempenho sejam alcançados. Essa estratégia é conhecida como *feedback scheduler* e está mostrada na Figura 4.2.

Em muitos casos, as aplicações de controle apresentam bruscas alterações na carga computacional exigida ao longo do tempo de execução, ou seja, um processo que controla uma planta que

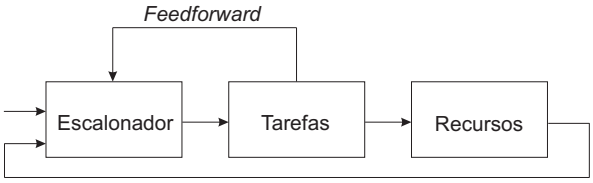


Figura 4.2: Estrura genérica do *feedback scheduling*

encontra-se em regime permanente pode liberar recursos para outro que controla uma planta que sofreu perturbação, por exemplo. Assim, a estratégia de informar ao escalonador sobre alterações nos parâmetros do controlador, como o período de amostragem, permite a otimização da utilização dos recursos disponíveis, cujo objetivo é melhorar o desempenho global do sistema.

Nessa linha, em (HENRIKSSON, 2006) são apresentados métodos para melhorar o desempenho do sistema através do reescalonamento dinâmico de tarefas aplicado a algoritmos do *Model Predictive Control* (MPC), onde o tempo computacional gasto para minimização da função custo desses algoritmos é a métrica para atribuição das prioridades. Essas técnicas buscam a utilização otimizada dos recursos, adaptando o sistema a chegadas de novas tarefas ou eventuais sobrecarga na rede.

Diante dessas estratégias de compensação de atrasos, questões pertinentes à implementação são levantadas em (MARTí; FUERTES, 2001). Nesse caso, são mostrados efeitos do custo computacional dessas soluções e seus impactos no desempenho do sistema, ou seja, a depender da estratégia adotada, excessivas alterações nos parâmetro dos controladores em tempo de execução podem significar instabilidade em algumas plantas. São propostas, então, duas possíveis implementações:

- Recálculo em tempo de execução: para estratégias que apresentam pequenas sobrecargas computacionais como simples controladores PID, o recálculo pode ser realizado *on-line*;
- Recálculo *off-line*: nesse caso os parâmetros a serem modificados serão pré-calculados e armazenados em uma *look-up table*, que armazenará os parâmetros do controlador para cada *sampling jitter* h_k. O tamanho da tabela é limitado, pois

$$h_{min} < h_k < h_{max}, \tag{4.2}$$

onde h_{min} corresponde ao *tick* do relógio do sistema, para o caso de início instantâneo da tarefa de controle, e h_{max} a no máximo quatro vezes a freqüência característica do processo.

Tendo em vista que o desempenho desses sistemas dependem de variáveis como a qualidade do

sinal amostrado, tempo de processamento da informação e tratamento dos sinais para transmissão, conclui-se que o sistemas de controle necessitam de uma abordagem específica de desenvolvimento, com etapas bem discretizadas e ferramentas especializadas de apoio, pois são mais sensíveis ou suscetíveis a falhas se comparados a outros sistemas. As soluções apresentadas acima podem ser úteis em sistemas complexos, com vários nodos de controle, ou sistemas com atrasos na comunicação, o que requer um amplo conhecimento em diversas áreas por parte dos desenvolvedores.

4.3 Engenharia de *Software*

A discussão sobre as etapas de desenvolvimento de sistemas embarcados em geral pode ser iniciada a partir das práticas já bem fundamentadas, utilizadas na engenharia de *software* (ES). Ao longo dos anos, o desenvolvimento de sistemas computacionais mostrou necessitar a adoção de métodos e técnicas bem definidas. Nesse sentido, a ES procura fornecer uma abordagem sistemática para o desenvolvimento, operação e manutenção de *software* (JALOTE, 1991).

Essas atividades são estruturadas nas seguintes etapas de desenvolvimento:

- Análise de Requisitos: refere-se ao detalhamento das necessidades e restrições que devem ser atendidas pelo sistema a ser desenvolvido;
- Projeto: planejamento da solução do problema, ou seja, modelagem do sistema e adoção de estratégias para atender às necessidades levantadas de modo eficiente;
- Codificação: tradução da solução adotada para uma linguagem de programação;
- Teste: detecção de possíveis erros no produto gerado.

Dentre os aspectos essenciais identificados nesse processo de desenvolvimento estão a compreensão detalhada do problema, o planejamento da solução, a tradução ou codificação da solução adotada, os testes e a documentação em cada etapa finalizada. Essas qualidades citadas podem ser utilizadas por outros tipos de aplicações, em particular o projeto de sistemas de controle tempo real embarcado. Nesse caso, apesar de apresentarem exigências distintas, algumas etapas e diagramas são comuns aos dois projetos.

4.4 Proposta para Engenharia de Sistemas de Controle Tempo Real Embarcado

4.4.1 Etapas de Desenvolvimento

Considerando o desenvolvimento propriamente dito, de acordo com (EDWARDS et al., 1997) e (LAVAGNO; SANGIOVANNI-VINCENTELLI; SENTOVICH, 1998), o projeto desse tipo de aplicação deve ser iniciado pela descrição do comportamento do sistema em níveis elevados de abstração, antes de qualquer etapa referente à decomposição do sistema em *hardware* e *software*. Desse modo, o processo de desenvolvimento de um sistema de controle difere em pontos específicos daquele apresentado na Secção 4.3. Ainda segundo esses autores, a implementação final do sistema deverá ser composta da síntese automática desses níveis de abstração, de modo a garantir, através de simulação ou verificação, a corretude do processo de construção.

A Figura 4.3 mostra as etapas básicas propostas nesse trabalho para o desenvolvimento de um sistema de controle tempo real, onde destacam-se algumas atividades como simulação, projeto da arquitetura alvo e calibração, que não são comumente realizadas em outros tipos de aplicação.

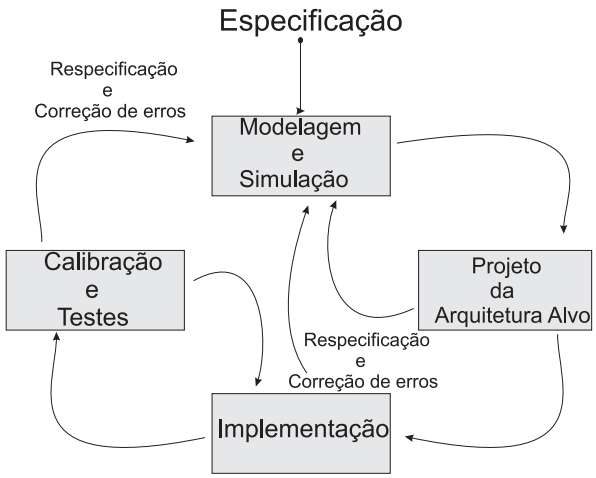


Figura 4.3: Ciclo básico do desenvolvimento de um sistema de controle

Modelagem e Simulação

Nessa etapa, haverá um detalhamento maior da aplicação a partir das especificações, onde é necessário a identificação de todos os modos de operação do sistema. Faz-se então necessário a elaboração de modelos computacionais que expressem fielmente o funcionamento global de toda a

aplicação.

Um modelo computacional (MoC) pode ser definido como uma abstração dos dispositivos reais que executam as diversas funções de um sistema. Diferentes modelos servem para objetivos e finalidades específicas, o que os fazem sempre suprimir algumas propriedades e detalhes que sejam irrelevantes naquele nível de abstração, focando, assim, em outros que sejam essenciais naquele momento. Os MoC's não tratam diretamente de questões como a capacidade de computação ou a possibilidade de realização de um determinado problema, mas auxiliam o projeto e a validação de sistemas concretos (JANTSCH, 2005).

A modelagem inicia-se com a escolha de uma linguagem adequada ao domínio ou natureza computacional em questão, de modo a facilitar a compreensão e representação dos desafios a serem superados.

Na engenharia de *software* apresentada anteriormente enfatiza-se principalmente o processamento de dados, ou seja, as atividades onde predominam a transformação e contenção de informações. Nesse caso, os modelos baseados no fluxo de dados permitem a identificação direta das entidades que enviam ou recebem dados do sistema e a representação da rota seguida por estes.

Nas aplicações de controle, além do fluxo de dados, há a necessidade de linguagens direcionadas à modelagem de sistemas reativos, pois auxiliam na representação de estados, transições e as condições de disparo que representam os modos de operação do sistema. As máquinas de estados finitos (FSM) são exemplos desse tipo de linguagem, onde a ordem na qual as operações são executadas são definidas por uma relação de precedência, ou seja, a execução de uma determinada operação está condicionada à execução da operação anterior (KRIAA, 2007). A principal característica desse tipo de modelo está ligado ao conceito de variáveis do sistema e estados globais, ou seja, o modelo de controle é caracterizado tanto pelo estado atual como pelo estado atual de um dado armazenado.

Em (JANTSCH; SANDER, 2005), organiza-se os modelos computacionais baseando-se em suas abstrações de tempo conforme segue:

- **Modelos Contínuos no tempo**: Para sistemas cujo comportamento é analisado ao longo do tempo, ou seja, de forma contínua como um conjunto não numerável de valores numéricos, realiza-se um modelo computacional contínuo no tempo. Nesse caso, a análise é tipicamente baseada em equações diferenciais, matrizes de estados ou funções de transferência.

- **Modelos a Eventos Discretos**: Nesse caso, todos os eventos estão associados a um instante de

tempo, onde esse é representado por conjunto numerável de valores numéricos. Esses modelos são freqüentemente utilizados para simulação de *hardware* como acontece com as linguagens VHDL e Verilog.

- **Modelos Síncronos**: Nos modelos síncronos não se define exatamente os tempos de cada atividade ou evento, mas apenas a limitação entre o começo e o fim de uma fatia de tempo (*time slot*). Assim, os eventos intermediários que não são vistos no final do *time slot* são ignorados.

- **Modelos independentes do tempo**:

  - **Modelos baseado em Fluxo de dados**: A modelagem através do fluxo de dados corresponde a identificação, modelagem e documentação de como a informação move-se pelo sistema de informação (KRIAA, 2007). O *Data flow model* é um grafo consistindo de nodos (atores) e arcos, que representam seqüências ordenadas de eventos (*tokens*), ou seja, sua execução é uma seqüência de disparos onde *tokens* são consumidos e produzidos. Um caso particular do *Data flow model* é o *Synchronous Data flow model* (SDF). Nesse caso, todos os *tokens* são consumidos ao mesmo tempo.
  - **Modelo baseado em Rendezvous**: O *rendezvous-based model* consiste de processos seqüenciais concorrentes. Os processos comunicam-se entre eles somente em pontos de sincronismo. Ou seja, para efetuar uma troca de informação, os processos devem alcançar esses pontos de sincronização, caso contrário o processo deve esperar pelos outros. Assim, cada processo possui uma *tag* de tempo que é compartilhada com outros processo em pontos de sincronismo.

- **Modelos Heterogêneos**:

  Para lidar com aplicações complexas, os sistemas são geralmente modelados utilizando-se uma composição de módulos. Cada módulo pode ser descrito como uma entidade independente, que usa um modelo computacional específico. A tendência atual é o projeto de módulos de forma independente do ambiente e da sua execução (KRIAA, 2007). Desse modo, permite-se a reutilização dos mesmos módulos em diferentes contextos ou com diferentes plataformas. Esta abordagem apresenta-se vantajosa pois MoC's distintos podem ser usados em partes do sistema onde mostram-se mais adequados. Por outro lado, como trata-se de vários modelos computacionais, a semântica de interação entre estes precisa ser bem definida (JANTSCH;

SANDER, 2005) e, a simulação, nesse caso, constitui-se na principal ferramenta de validação do modelo gerado. Além disso, embora MoC's heterogêneos forneçam um ambiente mais geral, flexível e útil aos projetistas, ao tratar de domínios diferentes (*cross-domain*), torna-se difícil aplicar técnicas de validação e otimização do modelo gerado.

A Tabela 4.1 mostra alguns exemplos de linguagens e ferramentas que facilitam a elaboração dos modelos supracitados.

Modelos	**Linguagens**	**Ferramentas**
Fluxo de Dados	Grafos de fluxo, Lustre	HP-Ptolemy,Lustre-SCADE
Modelos Contínuos	Blocos Funcionais	Simulink, Ptolemy, Scicos
Modelos Discretos	*StateFlow*, *StateChart*, ESTEREL	Simulink, Lustre-SCADE

Tabela 4.1: Ferramentas e linguagens para criação de modelos computacionais

Paralelo ao processo de elaboração dos modelos computacionais, realiza-se o projeto de controladores que venham garantir os requisitos de comportamento de cada subsistema da aplicação. Através de métodos analíticos ou técnicas de identificação de sistemas, levanta-se as equações matemáticas que representam o comportamento do processo a ser controlado ao longo do tempo. Nessa etapa, as ferramentas como Matlab/Simulink, Scilab/Scicos, MATRIX ou ASCET-SD costumam ser amplamente utilizadas para simulação do projeto.

Apesar de ser possível fazer uma avaliação de projeto de controladores com essas ferramentas de simulação, outras questões como, por exemplo, a qualidade dos sensores utilizados ou dos sinais a serem amostrados deixam de ser abordadas.

Com o intuito de permitir uma validação das soluções de controladores desenvolvidas em um ambiente mais realista, pode-se fazer uso das ferramentas integradas como dSpace e RT-Druid Code Generator discutidos capítulo 2. Através delas é possível gerar códigos automaticamente e executar os algoritmos propostos numa plataforma alvo. Além disso, através da estrutura *hardware in loop* acompanha-se em tempo de execução, o desempenho do sistema projetado. Essa abordagem mostra-se muito vantajosa, tendo em vista que antecede a correção de eventuais erros de projeto, além do que, em algumas ferramentas como o RTAI-Lab, é possível o ajuste de parâmetros durante a

execução.

Com os controladores definidos, o principal desafio é simular o funcionamento do sistema como um todo. Para tanto, lança-se mão de ferramentas que facilitem a integração dos modelos, onde inclui-se as interações com o usuário, mostra-se explicitamente os sinais de entrada e saída, os estados e os instantes de ativação das transições do sistema, juntamente com comportamento contínuo de todos os processos controlados.

Projeto da Arquitetura Alvo

O projeto da arquitetura alvo diz respeito à definição da plataforma *HW/SW*, ao modo como serão distribuídas as funcionalidades do sistema e onde cada tarefa será executada. Nessa etapa também são analisadas questões como, qual(is) processador(es) utilizar, utilização ou não de sistemas operacionais de tempo real e qual(is) são eles, e para aplicações distribuídas, qual protocolo de rede a ser utilizado. Além disso, nas aplicações de controle distribuído, a identificação de quais os nodos efetivos na rede e as mensagens trocadas por eles também são feitas nessa etapa.

A seleção dos processadores lida inicialmente com as vastas opções de arquiteturas disponíveis atualmente. De acordo com (VERBAUWHEDE; SCHAUMONT, 2005), essa escolha pode ser guiada por três importantes aspectos: custo, consumo de energia e flexibilidade.

Quanto às questões de consumo de energia, os processadores de propósito geral, apesar de apresentarem vantagens quanto ao custo, são menos eficientes na questão energética. O contrário observa-se nos ASIC, que são bastante utilizados para a construção de módulos de *hardware* específicos a uma dada aplicação como as conhecidas *glue logic*[1]. Essas arquitetura apresentam baixo consumo de energia, porém o custo efetivo é elevado. A Figura 4.4 mostra a relação entre os COTS e os ASIC's com relação a esses dois aspectos.

Além desses fatores apresentados, o tipo de aplicação determina quais características são desejáveis na arquitetura a ser proposta. Algumas aplicações, por exemplo, trabalham excessivamente com processamento de sinais, outras executam cálculos numéricos complexos e há ainda as que favorecem uma maior interatividade com o usuário.

As atividades de controle dos processos são geralmente realizadas por tarefas concorrentes e com restrições temporais (*deadlines*) associadas ao seu tempo de execução. Assim, o projeto é fortemente relacionado com as características da plataforma de *software* selecionada ou, especificamente, com o

[1] Estruturas de *hardware* específicas para compatibilizar interfaces de diferentes dispositivos.

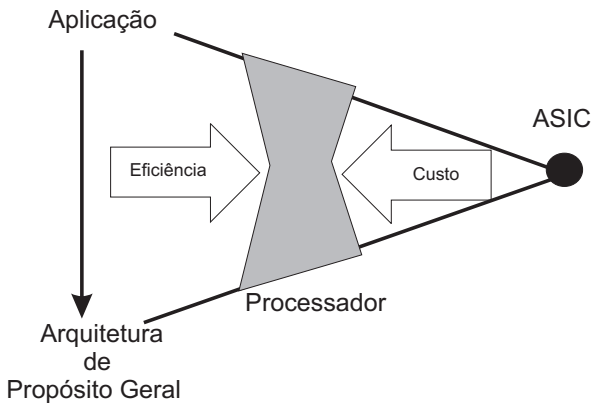


Figura 4.4: Aspectos de consumo de energia das arquiteturas (VERBAUWHEDE; SCHAUMONT, 2005)

sistema operacional utilizado. Diante desse cenário, o desafio no desenvolvimento de um sistema de controle tempo real embarcado é, além de projetar controladores, escaloná-los como tarefas, muitas vezes de tempo real, utilizando-se de recursos e processamento limitados, garantindo, no entanto, o desempenho otimizado do sistema (ÅRZÉN; CERVIN, 2005).

Como as aplicações de controle são geralmente realizadas em uma arquitetura descentralizadas, uma forma natural de realizar essa distribuição é separar o nodos de acordo com as funcionalidades desempenhadas, ou considerando os processos controlados envolvidos, ou ainda levando em conta os aspectos de posicionamento dos dispositivos. No entanto, os efeitos de fatores como a sobrecarga na rede e atrasos na comunicação devem ser considerados, pois impactam significativamente no comportamento do sistema. Assim, o processo de distribuição em unidades de processamento passa então a ser discutido como um problema de otimização, cujos objetivos são: minimização dos atrasos, balanceamento de carga e custo.

Cada atividade a ser desempenhada por um nodo é executada por um conjunto de funções, às quais estão associadas os modos de operação do sistema. As trocas de mensagens são realizadas via um barramento de comunicação e podem ser disparadas por tempo (*time-triggered*) ou por eventos (*event-triggerred*).

Implementação

A etapa de implementação envolve a definição da estratégia de execução das soluções, como, por exemplo, o número de tarefas presentes em cada nodo, os períodos de cada tarefa e os instantes de envio e recebimento de mensagens. Essa etapa envolve também a transformação automática ou manual dos modelos gerados em um código, através de uma sintaxe de programação.

Produzir algoritmos de controle partindo dos modelos abstratos e adaptá-los para os sistemas reais representa uma etapa que pode consumir bastante tempo de desenvolvimento. O uso de ferramentas de geração automática de código facilitam e reduzem o tempo de projeto até a geração de um protótipo do sistema. No entanto, em alguns casos, a codificação manual é necessária para um maior refinamento e otimização da solução gerada, além de possibilitar a utilização da API do SO selecionado.

De acordo com (VERBAUWHEDE; SCHAUMONT, 2005), um grande impacto nos resultados finais podem ser obtidos se os seguintes pontos forem observados:

1. Utilização da Memória:

   Muitos sistemas em tempo real executam operações com grandes matrizes e os fluxos de dados. Exemplos clássicos são as aplicações de vídeo, áudio e jogos. No entanto, as aplicações de controle, com o surgimento de novas técnicas, vêm exigindo cada vez mais velocidade na manipulação, acesso e envio de grandes quantidades de dados.

   De fato, o desempenho da maioria desses sistemas é limitado pela transferências de dados de ou para memória, entre o software e unidades aceleradoras em *hardware* e do tamanho da memória disponível. Desse modo, grade esforço deve ser realizado na elaboração ou refinamento do código para reduzir os acessos à memória e o tamanho requisitado. Para tanto, requer-se uma visão sobre o a produção e o consumo de dados armazenado em matrizes, análise do tempo de vida desses dados em matrizes multidimensionais, além da redução propriamente dita da quantidade de dados a serem armazenado e acessados. Essa redução beneficia o desempenho final e deve ser realizado independentemente da arquitetura a ser utilizada (Brockmeyer C. Kulkarni, 2001).

2. Desempenho do Sistema:

   O refinamento referente ao desempenho diz respeito às transformações que visam à melhora do paralelismo, *pipeline* e da melhor utilização de recursos. Muitas dessas transformações podem direcionar a soluções específicas de plataformas como múltiplos processadores ou processadores com comprimento de instruções muito grande (*very large instruction word*) (VERBAUWHEDE; SCHAUMONT, 2005).

3. Transformações de Ponto-Flutuante para Ponto-Fixo:

Os primeiros algoritmos referentes à solução do problema são, geralmente, desenvolvidos com representações numéricas do tipo *float* (HOLZER; BELANOVIC'; KNERR, 2005). Essa opção permite que sejam realizadas modificações no código sem, no entanto, haver preocupação com *overflow* ou erro de quantização, por exemplo.

Com a solução encaminhada, inicia-se o processo de transformação das operações em ponto-flutuante para ponto-fixo, cujas vantagens são: menor consumo de energia, menor área de silício ocupada, maior velocidade de transferência de dados, e menor latência. Diferentemente dos dois casos anteriores, essa transformação está fortemente relacionada ao plataforma final de execução.

Sob o ponto de vista de *hardware*, nessa etapa são tratados principalmente os aspectos da construção do protótipo e da sua instrumentação, que envolvem a seleção e posicionamento dos sensores e atuadores de acordo com as condições de operação do sistema.

Testes

Na etapa de testes, para os sistemas de controle em geral, a validação dos resultados é realizada a partir do protótipo gerado, pois permitem avaliar os fatores aos quais vários equipamentos estão sujeitos como vibrações, aquecimento e interferências eletromagnéticas.

É essencial que, na avaliação do sistema, sejam considerados todos sinais de entrada selecionados durante as etapas de simulação e comparados o sinais de saída com os gerados no protótipo.

4.5 Considerações Finais

Esse capítulo mostrou as principais etapas de desenvolvimento de um sistema de controle embarcado. Foram propostas e discutidas as principais etapas a serem cobertas durante a realização desse tipo de trabalho. Procurou-se apresentar as diferenças entre as etapas propostas e aquelas discutidas na literatura de engenharia de *software*.

No contexto do projeto SIAMES foram realizadas diferentes modelagens, de modo a tratar todos os aspectos da aplicação embarcada proposta. A Tabela 4.2 relaciona as etapas de projeto propostas, juntamente com os diagramas relacionados.

Etapa de Desenvolvimento	**Linguagens**
Especificação	Casos de Uso da linguagem UML
Modelagem e Simulação	Blocos Funcionais e *StateFlow*
Projeto da Arquitetura	Blocos Funcionais

Tabela 4.2: Linguagens utilizadas no desenvolvimento do SIAMES

Com a abordagem discutida nesse capítulo, busca-se que o processo de desenvolvimento seja realizado de maneira eficiente e produtiva, dirimindo erros e retrabalhos ao término do projeto.

Capítulo 5

Plataforma Embarcada para Sistemas de Controle Tempo Real

5.1 Introdução

Este capítulo descreve os componentes de *hardware* e *software* utilizados para a realização da plataforma utilizada no projeto do Sistema Autônomo de Manobras de Estacionamento (SIAMES). Essa plataforma também foi desenvolvida visando à criação de uma base consistente que facilite sua expansão e que permita o desenvolvimento de qualquer aplicação de controle tempo real embarcada.

Antes de apresentar a plataforma propriamente dita, são discutidos alguns desafios e soluções importantes referentes ao projeto de sistemas de controle distribuído.

5.2 Sistemas de Controle Tempo Real Distribuídos

As aplicações embarcadas de controle são tipicamente distribuídas, onde se utiliza redes de comunicação (*Network Control System*), facilitadas principalmente pela elevada capacidade e baixo custo dos microcontroladores e processadores em geral. As unidades de processamento, portanto, encontram-se geralmente espalhadas e acopladas aos sensores e atuadores através de um barramento de comunicação e são constituídas de um *kernel* multitarefas. Ou seja, esse avanço tecnológico permitiu que o processamento de dados fosse realizado localmente, o que eleva o grau de confiabilidade e modularidade da malha de controle.

Em sistemas de larga escala, essa distribuição do processamento apresenta vantagens em relação

a uma estrutura dedicada, como: melhor gerenciamento do processamento da informação e facilidade de reconfiguração da malha instalada. A Figura 5.1 mostra uma visão da interligação típica entre sensores e atuadores numa aplicação de controle distribuído.

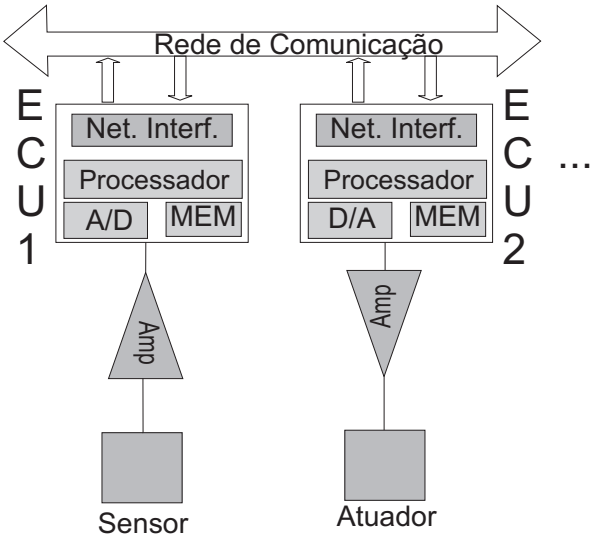


Figura 5.1: Arquitetura típica de um sistema de controle distribuído

Assim como os sistemas computacionais possuem tarefas que devem ser escalonadas para terem acesso ao processador e, portanto, o início de execução de uma tarefa pode ser bloqueado por outra, nos sistemas distribuídos o acesso ao barramento de comunicação também precisa ser realizado de maneira coordenada. Do mesmo modo que as tarefas, essas mensagens podem apresentar requisitos temporais rígidos (*Hard Real-Time*) ou não (*Soft Real-Time*). Esses requisitos tornam-se mais difíceis de serem atendidos quanto mais nodos são adicionados na rede ou quanto mais largura de banda é dedicada um só nodo.

Problema As aplicações de controle lidam com variáveis que não são comumente encontradas em sistemas computacionais convencionais. Por exemplo, um nível de descentralização apropriado, que geralmente depende de fatores que incluem custo, balanceamento de carga, além dos requisitos temporais. Assim, o projeto pode ser sensível a eventuais sobrecargas na rede devido a pequenos períodos de amostragem ou a um número excessivo de nodos ativos. Nesse caso, um balanceamento de carga melhora a flexibilidade do sistema e a margem de utilização do mesmo. Já quanto aos requisitos temporais, incluem-se a minimização ou limitação dos atrasos na comunicação e a sincronização das ações executadas em cada nodo.

5.3 Plataforma Proposta

A proposta desenvolvida no contexto desse trabalho apresenta-se como uma solução aos desafios encontrados na execução de um sistema de controle tempo real. Para tanto, os fatores mencionados de custo reduzido e requisitos temporais norteiam a tomada de decisão sobre quais componentes devem ser utilizados.

O desenvolvimento de sistemas tempo real implica não ser possível fazer uso de mecanismos de abstração de *hardware* como os sistemas operacionais de propósito geral e os protocolos de comunicação não determinísticos, pois as plataformas convencionais apresentam uma variação temporal significativa, ou seja, o tempo transcorrido entre o início e o término de uma tarefa, por exemplo.

Diante do exposto, o projeto SIAMES procurou reunir soluções de *hardware* e *software* que viabilizassem o desenvolvimento de aplicações de controle tempo real via rede CAN de modo fácil e rápido. Ao término, obteve-se uma estrutura flexível e de custo reduzido, pois é baseada em dispositivos reprogramáveis, onde a plataforma final e a de desenvolvimento são as mesmas, e de custo reduzido, e em soluções de *software* com código aberto.

5.3.1 Componentes de *Hardware*

Uma das etapas do processo de desenvolvimento de um sistema computacional embarcado é a escolha adequada da plataforma de *hardware*. Sabe-se que pequenas diferenças no custo dos componentes é determinante para o sucesso comercial da aplicação. Sendo assim, a noção de que processadores mais rápidos representam melhores soluções deve ser substituída pela idéia de que o suficientemente rápido é suficientemente bom, ou ainda, tendências de escolher máquinas de 64 bits ou mais quando os de 8, 16 ou 32bits podem atender à demanda da aplicação.

Como a arquitetura do SIAMES é multicamada, conforme explorado ao longo desse trabalho, a escolha da plataforma procura primeiramente atender a todos os requisitos apresentados em cada nível de controle. O projeto baseou-se numa estrutura onde existe uma unidade central de maior capacidade de armazenamento e processamento e outras menores mais próximas aos processos a serem controlados.

Camada Baixo Nível Para realização das atividades no baixo nível, ou seja, o controle da cinemática do veículo, o sensoriamento e atuação nos processos, são necessárias basicamente as seguintes estruturas:

- Geração de sinais *Pulse Width Modulation* (PWM);
- Conversores analógico-digital (ADC);
- Interrupções externas de *hardware*);
- Entradas e saídas digitais;
- Interface CAN de comunicação;

Para tanto, optou-se por um processador de 8 bits modelo AVR-AT90CAN128 devido a algumas características favoráveis a esse processador:

- Suporte aos SOTR's (FreeRTOS, EPOS, ERIKA);
- Sistema de alta performance e baixo consumo;
- Custo acessível;
- Controladoras CAN 2.0A e 2.0B.

Ressalta-se que o ambiente de desenvolvimento desse microprocessador (AVRStudio) é gratuito e integra-se perfeitamente ao FreeRTOS, onde pode ser realizado uma depuração melhor do próprio código do SO ou da solução desenvolvida.

Camada Alto Nível Para o nível de controle responsável pelo planejamento e coordenação da trajetória foi escolhido um processador que atendesse a demanda computacional exigida pelo algoritmo de controle preditivo. Conforme mencionado, essa técnica de controle apresenta um custo computacional elevado, com cálculos matriciais a cada período de amostragem e a solução de um sistema linear de alta ordem. Essa decisão também considerou futuras expansões do sistema com outras funcionalidades ou implementação de novas soluções. Assim, um processador de 32 bits tipo PowerPC-MPC5200 foi utilizado. Dentre suas características estão:

- Suporte aos STOR's (Linux/Xenomai, Linux/RTAI)

- Processador de 32 bits com capacidade para 760 MIPS a 400MHz;
- FPU (*Floating Point Unit*) de dupla precisão;
- MMU (*Memory Management Unit*);
- IU (*Integer Unit*) ;
- Controladoras CAN 2.0A e 2.0B.

Rede de Comunicação Para a realização das trocas de mensagens entre essas duas camada, utilizou-se o protocolo CAN no modo *event triggered*. Com o uso do CAN nessa rede de controle, alguns benefícios foram constatados:

- Redução do tempo de projeto (facilidade de acesso, variedade de componentes e ferramentas);
- Custo reduzido para conexão (simples com pequeno número de cabos e conectores);
- Confiabilidade (poucas conexões);

5.3.2 Componentes de *Software*

Na estrutura proposta, foram escolhidos dois sistemas operacionais de tempo real. A escolha correta de um SOTR, para uma determinada aplicação, requer principalmente a avaliação do equilíbrio entre o desempenho, flexibilidade de configuração, licenciamento, suporte técnico e as ferramentas de desenvolvimento.

Camada Baixo Nível Adotou-se para esse nível o FreeRTOS devido, principalmente, a fatores como: código aberto, desenvolvimento em linguagem C e portabilidade ao AT90CAN128. Algumas métricas importantes do SO em execução no AT90CAN128 foram levantadas para avaliação do seu desempenho, a constar:

- Tempo para criação de uma tarefa *task*: 208μs;
- Tempo para gravação, troca e restauração de contexto de tarefa: 9.8, 10.2, 9.6μs respectivamente;

- Tempo para primeira entrada da tarefa de maior prioridade, ou seja, o tempo desde sua criação até o estado de executando (*running*): 405μs;
- Tempo para execução do escalonador de tarefas: 41.8μs;

O FreeRTOS disponibiliza um escalonador baseado em prioridades que é executado a cada $1ms$ (*tick*). Como o *port* original do FreeRTOS para as arquiteturas AVR é destinado ao uso na família ATMega128, durante o desenvolvimento do sistema de controle, foram feitas alterações no código do SO para o seu funcionamento no AT90CAN128.

Camada Alto Nível Devido a sua ampla aceitação no meio acadêmico e facilidade de acesso, adotou-se o Linux como o sistema operacional para a plataforma responsável por esse nível de controle, juntamente com uma extensão (*patch*) de tempo-real, denominada Xenomai.

Conforme mencionado no capítulo 2, o Xenomai é um *framework* para desenvolvimento de aplicações de tempo real que trabalha em cooperação como o *kernel* do Linux, ou seja, o Xenomai representa um μ*kernel* (núcleo) com um escalonador tempo-real. Para utilização de ambos, Xenomai e Linux, utilizou-se o ADEOS, pois ele possibilita o compartilhamento de recursos de *hardware* entre eles. De modo geral, o Xenomai é executado em um domínio de mais alta prioridade do que o *kernel* do Linux , além de implementar diferentes API's, oferecendo serviços como: criação de tarefas de tempo-real (*real-time tasks*), *timers*, semáforos etc.

Algumas métricas do Linux/Xenomai em execução no PowerPC-MPC5200 foram levantas através de funções disponibilizadas após instalação:

- Valor médio [1] da latência na troca de contexto: 16μs;
- Latência entre momento esperado de expiração do *timer* e o tempo real: 7μs;
- Troca de contexto em modo primário e modo secundário: 427μs (modo primário), 411μs (modo secundário);
- *Time offset* comparado com a referência *gettimeofday()*: 0.009 $\frac{\mu s}{s}$ (*drift value*);

É válido observar que as tarefas de tempo-real (*RT task*) em espaço de usuário podem alternar o domínio entre "modo primário" e "modo secundário" durante a execução. Quando uma *RT task*

[1] Valores médios obtidos após mil iterações

é escalonada pelo Xenomai, diz-se estar no domínio primário, quando pelo *kernel* do Linux kernel, diz-se estar no domínio secundário.

Rede de Comunicação Apesar de sua ainda modesta aceitação em aplicações embarcadas automotivas, o ambiente Linux, a partir da versão 2.6.24, ganhou espaço com o suporte a redes CAN programáveis via conceito de *sockets* (socketCAN) (CORBET, 2007). Desse modo, o CAN passou a ser disponibilizado com todas as vantagens de um protocolo de rede, com uma programação orientada especificamente ao conceito de redes: *socket* API, controle da qualidade de serviço (QoS), fila de mensagens, filtros, etc. A Figura mostra o subsistema CAN no *kernel* do Linux.

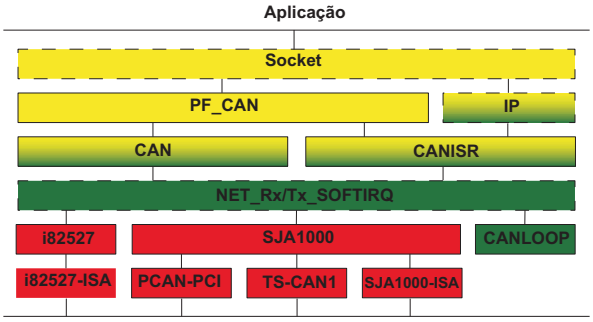


Figura 5.2: Subsitema CAN no *kernel* do Linux

Os *sockets* correspondem a pontos de entrada no espaço de *kernel* e o PF_CAN representa uma família de protocolos CAN: CAN_RAW, CAN_BCM, CAN_TP16, CAN_TP20, CAN_MCNET, CAN_ISOTP, CAN_BAP, CAN_NPROTO. As chamadas de sistema são: *socket, bind, connect, send, recv*. A camada mais abaixo mostra alguns *drivers* de controladoras CAN atualmente disponíveis. Além desses mostrados, um, em particular, possibilitou o uso de rede CAN com o PowerPC-MPC5200, o MSCAN/MPC5200.

A utilização da rede CAN via *sockets* pode representar uma fonte de indeterminismo no processo de comunicação. Para resolver esse problema, surgiu o RTCAN (RTCAN, 2008), que é um conjunto de funções que permitem transmissão e recepção de mensagens CAN em ambiente Xenomai através da utilização de um *driver* para dispositivos CAN baseado no RTDM (RTSocketCAN).

5.4 Considerações Finais

Esse capítulo apresentou alguns desafios encontrados no desenvolvimento de uma sistema de controle tempo real e, mais do que isso, detalhou uma plataforma que permite desenvolver tais

aplicações de modo a utilizar os benefícios que componentes como os sistemas operacionais fornecem sem comprometer significativamente o comportamento temporal desejado.

Apesar de ter sido aplicada com apenas dois nodos de controle, o sistema mostra-se capaz de suportar expansões sem grandes alterações no *lay-out* existente e com poucas mudanças no código implementado.

Para as situações onde um número grande de nodos de controle estão presentes, conforme discutido no capítulo anterior, faz-se necessário a realização de um estudo sobre a influência de atrasos da rede no desempenho da solução de controle, como mostrado em (SANTOS, 2006). Além disso, avaliar a escalonabilidade das mensagens, com a análise do pior caso do tempo de transmissão, conforme discutido em (TINDELL; BURNS; WELLINGS, 1995).

Capítulo 6

Sistema Autônomo para Manobras de Estacionamento - SIAMES

6.1 Introdução

Este capítulo apresenta o projeto e desenvolvimento de um sistema de controle tempo real embarcado aplicado ao controle de trajetória e, de modo mais específico, capaz de realizar manobras de estacionamento. São apresentados os requisitos do sistema e o veículo-protótipo utilizado, os modelos e algoritmos de controle desenvolvidos e a implementação dos mesmos no veículo-protótipo. Este estudo de caso foi desenvolvido de acordo com a metodologia apresentada no capítulo anterior.

6.2 Requisitos do Sistema

A primeira etapa definida no ciclo de desenvolvimento apresentado consiste no levantamento e na análise dos requisitos, os quais estão resumidos na Tabela 6.1 (a lista completa dos requisitos encontra-se no anexo A).

A Figura 6.1 exibe o diagrama de casos de uso elaborado. Observa-se que além das duas funcionalidades em destaque, existem também uma série de outras funcionalidades relacionadas. Além disso, o diagrama mostra também os elementos externos (atores) envolvidos no processo, neste caso o motorista, os sensores e o motor de tração.

F1 Início/Parada do sistema	
Descrição: O sistema deve ser explicitamente ativado pelo motorista para entrar em modo de operação.	
Nome	Restrição
NFR1.1 - Velocidade Máxima	Para iniciar o sistema a velocidade deve ser mantida $\leq$ 20Km/h.
F2 Procura por Vaga (*real-time operation*)	
Descrição: Quando ativado, o sistema deve iniciar a procura por uma vaga adequada enquanto o veículo move-se para frente.	
Nome	Restrição
NFR2.1 - Dimensões do Veículo	O sistema deve ser adaptável a veículos de dimensões diferentes.
F3 Estacionamento (*real-time operation*)	
Descrição: O motorista pode ativar o início de manobra somente com a vaga encontrada. O sistema controla a velocidade e a direção do veículo.	
Nome	Restrição
NFR3.1 - Distância Máxima	O veículo não deve estar a mais do que 20m da vaga.
NFR3.2 - Parada com intervenção do motorista	O sistema deve ser interrompido quando o motorista tocar a direção.

Tabela 6.1: Requisitos do sistema autônomo de manobra de estacionamento

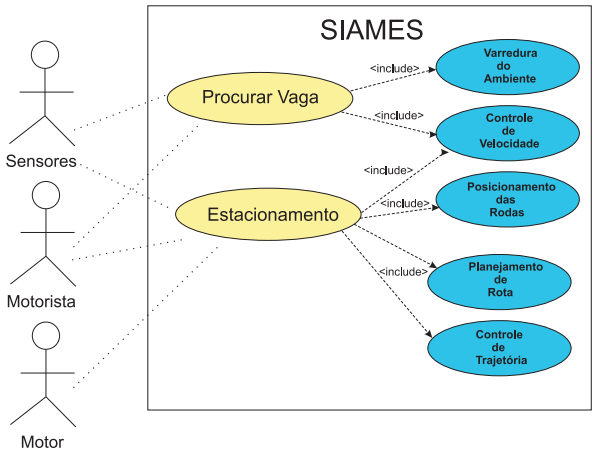


Figura 6.1: Diagrama de Casos de Uso - UML do SIAMES

6.2.1 O Veículo-Protótipo

Para projetar um sistema capaz de atender os requisitos apresentados, instrumentou-se um veículo de controle remoto com sensores capazes de obter informações sobre o ambiente. Além disso, alterou-se a eletrônica original do veículo para permitir a execução da manobra de estacionamento

de forma autônoma. Desta forma, utilizou-se um veículo dotado dos seguintes dispositivos:

- Sensores ultra-som lateral e traseiro para localizar uma vaga ideal e para evitar colisão respectivamente;
- Encoder incremental para medição de velocidade;
- Servo-motor para posicionamento das rodas dianteiras;
- Motor de corrente contínua para tração do veículo;
- Sistema de acionamento dos motores de tração;
- Unidades de processamento de dados;
- Rede de comunicação;

A Figura 6.2 mostra o veículo-protótipo instrumentado, utilizado para validação do sistema de estacionamento. Destaca-se o sensor lateral responsável pela identificação de um obstáculo e as unidades de processamento com o sistema de acionamento do motor logo acima.

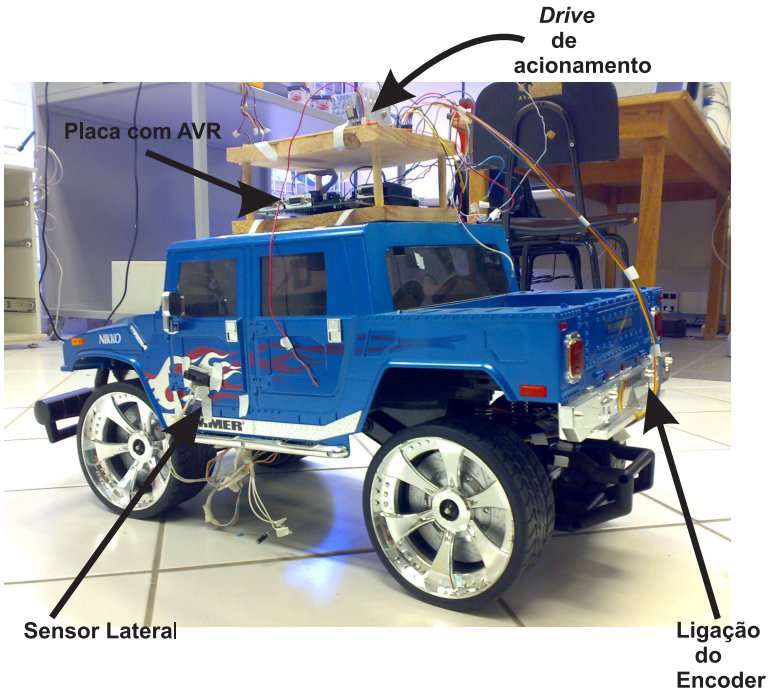


Figura 6.2: Veículo protótipo utilizado no SIAMES

A Figura 6.3 exibe o detalhe do *encoder* utilizado no projeto. Sua principal vantagem está no tamanho reduzido e excelente resolução (1000 pulsos/revolução).

Figura 6.3: *Encoder* da fabricante USDigtal utilizado no SIAMES

O *encoder* é ligado a um contador de pulsos (12bits), cujas saídas estão ligadas ao processador responsável pelo controle dos subsistemas. Os valores de saída desse contador são lidos em intervalos de tempo periódicos para o posterior cálculo da velocidade do veículo.

6.3 Modelagem e Simulação

A primeira etapa da modelagem consiste em destacar os vários elementos que compõem a aplicação e que são separados por suas funcionalidades. A Figura 6.4 mostra o modelo definido para o sistema em questão, onde foram utilizados os blocos funcionais disponíveis no Simulink.

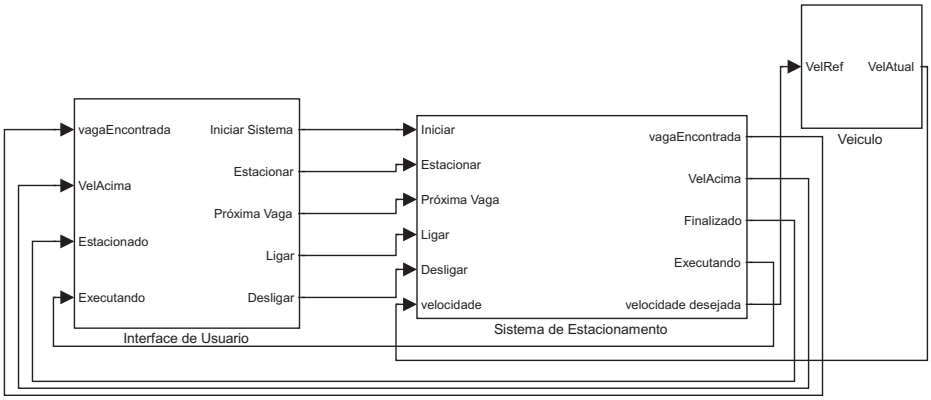


Figura 6.4: Diagram de blocos funcionais da interface de usuário, do sistema e do veículo

Essa etapa é vista como uma modelagem de alto nível, onde os detalhes do funcionamento são omitidos, apesar de serem estabelecidas quais informações são enviadas a cada elemento e quais são retornadas por ele. Conforme mostrado no diagrama, o sistema é separado por três blocos funcionais principais: a *Interface do Usuário*, que representa a entrada e saída de sinais enviada por/para este, o *Veículo*, que representa o processo a ser controlado e o *Sistema de Estacionamento*, que representa o sistema de controle, responsável pela busca de vaga e pela manobra de estacionamento.

Num segundo momento são identificados os modos de operação do sistema, que representam seus

possíveis estados de execução. A Tabela 6.2 detalha os estados identificados e as funções relacionadas. Neste caso, as transições entre os estados são geralmente ativadas por ações do usuário (motorista).

Estados do Sistema	**Funções Associadas**	**Variáveis Associadas**
Desligado	—	—
Ocioso	Aguardando evento início.	Evento da Interface
Varrendo Ambiente	Leitura dos sensores laterais; Controle de velocidade; Cálculo da distância percorrida.	Dados dos sensores; Velocidade do veículo.
Vaga Encontrada	Aguardando evento de aceitar e cancelar.	Evento da Interface
Estacionando	Leitura dos sensores traseiros; Controle de velocidade; Controle de direção; Controle de trajetória.	Dados dos sensores; Velocidade do veículo; Ângulo das rodas dianteiras.

Tabela 6.2: Estados, variáveis e funções associadas do sistema autônomo de manobra de estacionamento

Interface de Usuário Os comandos enviados ao sistema pelo usuário através da interface encontram-se resumidos na Tabela 6.3, enquanto mensagens retornadas pelo sistema ao usuário, também pela interface, encontram-se resumidas na Tabela 6.4.

Comandos enviados pelo Usuário	**Descrição**
Ligar	O sistema está ligado, porém inoperante, assim que é dada partida no veículo.
Iniciar sistema	O usuário solicita o início do sistema de estacionamento, que consiste primeiramente em buscar uma vaga adequada.
Estacionar	O usuário indica ao sistema que aceita estacionar na vaga encontrada.
Próxima Vaga	Caso não aceite estacionar na vaga encontrada, o usuário pode solicitar a busca por uma outra vaga.
Desligar	A qualquer momento, desde o início do sistema, o usuário pode desativar o sistema de estacionamento.

Tabela 6.3: Comandos enviados pelo usuário ao sistema de estacionamento

Sistema de Estacionamento O bloco referente ao sistema de estacionamento agrega todos os componentes do veículo (motores, sensores e atuadores) e é responsável pela execução da manobra de estacionamento e também das atividades relacionadas, como procurar uma vaga. Tendo em vista este conjunto de funcionalidades, foi necessário criar um bloco híbrido, composto de uma parte contínua

Mensagens retornadas pelo sistema	Descrição
Vaga Encontrada	O sistema indica ao usuário que uma vaga adequada foi encontrada.
Velocidade Acima (VelAcima)	Caso a velocidade esteja acima de um determinado valor, o usuário é notificado.
Executando	Após aceitar a vaga, o usuário é avisado quando a manobra é efetivamente iniciada.
Estacionado	Ao término, o sistema avisa que o carro encontra-se estacionado sem alguma operação a mais a ser realizada.

Tabela 6.4: Mensagens retornadas pelo sistema de estacionamento ao usuário

e outra discreta. Enquanto a primeira é responsável pelo controle preditivo, a segunda controla os comandos vindos do motorista.

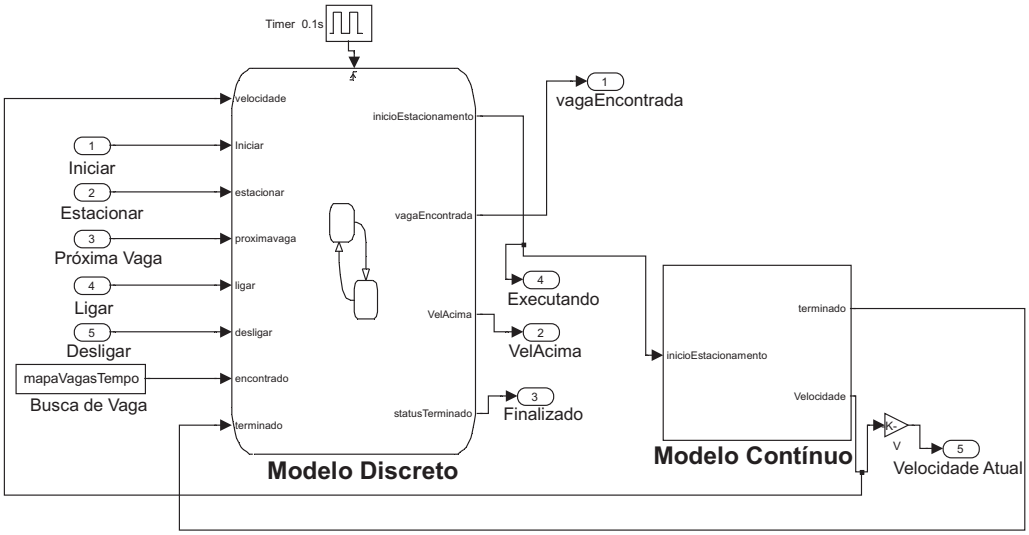


Figura 6.5: Ligação entre os modelos discreto e contínuo do SIAMES

Como pode ser visto na Figura 6.5, o modelo discreto recebe os sinais provenientes da interface de usuário, que ativam diferentes modos de operação. Com o sistema iniciado, dá-se início a busca por uma vaga e, uma vez encontrada a vaga, o usuário será avisado pelo envio do sinal “vagaEncontrada” para a interface. O usuário então confirma ou não a manobra através do sinal “estacionar”.

Após confirmar a intenção de estacionar o veículo, entra em funcionamento todo o modelo contínuo do sistema que executará a manobra. O valor de velocidade é monitorado pelo bloco de lógica que avisa ao usuário, através do sinal “VelAcima”, a ocorrência de um valor de velocidade acima do limite. Quando a manobra é finalizada, o usuário é notificado pelo sinal “terminado”.

6.3.1 Modelo Discreto

Conforme discutido no capítulo anterior, as máquinas de estados finitos (*Finite State Machines*, *Statecharts*, *Stateflow*) são linguagens freqüentemente utilizadas para representar os estados e transições do sistema. Neste trabalho optou-se pelo uso da linguagem *Stateflow* para representar o modelo discreto devido a sua compatibilidade com o ambiente Simulink.

A Figura 6.6 o corpo principal do diagrama desenvolvido. Quando o sistema é ligado, ele entra no estado *Ocioso*, onde espera por um comando do usuário. Uma vez iniciado o sistema, ele inicia o monitoramento da velocidade do veículo e as etapas que compõem o estacionamento, ambos representados pelo estado *Executando*. A velocidade do veículo deve ser mantida inferior a 20 cm/s durante a manobra.

Observa-se, mais uma vez, que o sistema pode ser abortado a qualquer momento, através do comando na interface de usuário. Para a realização da simulação e considerando a distância percorrida, considerou-se que após 20 segundos ([after(20,tempo)]) a manobra estaria finalizada, o que mostra o estado *Estacionado* .

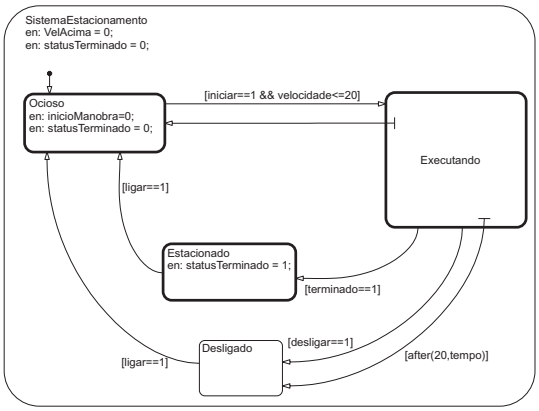


Figura 6.6: Diagrama *Stateflow* dos modos de operação o SIAMES

O estado *Executando* é detalhado através de outro diagrama de estados, mostrado na Figura 6.7. Inicialmente é feito a busca por uma vaga, que é representada pelo estado *Varrendo Ambiente*. Uma vez que a vaga foi encontrada, e em sendo desejável iniciar a manobra automaticamente pelo motorista, inicia-se o estacionamento, que está identificado pelo estado *Estacionamento*. Caso o motorista não esteja satisfeito com a vaga encontrada, pode-se optar por procurar uma próxima vaga

ou então, após um intervalo de tempo, deixar o sistema retornar ao estado *Varrendo Ambiente* e continuar a procura de uma nova vaga.

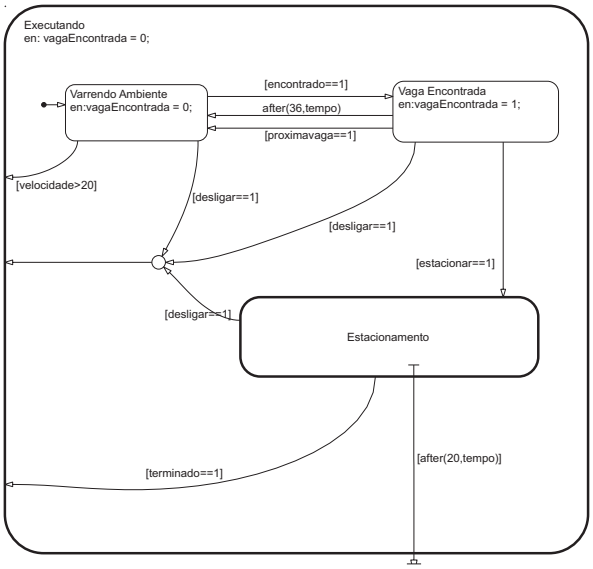


Figura 6.7: Diagrama *Stateflow* da manobra de estacionamento

É requisito do projeto que a execução da manobra possa ser realizada com a velocidade variável. Portanto, durante o estacionamento, é importante que a velocidade do veículo seja monitorada afim de que todo o processo ocorra normalmente sem desvio de trajetória. A Figura 6.8 apresenta o digrama *StateFlow* responsável pelo monitoramento da velocidade durante a manobra.

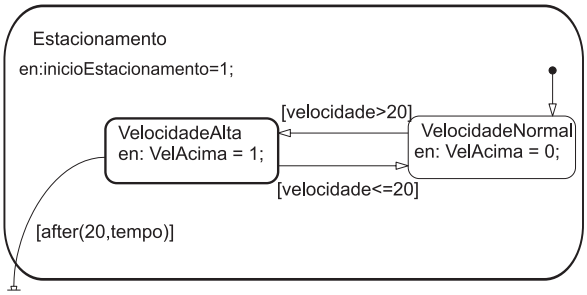


Figura 6.8: Diagrama *Stateflow* do controle de velocidade

6.3.2 Modelo Contínuo

No modelo contínuo foram projetados os controladores envolvidos e os modelos dos processos a serem controlados, conforme mostrado na Figura 6.9.

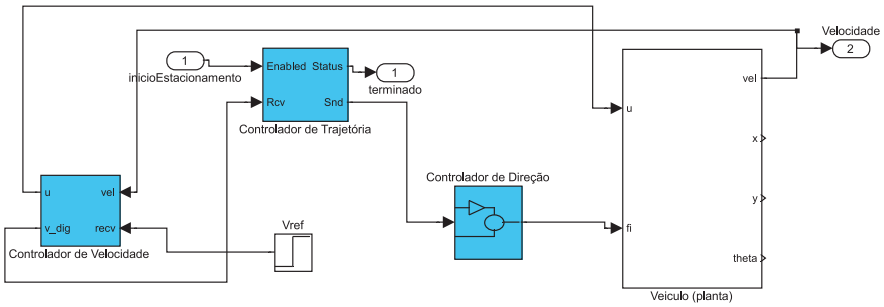


Figura 6.9: Modelagem contínua do sistema de estacionamento

Após a análise de desempenho dos controladores projetados, a modelagem segue com a inserção dessas funções nos blocos de controle como o de velocidade, de direção e de trajetória mostrados. As funções que modelam o movimento do veículo, o motor de tração e o de direção estão inseridos no bloco do processo (planta) em questão. A Figura 6.10 mostra essas funções expressas em diagramas de blocos.

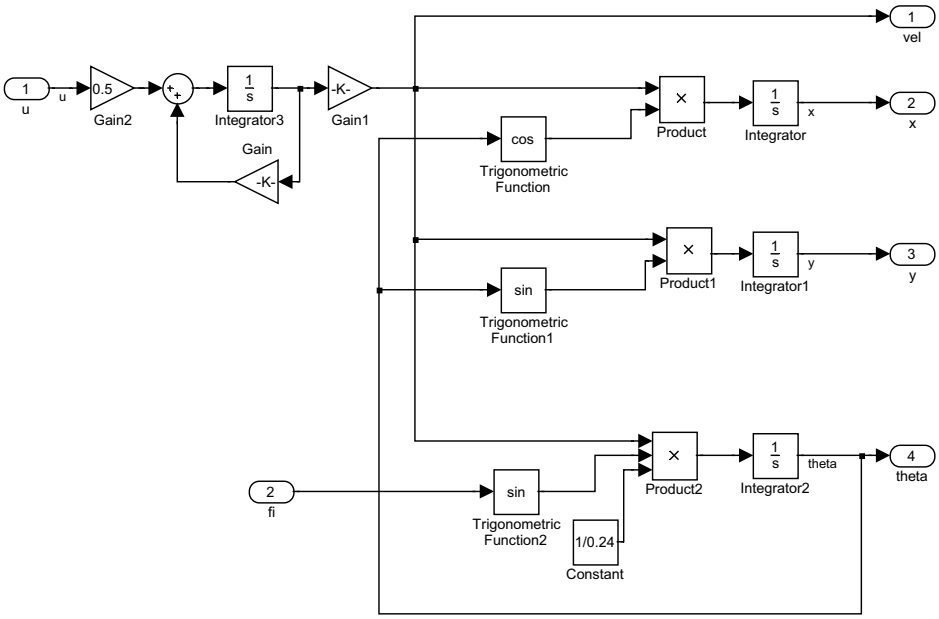


Figura 6.10: Diagram de blocos dos subsistemas do veículo

É importante ressaltar no modelo contínuo a forma como os blocos interagem, ou seja, quais são os sinais de entrada (realimentação) retornados pelo processo, quais os sinais de saída (atuação) dos controladores e quais sinais são necessários por cada bloco de controle envolvido no sistema. Como visto na Figura 6.10, o sinal u corresponde ao sinal de controle que atua no motor de tração e o sinal fi representa o ângulo em que devem ser ajustadas as rodas dianteiras para que a trajetória seja seguida corretamente. O processo retorna sua velocidade atual e o ângulo de suas rodas como sinais de realimentação para os blocos de controle.

6.4 Projeto da Arquitetura

Com o comportamento do sistema validado através de simulações, é necessário avaliar como será feita a implementação do mesmo. Assim, são feitas decisões de projeto como, por exemplo, optar por processamento distribuído ou centralizado, escolher a melhor interface de comunicação e organizar as tarefas computacionais. No SIAMES, decidiu-se por separar o modelo contínuo da Figura 6.9 em duas partes principais: controle de trajetória, denominado alto nível e o controle dos subsistemas (velocidade, direção e sensores), denominado baixo nível. Inclui-se ainda a parte responsável por representar a interface com o usuário e disparo de eventos.

O projeto então iniciou-se identificando cada tarefa de controle, seguindo suas características, como o tipo e a quantidade de informação tratada. O controle de alto nível é o responsável pelo planejamento e supervisão do seguimento de trajetória, enquanto o baixo nível controla a velocidade e o ângulo das rodas, além de adquirir os dados do ambiente externo. A Figura 6.11 mostra como ficou a separação em camadas dessas responsabilidades citadas.

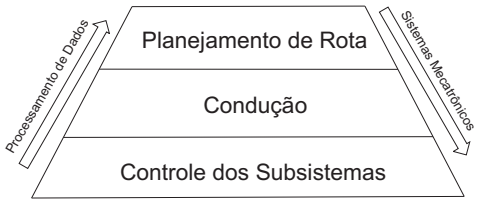


Figura 6.11: Ações executadas no SIAMES

A camada *Planejamento de Rota* corresponde a geração de trajetória de estacionamento a ser seguida pelo veículo, enquanto a camada *Condução* representa o controle ou supervisão do seguimento dessa trajetória. A camada responsável pelo sensoriamento e atuação no veículo está representada pela camada *Controle dos Subsistemas*.

A partir dessas definições de responsabilidades, juntamente com a integração dos modelos realizada anteriormente, tem-se uma idéia sobre o conjunto de tarefas necessárias a serem utilizadas para execução do sistema elaborado. Faz-se necessário agora identificar quais os processadores e quantos estarão envolvidos no processo, o que corresponde ao planejamento da arquitetura ou onde efetivamente o sistema será implementado e como estarão distribuídas essas tarefas. As unidades de processamento serão vistas como nodos da aplicação de controle. A Tabela 6.5 resume as principais tarefas computacionais, as informações processadas por cada uma e sua localização no sistema de controle adotado no SIAMES.

Tarefa	**Período**	**Descrição**	**Localização da execução**
τ_1	T=10ms	Cálculo da velocidade atual do veículo; Cálculo do sinal de controle de velocidade; Envio de mensagens CAN com o valor de velocidade atual.	AVR
τ_2	T=1ms	Leitura do sensor lateral; Cálculo da distância percorrida.	AVR
τ_3	Aperiódica	Aguardar a aceitação da vaga encontrada.	PowerPC
τ_4	T=10ms	Recebimento de mensagens CAN com o valor de velocidade atual.	PowerPC
τ_5	Aperiódica	Cálculo da nova ação de controle para seguimento de trajetória; Envio de mensagem CAN com o novo ângulo a ser aplicado às rodas dianteiras.	PowerPC
τ_6	T=1ms	Leitura do sensor traseiro.	AVR
τ_7	T=10ms	Recebimento de mensagens CAN com o ângulo a ser aplicado às rodas dianteiras; Atuação nas rodas dianteiras.	AVR

Tabela 6.5: Tarefas executadas no sistema de estacionamento.

Com todas essas definições, utilizou-se a ferramenta *True Time* (OHLIN; HENRIKSSON; CERVIN, 2007), que é um simulador baseado em Matlab/Simulink, para auxiliar na determinação de qual o meio a ser utilizado para transmitir dados, uma vez que o processamento apresenta-se descentralizado. A Figura 6.12 mostra a modelagem resultante, onde são identificados os processadores responsáveis pelo sistema de controle de velocidade e o controle de trajetória. Nessa representação, estão também presentes as mensagens trocadas entre os nodos e os sinais enviados aos subsistemas do veículo, onde foi utilizada uma rede CAN para essa troca de dados.

1. AVR: Sensoriamento e o processamento do controle de velocidade;

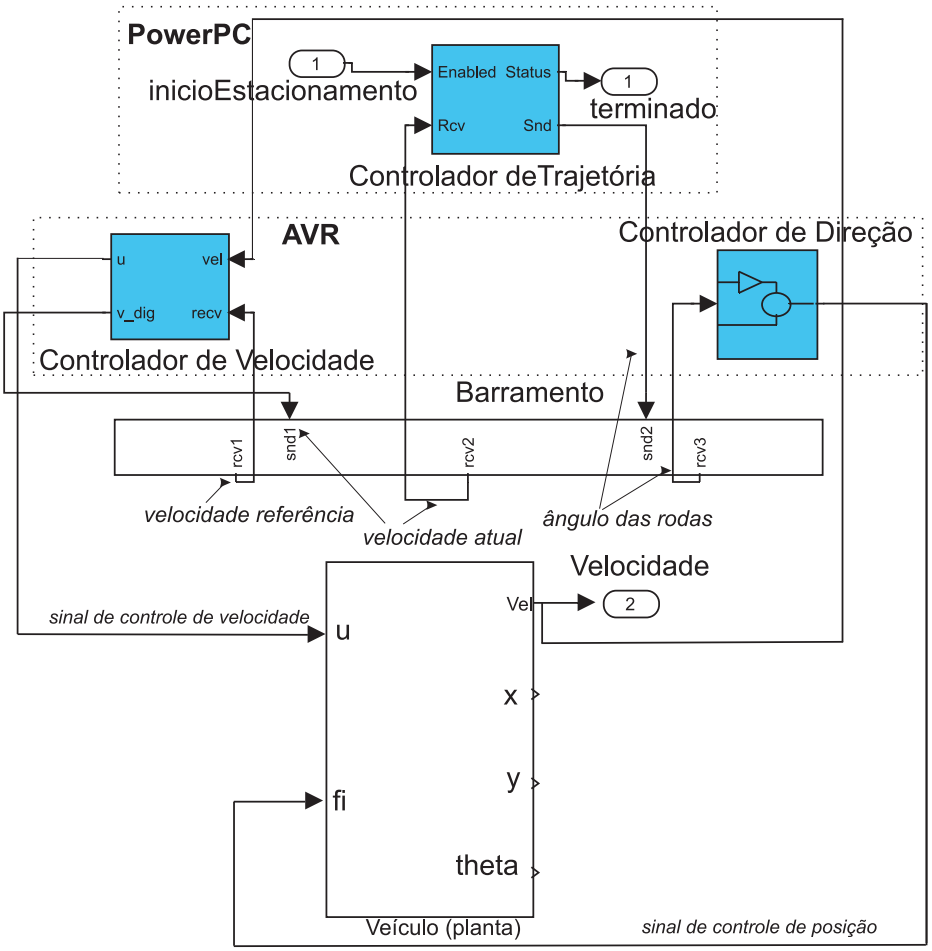


Figura 6.12: Representação da arquitetura do sistema realizado com o True Time

- Identificação da velocidade do veículo;
- Sinal de atuação nas rodas dianteiras.
- Mensagem de envio : velocidade atual do veículo;
- Mensagem de recebimento : ângulo ϕ da rodas dianteiras;

2. PowerPC: planejamento e controle de trajetória;

- Mensagem de envio : ângulo ϕ da rodas dianteiras;
- Mensagem de recebimento : velocidade atual do veículo;

Observa-se pela figura que o *True Time* permitiu que o projeto modelado representasse fielmente o que passaria a ser implementado no veículo protótipo mostrado na seção 6.2.1. Esse modelo mostrado vem substituir o modelo contínuo elaborado mostrado na Figura 6.9 e que passará a interagir com o modelo discreto.

6.4.1 Simulação e Análise

Com o ambiente do *True Time* mostrado anteriormente, integrou-se os blocos do modelo discreto, que representa a lógica de funcionamento, em (*Stateflow*) e mostrados na Seção 6.3.1 com o novo modelo contínuo do sistema, mostrado Figura 6.5. A simulação é então analisada a partir da integração dos dois modelos, ou seja, enquanto o veículo está procurando por uma vaga apenas o controle de velocidade é acionado. O a etapa de estacionamento é iniciada quando o sistema recebe uma sinalização (*inicioEstacionamento*) indicando que o usuário deseja estacionar o veículo na vaga encontrada. Nesse momento, uma mensagem é enviada para o modelo contínuo, onde então dá-se início à troca de mensagens via rede CAN entre os blocos *Controlador de Velocidade*, *Controlador de Direção* e *Controlador de Trajetória* que controla os movimentos do veículo (planta).

Nesse conjunto de ferramentas ainda é possível gerar um relatório onde são verificadas todas as transições ativadas e estados em que o sistema esteve. Com o *True Time* especificamente, é possível realiza várias alterações na estrutura do controle, avaliando a eficiência do projeto com diferentes protocolos de rede, como *ethernet* ou simulando mais nodos de controle.

Antecedendo a etapa de implementação do sistema, além da integração dos modelos do sistema com os projetos dos controladores, é relevante obter informações, durante a simulação, sobre o código elaborado. Através de ferramentas como *MatlabProfile* e *LightSpeed* (MINKA, 2006) foi possível identificar quais funções implementadas demandam maior carga computacional, quantas operações em ponto flutuante (*flops*) são executadas por uma determinada função, quantas vezes e por quem uma determinada função é chamada. Todas essas informações constituem dados relevantes para as etapas posteriores do desenvolvimento, como o projeto da arquitetura e implementação final. O anexo B mostra uma saída do *MatlabProfile* para o código da trajetória de estacionamento, bem como detalhes desse código.

6.5 Implementação do Sistema

As seções anteriores exploraram os modelos computacionais que constituem o SIAMES, ou seja, foi descrito o comportamento do sistema e o que o ele se propõe a fazer. Nessa secção, procura-se descrever o processo de implementação ou detalhar o "como fazer" aquilo que foi proposto. Essa etapa corresponde à tradução dos modelos computacionais desenvolvidos até então.

O processo de implementação é composto basicamente de três etapas:

- **Etapa1 :** Transcrição dos controladores projetados em Matlab/Simulink para a linguagem a ser utilizada no projeto. No SIAMES utilizou-se a liguagem C e os resultados foram validados através de gráficos gerados com os dados obtidos a partir da execução dos códigos nas próprias plataformas.

- **Etapa2 :** Utilizando a API do sistema operacional escolhido juntamente com as informações do comportamento dinâmico do sistema, agrupa-se o código elaborado em tarefas, onde são estabelecidos seus períodos e instantes de ativação. Além disso, identifica-se os dados a serem manipulados e os mecanismos de sincronização para acessá-los, além das mensagens trocadas.

- **Etapa3 :** Por fim, identifica-se as mensagens trocadas entre os níveis de controle e estabelece-se os pontos no código onde ocorrerá o envio dessas mensagens, ou seja, os instantes em que inicia-se a preparação do pacote de dados e o processo de envio.

As etapas destacadas foram utilizadas durante a implementação do sistema em cada nível de controle, conforme discutido a seguir:

6.5.1 Controle dos Subsistemas

- **Etapa1 :** Com o projeto do controlador de velocidade PI apresentado no cap. 3, realizado no domínio do tempo em ambiente Matlab/Simulink, obtém-se sua representação digital a partir da discretização da função de transferência com o período de amostragem adotado de acordo com as características da planta. Essa representação digital é que será executada pelo processador, conforme fora planejado para a arquitetura. Para o motor de tração utilizado no SIAMES, uma taxa de $10ms$ mostrou resultados satisfatórios quanto ao desempenho do controle.

  Para obter uma precisão que atendesse às necessidades da aplicação, além de um torque que garantisse a posição das rodas durante a manobra, utilizou-se servo-motores para o ajuste do ângulo das rodas. Nesse caso, a simples determinação da relação entre a largura de pulso do sinal de saída PWM aplicado ao motor das rodas e o ângulo realizado pelo mesmo permite que seja utilizado esse sinal para o ajuste ideal do posicionamento das rodas dianteiras. Esse controle também é realizado pelo AVR e aplicado ao veículo, como mostra a arquitetura proposta..

- **Etapa2 :** Até esse momento no entanto os códigos escritos estão isolados e sem aparente

interação entre si. A partir da modelagem discreta e das variáveis associadas a cada estado do sistema, conforme resumido na Tabela 6.2, é possível identificar algumas tarefas chave do processo e a informações manipuladas por elas.

Conforme pode ser constatado tanto na Tabela 6.2 como pelo diagrama da Figura 6.8, o monitoramento da velocidade do veículo é realizado em todos os momentos em que o mesmo está em movimento, ou seja *Varrendo Ambiente e Estacionando*. Sendo assim, ao controle de velocidade é atribuída uma tarefa periódica de alta prioridade cujo período é determinado de acordo com o valor utilizado durante a discretização do controlador de velocidade ($10ms$). Nessa mesma tarefa, realiza-se a leitura do contador de pulsos ligado ao *encoder* da roda para determinação da velocidade atual (realimentação) conforme descrito na seção 6.2.1.

Uma das atribuições dada ao sistema consiste na identificação da vaga adequada à manobra, quando ele recebe a indicação do usuário. Para tanto, duas informações são fundamentais: velocidade atual do veículo e detecção da presença de obstáculo (veículos laterais). O valor de velocidade pode ser obtido a partir do cálculo realizado na tarefa descrita anteriormente, sendo portanto necessário tratar a velocidade como uma variável global. Com base na velocidade do veículo, uma tarefa periódica específica, ativada quando o sistema está em *Varrendo Ambiente*, realiza o cálculo da distância percorrida enquanto lê os sensores laterais de identificação de obstáculos. Para evitar que o veículo percorra distâncias maiores quando a vaga for encontrada, essa tarefa possui um período menor de $1ms$ fazendo o veículo parar imediatamente após o encontro da vaga.

- **Etapa3 :** Com a vaga encontrada e o veículo pronto para iniciar a manobra, dá-se início a interação entre os dois níveis de controle. A partir desse momento, uma mensagem CAN é enviada ao controle de trajetória indicando início do estacionamento propriamente dito. Como o controle realizado no PowerPC (alto nível) também depende do valor de velocidade do veículo, decidiu-se por enviar o valor de velocidade logo após o seu cálculo na tarefa de controle de velocidade.

  Para o ajuste do ângulo das rodas criou-se uma nova tarefa responsável pelo recebimento de mensagens advindas do controle de trajetória com no novo ângulo das rodas. Com base nesses valores, calcula-se então a nova largura de pulso do sinal aplicado aos servo-motores.

A utilização do FreeRTOS, sistema operacional escolhido para rodar no AVR, auxiliou sobremaneira a organização do código em tarefas de tempo real, além de facilitar o acesso aos recursos de *hardware* do processador, como o uso da interface CAN. O trecho do código 6.1 ilustra como a utilização de um SO reduz significativamente o tempo de projeto tanto para sua elaboração como para correção de erros.

Código 6.1: Envio de velocidade pela interface CAN

```
//Enviando Velocidade atual
        packet.id == 0x102 // id do pacote com valores de velocidade
    fracpart = modf(new_velocity[0],&intpart);
    data[0] = (int)intpart;
    data[1] = (int)(fracpart*100);//montagem do pacote de dados
    if(intpart<0)
        data[2] = 1;
    else
        data[2] = 0;
    can_send(0x102,3,data,13,1000);//envio das informações
```

6.5.2 Controle de Trajetória e Interface com Usuário

- **Etapa1 :** Para o controle de trajetória, há a necessidade de transcrever em código a modelagem do veículo, ou seja, escrever as equações que descrevem o movimento do veículo dado um ângulo para suas rodas dianteiras. Além disso, como o GPC necessita de valores futuros da trajetória a ser seguida, o equacionamento elaborado em Matlab/Simulink para realizar a trajetória em "S", mostrado no cap. 3, também deve ser transcrito para a linguagem de projeto adotada.

  Com as equações que descrevem o movimento do veículo, é possível determinar a resposta livre desse sistema quando aplicado a ele um sinal de controle. É a comparação entre essa resposta livre e a trajetória de referência "S" elaborada que gera o sinal de erro a ser aplicado ao GPC discretizado. Assim, determina-se a seqüência com que ocorrem os eventos na implementação do controle executado no PowerPC de acordo com a arquitetura proposta.

  Nesse nível, também são codificadas as informações trocas com o usuário através da interface do sistema e que determinarão os instantes de transição de estados durante a manobra.

- **Etapa2 :** Para realizar o agrupamento em tarefas de tempo real nesse nível de controle, pode-se partir da distinção das principais atividades a serem realizadas nesse nível. Para a função de controle, decidiu-se separar a atividade de recebimento de mensagens CAN da execução do GPC em tarefas distintas. Desse modo, após ser avisado sobre o início de manobra ou quando recebe um novo valor de velocidade, a tarefa periódica ($10ms$) que recebe a mensagem CAN "acorda" aquela que calcula a nova ação de controle e envia o novo valor via CAN para o baixo nível.

- **Etapa3 :** Nesse nível de controle optou-se por ter uma tarefa específica para recebimento de mensagens CAN. Já o novo valor de ângulo a ser aplicado nas rodas é enviado logo após ter sido calculado pela tarefa que executa o GPC.

Do mesmo modo que o realizado anteriormente, no AVR, fez-se uso aqui de um sistema operacional de tempo real que não só facilitasse o desenvolvimento da solução de controle mas que permitisse o desenvolvimento futuro da interface de comunicação com o usuário de maneira otimizada. O Linux/Xenomai, permite rodar soluções que exigem certo grau de determinismo, usufruindo de um ambiente amigável de desenvolvimento. Para ilustrar as vantagens do uso de um SO desse porte, o código 6.2 ilustra a configuração de uma rede CAN utilizando o RT_SocketCAN disponível com o Xenomai.

Código 6.2: Configuração da rede CAN com o RT_SocketCAN

```
//configuracão da interface CAN
    mlockall(MCL_CURRENT | MCL_FUTURE);
    ret = rt_dev_socket(PF_CAN, SOCK_RAW, CAN_RAW);
        if (ret < 0) {
            fprintf(stderr, "rt_dev_socket: %s\n", strerror(-ret));
            return -1;
        }
    strncpy(ifr.ifr_name, "rtcan0", IFNAMSIZ);
    ret = rt_dev_ioctl(s, SIOCGIFINDEX, &ifr);
    memset(&to_addr, 0, sizeof(to_addr));
     to_addr.can_ifindex = ifr.ifr_ifindex;

    ret = rt_dev_bind(s, (struct sockaddr *)&to_addr, sizeof(to_addr));
```

Análise A análise da solução desenvolvida pode ser realizada avaliando a arquitetura proposta, a distribuição de tarefas nos diferentes nodos, a influência da rede de comunicação na transmissão de dados, ou ainda na eficiência do código elaborado.

Com o *True Time* todos os aspectos referentes à arquitetura do sistema adotada, conforme Figura 6.12, poder avaliada, ou seja, nesse ambiente podem ser utilizados protocolos diferentes, diferente distribuição de tarefas entre os nodos e avaliar o efeito de sobrecarga na rede. Quanto aos código elaborados, de modo parecido ao MatlabProfile, o *GNU profile* (OSIER, 1993) é uma ferramenta valiosa na análise de códigos C quando executados na plataforma alvo. O anexo C mostra um exemplo de arquivo gerado com o *GNU profile* para o código de controle de trajetória, executado no PowerPC, onde pode-se fazer o levantamento do tempo computacional gasto por cada função utilizada na solução.

Os tempos computacionais das principais tarefas de controle são mostrados na Tabela 6.6. Observa-se que esses valores de tempo são bastante inferiores ao período de amostragem da velocidade de $10ms$, o que causa uma pequena ocupação nas respectivas CPU e atesta a sua eficiência. Além disso, esses valores puderam então ser retornados à arquitetura do sistema modelada com o *True Time* implicando numa configuração ainda mais realista do sistema.

Tarefas de controle	**Tempos de execução**
PID para controle de velocidade (τ_1)	$550\mu s$
Controle Preditivo para controle de trajetória (τ_5)	$700\mu s$

Tabela 6.6: Tempos de execução das tarefas de controle.

6.6 Resultados Finais

Sob o ponto de vista de *software*, o FreeRTOS mostrou-se uma excelente opção para aplicações multitarefas devido principalmente a sua simplicidade e fácil compreensão. O sistema ainda integra-se facilmente à ambientes de desenvolvimento, como o AVRStudio, o que permite a utilização de ferramentas de depuração e um rápido entendimento da dinâmica do SO.

Além disso, o código gerado requer pouca memória, o que é ideal para pequenos processadores. No SIAMES, todo o processo de estacionamento é realizado por cinco tarefas disparadas em instantes distintos, e todo o código ocupa aproximadamente 60Kb de memória RAM dos 128Kb disponíveis.

No PowerPC, conforme mencionado, a opção pelo Linux é motivada pela popularidade do SO e

de suas ferramentas, e objetiva uma maior disseminação das suas extensões de tempo real. Apesar da aplicação não ter mostrado requisitos temporais que justificassem a utilização de um sistema operacional de tempo real, os tempos levantados e mostrados no capítulo 4 refletem a possibilidade de utilização desse sistema em diversas situações e com diferentes exigências.

Os códigos de controle executados nessas plataformas pode ser utilizado para o seguimento de diversas trajetórias, bastando passar o equacionamento do trajeto desejado. Adaptações no código precisam ser constantemente realizadas quando são realizados testes em diferentes superfícies ou quando utiliza-se outro veículo.

Sob a óptica do *hardware*, a instrumentação do protótipo mostra-se ser uma etapa desafiadora, devido aos aspectos construtivos do veículo e a disponibilidade de sensores no mercado. O espaço disponível para instrumentação, o pequeno ângulo de esterçamento das rodas e seu alinhamento são os maiores problemas a serem trabalhados no protótipo. O *encoder* mostrado na Figura 6.9 enquadra-se perfeitamente aos requisitos dimensionais, ou seja, pequeno suficiente para viabilizar instalação no protótipo, e operacionais, devido a sua alta resolução (mil pontos/volta) e, conseqüentemente, diminuição dos ruídos de medição. Apesar de terem sido utilizadas as próprias placas de desenvolvimento, em uma instalação comercial pode-se confeccionar unidades menores, onde estará presente apenas o que será efetivamente utilizado.

A rede CAN foi de extrema utilidade pois necessita de poucos cabos para ser estabelecida, sendo possível transferir dados com apenas um cabo para taxas menores que 125Kbps. Além disso, através do *sockets*, tornou-se fácil configurar e testar a rede CAN.

O protótipo mostrado na Figura 6.2 foi utilizado para realizar efetivamente a manobra de estacionamento, onde foram então colocados obstáculos representando os veículos delimitadores da vaga. O sistema comportou-se exatamente como mostrado nos gráficos gerados em simulação, executando perfeitamente a trajetória planejada. Por fim, foram efetuadas filmagens de modo a ratificar o que foi exposto.

6.7 Considerações Finais

Esse capítulo mostrou os detalhes do projeto SIAMES, seus requisitos, desafios e soluções adotadas. O mesmo permitiu mostrar o uso da metodologia de desenvolvimento apresentada no capítulo anterior.

Através do uso de uma metodologia, torna-se assegurar que eventuais erros não passem desapercebidos e que todos os requisitos sejam atendidos. Assim, cada etapa do projeto é detalhada e seus resultados são analisados.

Neste processo, a ferramenta Simulink mostrou ser muito valiosa, principalmente devido a possibilidade de integração de diferentes modelos e por ser especialmente adaptado aos sistemas de controle. As ferramentas *StateFlow* e *TrueTime* permitiram que todo o sistema fosse inicialmente modelado antes de sua implementação. Além disso, ferramentas como o *MatlabProfiler* (REUTHER, 2008) e o *LightSpeed* (MINKA, 2008) auxiliaram nas etapas de simulação e validação do projeto, pois permitiram, entre outras coisas, que fossem coletadas informações sobre os tempos de execução dos algoritmos projetados.

Por fim, todos esses passos permitiram que se chegasse ao final com uma versão operacional e estável do sistema de estacionamento proposto.

Capítulo 7

Conclusões e Trabalhos Futuros

Neste trabalho discutiu-se o processo de desenvolvimento de um sistema de controle tempo real embarcado, objetivando, entre outras coisas, aproximar a área de engenharia de controle e automação da área de ciência da computação. Através da definição de uma metodologia de projeto, essa dissertação buscou sistematizar o desenvolvimento desse tipo de aplicação. Aos projetistas de controle, o conjunto de ferramentas apresentadas, comuns entre profissionais de computação, constituem-se de poderosas formas de análise e validação do sistema projetado. Aos desenvolvedores de *software*, os desafios comumente encontrados nos projetos de controle, discutidos ao longo do trabalho, podem servir de guia para o estabelecimento de estratégias mais adequadas de implementação. O trabalho procurou focar, principalmente na fase de implementação, no uso de ferramentas computacionais gratuitas como os sistemas operacionais FreeRTOS e Linux/Xenomai. Na etapa de projeto do sistema, algumas atividades, como o projeto dos controladores, realizadas aqui em Matlab/Simulink podem ser refeitas no ambiente gratuito do Scilab/Scicos com excelentes resultados.

A indústria, em particular, pode fazer uso das ferramentas de desenvolvimento gratuitas apresentadas, o que reduz o custo e o tempo de projeto. Nesse sentido, o amadurecimento quanto ao uso de tais instrumentos no meio acadêmico permite o seu melhoramento, sua abrangência mas, principalmente, aumenta a confiança nessas soluções. Como resultado, atinge-se uma maior aproximação entre o setor produtivo e a pesquisa acadêmica.

Tendo como base a plataforma apresentada, abre-se a possibilidade de desenvolvimento de um número de aplicações como os sistemas de direção elétrica e as interfaces de usuário com visão do

ambiente, por exemplo. No âmbito dos sistemas controle, diferentes estratégias podem ser utilizadas com um número maior de processos a serem controlados. Quanto à metodologia, pode-se discutir a necessidade de diferentes atividades, como a verificação formal, nas etapas mencionadas, associadas às diferentes linguagens e ferramentas como Esterel, Lustre e Ptolemy, onde serão então encontradas facilidades e limitações a essa aplicação.

Como proposta de trabalho futuro, sugere-se equipar o sistema com eletrônica que permita explorar as técnicas de *Dynamic Voltage Scheduling* para redução do consumo de energia. Para a etapa de implementação do sistema, seria válido utilizar as ferramentas de geração de código automática de *Code Generator* e comparar o desempenho em relação a geração manual. Sob o ponto de vista das redes de comunicação, outra importante contribuição pode ser realizada com a utilização da rede CAN para envios de mensagens disparados por tempo, como o mencionado TTCAN ou ainda utilizar outros protocolos como o RTNET, com o objetivo de avaliar a adequação a sistemas de tempo real e facilidade de implementação.

O estudo de caso mostra claramente que a utilização de uma metodologia específica aliada a linguagens e ferramentas adequadas fornece uma maior confiabilidade ao projeto dos sistemas de controle embarcado, facilitando a identificação de erros e evitando futuros retrabalhos. No que tange ao protótipo montado, é possível adicionar uma maior quantidade de sensores ao veículo, capacitando-o a deslocar-se num meio evitando colisões. Ou, em casos ainda mais interessante, instrumentar o veículo com câmeras para mapeamento do ambiente e o comando remoto por parte do usuário.

Apêndice A

Sistema Autônomo de Estacionamento

A.1 Análise de Requisitos

F1 Início e Parada do Sistema				**Oculto()**
Descrição: O sistema deve ser explicitamente iniciado pelo usuário para entrar no modo de operação. Quando desligado, o sistema deve parar o veículo.				
Requisitos Não-Funcionais				
Nome	Restrição	Categoria	Tempo Real	Permanente
NFR1.1- Velocidade Máxima	Para dar início ao sistema, a velocidade deve ser mantida $\leq$ 20km/h.	Segurança	*hard real time*	(X)
NFR1.2- Aviso Sonoro	O sistema deve emitir um som quando estiver operando.	Interface	*soft real-time*	(X)
NFR1.3- Aviso Sonoro	O sistema deve emitir um som mais forte quando finalizar a operação.	Interface	*soft real-time*	(X)

F2 Busca por uma Vaga *Real-Time Operation*				**Oculto()**
Descrição: Ao ser inicializado, o sistema deve iniciar a busca por uma vaga ideal considerando as dimensões do veículo, enquanto o mesmo move-se para frente.				
Requisitos Não-Funcionais				
Nome	Restrição	Categoria	Tempo Real	Permanente
NFR2.1- Dimensões do veículo	O sistema deve ser adaptável a veículos de dimensões diferentes.	Implementação	—	(X)
NFR2.2- Velocidade máxima	O sistema deve avisar ao motorista que a velocidade máxima permitida foi ultrapassada.	Segurança e Interface	*hard real-time*	(X)
NFR2.3- Aviso ao Motorista	O sistema deve avisar ao motorista quando a vaga for encontrada	Interface	*soft real-time*	(X)

F3 Estacionamento Autônomo - *Real-Time Operation*			**Oculto()**	
Descrição: O motorista só pode iniciar a manobra quando um vaga tiver sido encontrada. Durante a manobra o sistema controlará a direção e opcionalmente a velocidade.				
Requisitos Não-Funcionais				
Nome	Restrição	Categoria	Tempo Real	Permanente
NFR3.1- Velocidade Máxima	O sistema só pode começar a manobra se a velocidade inicial for igual a zero.	Segurança	—	(X)
NFR3.2- Distância Máxima	A distância entre o veículo e a vaga encontrada não pode ser maior que 20m.	Implementação	—	(X)
NFR3.3- Parada com Intervenção do motorista	O sistema de parar caso o motorista mexa o volante.	Segurança	*hard real-time*	(X)
NFR3.4- Aviso ao motorista	O sistema deve avisar ao motorista que a manobra foi finalizada.	Interface	*soft real-time*	(X)

F4 Geração da trajetória de referência			**Oculto(x)**	
Descrição: Com a manobra ativada (F3), o sistema deve gerar a trajetória a ser seguida para o estacionamento.				
Requisitos Não-Funcionais				
Nome	Restrição	Categoria	Tempo Real	Permanente
NFR4.1- Coordenadas Globais	A trajetória deve ser definida em termos de coordenadas globais.	Implementação	—	(X)
NFR4.2 - Tempo Limite	Essa operação não deve levar mais do que 2seg.	Desempenho	*hard real-time*	(X)

F5 Controle da direção do veículo			**Oculto(x)**	
Descrição: Com a manobra ativada (F3) e a trajetória gerada (F2), o sistema deve iniciar o controle de direção para manter o veículo na trajetória desejada.				
Requisitos Não-Funcionais				
Nome	Restrição	Categoria	Tempo Real	Permanente
NFR5.1- Máximo desvio	O desvio máximo tolerável é de 10cm.	Desempenho	—	(X)
NFR5.2 - Variação de velocidade	O sistema deve tolerar variações de velocidade.	Desempenho	—	(X)
NFR5.3 - Restrições temporais	O período e o *deadline* da função de controle deve ser ajustado de modo a atender NFR's 5.1 e 5.2.	Tempo	*hard real-time*	(X)
NFR5.4 - Parada de emergência	O sistema deve para imediatamente (*deadline* de 10ms) caso o motorista assuma o volante.	Segurança e Tempo	*hard real-time*	(X)
NFR5.5 - Velocidade Máxima	O sistema deve avisar ao motorista caso a velocidade esteja acima da permitida. Caso permaneça nesse estado por mais do que 2seg., o sistema deve entrar em parada de emergência.	Segurança	*hard real-time*	(X)

F6 Controle de Velocidade			**Oculto(x)**	
Descrição: Em veículos com motores elétricos é possível realizar o controle de velocidade enquanto realiza-se a manobra de estacionamento. Essa operação trabalha em cooperação com F5.				
Requisitos Não-Funcionais				
Nome	Restrição	Categoria	Tempo Real	Permanente
NFR6.1- Parada de Emergência	O sistema deve entrar em parada de emergência caso o freio ou acelerador sejam pressionados.	Segurança, Tempo	*hard real-time*	(X)
NFR6.2 - Tempo Limite	Essa operação não deve levar mais do que 2seg.	Segurança	*hard real-time*	(X)

Tabela A.1: Requisitos do sistema de estacionamento

A.2 Casos de Uso

Casos de Uso			
Nome	Atores	Descrição	Referências Cruzadas
Procurar uma vaga	Motorista, Sensores (distância, velocidade)	O motorista dispara o sistema mantendo-o sob controle e dentro de uma faixa de velocidade	F3
Estacionamento	Motorista, Sensores (distância, velocidade)	O sistema assume controle total do veículo, gera a trajetória de referência, e finalmente estaciona.	F1, F2, F4

Tabela A.2: Casos de Uso do sistema de estacionamento

Apêndice B

Arquivo Gerado pelo Matlab Profile

B.1 Profile aplicado ao código trajetória de estacionamento

This is a static copy of a profile report
Home
trajEstacionamento (1 call, 0.016 sec)
Generated 05-Dec-2008 09:46:14
M-function in file C:\MATLAB7\work\Carrov3\Carro\trajEstacionamento.m
[Copy to new window for comparing multiple runs]

Parents (calling functions)
No parent
Lines where the most time was spent

Line Number	Code	Calls	TotalTime	% Time	Time Plot
80	xref(k+1) = xref(k) + vref(k)*...	506	0.002 s	13.0%	
82	thetaref(k+1) = thetaref(k) + ...	506	0.002 s	10.2%	
63	vref(k)=vd_inic;	506	0.002 s	10.0%	
81	yref(k+1) = yref(k) + vref(k)*...	506	0.001 s	9.2%	
79	firef(k) = fi;	506	0.001 s	8.4%	
Other lines & overhead			0.008 s	49.3%	
Totals			0.016 s	100%	

Children (called functions)
No children
M-Lint results

Line number	Message
18	The value assigned here to variable 'dt' is never used
35	The value assigned here to variable 'dx' is never used
41	The value assigned here to variable 'Ty' is never used
42	The value assigned here to variable 'Tx' is never used
63	Array 'vref' is constructed using subscripting. Consider preallocating for speed
68	The value assigned here to variable 'vd' is never used
72	The value assigned here to variable 'vd' is never used
76	The value assigned here to variable 'vd' is never used

Coverage results
[Show coverage for parent directory]
Total lines in file 143
Non-code lines (comments, blank lines) 94
Code lines (lines that can run) 49
Code lines that did run 45
Code lines that did not run 4
Coverage (did run/can run) 91.84 %

B.1.1 Código da trajetória de estacionamento

Function details for trajEstacionamento file:///C:/MATLAB7/work/Carrov3/Carro/results1/file1.html

File listing

time	calls	line	
		1	%Criacao de Referencias para trajetoria de Estacionameto
		2	% by Treebeard, 2008
		3	
		4	%vd_inic = velocidade inicial
		5	%T = tempo de amostragem
		6	%cm = distancia da traseira ao centro de massa
		7	%ET = distancia da traseira ao eixo traseiro
		8	%ED = distancia da traseira ao eixo dianteiro
		9	%R = minimo raio de curvatura do veiculo
		10	%Lcar = comprimento do carro
		11	%Wcar = largura do carro
		12	%dy = distancia da lateral do veiculo ateh o limite horizontal da vaga (achar termo melhor =p)
		13	
		14	function [xref,yref,thetaref,firef,y,TFin]=trajEstacionamento(vd_inic,T,ET,cm,ED,R,Lcar,Wcar,ddy)
		15	
		16	%Calculo da vaga minima para nao haver choque
		17	
< 0.01	2	**18**	dt = cm-ET;
< 0.01	1	**19**	dd = ED-cm;
		20	
< 0.01	1	**21**	L = ((2*R*(Wcar)+(Lcar-cm)^2)^(1/2))+cm;
		22	
		23	
		24	%Centro do circulo 1
< 0.01	1	**25**	Cx1 = cm;
< 0.01	1	**26**	Cy1 = R;
		27	
		28	%Centro do circulo 2
< 0.01	1	**29**	Cy2 = Wcar+ddy-R;
		30	
< 0.01	1	**31**	l = ((2*R)^2-(Cy2-Cy1)^2)^(1/2);
< 0.01	1	**32**	Cx2 = l+cm;
		33	
		34	%distancia em que o centro de massa veiculo deve estar referente a L
< 0.01	1	**35**	dx = Cx2-cm-L;
		36	
		37	%Calculo do angulo alfa formado pelas duas circunferencias
< 0.01	1	**38**	alfa = atan((Cx2-Cx1)/(Cy1-Cy2));
		39	
		40	%Calculo do ponto de Giro
< 0.01	1	**41**	Ty = Cy2+R*cos(alfa);
< 0.01	1	**42**	Tx = Cx2-R*sin(alfa);
		43	
		44	%Calculo do tempo total necessario para percorrer a trajetoria com a velocidade dada
< 0.01	1	**45**	distanciaT1 = alfa*R;
< 0.01	1	**46**	tr1 = abs(distanciaT1/vd_inic);
< 0.01	1	**47**	T1 = ceil(tr1/T);
		48	
< 0.01	1	**49**	T2 = T1;
< 0.01	1	**50**	Tfinal = T1+T2;
		51	
		52	% Inicializacao:
< 0.01	1	**53**	fi =0;
< 0.01	1	**54**	xref(1) = Cx2;
< 0.01	1	**55**	yref(1) = Wcar+ddy;
		56	
< 0.01	1	**57**	thetaref(1) = 0;
< 0.01	1	**58**	y(1)=vd_inic;
< 0.01	1	**59**	firef(1) = atan(dd/R);
		60	
		61	

```
< 0.01      1    62      for k = 1:(Tfinal)
< 0.01    506    63          vref(k)=vd_inic;
< 0.01    506    64          y(k+1)=vref(k);
              65
< 0.01    506    66          if(k<=(Tfinal/2))
< 0.01    253    67              fi = -atan(dd/R);
< 0.01    253    68              vd = vd_inic;
< 0.01    253    69          end
< 0.01    506    70          if(k>(Tfinal/2))
< 0.01    253    71              fi = atan(dd/R);
< 0.01    253    72              vd = vd_inic;
< 0.01    253    73          end
< 0.01    506    74          if(k>Tfinal)
              75              fi = 0;
              76              vd = vd_inic; %mudar para vd_inic?
              77          end
< 0.01    506    78          vref(k)=vd_inic;
< 0.01    506    79          firef(k) = fi;
< 0.01    506    80          xref(k+1) = xref(k) + vref(k)*cos(thetaref(k))*T;
< 0.01    506    81          yref(k+1) = yref(k) + vref(k)*sin(thetaref(k))*T;
< 0.01    506    82          thetaref(k+1) = thetaref(k) + (vref(k)*sin(firef(k))*T)/dd;
```



```
< 0.01    506    83      end
          1      84      xref=xref-xref(1);
          1      85      yref=yref-yref(1);
< 0.01    1      86      TFin = size(xref);
          1      87  end
                 88
                 89
                 90
                 91  %%%%%       Graficos utilizados para depuracao
                 92  % % Vetores de tempo:
                 93  % tu = 0:T:(k-1)*T;
                 94  % tx = 0:T:(k-2)*T;
                 95  %
                 96  % % Graficos:
                 97  % figure('name','Estados x, y e theta','numbertitle','off');
                 98  % subplot(3,1,1);
                 99  % hold on; box on; grid on;
                100  % plot(tx,xref(1:k-1),'r');
                101  % legend('x_{ref}',1);
                102  % ylabel('x');
                103  % hold off;
                104  %
                105  % subplot(3,1,2);
                106  % hold on; box on; grid on;
                107  % plot(tx,yref(1:k-1),'r');
                108  % legend('y_{ref}',1);
                109  % ylabel('y');
                110  % hold off;
                111  %
                112  % subplot(3,1,3);
                113  % hold on; box on; grid on;
                114  % ;plot(tx,thetaref(1:k-1),'r');
                115  % legend('\theta_{ref}',1);
                116  % ylabel('\theta');
                117  % xlabel('tempo [s]');
                118  % hold off;
                119  %
                120  % figure('name','Controle fi','numbertitle','off');
                121  % hold on; box on; grid on;
                122  % plot(tu,firef(1:k),'r');
                123  % legend('fi',1);
                124  % ylabel('fi');
                125  % xlabel('tempo [s]');
```

```
126  % hold off;
127  %
128  % figure('name','Trajetoria XY','numbertitle','off');
129  % hold on; box on; grid on;
130  % plot(xref(1:k-1),yref(1:k-1),'r');
131  % legend('referencia',1);
132  % xlabel('x'); ylabel('y');
133  % %circunferencia 1
134  % rectangle('Position',[Cx1-R,Cy1-R,2*R,2*R],'Curvature',[1,1],'LineStyle',':','EdgeColor',[0 0 1])
135  % %circunferencia 2
136  % rectangle('Position',[Cx2-R,Cy2-R,2*R,2*R],'Curvature',[1,1],'LineStyle',':','EdgeColor',[0 0 1])
137  %
138  % if((Cx2+Lcar)<(L+Lcar+1))
139  %       axis([-(Lcar+1) (L+Lcar+1) Cy2 Cy1])
140  % else
141  %       axis([-(Lcar+1) (Cx2+Lcar) Cy2 Cy1])
142  % end
143  % hold off;
```

12/23/2008 7:1 9 PM

Figura B.1: Arquivo profile gerado pelo Matlab

Apêndice C

Exemplo de Arquivo Gerado pelo GNU Profile

C.1 Profile aplicado ao código de controle e supervisão de trajetória

```
                     Flat profile

       Each sample counts as 0.01 seconds.

\textbf{ \%      cumulative self       self     total}
\textbf{time      seconds  seconds     calls    ms/call ms/call  name}
19.44      0.07     0.07       51414    0.00     0.00  abs2local
19.44      0.14     0.07       1254     0.06     0.06  processFreeResponseOP
19.44      0.21     0.07       1254     0.06     0.18  purePursuit
16.67      0.27     0.06      25080     0.00     0.00  circ
11.11      0.31     0.04       1254     0.03     0.03  processCommand
 8.33      0.34     0.03      25080     0.00     0.00  local2abs
 2.78      0.35     0.01        8       1.25     1.25  exportaArray
 2.78      0.36     0.01                              main
 0.00      0.36     0.00        2       0.00     0.00  exportaArrayxy
 0.00      0.36     0.00        1       0.00     0.00  choleskyDec
 0.00      0.36     0.00        1       0.00     0.00  choleskySol
 0.00      0.36     0.00        1       0.00     0.00  choleskySolLinSys
 0.00      0.36     0.00        1       0.00     0.00  mpcG
 0.00      0.36     0.00        1       0.00     0.00  mpcGG
 0.00      0.36     0.00        1       0.00     0.00  mpcK
 0.00      0.36     0.00        1       0.00     0.00  processControlLaw
 0.00      0.36     0.00        1       0.00     0.00  systemInitialize
```

```
  0.00      0.36     0.00        1       0.00     0.00  systemShutdown
  0.00      0.36     0.00        1       0.00     0.00  trOitoDir
```

```
\%  time     the percentage of the total running time of the
                 program used by this function.

cumulative  a running sum of the number of seconds accounted
seconds      for by this function and those listed above it.

self          the number of seconds accounted for by this
seconds   function alone.  This is the major sort for this
                listing.
                calls        the number of times this function was invoked, if
               this function is profiled, else blank.

 self         the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
                   else blank.

 total       the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
                   function is profiled, else blank.

name      the name of the function.  This is the minor sort
               for this listing.
```

```
granularity: each sample hit covers 4 byte(s) for 2.78\% of 0.36 seconds

index \% time    self  children    called        name
                                                 <spontaneous>
[1]         100.0    0.01    0.35                 main [1]
                0.07    0.16    1254/1254        purePursuit [2]
                0.07    0.00    1254/1254        processFreeResponseOP [5]
                0.04    0.00    1254/1254        processCommand [6]
                0.01    0.00       8/8           exportaArray [8]
                0.00    0.00    1254/51414       abs2local [4]
                0.00    0.00       2/2           exportaArrayxy [9]
                0.00    0.00       1/1           systemInitialize [17]
                0.00    0.00       1/1           processControlLaw [16]
                0.00    0.00       1/1           trOitoDir [19]
                0.00    0.00       1/1           systemShutdown [18]
-----------------------------------------------
                0.07    0.16    1254/1254        main [1]
```

```
[2]           63.4    0.07    0.16    1254         purePursuit [2]
                0.06    0.06   25080/25080       circ [3]
                0.03    0.00   25080/51414       abs2local [4]
-----------------------------------------------
                0.06    0.06   25080/25080       purePursuit [2]
[3]           34.5    0.06    0.06   25080         circ [3]
                0.03    0.00   25080/51414       abs2local [4]
                0.03    0.00   25080/25080       local2abs [7]
-----------------------------------------------
                0.00    0.00    1254/51414       main [1]
                0.03    0.00   25080/51414       purePursuit [2]
                0.03    0.00   25080/51414       circ [3]
[4]           19.4    0.07    0.00   51414         abs2local [4]
-----------------------------------------------
                0.07    0.00    1254/1254        main [1]
[5]           19.4    0.07    0.00    1254         processFreeResponseOP [5]
-----------------------------------------------
                0.04    0.00    1254/1254        main [1]
[6]           11.1    0.04    0.00    1254         processCommand [6]
-----------------------------------------------
                0.03    0.00   25080/25080       circ [3]
[7]            8.3    0.03    0.00   25080         local2abs [7]
-----------------------------------------------
                0.01    0.00       8/8           main [1]
[8]            2.8    0.01    0.00       8         exportaArray [8]
-----------------------------------------------
                0.00    0.00       2/2           main [1]
[9]            0.0    0.00    0.00       2         exportaArrayxy [9]
-----------------------------------------------
```

```
This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index        A unique number given to each element of the table.
                  Index numbers are sorted numerically.
                  The index number is printed next to every function name so
                  it is easier to look up where the function in the table.
```

```
   \% time        This is the percentage of the `total' time that was spent
               in this function and its children.  Note that due to
               different viewpoints, functions excluded by options, etc,
               these numbers will NOT add up to 100%.

    self        This is the total amount of time spent in this function.

    children        This is the total amount of time propagated into this
               function by its children.

    called        This is the number of times the function was called.
               If the function called itself recursively, the number
               only includes non-recursive calls, and is followed by
               a `+' and the number of recursive calls.

    name        The name of the current function.  The index number is
               printed after it.  If the function is a member of a
               cycle, the cycle number is printed between the
               function's name and the index number.

For the function's parents, the fields have the following meanings:

    self        This is the amount of time that was propagated directly
               from the function into this parent.

    children        This is the amount of time that was propagated from
               the function's children into this parent.

    called        This is the number of times this parent called the
               function `/' the total number of times the function
               was called.  Recursive calls to the function are not
               included in the number after the `/'.

    name        This is the name of the parent.  The parent's index
               number is printed after it.  If the parent is a
               member of a cycle, the cycle number is printed between
               the name and the index number.

If the parents of the function cannot be determined, the word
`<spontaneous>' is printed in the `name' field, and all the other
fields are blank.
```

```
For the function's children, the fields have the following meanings:

    self        This is the amount of time that was propagated directly
                from the child into the function.

    children        This is the amount of time that was propagated from the
                child's children to the function.

    called        This is the number of times the function called
                this child '/' the total number of times the child
                was called.  Recursive calls by the child are not
                listed in the number after the '/'.

    name        This is the name of the child.  The child's index
                number is printed after it.  If the child is a
                member of a cycle, the cycle number is printed
                between the name and the index number.

If there are any cycles (circles) in the call graph, there is an
entry for the cycle-as-a-whole.  This entry shows who called the
cycle (as parents) and the members of the cycle (as children.)
The '+' recursive calls entry shows the number of function calls that
were internal to the cycle, and the calls entry for each member shows,
for that member, how many times it was called from other members of
the cycle.
```

Figura C.1: Exemplo de arquivo gerado pelo GNU Profile

Referências Bibliográficas

ABU-AYYAD, M.; DUBAY, R. Real-time comparison of a number of predictive controllers. *ISA Transactions*, v. 46, p. 411–418, 2007.

ADEOS. *ADEOS - Adaptative Domain Environment for Operating System*. 2004. Website. `http://home.gna.org/adeos/`. Acesso em: Setembro 2008.

ÅRZÉN, K.-E.; CERVIN, A. Control and embedded computing: Survey of research directions. In: *Proc. 16th IFAC World Congress*. Prague, Czech Republic: [s.n.], 2005.

ÅSTRöM, K. J.; HäNGGLUND, T. *PID Controllers: Theory, Design and Tuning*. [S.l.]: Instrument Society of America, 1995.

BARBALACE A. LUCHETTA, G. M. M. M. A. S. A.; TALIERCIO, C. Performance comparision of Vxworks, Linux, Rtai, and Xenomai in a Hard Real Time Application. In: *IEEE Transaction on Nuclear Science*. [S.l.: s.n.]. v. 55.

BARBALACE A. LUCHETTA, G. M. M. M. A. S. A.; TALIERCIO, C. Performance comparison of vxworks,linux,rtai, and xenomai in hard real-time application. *In:IEEE Trasactions on Nuclear Science*, v. 55, p. 435–439, 2008.

BARRAQUAND, J.; LATOMBE, J. On nonholonomic mobile robots and optimal maneuvering. In: . Albany, USA: [s.n.], 1989. p. 340–347.

BAZANELLA, A. S.; J. M. GOMES da SILVA Jr. Ajuste de controladores pid. 2003. Disponível em: <http://www.ece.ufrgs.br/ jmgomes/pid/Apostila/apostila/apostila.html>.

BOSCH, R. *CAN Specification*. Stuttgart, 1991.

Brockmeyer C. Kulkarni, A. V. P. G. K. P. R. P. F. C. N. D. D. K. D. . E. Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, ACM Press, New York, NY, USA, v. 6, n. 2, p. 149–206, 2001.

BRUYNINCKX, H.; K.U.LEUVEN. *Real-Time and Embedded Guide*. [S.l.], 2000.

BUCHER, R.; MANNORI, S.; NETTER, T. *RTAI-Lab tutorial: Scilab, Comedi, and real-time control*. [S.l.], 2008. Disponível em: <https://www.rtai.org/RTAILAB/RTAI-Lab-tutorial.pdf>.

CAMACHO, E. F.; BORDONS, C. *Model Predictive Control*. New York,USA: Springer-Verlag, 2007.

CERVIN, A. et al. The jitter margin and its application in the design of real-time control systems. In: *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*. Göteborg, Sweden: [s.n.], 2004. Best paper award.

CLARKE, D. W. Application of generalized predictive control to industrialprocesses. *Control Systems Magazine, IEEE*, v. 8, n. 32, p. 49–55, April 1988.

CORBET, J. Linux gets can support. 2007. Disponível em: <http://www.pcworld.idg.com.au/index.php/id;11338500>.

DSPACE. *MicroAutoBox*. 2008. Website. `http://www.dspace.de`.

EDWARDS, S. et al. *Design of embedded systems: formal models, validation, and synthesis*. [S.l.], 1997.

EVIDENCE. *Erika Enterprise - A modular and optimized RTOS for 8, 16, 32 bit microcontrollers*. 2008. Website. `http://www.evidence.eu.com/images/Documents/brochure_erika_enterprise_1_4.pdf`.

EVIDENCE. *RT-Druid Code generator Plugin reference manual*. 2008. Website. `http://www.evidence.eu.com/download/manuals/pdf/rtdruid\_refman\_1\_4\_9.pdf`.

FüHRER, T. et al. *Time triggered communication on CAN*. [S.l.], 2000. Disponível em: <http://www.can-cia.org/can/ttcan/fuehrer.pdf>.

FLEXRAY. *FlexRay: The Communication System for Advanced Automotive Control Application*. 2008. Website. `http://www.flexray.com`. Acesso em: 08 Setembro 2008.

FREERTOS. *FreeRTOS - The Standard Solution for Small Embedded Systems*. 2008. Website. `http://www.freertos.org/`.

FREESCALE. *Braking Systems*. 2008. Website. `http://www.freescale.com/webapp/sps/site/application.jsp?nodeId=02Wcbf07jSKy5P`.

FRöHLICH, A. A. *Application-Oriented Operating Systems*. Tese (Doutorado) — GMD-Forschungszentrum Informationstenchnik, Sankt Augustin, 2001.

FU, L.; SCHWEBEL, R. RT PREEMPT HOWTO. 2006. Disponível em: <http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO>. Acesso em: Novembro 2008.

GAGNE, A. S. B. G. *Operating System Concepts*. seventh. New Jersey,USA: John Wiley & Sons, Inc., 2004.

GERUM, P. *Xenomai - Implementing a RTOS Emulation Framework on GNU/Linux*. 1st. ed. [S.l.], April 2004.

GOMES, G. K. *Controle Preditivo em Tempo Real para Seguimento de Trajetória de Veículos Autônomos*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, mar. 2006.

HENRIKSSON, D. *Resource-Constrained Embedded Control and Computing Systems*. Tese (Doutorado) — Department of Automatic Control, Lund Institute of Technology, Sweden, jan. 2006.

HERMANN, N. Ein mathematisches modell zum parallelparken. nov. 2003.

HOLZER, M.; BELANOVIC', P.; KNERR, B. *Automatic Design Techniques for Embedded Systems*. 2005.

HOUPIS, J. J. D. e C. H. *Análise e projeto de sistemas de controle lineares*. Rio de Janeiro, RJ,Brasil: Editora Guanabara Dois S.A., 1981.

HOYLE, R. Requirements for a perfect s-shaped parallel parking manoeuvre in a simple mathematical model. may 2003.

JALOTE, P. *An Integrated Approach to Software Engineering*. New York, USA: Springer-Verlag, 1991. 375 p.

JANTSCH, A. Models of embedded computation. In: *Embedded Systems Handbook*. [S.l.]: CRC Press, 2005.

JANTSCH, A.; SANDER, I. Models of computation and languages for embedded system design. In: *Computers and Digital Techniques, IEEE Proceedings* -. [S.l.: s.n.], 2005. v. 152, p. 114–129.

JOHANSSON, K. H.; TÖRNGREN, M.; NIELSEN, L. Vehicle applications of controller area network. In: *Handbook of Networked and Embedded Control Systems*. [S.l.: s.n.], 2005. p. 741–766.

KISZKA, J. *The Real-Time Driver Model and First Applications*. [S.l.], 2007. Disponível em: <http://www.xenomai.org/documentation/branches/v2.3.x/pdf/RTDM-and-Applications.pdf>.

KOPETZ, H. *Real Time Systems: Principles for Distributed Applications*. Massachusetts, USA: Klumer, 1998. 338 p.

KRIAA, L. Heterogeneous systems modeling: Basic concepts. In: SPRINGER (Ed.). *Global Specification and Validation of Embedded Systems*. Grenoble, France: [s.n.], 2007. p. 5–28.

LAN, I.-V. *CAN Bus*. [S.l.], 2008.

LAVAGNO, L.; SANGIOVANNI-VINCENTELLI, A.; SENTOVICH, E. Models of computation for embedded system design. In: *in NATO ASI Proc. on System Synthesis*. [S.l.]: Kluwer Academic Publishers, 1998. p. 45–102.

LI, C. Y. Q. *Real-Time Concepts for Embedded Systems*. San Francisco, CA, USA: CPM Books, 2003. ISBN 1578201241.

LIN. *Local Interconnect Network*. [S.l.], 1999. Disponível em: <http://www.lin-subbus.org>. Acesso em: 08 Setembro 2008.

LINCOLN, B. Jitter compensation in digital control systems. In: *Proceedings of American Conference*. Anchorage, USA: [s.n.], 2002.

LINEO. *DIAPM RTAI Programming Guide*. [S.l.], September 2000.

MARTí, P.; FUERTES, J. M. Jitter compensation for real-time control systems. In: *in 22nd IEEE Real-Time Systems Symposium*. [S.l.: s.n.], 2001.

MASSA, M. B. A. *Programming Embedded Systems*. [S.l.]: O'Reilly, 2006. 304 p.

MELLICHAMP, D. E. S. T. F. E. D. A. *Process Dynamics and Control*. New Jersey, USA: John Wiley & Sons, Inc., 2004. 704 p.

MINKA, T. The lightspeed matlab toolbox. 2006. Disponível em: <http://research.microsoft.com/en-us/um/people/minka/software/lightspeed/>.

MINKA, T. *The Lightspeed Matlab toolbox*. 2008. Website. `http://research.microsoft.com/en-us/um/people/minka/software/lightspeed/`. Acesso em: 18 dezembro 2008.

MORARI, M.; LEE, J. H. Model predictive control: past, present and future. *Computers and Chemical Engineering*, v. 23, p. 667–682, 1999.

MURRAY, R.; SASTRY, S. Nonholonomic motion planning: steering using sinusoids. *In: IEEE Transactions on Automatic Control*, v. 38, p. 700–717, may 1993.

NIKOLAOU, M. Model predictive controllers: A critical synthesis of theory and industrial needs. In: *Advances in Chemical Engineering Series*. [S.l.: s.n.], 2001.

NORMEY-RICO, J. E. Introdução à análise e projeto de sistemas de controle lineares. 2005. Disponível em: <http://www.das.ufsc.br/disciplinas/das5121/>. Acesso em: Dezembro 2008.

OHLIN, M.; HENRIKSSON, D.; CERVIN, A. *TrueTime 1.5—Reference Manual*. [S.l.], 2007.

OSIER, J. *GNU gprof*. [S.l.], 1993. Disponível em: <http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html>.

PARET, C. E. G. D. M.; RAMARKER, B. L. *The Second Shell Process Control Workshop: Solutions to the Shell Standard Control Problem*. Oxford, UK: Butterworth-Heinemann, 1990. 696 p.

PARET, D. *Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire...* [S.l.]: John Wiley & Sons, Ltd, 2007.

PEREZ U. F. MORENO, C. B. M. D. A. Can networked control systems with remote controllers using jitter margin. In: *Proceedings of the IECON06, 2006*. Paris, France: [s.n.], 2006. p. 252–257.

QIN, S. J.; BADGWELL, T. A. An overview of industrial model predictive control technology. In: . [S.l.: s.n.], 1997. p. 232–256.

RAMMIG, F. J. (Ed.). *Distributed and Parallel Embedded Systems*. Distributed and parallel embedded systems. [S.l.]: Kluwer Academic Publishers, 1999.

REUTHER, H. K. *Profiling pMatlab and MatlabMPI Applications Using the MATLAB 7 Profiler*. 2008. Website. `http://www.ll.mit.edu/mission/isr/pmatlab/Profiling_pMatlab_MatlabMPI.pdf`. Acesso em: 18 dezembro 2008.

RTAI. *RTAI: Real-Time Application Interface*. 2006. Website. `https://www.rtai.org/`. Acesso em: Setembro 2008.

RTCAN. *Xenomai API*. [S.l.], 2008. Disponível em: <http://www.xenomai.org/documentation/branches/v2.4.x/html/api/index.html>. Acesso em: Outubro 2008.

SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. In: *Design and Test of Computers, IEEE*. [S.l.: s.n.], 2003. p. 23–33.

SANTOS, T. L. M. *UMA CONTRIBUIÇÃO AO DESENVOLVIMENTO DE SISTEMAS DE CONTROLE VIA REDES USANDO A MARGEM DE JITTER*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, mar. 2006.

SAROLAHTI, P. *Real Time Application Interface*. [S.l.], February 2001.

SCHLEEF, D.; HESS, F.; BRUYNINCKX, H. Comedi - the control and measurement device interface. 2007. Disponível em: <http://www.comedi.org/doc/>. Acesso em: Dezembro 2008.

SMITH, C. A.; CORRIPIO, A. *Principles and Pratice of Automatic Process Control*. New Jersey, USA: John Wiley & Sons, Ltd, 2006. 505 p.

Tanenbaum, A. S. *Structure Computer Organization*. New Jersey, USA: Prentice-Hall,Inc, 2000. 460 p.

TINDELL, K.; BURNS, A.; WELLINGS, A. Calculating controller area network (can) message response times. *Control Engineering Practice*, v. 3, p. 1163–1169, 1995.

VERBAUWHEDE, I.; SCHAUMONT, P. Skiing the embedded systems mountain. *ACM Transactions on Embedded Computing Systems*, v. 4, p. 529–548, 2005.

WANNER, L. F. *Suporte de Sistema Operacional para Redes de Sensores Sem Fio.* Dissertação (Mestrado) — Dissertação de Mestrado, Universidade Federal de Santa Catarina, Programa de Pós-Graduação em Ciência da Computação, Florianópolis, SC, Brasil, 2006.

XENOMAI. *Xenomai: Real-Time Framework for Linux*. 2007. Website. `http://www.xenomai.org`. Acesso em: Setembro 2008.

YAGHMOUR, K. *Building Embedded LINUX Systems*. Sebastopol,CA,USA: O'Reilly Media,Inc, 2003.