Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Sistema Tempo-Real Embarcado para Controle de um Robô Tele-Operado

Autor:

Vitor Paranhos de Oliveira Carneval

Orientador:

Prof. Flávio Luis de Mello, D. Sc.

Orientador:

Prof. Heraldo Luis Silveira de Almeida, D. Sc.

Examinador:

Prof. Marcelo Luiz Drumond Lanza, M. Sc.

DEL

Maio de 2011

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

AGRADECIMENTO

Agradeço imensamente aos meus pais, Ricardo Carneval e Maria da Glória Paranhos, por toda a confiança em mim depositada e por todo o investimento realizado em meus estudos.

Gostaria também de agradecer à minha namorada Techandra Sodré, por toda a compreensão nos momentos de ausência para a realização deste trabalho e pelo incondicional apoio em todas as minhas empreitadas.

Agradeço aos meus orientadores, Prof. Flávio Luis de Mello e Prof. Heraldo Luis Silveira de Almeida, por toda a atenção, dedicação e orientação.

Agradeço também a todos os colegas do Laboratório de Robótica do Centro de Pesquisas da PETROBRAS: Auderi Santos, Alexandre Seixas, José Almir de Sena, Rômulo Curty, Pedro Panta, Rodrigo Ferreira e Patrick Merz. Sem eles com certeza não seria possível realizar esse projeto.

Um agradecimento especial para os colegas, também do referido laboratório, Ney Robinson dos Reis pela oportunidade concedida de participar desse grupo e desse projeto, e Gustavo Freitas pela grande ajuda e por uma série de orientações e idéias, durante o projeto.

RESUMO

O presente trabalho apresenta o projeto de um sistema embarcado em tempo-real para controle de um robô móvel tele-operado. Será discutida sua arquitetura, as classes projetadas para o funcionamento, e as informações pertinentes para sua implementação. Serão também apresentados e discutidos resultados gerados através de testes com esse sistema.

Será apresentado também o sistema operacional de tempo-real Xenomai de maneira a viabilizar que tarefas periódicas de tempo-real possam ser realizadas, como por exemplo algoritmos de controle.

Será apresentada a estrutura de hardware do protótipo pequeno do Robô Ambiental Híbrido, para o qual esse sistema será projetado, e uma tarefa de controle a ser realizada no mesmo. Um algoritmo de controle proposto com essa finalidade será apresentado e implementado, gerando resultados que também serão discutidos.

Por fim serão avaliados todos os trabalhos futuros que podem ser desenvolvidos a partir do sucesso do presente trabalho.
Palavras-Chave: sistema embarcado, tempo-real, robô móvel, xenomai.

ABSTRACT

This dissertation presents the design of an embedded system in real-time to control a mobile tele-operated robot. The system architecture, the classes designed for the operation, and relevant information for its implementation will be discussed. The results generated by tests with this system will also be presented and discussed.

The real-time operating system Xenomai will be introduced in order to enable that periodic tasks of real-time can be attained by this system, such as control algorithms.

The dissertation will show the hardware structure of the small prototype of Environmental Hybrid Robot, for which this system will be designed, and a control task to be performed on it. A proposed control algorithm for this task will be presented and implemented, leading to results that will be discussed.

Finally will be assessed all future work that could be developed after the success of this work.

Key-words: embedded system , real-time, mobile robot, xenomai.

SIGLAS

ADEOS - Adaptive Domain Environment for Operating Systems

API - Application Programming Interface

CAN - Controller Area Network

CC - Corrente Contínua

DD - Dianteira Direita

DE - Dianteira Esquerda

IPC - Inter-Process Communication

HAL - Hardware Abstraction Layer

QoS - Quality of Service

RAH - Robô Ambiental Híbrido

RAHM - Robô Ambiental Híbrido Médio

RAHP - Robô Ambiental Híbrido Pequeno

RAHT - Robô Ambiental Híbrido Tripulado

RTDM - Real Time Driver Model

RTOS - Real Time Operating System

SSH - Secure Shell

TD - Traseira Direita

TE - Traseira Esquerda

UFRJ - Universidade Federal do Rio de Janeiro

USB - Universal Serial Bus

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Tema

O tema deste trabalho é o projeto de um sistema de softwares embarcados para controle em tempo-real para um robô móvel teleoperado.

1.2 Delimitação

Esse trabalho está delimitado ao projeto do sistema de controle para o protótipo pequeno do Robô Ambiental Híbrido, que será melhor descrito no capítulo 2. Desta maneira, todo o estudo apresentado levará em conta a estrutura do protótipo citado, embora toda a teoria e metodologia possa ser aproveitada para o desenvolvimento de sistemas de controle para qualquer robô móvel.

A delimitação se da também devido a estrutura de hardware instalada no citado protótipo, e no uso do sistema operacional Linux com o framework de tempo-real Xenomai para a implementação do sistema.

1.3 Justificativa

Robôs móveis são amplamente utilizados em terrenos perigosos, nos quais é complicado o acesso e a locomoção, para diversas aplicações, incluindo mineração, silvicultura, agricultura, e atividades militares [1].

Ferramentas desse tipo podem ser extremamente úteis para a sociedade, evitando que vidas humanas precisem ser colocadas em situações de risco. Robôs móveis podem por exemplo realizar tarefas em cenários de catástrofes, ajudando na execução de resgates e identificação das áreas atingidas.

Nos últimos meses vimos uma série de eventos como estes nos quais essas ferramentas poderiam ser de grande ajuda, como o temporal que atingiu a região metropolitana do estado do Rio de Janeiro em Abril de 2010 e a região serrana do mesmo estado em Janeiro de 2011.

Uma outra utilidade pode ser a de exploração de ambientes desconhecidos. Os chamados *Rovers*, citados em [1], tem o objetivo de explorar e coletar dados no planeta Marte. Recentemente um robô móvel foi utilizado, da mesma maneira, para explorar um túnel de 2000 anos com o propósito de estudos arqueológicos [2].

Como exemplificado existem diversas aplicações para robôs deste tipo. Para que esses robôs possam funcionar, de maneira a realizar as tarefas para as quais foram idealizados, é necessário que estes tenham sistemas de controle embarcados que possam comandar de maneira correta suas funcionalidades e controlar as partes mecânicas presentes nos mesmos.

Dessa maneira estudos e pesquisas envolvendo sistemas de controle para robôs móveis podem ser muito importantes e de grande relevância para a sociedade.

1.4 Objetivos

O objetivo deste trabalho é desenvolver um sistema embarcado de controle capaz de garantir algumas características de tempo-real para a execução de algoritmos de controle.

Desta forma tem-se como objetivos específicos: (1) receber comandos de um computador chamado de base, através do protocolo TCP/IP e conexão Ethernet; (2) Interpretar os comandos recebidos e se comunicar com os atuadores para realização dos mesmos; (3) Fazer leituras nos sensores de operação e de telemetria e enviar os

dados para o computador base, e; (4) Permitir a implementação de algoritmos de controle com características de tempo-real.

1.5 Metodologia

Este trabalho irá utilizar uma plataforma Debian Linux para, através do desenvolvimento de um sistema, proporcionar o controle de um robô tele-operado. Para o desenvolvimento desse sistema serão utilizadas as linguagens C e C++.

Será estudado o framework de tempo-real Xenomai, que permite ter no ambiente Linux o suporte a tarefas de tempo-real.

Uma arquitetura multi-processos será proposta para o sistema de forma que algum eventual problema em um dos processos não influencie nos outros. A nível prático isso significa que quando alguma funcionalidade do robô for comprometida, não necessariamente as outras também serão. Devido a essa característica se fará necessária a utilização de comunicação entre processos.

Para a comunicação com a base será utilizada uma rede Ethernet e o protocolo TCP/IP, implementado através de sockets.

Para se comunicar com os sensores e atuadores embarcados no robô serão utilizadas as portas de entrada e saída do PC104, do tipo USB e serial. Algumas bibliotecas de comunicação, de autoria dos fabricantes, serão utilizadas nesse processo. Será necessário também estabelecer um controle em tempo-real da porta serial, para comunicação com os hardwares atuadores dos motores de forma a permitir a execução de algoritmos de controle.

1.6 Descrição

No capítulo 2 será apresentado o projeto do Robô Ambiental Híbrido e toda a estrutura de um de seus protótipos, que será utilizada nesse trabalho.

O capítulo 3 apresenta a arquitetura do sistema de tempo-real Xenomai e o seu funcionamento.

Um controle proposto para os mecanismos de suspensão do protótipo pequeno do Robô Ambiental Híbrido será apresentado no capítulo 4. Nele serão explicitados detalhes e resultados da implementação. Esse controle será usado como prova de conceito para o sistema.

No capítulo 5 será apresentada a arquitetura proposta para o sistema de softwares que fará o controle do robô.

O capítulo 6 apresenta os resultados dos testes propostos para o funcionamento do sistema de controle.

Por fim, no capítulo 7, são apresentas as conclusões desse trabalho, e os trabalhos futuros que podem decorrer do mesmo.

Capítulo 2

Robô Ambiental Híbrido

2.1 O Projeto

A Floresta Amazônica é a maior floresta tropical e contém a maior biodiversidade do mundo. É formada basicamente por matas de terra firme, matas de várzea alagadas pelos rios de água barrenta na estação das cheias, e matas de igapós inundadas quase permanentemente por rios de água preta. A região possui vegetação nativa, com alta densidade vegetal, e clima constituído de altas temperaturas e umidade [3].

A chamada Província Petrolífera de Urucu foi instalada a cerca de 20 anos, marcando a presença da Petrobrás nesse valioso ecossistema. A Unidade de Produção de Gás Natural 3 (UPGN3), localizada em Urucu, responsável pela produção de 6 milhões de m^3 por dia, é hoje a maior unidade de processamento de gás natural do Brasil.

A fim de obter um melhor escoamento do Gás Natural a Petrobrás construiu o gasoduto Coari-Manaus, ao longo das margens do Rio Solimões. O gasoduto tem uma extensão de cerca de 400Km, passando por regiões de floresta, áreas alagáveis, além de sete municípios.

Ficou acordado entre a Petrobrás e os órgãos de regulação ambiental nacionais que para o licenciamento da operação do gasoduto a empresa petrolífera deveria fazer um monitoramento ambiental, completo e detalhado, da região.

A Petrobrás firmou então diversas parcerias com instituições acadêmicas, como UFAM e COPPE/UFRJ, participando de projetos como o PIATAN, um projeto de pesquisa socioambiental criado para monitorar as atividades de produção e transporte de petróleo e gás natural oriundos de Urucu [4], e o COGNITUS, que visa a aplicação de ferramentas cognitivas para a gestão ambiental na Amazônia [5].

O projeto COGNITUS tem como uma de suas linhas de pesquisa a chamada AmazonBOTS. A referida linha tem como objetivo o desenvolvimento de sistemas robóticos com aplicação para monitoração ativas de lagos amazônicos, visando a construção de séries históricas de parâmetros limnológicos considerados relevantes.

Foi nesse contexto que o Laboratório de Robótica do CENPES criou o projeto Robô Ambiental Híbrido Chico Mendes. Esse projeto é de um veículo robótico, desenvolvido para uma locomoção otimizada nos ambientes amazônicos, com o objetivo de fazer telemetria de parâmetros ambientais e televisão dos ambientes percorridos pelo gasoduto Coari-Manaus. Um dos protótipos do robô é apresentado na figura 2.1.

Figura 2.1: Protótipo Médio em testes realizados na Amazônia.

2.2 O Robô

O RAH (Robô Ambiental Híbrido) foi concebido como uma ferramenta robótica para ambientes inóspitos, onde o homem não deve chegar, ou onde existam grandes riscos pela presença humana, de forma a auxiliar nas pesquisas na região amazônica e na colheita de parâmetros e amostras para o projeto COGNITUS.

Sob esta ótica, para solucionar as dificuldades em percorrer a região foi desenvolvido um novo conceito em estrutura de locomoção. São partes constituintes desse novo conceito as rodas, que também funcionam como flutuadores nas regiões alagadas, e as suspensões ativas e independentes, capazes de alterar o ângulo de cambagem das rodas, e a altura do robô em relação ao solo, com o fim de variar o atrito com o terreno e permitir a transposição de obstáculos.

Os modos de operação idealizados para esse Robô foram tanto a tele-operação quanto a operação por um tripulante no interior do veículo, isolado do ambiente externo.

O primeiro protótipo do projeto, apresentado na figura 2.2, tinha suas rodas confeccionadas em isopor, conforme descrito em [6], e seu objetivo era de testar o conceito de movimentação do projeto.

Figura 2.2: Primeiro protótipo do Robô Ambiental Híbrido.

O segundo protótipo do projeto, presente na figura 2.1, foi identificado como Robô Ambiental Híbrido Médio, RAHM, e possui todas as funcionalidades idealizadas inicialmente no projeto. Ele é capaz de realizar medições e pequenas tarefas, conforme descrito em [3]. Por se tratar de um protótipo maior do que o primeiro protótipo citado anteriormente, é possível que nele seja embarcado uma quantidade maior de equipamentos eletrônicos.

2.3 Novo Protótipo Pequeno

O RAHP, Robô Ambiental Híbrido Pequeno, objeto de estudo desse trabalho, é o terceiro protótipo vinculado ao projeto do RAH. Seu modelo 3D é apresentado na figura 2.3.

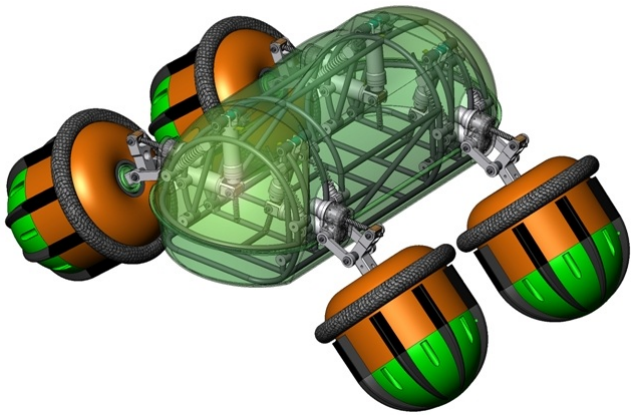

Figura 2.3: Modelo em 3D do Protótipo Pequeno.

O robô pesa 35Kg, tem a distância entre os eixos de 490mm e largura de faixa de 710mm.

2.3.1 Estrutura de Suspensão

No decorrer do desenvolvimento do projeto, viu-se que era necessário ampliar a movimentação da suspensão do RAH. A nova estrutura de suspensão, presente pela primeira vez no protótipo RAHP, permite uma melhor movimentação vertical da roda, mantendo o ângulo de cambagem igual a zero. Isso permite ao robô alterar

sua altura em relação ao solo sem ter que obrigatoriamente alterar o ângulo de cambagem das rodas, o que não era possível no RAHM.

Dessa forma cada roda é acoplada a um sistema de suspensão independente, aqui referidas como perna, ilustrada na figura 2.4, composta por uma mola e dois atuadores elétricos. Os motores das suspensões são acoplados a fusos, compondo as juntas prismáticas ativas.

Figura 2.4: Suspensão do RAHP com cambagem e altura ajustável.

O mecanismo paralelo de suspensão foi projetado para rigidez da estrutura, permitindo ao robô transpor obstáculos, e aumentar a tração das rodas. O mecanismo permite também que o robô opere de cabeça para baixo. Na figura 2.5 diferentes configurações da suspensão são apresentadas.

Os motores ligados aos mecanismos de suspensão possuem *encoder*s instalados. Definindo uma posição como sendo a posição zero para aquela junta do mecanismo, quando o motor se movimenta, através da contagem de pontos do *encoder*, tanto no sentido positivo quanto no negativo, é possível saber a posição da junta do mecanismo.

Sabendo o número de pontos de *encoder* decorridos, e de posse do número de pontos de uma volta do *encoder*, conhecendo o mecanismo de redução do motor, e as características do fuso instalado, é possível fazer a transformação de variáveis para saber o posicionamento do mecanismo.

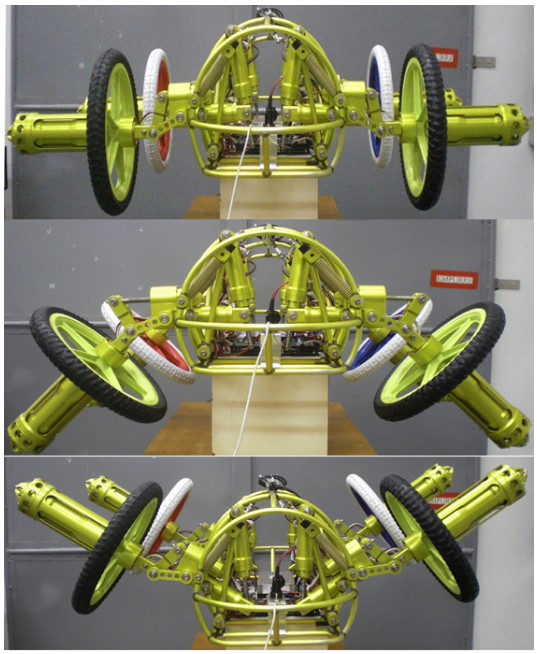

Figura 2.5: Suspensão do RAHP em diferentes configurações.

A tabela 2.3.1 mostra o curso de movimentação das juntas do mecanismo de suspensão, dado em pontos de *encoder*. Um limite inferior e um superior é estipulado para evitar que as juntas colidam, durante a operação, com o final do fuso, evitando desgastes ou o travamento do mecanismo.

Tabela 2.1: Curso das suspensões do RAHP.

Junta	**Mínimo**	**Máximo**	**Limite Inferior**	**Limite Superior**
Interna	0	143000	1500	141500
Externa	0	152000	1000	151000

Um outro problema decorrente dessa implementação é que sempre que o hardware do robô é desligado a orientação de qual é o ponto zero da junta é perdida. Quando ele é religado, o hardware considera o ponto atual é como sendo o ponto zero. Para resolver essa falta de referência os hardwares controladores possuem uma função denominada "*homing*", que permite achar o ponto zero. Essa função é configurável, podendo ser implementada de uma série de maneiras diferentes. No caso do RAHP ela irá rodar o motor com uma velocidade negativa até que um pico de corrente

de 700mA aconteça. Esse valor é um valor definido para indicar que o mecanismo colidiu com o fim do fuso, sem que haja um pico de corrente muito alto nos motores. Realizado esse procedimento, esse ponto de colisão passa a ser o ponto zero do mecanismo.

2.3.2 Arquitetura de Hardware

A figura 2.6 mostra o fluxograma da arquitetura de hardware do protótipo, e a figura 2.7 apresenta uma foto da montagem no protótipo.

2.3.2.1 PC104

O Puma PC104, demonstrado na figura 2.8, é um computador de placa única compacto, com baixo consumo de potência e sem partes móveis. Esse computador possui uma série de funcionalidades on-board, como vídeo, rede, portas USB, portas seriais, portas paralelas e interface para memória flash. Suas principais características são apresentadas na tabela 2.2.

Tabela 2.2: Características do PUMA PC104

Processador	AMD Geode GX 500
Chipset	AMD Geode CS5536
Alimentação	+5V ±5% @ 1A (5W)
Temperatura de Operação	0° a +60°C
Memória RAM	256 MB DDR SDRAM

Devido a essas características essa compacta placa se mostra interessante para sistemas de controle embarcados, sendo o hardware escolhido para ser o núcleo funcional do robô. Sua memória flash possui um sistema operacional instalado e nele são executados os softwares de controle que gerenciaram todas as ações do RAHP.

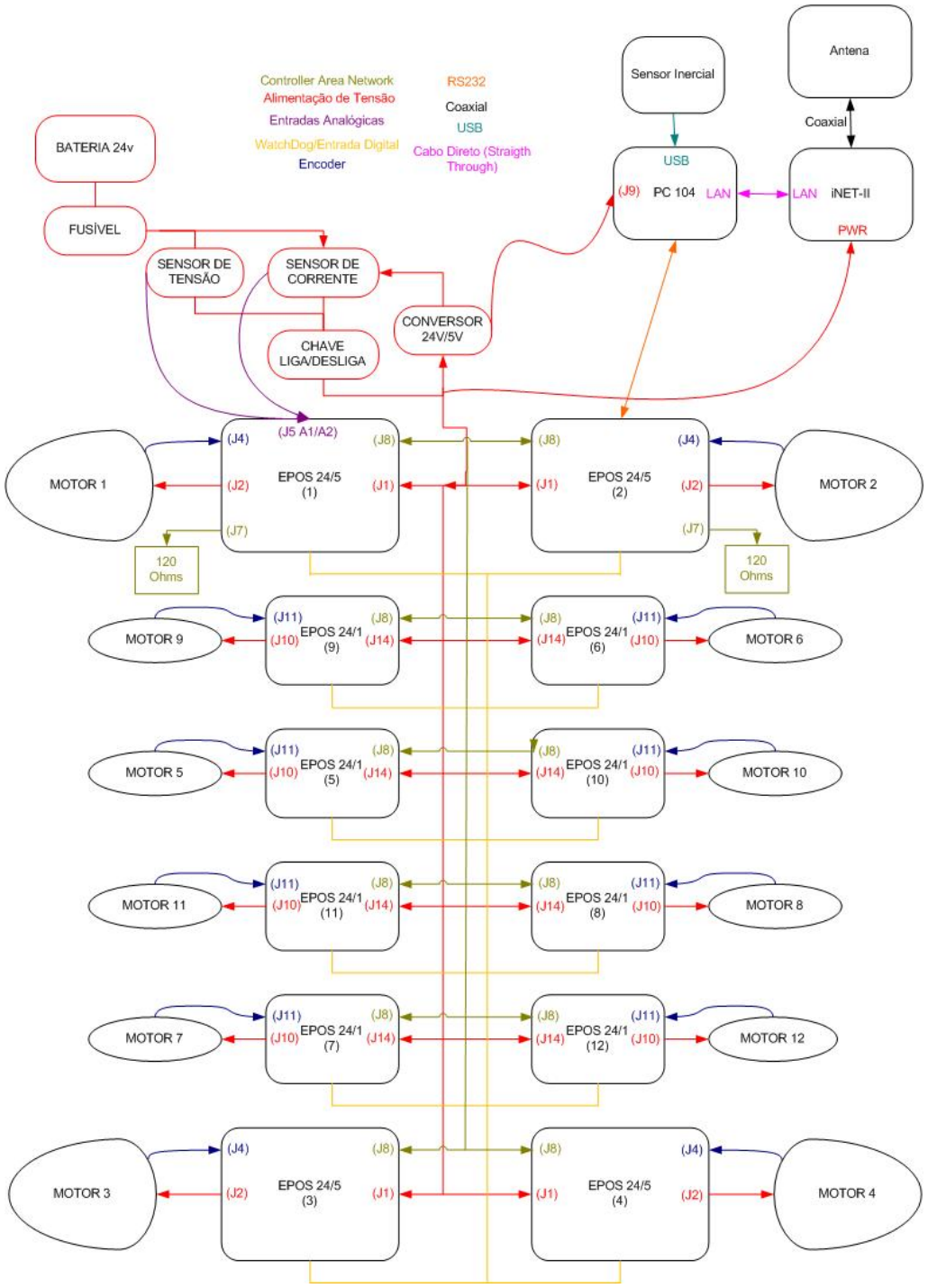

Figura 2.6: Fluxograma do Hardware do RAHP.

Esse computador é ligado via rede Ethernet ao rádio embarcado, e deste modo, sendo capaz de se comunicar com um outro computador que servirá de base para tele-operação do protótipo.

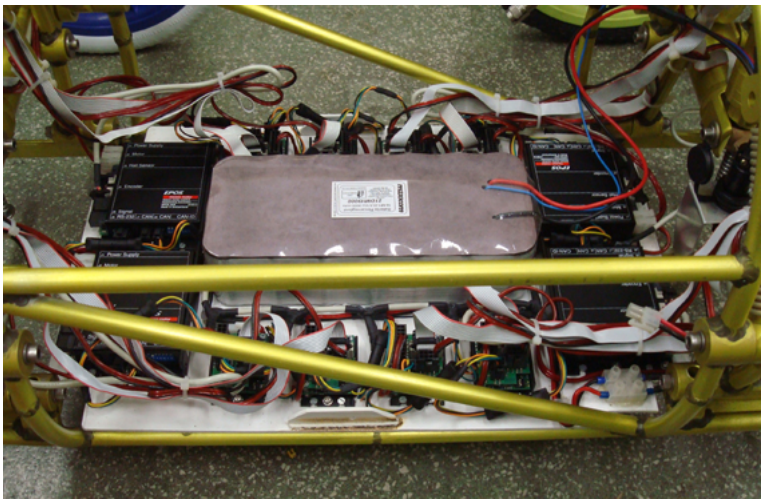

Figura 2.7: Montagem dos componentes eletrônicos do robô.

Figura 2.8: Computador PC104.

2.3.2.2 Controladores MAXON EPOS

Trata-se de um controlador de motores DC da Maxon [7] que deve ser configurado com os parâmetros do motor que irá controlar. Durante a operação ele pode ser acessado tanto por RS232 como por rede CAN. Através de um protocolo bem definido em seus manuais é possível fazer alterações na máquina de estados do controlador, comandando o mesmo, que é capaz de controlar velocidade, corrente ou posição do motor.

Nesse protótipo existem doze controladores, um para cada motor instalado no robô. Quatro desses controladores são do tipo 24/5, com máximos de 24V de tensão

e 5A de corrente, e oito deles são do tipo 24/1, com máximos de 24V de tensão e 1A de corrente. Os controladores 24/5 são ligados aos motores das rodas, enquanto que os controladores 24/1 são ligados aos motores destinados a controlar o mecanismo de suspensão do protótipo.

Esses doze controladores são interligados através de uma rede CAN. O controlador numerado como 2 é conectado então via RS232 ao PC104 do protótipo e funciona como um gateway, para que o computador possa se comunicar com qualquer um dos nós dessa rede de controladores.

2.3.2.3 Rádio MDS-Inet II

Figura 2.9: Rádio Industrial MDS INET-II.

O Rádio MDS-Inet II é um rádio capaz de gerar uma conectividade IP/Ethernet de longa distância. A figura 2.9 mostra o modelo do rádio e a tabela 2.3 apresenta suas características.

2.3.2.4 Central Inercial - Xsens MTi

O Xsens MTi é considerado uma miniatura de "Sensor da Referência da Atitude e do Título", um conjunto de três sensores em linha capazes de fornecer ângulos de atitude, título e guinada. É considerada uma unidade de medida para controles de robôs. A foto de uma central como essas é apresentada na figura 2.10. Esta central possui acelerômetros com escala de até 5g e giroscópios com escala de 300°/s.

As leituras desse sensor são obtidas através de conexão USB com o PC104. Esses dados obtidos são:

Tabela 2.3: Características do Rádio Inet II

Data Rate	512Kbps
Frequência de Trabalho	902MHz - 928MHz
Alcance em movimento	1 milha
Temperatura	-30°C - +60°C
Alimentação	10.3Vdc - 30Vdc
Potência	7W

Figura 2.10: Sensor Inercial Xsens.

- Ângulo de Orientação 3D
- Aceleração 3D
- Velocidade de giro 3D
- Campo Magnético 3D

2.4 Base de Controle

Para que o robô funcione tele-operado uma estrutura é montada como base de operação. A figura 2.11 mostra o esquema da base montado em uma missão na

Amazônia. O esquemático da figura 2.12, por sua vez, ilustra os equipamentos utilizados e as ligações necessárias para a montagem dessa base.

Figura 2.11: Base de tele-operação do robô em missão na Amazônia.

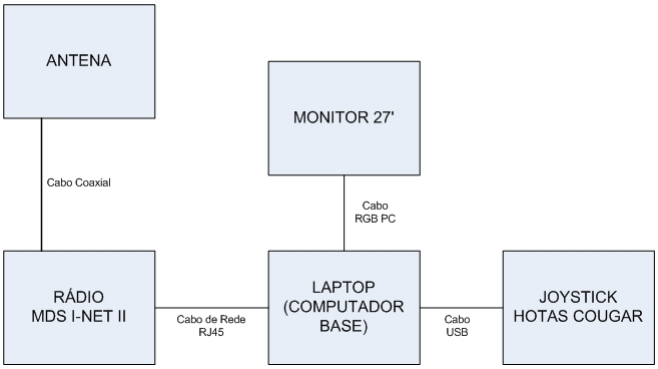

Figura 2.12: Esquemático de montagem da base de controle.

A antena tem como objetivo transmitir e receber o sinal de comunicação da base com o robô. Ela deve ser ligada através de um cabo coaxial ao rádio da base, que é o mesmo modelo, MDS Inet-II, instalado no interior do robô e apresentado na figura 2.9.

O rádio da base, por sua vez, é conectado ao computador através de um cabo ethernet com conectores RJ45. O computador, por conveniência, é ligado por cabo

USB a um joystick, que tem como objetivo permitir uma melhor controlabilidade do robô durante sua operação. O computador é ligado também a um monitor para propiciar uma melhor visualização do painel de controle.

Nesse computador da base, um notebook, é executado um software, desenvolvido em Java, que tem como interface gráfica o painel de controle do robô. Esse software tem como objetivo ser a interface entre todo o sistema do robô e seu operador. Ele envia comandos para o robô e recebe informações do mesmo através de *socket*s de rede.

Embora não seja estritamente necessário, a sugestão de utilização dos monitores desse sistema é que se use o monitor de 27' para apresentar a interface gráfica do software de controle e o monitor do notebook para apresentar uma janela de linhas de comando conectada via SSH ao computador embarcado no robô, de maneira que se possa monitorar de forma mais extensa os acontecimentos no sistema embarcado.

2.4.1 Software de Controle

O Painel de Controle é a interface gráfica do software de controle, e é apresentado na figura 2.13, onde é possível perceber que o controle do robô é realizado de maneira intuitiva, e que todos os dados lidos pelos sensores embarcados no robô serão apresentados para o operador nessa interface.

O software de controle do robô foi desenvolvido com o uso da linguagem Java, empregando um paradigma de gerenciamento de recursos conhecido como Produtor-Consumidor [8]. Um diagrama esquemático do funcionamento desse software pode ser observado na figura 2.14.

Conforme ilustrado no diagrama, as alterações feitas pelo operador, na interface do software são identificadas pelas classes do grupo *ChicoUI*. Essas classes interpretam o comando e fazem a passagem dos dados para as classes do grupo *Information*. Um outro grupo de classes, responsável por gerenciar as conexões de rede com o robô, denominado de *Sockets* irá fazer a leitura dos dados contidos nas classes *Information* e enviar os comandos via *socket*s de protocolo TCP/IP.

Figura 2.13: Painel de controle do Robô.

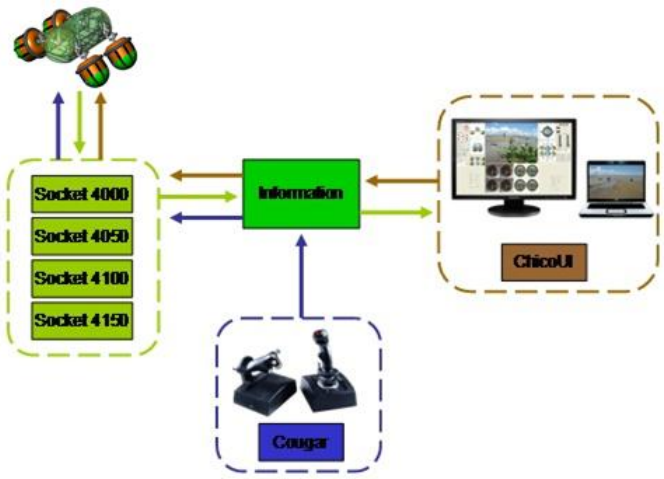

Figura 2.14: Diagrama de funcionamento do software da base.

As classes identificadas como do tipo *Joystick* também podem fazer alterações nos dados contidos nas classes *Information*. Dessa forma, comandos realizados no controlador também serão enviados ao robô.

A comunicação com o robô é feita por meio do protocolo TCP/IP, o qual trabalha com o modelo cliente-servidor. No caso desse sistema em específico o sistema do robô irá funcionar como o servidor e estará aguardando por conexões nas portas definidas no projeto. Essas portas são listadas na tabela 2.4. No caso do RAHP, não existe ainda nenhum sensor de parâmetros ambientais embarcados. Neste sentido, esse robô utiliza apenas as três primeiras portas.

Tabela 2.4: Portas de comunicação no sistema do robô

Porta	**Função**
4000	Conexão destinada a comandos enviados da base para o robô
4050	Conexão destinada a enviar dados operacionais do robô para a base
4100	Conexão destinada a enviar dados de navegação do robô para a base
4150	Conexão destinada a enviar dados de parâmetros ambientais para a base

As mensagens enviadas através da rede são cadeias de caracteres. Um protocolo é definido para enviar os comandos através dessas cadeias e será agora apresentado.

- Protocolo para a Porta 4000 - Comandos da base para o robô

  1. DC_1C_2XXC_3XC_4

     C_1 Indica a leitura da velocidade das rodas do robô.

     C_2 Indica a leitura das posições do mecanismo de suspensão do robô.

     C_3 Indica a leitura ou não do atual estado da bateria.

     C_4 Indica a leitura da corrente dos dos motores da roda do robô.

Os caracteres marcados como X são caracteres do tipo "*don't care*". Os demais caracteres podem assumir o caractere 0 ou o caractere 1. O caractere 1 indica que a leitura deve ser feita, e o caractere 0 indica que a leitura não é necessária.

2. R,vel1,vel2,vel3,vel4

   É o comando de velocidades para as rodas. Cada uma das quatro rodas será controlada com a velocidade no valor descrito no comando.

   Os valores de vel1 a vel4 podem ser um número inteiro de 0 a 7000. Esse valor deverá ser normalizado para ser compatível com a velocidade máxima dos motores das rodas do RAHP.

3. SP,pos1,pos2,pos3,pos4

   É comando de posição dos mecanismos de suspensão. Os campos pos1 a pos4 podem assumir como valor um número de 0 a 1000, o caractere h, ou o caractere O.

   Sendo o comando para aquele mecanismo o caractere h a frenagem dos motores associados a aquele mecanismo deve acontecer imediatamente. Quando o comando for o caractere O o mecanismo devera executar o procedimento de *homing*, que já foi descutido anteriormente. Se o comando for um número esse valor deverá ser normalizado para uma referência de altura para o mecanismo em questão, procedimento que será melhor discutido na seção 4.2.

4. LC_1XXXC_2XXXXXC_3

   C_1 Indica o funcionamento do controlador dos atuadores do robô.

   C_2 Indica o funcionamento do sensor inercial do robô.

   C_3 Indica o funcionamento do controle das suspensões do robô.

   Os caracteres marcados como X são caracteres do tipo "*don't care*". Os demais caracteres podem assumir o caractere 0 ou o caractere 1. O carac-

tere 1 indica que o funcionamento deve acontecer, enquanto o caractere 0 indica que o funcionamento não é necessário.

- Protocolo para a Porta 4050 - Dados operacionais do robô

  1. v,vel1,vel2,vel3,vel4,velref1,velref2,velref3,velref4

     Tem por objetivo informar à base a velocidade atual das 4 rodas do robô assim como suas velocidades de referência.

  2. p,pos1,pos2,pos3,pos4,posref1,posref2,posref3,posref4

     Tem por objetivo informar à base a atual posição do mecanismo de suspensão do robô assim como sua posição de referência.

  3. b,bat1,bat2,bat3,bat4

     Tem como objetivo enviar as informações sobre o atual estado da bateria do robô para a base.

  4. c,cur1,cur2,cur3,cur4

     Informa a base a respeito da corrente de cada um dos motores das rodas do robô.

- Protocolo para a Porta 4100 - Dados de navegação do robô

  1. x,informação_tempo,informação_inercial
     - informação_tempo = tm_mon,tm_mday,tm_hour,tm_min,tm_sec,tv_usec
     - informação_inercial = euler3D,acel3D,gyr3D,mag3D
     - euler3D = euler_pitch,euler_roll,euler_yaw
     - acel3D = acel_pitch,acel_roll,acel_yaw
     - gyr3D = gyr_pitch,gyr_roll,gyr_yaw
     - mag3D = mag_pitch,mag_roll,mag_yaw

     As informações do tempo enviadas na cadeia de caracteres são conforme a biblioteca "time.h"padrão do C ANSI. As demais informações são retiradas do sensor inercial MTi Xsens e formatadas em caracteres.

2.5 Endereços de Rede

Para que a rede e o roteamento dos pacotes funcione os dispositivos devem ter seus endereços corretamente definidos e configurados. Algumas estratégias de QoS podem ser utilizadas para o melhor tráfego de informações sem comprometer o correto funcionamento do robô, porém esse assunto não faz parte do escopo do presente projeto. A tabela 2.5 apresenta os endereços IP definidos para os dispositivos necessários para o funcionamento do protótipo.

Tabela 2.5: IP dos dispositivos na rede de operação do robô

Dispositivo	**Endereço IP**
Computador Base	10.241.4.135
Rádio Base	10.241.4.141
Rádio Embarcado	10.241.4.130
PC/104	10.241.4.136

Capítulo 3

Xenomai

3.1 Introdução

O Xenomai (http://www.xenomai.org) é um subsistema de tempo real, que pode ser fortemente integrado ao *kernel* do Linux a fim de garantir tempos de resposta previsíveis para as aplicações [9].

Dessa forma o sistema permite rodar tarefas de tempo real no espaço de usuários do sistema operacional. As tarefas de tempo real devem ser controladas exclusivamente pelo *co-kernel* do Xenomai, quando houver a necessidade de executar tarefas críticas em relação ao tempo, com o objetivo de obter latências baixas.

Em seu nascimento, o Xenomai tinha como principal objetivo ser um sistema de tempo real no ambiente GNU/Linux de licença livre para o qual seria possível portar, com mínimas alterações, os softwares desenvolvidos previamente para os principais sistemas de tempo real com licença proprietária.

Devido a esse objetivo a arquitetura do Xenomai apresenta o que são chamadas de *skins*. Uma série de diferentes APIs que o sistema dispõe para que seja possível compilar aplicações escritas conforme a semântica de um determinado sistema operacional.

Além disso o sistema apresenta também uma *skin* chamada de *Native*, que tem como objetivo introduzir uma API que atinja todas as capacidades do núcleo de

tempo real e faça uso completo do alto nível de integração com o ambiente GNU/Linux.

3.2 A Arquitetura do Xenomai

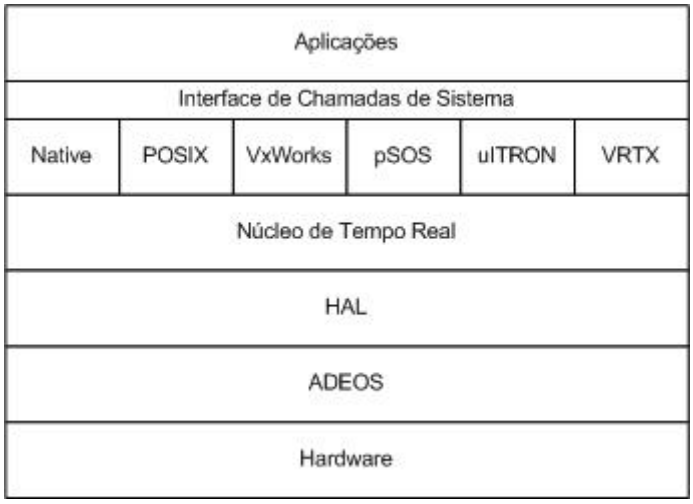

Figura 3.1: Arquitetura de camadas do Xenomai. Fonte: [10].

A figura 3.1 apresenta a arquitetura de camadas do Xenomai observada de forma gráfica. A primeira camada é obviamente o hardware da máquina onde o sistema estará em execução. Acima do hardware encontra-se o *nano-kernel* do ADEOS, que fornece um "ambiente adaptativo que pode ser utilizado para compartilhar os recursos de hardware entre múltiplos sistemas operacionais ou entre múltiplas instâncias de um mesmo sistema operacional", conforme definido por [10]. Essa estrutura será melhor explicada na seção seguinte.

A camada superior a essa é a chamada HAL, sigla em inglês para *Hardware Abstraction Layer*. Essa camada tem por objetivo funcionar como uma camada de abstração ao hardware, isso é, reunir toda a parte dos códigos de implementação que são dependentes da arquitetura de processador que está sendo usada. Dessa forma, tudo que for realizado nas camadas superiores não será dependente de qual a arquitetura está sendo utilizada.

Acima da HAL, esta posicionado o núcleo do sistema operacional de tempo real, o *co-kernel* do Xenomai. Esse núcleo será melhor discutido mais adiante nesse

trabalho.

A camada seguinte é composta por diversas *skins* que são oferecidas pelo Xenomai. Essas *skins* permitem aos usuários diversas semânticas diferentes para serem feitas as chamadas de sistema na camada acima chamada de Interface de Chamadas de Sistema.

Por fim, a última camada chamada Aplicações é onde ficam posicionados os utilitários do sistema.

3.2.1 Pipeline de Interrupções

Para garantir que as latências serão previsíveis para as tarefas de tempo real é necessário garantir que o *kernel* do Linux jamais irá adiar o tratamento de interrupções externas. Da mesma forma, o *kernel* do Linux também não pode disparar um tratador de interrupção quando uma tarefa de tempo real estiver em uma seção que bloqueie interrupções.

Para esse fim, e também para que seja possível o compartilhamento do hardware entre o Xenomai e o *kernel* padrão do Linux, é utilizado o conceito do *Adaptive Domain Environment for Operating Systems* [11], o ADEOS, que tem como objetivo gerenciar um ambiente capaz de permitir o compartilhamento de um mesmo hardware por múltiplos sistemas operacionais.

Nessa implementação, o conceito mais importante utilizado é o de Domínio de Sistema Operacional, onde cada sistema é encapsulado em um domínio, no qual o sistema tem o total controle, podendo esse domínio incluir processos, memórias virtuais, sistemas de arquivo, entre outros recursos.

Esses recursos não serão necessariamente exclusivos. Uma vez que alguns sistemas operacionais reconhecem a existência do ADEOS e são capazes de interagir com o mesmo, eles são capazes de compartilhar recursos com outros domínios e acessar os recursos dos outros domínios.

A figura 3.2 exemplifica a arquitetura implementada com o ADEOS. Nela é possível visualizar os domínios de sistema operacional e as possíveis interações. Interações do tipo A são o uso normal do hardware, feito por qualquer um dos domínios. Interações do tipo B mostram o ADEOS recebendo controles a partir do hardware devido a interrupções, assim como todos os tipos de comando que o ADEOS pode gerar para o hardware.

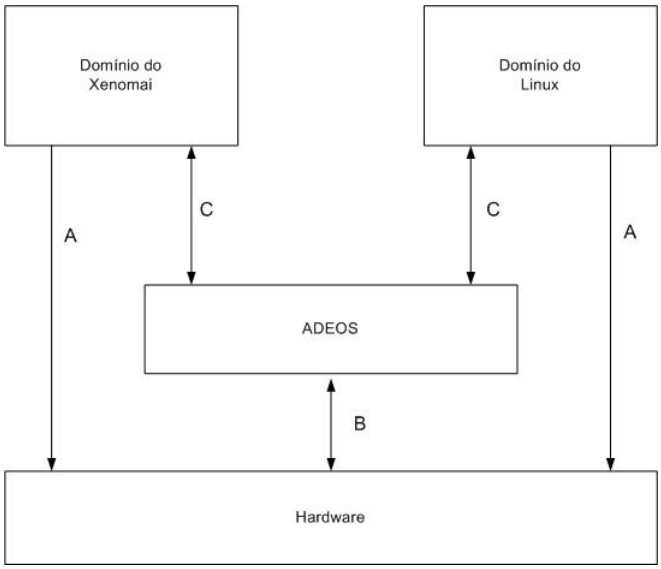

Figura 3.2: Arquitetura de funcionamento do ADEOS. Fonte: [11].

As interações do tipo C, por sua vez, representam a comunicação entre os domínios de sistema operacional e o ADEOS. Nesse tipo de comunicação, onde o sistema operacional tem conhecimento da existencia do ADEOS e se comunica com o mesmo, é onde o maior uso da capacidade do ADEOS, pois o sistema operacional pode garantir sua prioridade no tratamento de interrupção acima dos outros domínios, como é o caso do Xenomai.

É prevista ainda um outro tipo de interação, onde o SO não tem conhecimento da existência do ADEOS, e recebe as interações do mesmo como se fossem provenientes do hardware e de sua máscara de interrupções. Esse caso não e utilizado na implementação Xenomai/Linux, embora o *kernel* do Linux quase não interaja com o ADEOS e na prática funcione a maior parte do tempo sem perceber sua existência.

O ADEOS implementa também o Pipeline de Interrupções, também chamado de I-pipe. Essa estrutura tem como função criar uma máscara para interrupções virtual, implementada por software, de forma que possa ser ajustada conforme a necessidade.

No I-pipe cada um dos domínios de SO tem um determinado nível de prioridade. No caso em questão o Xenomai é o domínio de prioridade mais alta, a frente do *kernel* do Linux. Sendo assim, o I-pipe vai despachar os diversos eventos, como por exemplo interrupções, chamadas ao sistema, falhas do processador e outras exceções primeiro ao Xenomai.

O Xenomai, por sua vez, registra tratadores para os diferentes eventos que podem ser provenientes do I-pipe. Essa estrutura descrita permite ao Xenomai pré-processar os eventos fazendo com que tanto o sistema de tempo real quanto o *kernel* comum do Linux possam compartilhar o hardware adequadamente.

A figura 3.3 representa a arquitetura do I-pipe. Os eventos serão propagados através dos domínios de sistema operacional, que poderão decidir se vão, aceitar, ignorar, descartar ou terminar esses eventos. Cada uma dessas possíveis escolhas terá um diferente efeito e será controlado pelo ADEOS de maneira diferente.

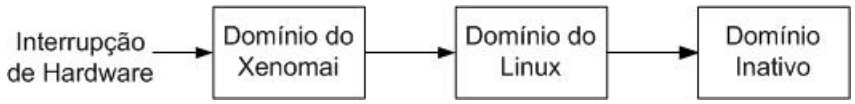

Figura 3.3: Pipe de interrupções do ADEOS. Fonte: [11].

3.2.2 Núcleo do Xenomai

O *kernel* do Xenomai tem o objetivo de fornecer todos os recursos de sistema operacional necessários para que as skins implementem as APIs de tempo real, uma vez que o Linux não possui chamadas de sistema com essa finalidade.

Em sua construção o *kernel* do Xenomai é descrito como um RTOS abstrato, pois define blocos de construções genéricos, com o objetivo de implementar qualquer API

de tempo real que possa se encaixar nesse modelo genérico [9].

Esses blocos de construção são então integrados em um só módulo denominado *nucleus*. Os principais componentes do *nucleus* do Xenomai são descritos a seguir:

- Alocador de memória com latência previsível.
- Objeto de sincronização extensível, utilizado para bloquear threads por todos os serviços do RTOS. Esse recurso permite ao Xenomai realizar *timeouts* e ordens de fila FIFO ou baseada em prioridade quando múltiplas *threads* precisam bloquear um só recurso.

  Dessa forma, todos os semáforos, mutexes, variáveis de condição, filas de mensagens e caixas de correio implementados nas Skins do Xenomai se baseiam nessa abstração do *nucleus*.
- Gerenciamento de temporizador, permitindo que qualquer serviço relacionado ao tempo crie temporizadores de software.
- Um objeto genérico de interrupção, capaz de conectar a skin do RTOS a qualquer número de linhas de IRQ fornecidas pelo hardware.
- Um objeto de *thread* tempo real controlado pelo escalonador do Xenomai. Esse escalonador do Xenomai e preemptivo e da suporte a vários níveis de prioridade de *thread*. Dessa forma, a ordem FIFO se aplica dentro de um nível de prioridade.

É possível perceber que o *nucleus* do Xenomai adiciona mais uma camada de abstração, permitindo que as *skins* do Xenomai tenham disponíveis todas as chamadas de sistema necessárias para a implementação das APIs de RTOS.

3.2.3 Skins do Xenomai

Conforme já citado anteriormente o Xenomai nasceu de um objetivo de permitir que códigos de diversos RTOS proprietários fossem portados para o ambiente GNU/Linux. Devido a isso o sistema Xenomai não possui uma API mestre para o desenvolvimento de softwares, mas sim um leque de diversas APIs que permitem ao

desenvolvedor escolher qual em qual semântica desenvolver sua aplicação de tempo real, ou portar determinado código já existente através da API conveniente.

As *skins* do Xenomai são construídas utilizando os blocos de códigos genéricos do *nucleus* do Xenomai, descrito anteriormente.

As diversas *skins* na prática vão aparecer como módulos do Linux, que o administrador do sistema poderá carregar estaticamente na imagem do *kernel* ou carregar dinamicamente conforme a necessidade.

O Xenomai possui APIs para emular sistemas POSIX, VxWorks, pSOS+, VRTX, e uITRON.

Durante o desenvolvimento do sistema, foi introduzida também uma API original do Xenomai, denominada *Native API*, voltada para desenvolvedores que não tivessem interesse em questões de portabilidade do código para outros RTOSs e quisessem se aproveitar de alto nível de integração com o ambiente GNU/Linux. Essa API será melhor descrita na próxima seção.

Existe ainda uma outra API, denominada de RTDM, que não representa uma das *skins* do Xenomai. Essa API tem por objetivo fornecer uma interface para usuários e desenvolvedores de *drivers* para dispositivos de tempo real, abordando as restrições de sistemas baseados em *kernel* dual.

3.2.3.1 Native API

Como já citada anteriormente a *skin* chamada de *Native* tem como objetivo introduzir uma API que atinja todas as capacidades do núcleo de tempo real e faça uso completo do alto nível de integração com o ambiente GNU/Linux.

Essa API disponibiliza uma série de chamadas de sistema, com o objetivo de prover a maioria das chamadas existentes nos diversos RTOS que serviram de objeto de estudo para a criação das outras *skins* do Xenomai.

Sendo assim, é possível enumerar os serviços documentados para essa API, que podem ser melhor observados em [12] e [13], sendo eles divididos em cinco categorias:

1. **Gerenciamento de Tarefas:** Essa categoria define uma série de tarefas relacionadas com o agendamento de tarefas e o gerenciamento geral das mesmas. Uma aplicação precisa dessas chamadas de sistema para criar *tasks* e para controlar o comportamento das mesmas.

2. **Serviços de Tempo:** Esse grupo engloba todas as chamadas de sistema relacionadas ao gerenciamento do temporizador do sistema e *queries* ao temporizador, além de prover temporizadores chamados de *Alarms* para a implementação de *watchdogs*.

3. **Suporte a Sincronização:**

   - Semáforos Contadores
   - Mutexes
   - Variáveis de Condições
   - Flags de Eventos

4. **Mensagens e Comunicação:**

   - Mensagens Inter-task Síncronas
   - Message Queues
   - Pilhas de Memória
   - Pipes de Mensagem

5. **Manipulação de E/S de Dispositivos:** Mecanismos simples para lidar com interrupções e acesso a memórias de dispositivos de E/S a partir do espaço de usuários.

3.3 Como o Xenomai Funciona

O Xenomai é um sistema operacional de tempo real que se propõe a ter um alto nível de integração com o ambiente GNU/Linux. Segundo [9], "Para o projeto Xenomai, manter o modelo normal de programação para o Linux disponível

para aplicações de tempo real sempre foi considerado tão importante como garantir latências mais baixas em qualquer hardware fornecido".

Em fato, o que é observado pelo Xenomai é que as aplicações de tempo real podem necessitar dos serviços do Linux, como acessar arquivos de disco para registrar dados em logs ou estabelecer comunicação pela rede. O problema desta questão quando se diz respeito a arquitetura do sistema, é como fazer uma aplicação que está atrelada a uns dos kernels chamar serviços do outro *kernel* se o Linux mal sabe da existência do cokernel Xenomai.

3.3.1 Migração de Domínio

Entendendo a funcionalidade do Xenomai é possível perceber que uma *thread* de tempo real pode chamar tanto os serviços padrão da *glibc* quanto os serviços introduzidos pelo Xenomai. A questão de qual chamada de sistema é utilizada depende da conveniência para o objetivo que a *thread* deseja realizar no momento.

Segundo [9], uma *thread* pode por exemplo se utilizar de chamadas do Linux para realizar tarefas de configuração ou limpeza, como acessar o sistema de arquivos ou configurar comunicações com algum driver de dispositivo. Porém durante o trabalho de processamento principal, o qual é o objetivo daquela *thread*, ela pode exigir garantias rígidas de tempo real, com latências baixas e limitadas, e para isso se utilizar de chamadas ao sistema Xenomai.

Para tirar o maior proveito possível dos recursos do Linux, o Xenomai implementa então o que ele chama de migração de domínio. O sistema permite que as aplicações sejam executados no domínio do Xenomai ou do Linux.

Quando executada no domínio do Xenomai a *thread* será controlada pelo escalonador do Xenomai. O pior caso de latência para tarefas como essa é proxima ao limite do hardware,e previsível.

Quando uma tarefa de tempo real, no domínio do Xenomai, realizar uma chamada de sistema do Linux, será imediatamente migrada do domínio do Xenomai para o

do Linux, uma vez que a chamada não poderia ser atendida a partir do domínio do Xenomai. A tarefa vai então entrar no *kernel* Linux no ponto de escalonamento mais próximo. Obviamente, o custo de tempo dessa migração depende da granularidade do *kernel* do Linux.

Quando executada no domínio do Linux, a *thread* tem acesso a todas as chamadas de sistema do Linux, porém essas chamadas de sistema em sua maioria foram projetadas para serem justas em relação a distribuição de tempo do processador, e portanto não possuem um determinismo no seu tempo de execução, o que é critico no caso de uma *thread* de tempo real.

É importante observar também, que quando uma *thread* de tempo real migra para o domínio do Linux, o *kernel* do Linux como um todo herda a prioridade dessa *thread* de tempo real, e passa a competir por recursos da CPU com outras tarefas de tempo real.

Quando uma *thread* estiver executando no domínio do Linux e fizer uma chamada de sistema do Xenomai, ela deve ser da mesma forma migrada de volta para o domínio do Xenomai.

Por convenção, todos os serviços do Xenomai que podem bloquear a *thread* que está requisitando o serviço realizam essa migração para o domínio do Xenomai. Isso quer dizer que todas as tarefas bloqueadas serão mantidas dormindo no domínio do Xenomai, controladas pelo mesmo.

Dessa forma, é possível concluir que as tarefas serão mantidas no domínio do Xenomai até realizarem uma chamada ao sistema do Linux, quando serão migradas para o domínio do Linux, e permanecerão nele até que alguma chamada seja feita ao domínio do Xenomai, quando serão migradas de volta.

3.4 Como Escrever As Tarefas de Tempo-Real

Através do entendimento da seção anterior podemos perceber que para tarefas de tempo-real o ideal é que façam chamadas de sistema apenas ao Xenomai, de forma a

manter o determinismo de seus tempos de execução. Para o simples processamento de dados presentes em variáveis do programa não existe nenhum problema nisso, o problema é com as chamadas de sistema, atividades como fazer comunicação com o hardware e imprimir caracteres na tela, escrever informações em arquivos texto.

De maneira a realizar todas as tarefas necessárias no domínio do Xenomai, é necessário utilizar para a comunicação uma das estruturas disponíveis para comunicação em tempo real, como o RT_SOCKET_CAN utilizada para comunicação em CAN ou a RT_SERIAL. A partir daí chamadas ao sistema Xenomai serão utilizadas como por exemplo, rt_dev_open ou rt_dev_write.

Dessa maneira é preciso apenas conseguir uma estratégia para imprimir caracteres na tela, ou em algum arquivo texto. O ideal seria iniciarmos uma outra thread não-RT e passar para ela as strings que precisam ser impressas na tela ou em um arquivo. Felizmente isso já foi desenvolvido no Xenomai, embora ainda não seja bem documentado.

A biblioteca *librtdk* pertencente ao pacote do Xenomai é uma biblioteca para impressão em tempo real. A idéia básica da biblioteca é manter a tarefa tão leve o quanto possível. Cada *thread* que utilizar uma chamada como por exemplo a *rt_printf* tem seu buffer local próprio. Uma *thread* central não-RT se encarrega de direcionar o conteúdo do buffer de cada uma das *threads* que estiverem utilizando o serviço para o stream de saída. Essa *thread* de saída utiliza *pooling* periódico, de forma a evitar que qualquer mecanismo de sinalização seja necessário.

Qualquer programa que for se utilizar desses mecanismos deve fazer uma chamada de sistema para criar o buffer de saída e iniciar a *thread* de saída não-RT. A chamada a seguir pode ser utilizada com esse intuito:

```
rt_print_auto_init(1);
```

Com essas particularidades discutidas temos todas as ferramentas para escrever as tarefas de tempo-real do sistema que será projetado.

Capítulo 4

Controle Proposto para o RAHP

4.1 Introdução

Conforme descrito no capítulo 1, existe uma imensa série de aplicações para robôs móveis tele-operados, e para que esses possam funcionar de maneira correta são necessários diferentes tipos de controles, com diferentes objetivos.

Se nos restringíssemos ao caso do Robô Ambiental Híbrido, poderíamos elaborar uma série de controles, tanto necessários para sua operação, quanto com o objetivo de facilitar a vida do operador. Apenas como um exemplo, poderíamos propor que fosse controlada a posição GPS do Robô, de forma que passada uma posição de referência esse, de maneira autônoma, tomasse uma direção, propondo caminhos que desviassem de obstáculos, e fosse capaz de chegar a posição de referência.

Controles como esses são possíveis, mas requerem uma quantidade imensa de estudos e algoritmos de controle complexos.

O objetivo desse sistema é permitir que algoritmos como esse possam ser implementados, mas não precisamos provar o conceito com um algoritmo tão complicado. Podemos nos valer de um controle não tão complicado, mas talvez até mais valioso para o funcionamento do protótipo: o controle dos independentes mecanismos de suspensão.

Nesse capítulo será apresentada a estratégia proposta para controlar esses mecanismos, o que funcionará como uma maneira de mostrar que esse sistema pode rodar algoritmos de controle de maneira correta.

4.2 Controle dos Mecanismos de Suspensões

Conforme já descrito anteriormente, as suspensões do RAHP são compostas por um mecanismo paralelo com 2 graus de liberdade. Esse mecanismo precisa ser controlado para que a estrutura possa, na prática, ajudar na locomoção do RAHP.

Um estudo completo para o controle do mecanismo é apresentado no Anexo A. O presente texto resume-se a explicar o estudo desenvolvido envolvendo a sua implementação.

4.2.1 Sistema e Modelo

A figura 4.1 mostra o modelo da estrutura a ser controlada. O ponto ao fim do Link6 é o ponto de contato entre a roda e o solo em que o robô se encontra e portanto é considerado o ponto efetuador da estrutura.

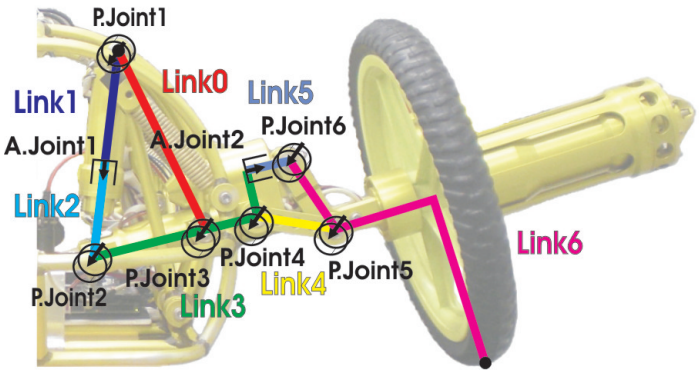

Figura 4.1: Suspensão do RAH: 7 elos com juntas ativas e passivas.

A figura 4.2 apresenta o modelo de onde pode ser obtida a cinemática direta do mecanismo. Essa cinemática, quando dadas as posições das duas juntas ativas,

$\theta_a = [d_1, d_2]^T$, permite a determinação da posição de todas as juntas passivas do mecanismo $\theta_p = [\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6]^T$.

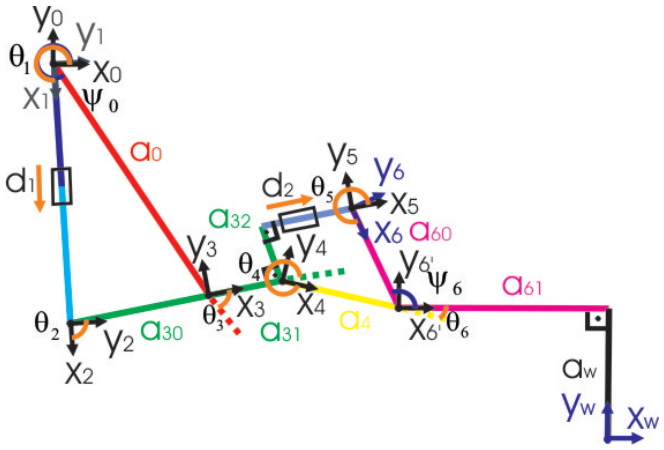

Figura 4.2: Suspensão do RAH: sistema de coordenadas do mecanismo de 7 elos.

A atuação utilizada para o controle do mecanismo é realizada na velocidade das juntas ativas, e a realimentação obtida é a posição atual dessas juntas.

O bloco controlador, exemplificado na figura 4.3, deverá receber as referências da coordenada y do ponto de contato entre a roda e o solo (altura) e o ângulo de orientação desse ponto, que dará por consequência o ângulo de orientação da roda do robô. Essas são as duas variáveis a serem controladas. O bloco controlador deverá também receber os valores realimentados da posição das juntas ativas.

Dessa forma é computada a cinemática direta, e então, através de uma lei de controle, é gerada a referência de velocidade para as juntas ativas do mecanismo.

Para a implementação em questão, o ângulo de referência para o mecanismo será gerado a partir do conhecimento de qual a altura de referência, de maneira otimizar o posicionamento do efetuador. O calculo do ângulo de referência se dará segundo o algoritmo a seguir escrito em C:

```
double thetaRef(double href){
const double pi=3.1415;
if (href >= -9)
```

```
    return (6*pi - pi/30);
else
    return (6*pi - pi/30 - (6*pi - 18.2187)*(-9 - href)/(-9 + 14.27));
}
```

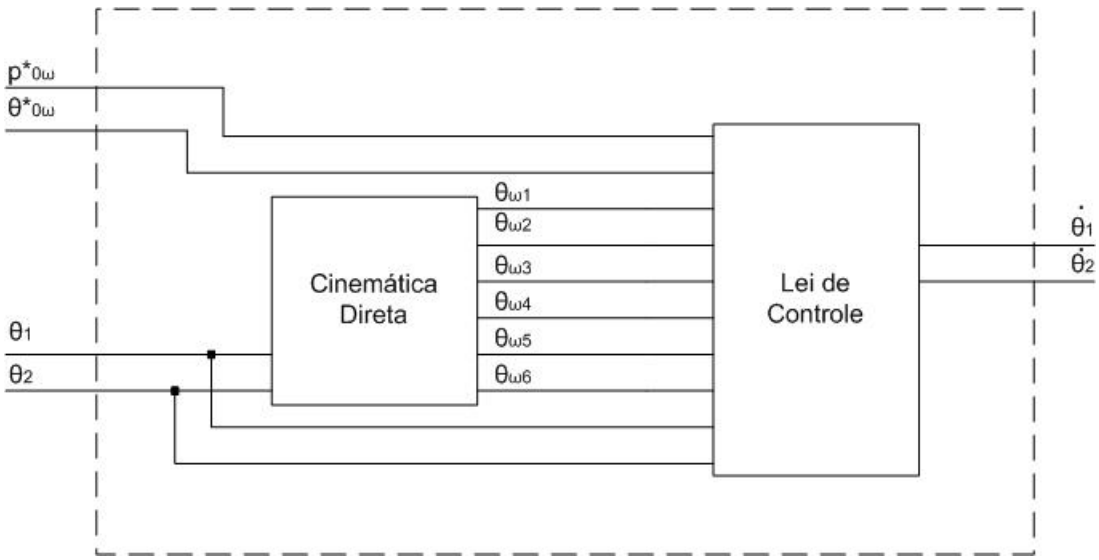

Figura 4.3: Controlador do mecanismo de suspensão do RAHP.

O sistema de controle pode ser observado na figura 4.4.

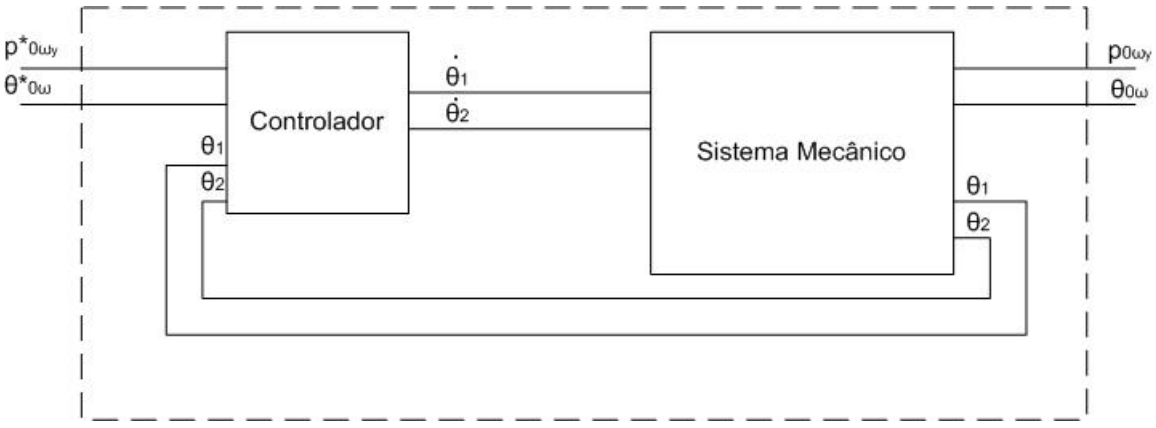

Figura 4.4: Sistema de controle das suspensões do RAHP.

4.2.2 Implementação

Essa seção tem por objetivo descrever a implementação do controle cinemático das suspensões do RAHP, proposto no anexo B.

Para realizar os primeiros testes da implementação do controle um software foi desenvolvido com o algorítmo conforme descrito a seguir:

```
abrirArquivoDeLog();
iniciarComunicacaoComHardware();
configurarParametrosDoControle();
while(true){
    for(todosOsMecanismos){
        lerPosicaoAtual();
        calcularCinematicaDireta();
        receberReferencias();
        calcularErros();
        calcularVelocidades();
    }
    for(todosOsMecanismos){
        enviarVelocidadesParaOHardware();
        salvarLogs();
    }
}
```

Para tornar a implementação possível uma biblioteca em C foi desenvolvida e é utilizada, sendo composta pelas seguintes funções:

- *convertEncoderPointsToMillimeter()*: Converte uma dada posição das juntas ativas $\theta_a = [d_1, d_2]^T$ em pontos de encoder para comprimento em milímetros;
- *forwardKinematics()*: Dadas as posições das juntas ativas θ_a, a função calcula a cinemática direta do mecanismo $\vec{p}_{0w}, \theta_{0w}$;

- *suspensionControl()*: Dada a posição das juntas θ_a, θ_p e os erros de malha fechada de posição e_h e orientação e_o a função calcula o comando do controle $u = \dot{\theta}_a$.

A função *forwardKinematics* foi obtida usando o comando emphccode do emph-Matlab. A expressão matemática para a cinemática direta de posição é declarada como uma expressão simbólica no *Matlab*, e o emphccode é usado para gerar um código em C implementando essa expressão.

O comando *ccode* não gera necessariamente um código computacionalmente eficiente, e esse deveria ser pós-otimizado. Para isso, valores que são calculados com frequência durante a execução do código são armazenados em variáveis e depois reutilizados, reduzindo a média do tempo de computação conforme mostrado na tabela 4.1.

Para implementar *suspensionControl*, o *ccode* foi novamente utilizado, e depois o código foi otimizado como descrito anteriormente.

A biblioteca OpenCV [14] é utilizada para executar a inversão da matriz $J_{c_p}^{-1}$. Ela possui uma função para a inversão de matrizes tanto por método SVD (decomposição de valores singulares) e método LU (eliminação gaussiana com escolha de pivô otimizada). A média do tempo de computação para *suspensionControl* usando ambos os métodos com otimização ou não é apresentada na tabela 4.2.

Tabela 4.1: Tempo médio de computação (E[t]) e desvio padrão associado (σ) da função *forwardKinematics*.

Algoritmo	**E[t]**	σ
Não Otimizado	165.79μs	74.72μs
Otimizado	35.19μs	6.50μs

Tabela 4.2: Tempo médio de computação (E[t]) e desvio padrão associado (σ) da função *suspensionControl*.

Algoritmo	**E[t]**	σ
Não Otimizado com inversão LU	353.60μs	99.79μs
Não Otimizado com inversão SVD	1074μs	158.93μs
Otimizado com inversão LU	219.07μs	65.05μs
Otimizado com inversão SVD	942.57μs	175.61μs

As médias de tempo de computação apresentadas nas tabelas 4.1 e 4.2 levam em consideração apenas o tempo de processamento, dessa forma não incluem o tempo de comunicação com o hardware do robô. A tabela 4.1 mostra que *forwardKinematics* é reduzida a 20% do seu tempo original de computação. A tabela 4.2 mostra que *suspensionControl* tem seu tempo de computação reduzido a 60% do valor original e também que a inversão através do método LU, para essa aplicação é $\sim 4\times$ mais rápida que através do método SVD.

4.2.3 Testes e Resultados

Com o objetivo de validar a implementação do controle cinemático, os mesmos valores utilizados nas simulações presentes no Anexo A foram usadas para controlar uma das suspensões do RAHP. Os resultados são apresentados na figura 4.5.

De maneira similar à simulação, o objetivo primário, controlar a posição vertical do ponto de contato entre a roda e o terreno p_{0w_y} é prontamente atendido, e $e_h \to 0$ depois de cada uma das mudanças no sinal de referência $p^*_{0w_y}$.

O erro de orientação e_o também diminui para cada diferente referência, de forma que o objetivo secundário é eventualmente atingido, quando as juntas ativas não estão em suas posições limite.

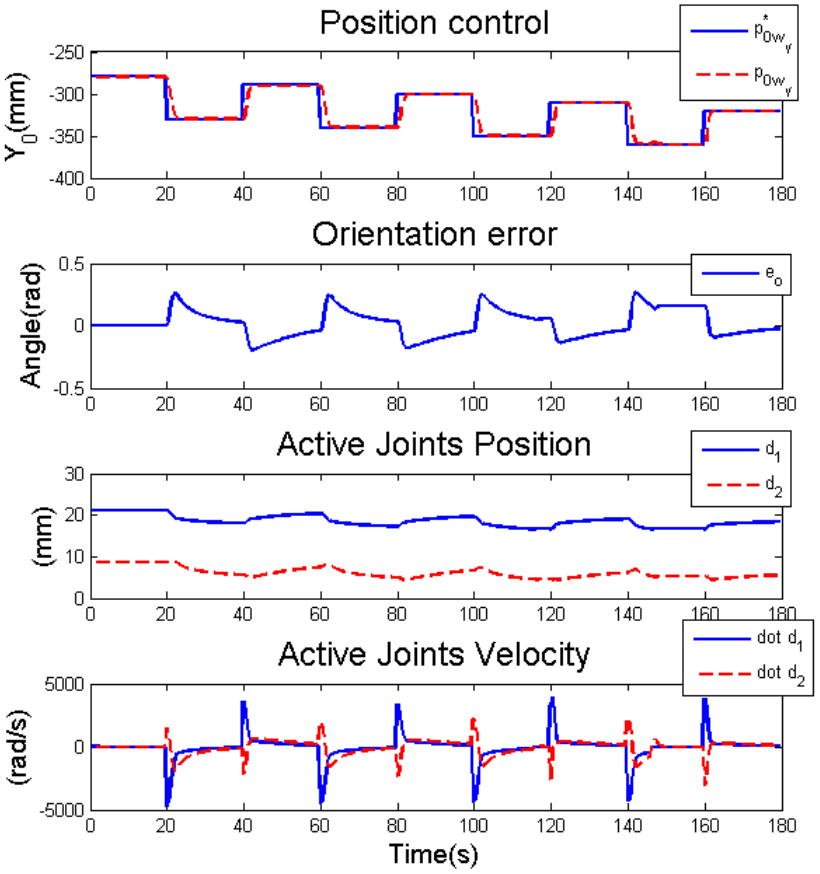

Figura 4.5: Resultados experimentais do controle cinemático do RAHP.

Capítulo 5

Especificações do Sistema

5.1 Sistema

Segundo [15], um sistema é um mapeamento de uma série de entradas em uma série de saídas. Essa visão pode auxiliar em muito a compreensão do sistema que deve ser projetado.

Tomando como base o protótipo apresentado no capítulo 2 e o sistema de controle apresentados no capítulo 4 uma primeira visão do sistema é apresentada na figura 5.1, sendo sua implementação ainda abstraída.

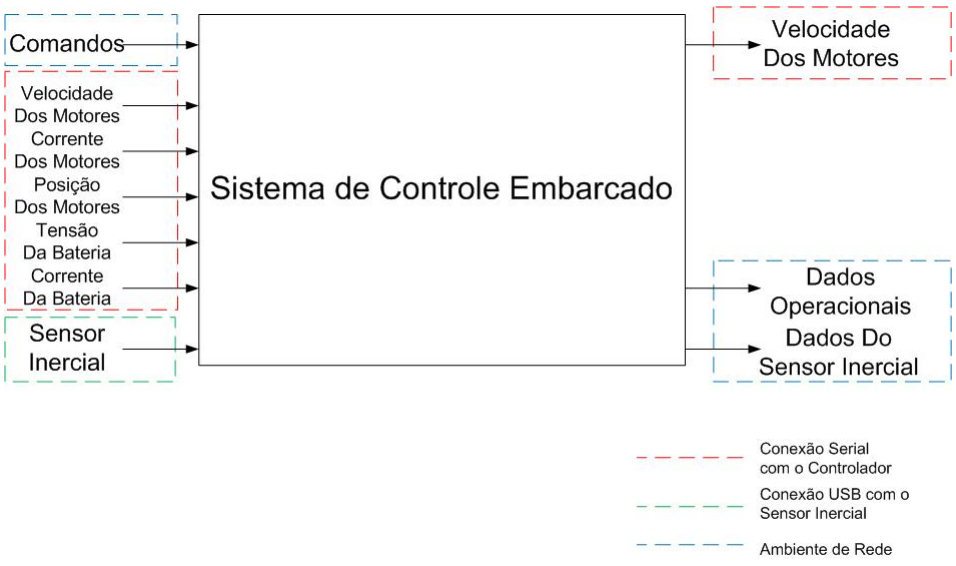

Figura 5.1: Sistema de controle embarcado.

A figura 5.1 apresenta ainda as interfaces do sistema com os outros sistemas do robô. Uma conexão USB é apresentada e será utilizada para receber as entradas provenientes do sensor inercial. Da mesma maneira uma conexão serial é utilizada para comunicação com o hardware de controle dos motores, e será responsável por uma série de entradas e pelas saídas referentes a comando dos motores. As conexões de rede das diversas portas TCP/IP são também apresentadas no desenho.

5.2 Requisitos do Sistema

Para que o sistema possa ser desenvolvido de maneira a satisfazer todas as necessidades de operação do robô é preciso primeiro fazer uma avaliação de quais são essas necessidades, o que possibilitará também a idealização da arquitetura do sistema.

O protótipo será avaliado conforme suas funcionalidades, sua instrumentação embarcada, e o funcionamento de suas partes mecânicas para que o sistema possa ser dimensionado e projetado.

Independente da arquitetura de sistema definida, também é objetivo que o sistema, através de poucas alterações, possa atender a novos protótipos que sejam desenvolvidos dentro do mesmo projeto, com maior ou menor quantidade de instrumentação e partes mecânicas comandadas, através da reutilização dos códigos.

5.2.1 Leitura de Sensores

O sistema precisa fazer a leitura nos diversos sensores existentes no robô para poder tanto a manutenção desses dados quanto enviá-los para a base de operação. A seguir são listados os dados que serão lidos:

1. Velocidade dos motores das quatro rodas do robô
2. Corrente dos motores das quatro rodas do robô
3. Posição dos oito motores das suspensões do robô
4. Dados da bateria de alimentação do robô (Tensão e Corrente)
5. Dados de orientação lidos pela central inercial do robô

5.2.2 Atuadores

O sistema deverá se comunicar com os drivers dos motores (hardware) do robô de forma a garantir a atuação que foi especificada pelo operador da base e que foi enviada na forma de comandos para esse sistema. Essas atuações são listadas abaixo:

1. Velocidade dos motores das quatro rodas do robô
2. Posição dos motores das suspensões do robô
3. Velocidade dos motores das suspensões do robô

5.2.3 Comunicação

O sistema deverá se comunicar com a base de operação para receber comandos e enviar dados. Para tal será necessário manter as conexões do tipo socket listadas abaixo:

1. Uma conexão para receber comandos provenientes da base de operação
2. Uma conexão para enviar dados de telemetria para a base de operação
3. Uma conexão para enviar dados de navegação para a base de operação

5.2.4 Tolerância a Falhas

O sistema deverá possuir condições mínimas de tolerância a falha, garantindo a segurança do protótipo.

Dessa forma, caso alguma falha ocorra o robô deve tentar contorná-la. Caso isso não seja possível, o mínimo que se espera do sistema é que ele realize a parada do protótipo, evitando qualquer acidente.

5.2.5 Implementação de Controles em Malha Fechada

O sistema deve permitir que sejam implementados controles em malha fechada, fazendo leituras nos sensores de realimentação, realizando o processamento necessário, e se comunicando com o hardware para que aconteça a atuação calculada.

5.2.6 Diagrama de Casos de Uso

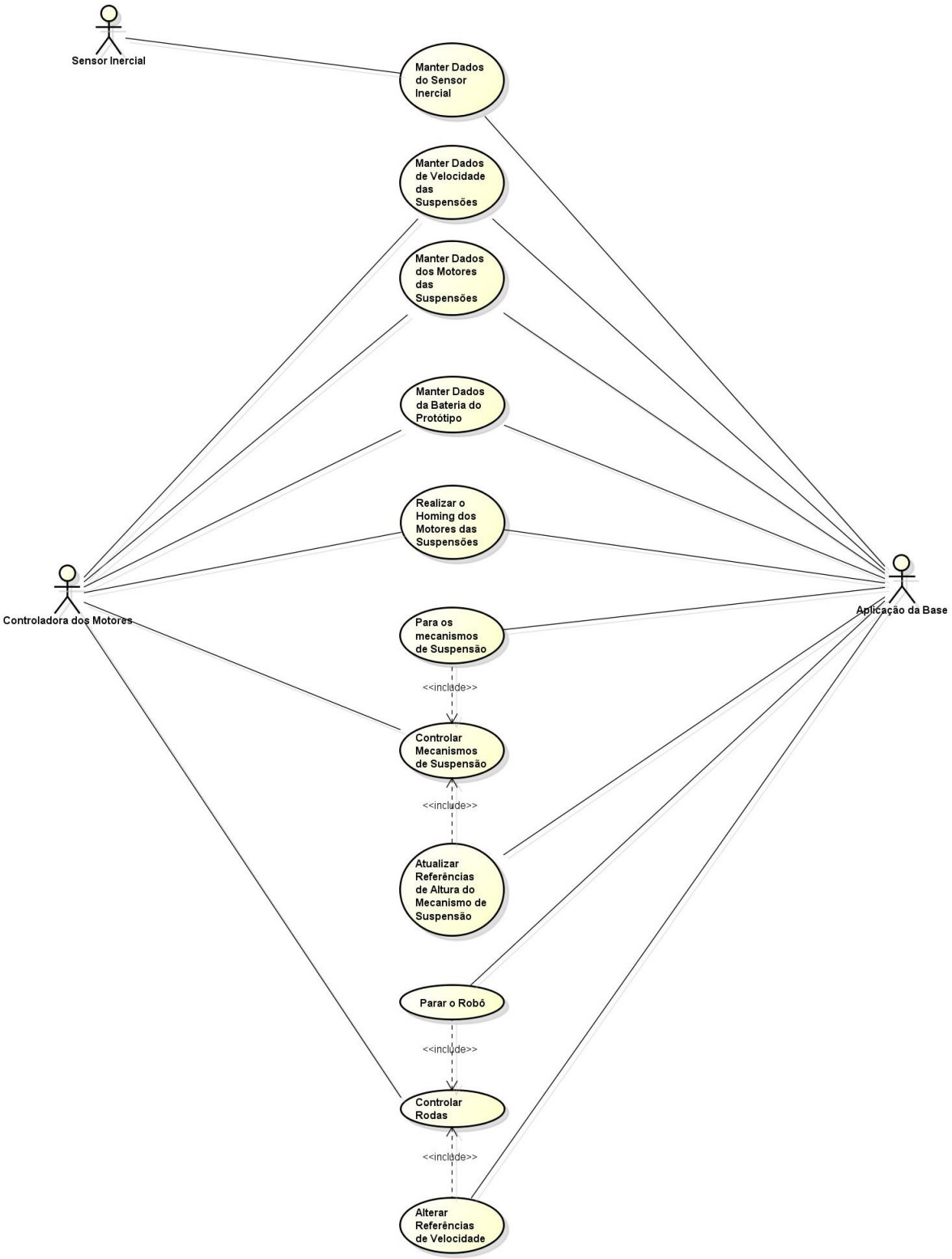

Figura 5.2: Diagrama de casos de uso do sistema.

Conforme já mencionado nas seções anteriores, o primeiro passo para projetar uma arquitetura para o sistema é entender os seus requisitos. Segundo [16], "...diagramas de caso de uso são tipicamente utilizados para capturar os requisitos de um sistema, isso é, o que um sistema supostamente deve fazer".

Dessa maneira, reunindo todos os requisitos listados nas seções anteriores podemos chegar a um diagrama de casos de uso para o sistema. Esse diagrama está representado na figura 5.2.

5.3 Arquitetura do Sistema

Foi definida uma arquitetura para o sistema de controle, que pode ser observada nas figuras 5.3 e 5.4, seus diagramas de componentes e implantação respectivamente. Essa arquitetura será melhor entendida através do entendimento de cada processo em específico.

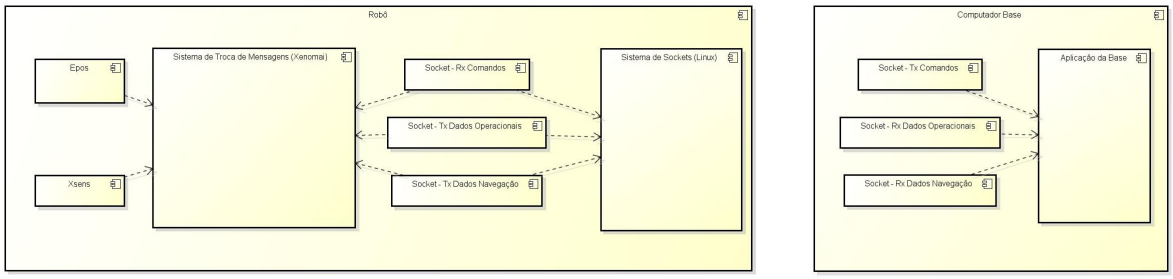

Figura 5.3: Diagrama de componentes do sistema.

5.3.1 Classes

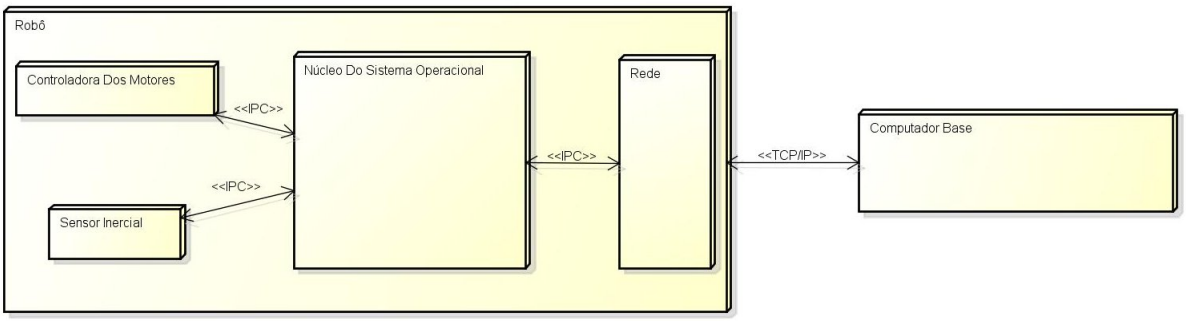

Figura 5.4: Diagrama de Implantação do sistema.

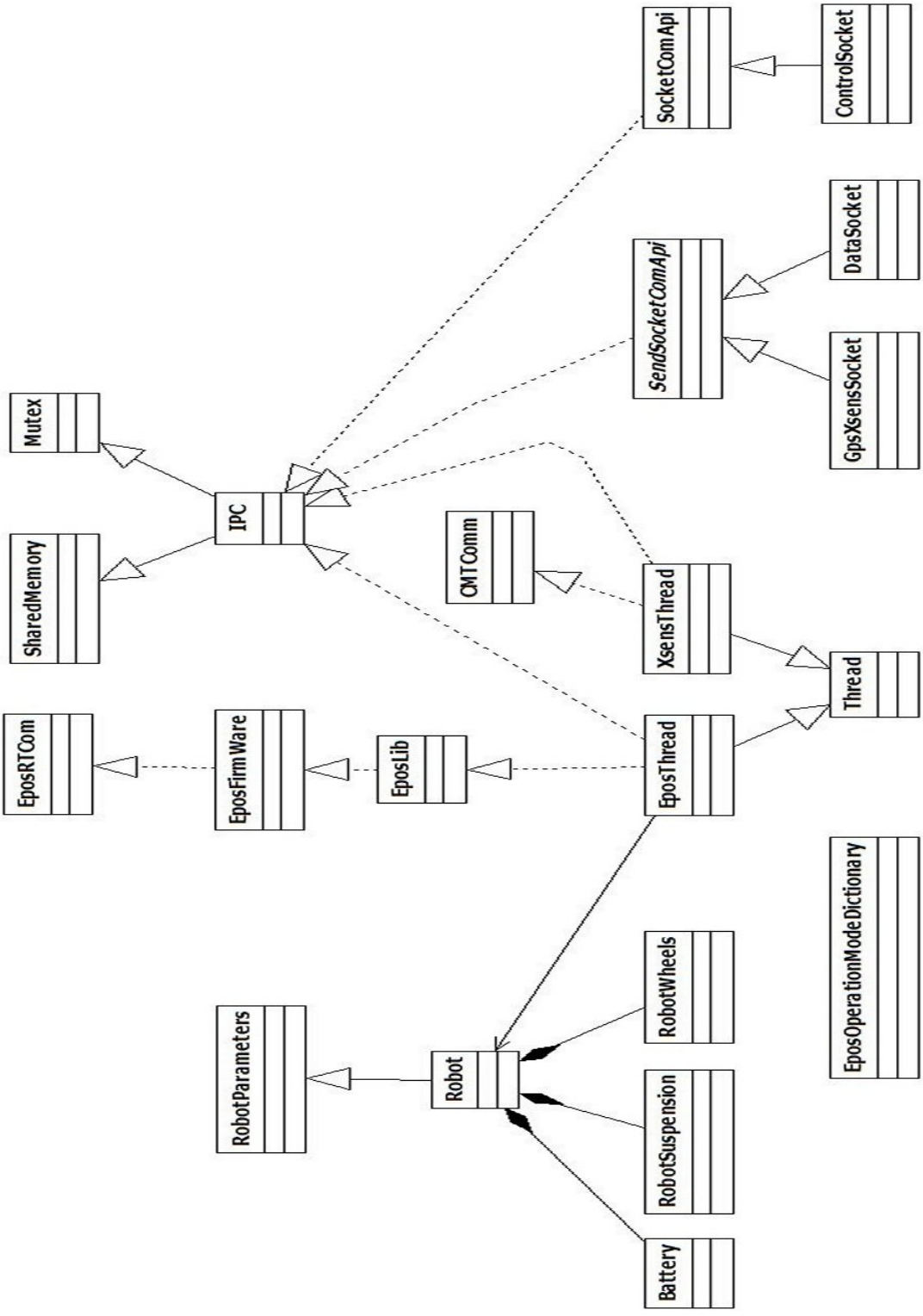

Figura 5.5: Diagrama da classes do sistema.

5.3.1.1 Mutex

A classe *Mutex*, representada na figura 5.6, implementa todos os métodos necessários para que seja utilizado o sistema de *Mutual Exclusion* do *co-kernel* Xenomai, funcionando como uma camada de abstração . A classe define ainda todos os *Mutex*'s utilizados no sistema do Robô.

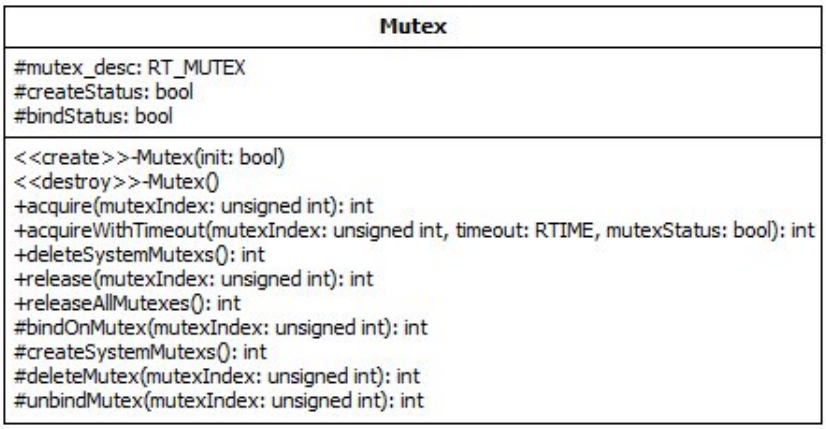

Figura 5.6: Diagrama da classe *Mutex*.

Métodos ressaltados:

- Mutex(bool init=false);

  O construtor da classe recebe uma variável booleana que por definição é *false*. Caso o construtor seja chamado com o valor lógico dessa variável sendo *true* serão criados todos os mutexs necessários para o sistema.

  Dessa forma, é necessário que o primeiro objeto dessa classe criado, quando o sistema for iniciado, receba essa variável como *true* para que posteriormente todos os objetos acessem os Mutex's já criados.

5.3.1.2 SharedMemory

A classe *SharedMemory*, representada na figura 5.7, implementa uma camada de abstração com todos os métodos necessários para que seja utilizado o serviço de

memórias compartilhadas através do Xenomai, de maneira análoga à classe *Mutex*. A classe define ainda todas as memórias compartilhadas que são necessárias para o sistema.

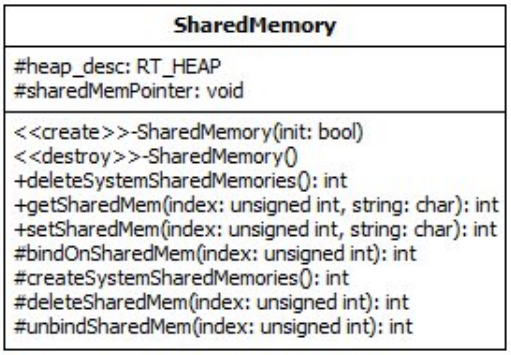

Figura 5.7: Diagrama da classe *SharedMemory*.

Métodos ressaltados:

- SharedMemory(bool init=false);

  Da mesma maneira que na classe *Mutex* o construtor da classe recebe uma variável booleana que por definição é *false* e caso o construtor seja chamado com o valor lógico dessa variável sendo *true* serão criados todas as memórias compartilhadas necessárias para o sistema.

  Dessa forma, também é necessário que o primeiro objeto dessa classe, quando criado no momento em que o sistema for iniciado, receba essa variável como *true* para que posteriormente todos os objetos possam acessar as memórias compartilhadas já criadas.

5.3.1.3 IPC

A classe *IPC*, representada na figura , implementa toda a estrutura necessária para a comunicação entre processos no sistema de controle do robô. Essa classe introduz também uma camada de proteção de exclusão mútua para o acesso às memórias

compartilhadas do sistema, evitando qualquer tipo de colisão indesejada no acesso às mesmas.

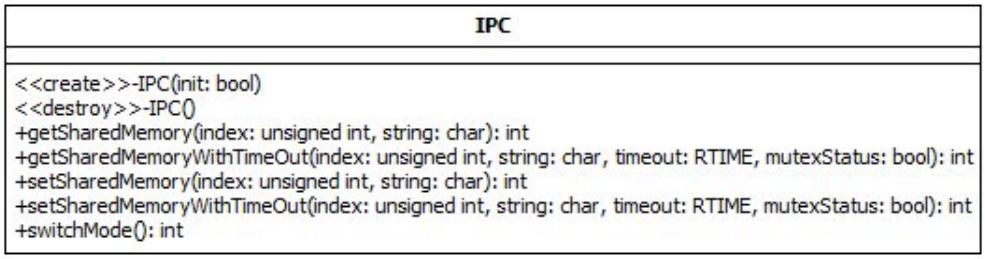

Figura 5.8: Diagrama da classe *IPC*.

Para implementar essa comunicação a classe irá se utilizar dos mecanismos implementados nas classes *Mutex* e *SharedMemory*.

Métodos ressaltados:

- IPC(bool init=false);

  Da mesma maneira que nas classes *Mutex* e *SharedMemory* o construtor da classe recebe uma variável booleana que por definição é *false* e que se tiver lógico *true* faz com que seja criada toda a estrutura de comunicação do sistema.

  Dessa forma, também é necessário que o primeiro objeto dessa classe, quando criado no momento em que o sistema for iniciado, receba essa variável como *true* para que posteriormente todos os objetos criados possams e utilizar da estrutura já criada.

5.3.1.4 SendSocketComApi

A classe *SendSocketComApi*, representada na figura 5.9, implementa de maneira abstrata todos os métodos necessários para a manutenção de uma conexão TCP/IP com o objetivo de enviar dados para a base de operação.

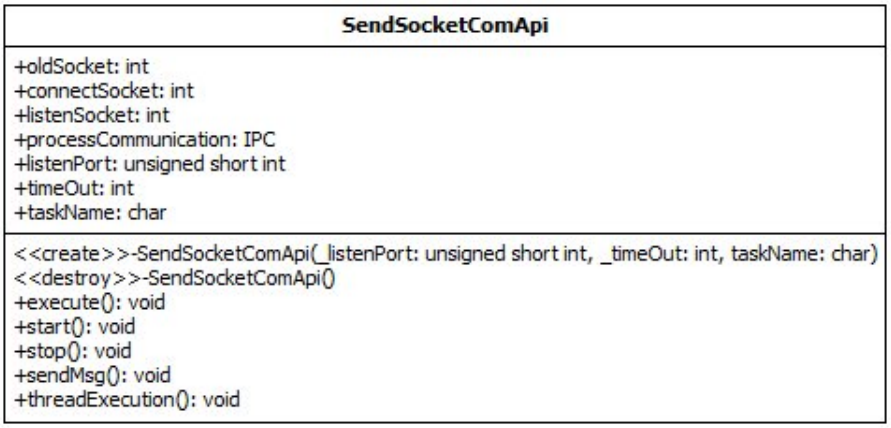

Figura 5.9: Diagrama da classe *SendSocketComApi*.

5.3.1.5 SocketComApi

A classe *SocketComApi*, representada na figura 5.10, implementa de maneira abstrata todos os métodos necessários para a manutenção de uma conexão TCP/IP com o objetivo de receber dados da base de operação.

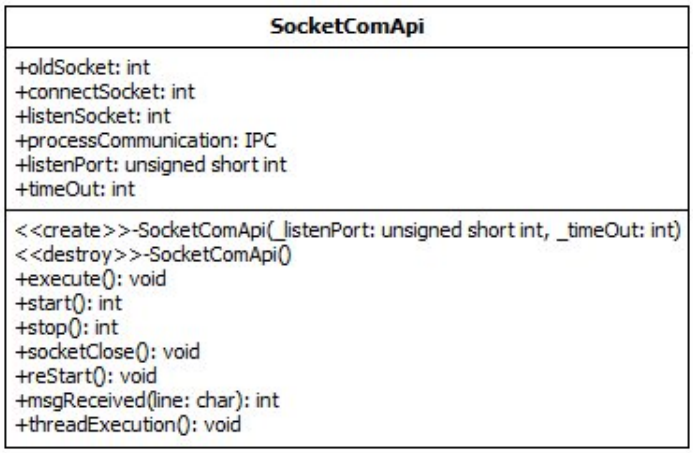

Figura 5.10: Diagrama da classe *SocketComApi*.

5.3.1.6 ControlSocket

A classe *ControlSocket* além de herdar todas as funcionalidades da classe *SocketComApi* implementa todos os métodos a mais necessários para a manutenção da

conexão TCP/IP responsável por receber comandos da base.

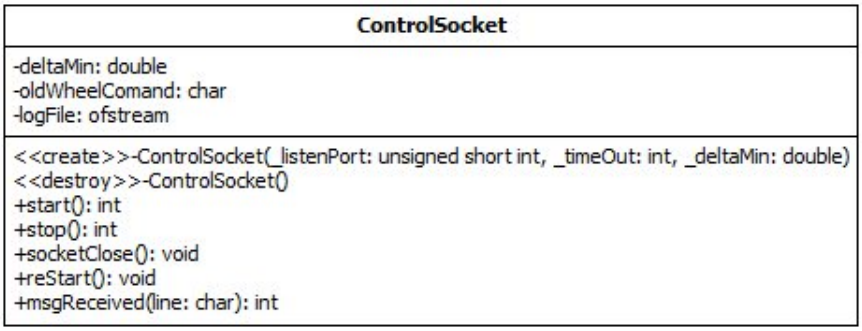

Figura 5.11: Diagrama da classe *ControlSocket*.

5.3.1.7 DataSocket

A classe *DataSocket*, representada na figura 5.12, além de herdar todas as funcionalidades da classe *SendSocketComApi* implementa todos os métodos necessários para a manutenção da conexão TCP/IP responsável por enviar os dados operacionais do robô para a base de operação.

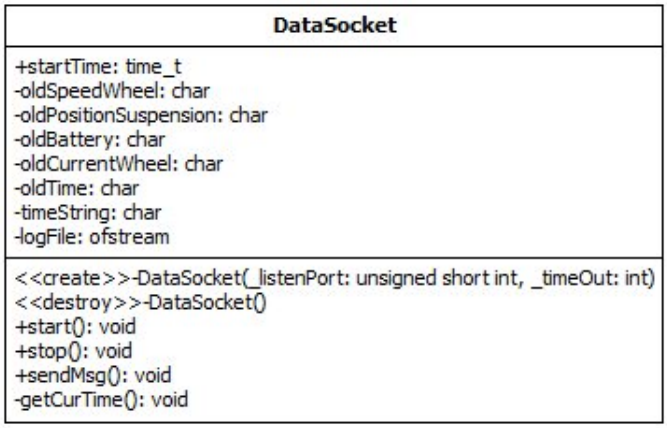

Figura 5.12: Diagrama da classe *DataSocket*.

5.3.1.8 GpsXsensSocket

A classe *GpsXsensSocket*, representada na figura 5.13, além de herdar todas as funcionalidades da classe *SendSocketComApi* implementa todos os métodos necessários

para a manutenção da conexão TCP/IP responsável por enviar os dados de navegação para a base de operação.

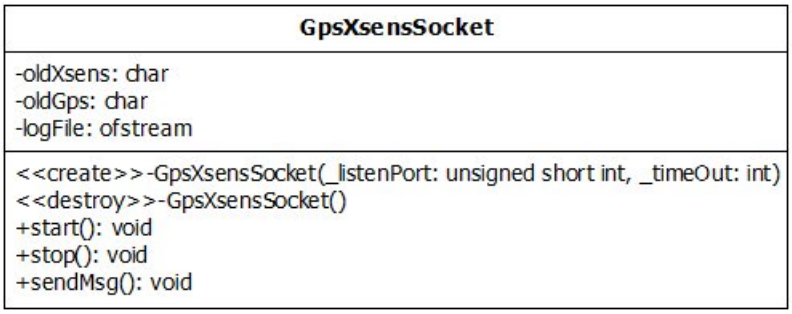

Figura 5.13: Diagrama da classe *GpsXsensSocket*.

5.3.1.9 Robot

A classe *Robot*, representada na figura 5.14, introduz uma camada de abstração para o controle dos mecanismos do robô. Ela implementa métodos que através das mensagens provenientes da base de operação, são capazes de acionar o fragmento do sistema necessário para realizar determinado comando.

Métodos ressaltados:

- int runPeriodicTasks(char *wheelCommand,char *suspensionCommand, char *dataReadMemory, char *onOrOffMemory);

  O método é responsável por executar todas as tarefas periódicas de controle no robô, inclusive de leitura dos dados. Esse é o método que deve ser chamado uma vez por loop.

- void get*****String(char *returnString);

  Os métodos do tipo *getString* são utilizados para receber strings com os dados lidos durante a execução do método *runPeriodicTasks*, e dessa forma o ideal é que sejam chamados depois da execução do mesmo.

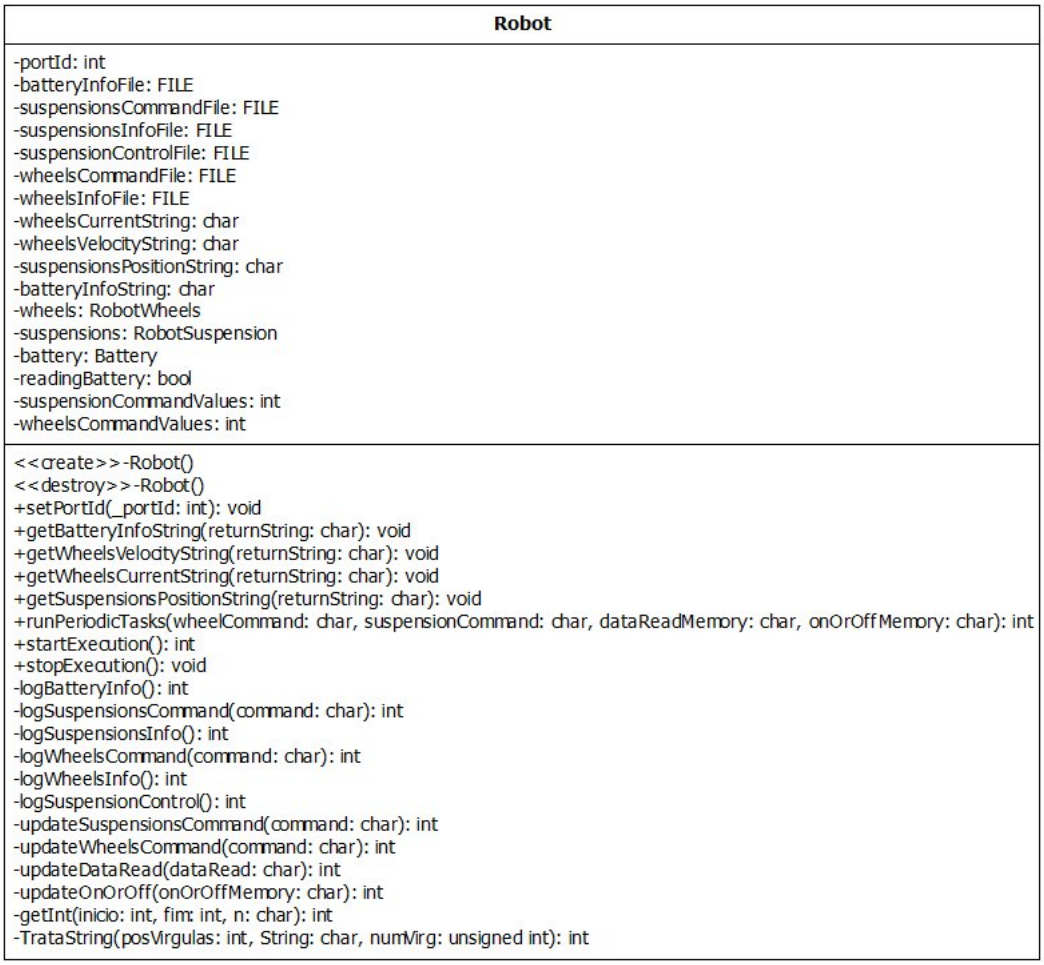

Figura 5.14: Diagrama da classe *Robot*.

5.3.1.10 RobotParameters

A Classe *RobotParameters*, representada na figura 5.15 define todos os parâmetros de configuração a respeito do robô, e sendo assim reúne todas as informações e características do protótipos necessárias para o funcionamento do sistema, como número de motores, configurações da rede de controladores, entre outras informações.

Essa classe pretende modularizar o código de forma que seja possível reutilizar um sistema em um novo protótipo, com mecanismos diferentes que sigam o mesmo conceito.

Figura 5.15: Diagrama da classe *RobotParameters*.

5.3.1.11 RobotSuspension

A classe *RobotSuspension*, representada na figura 5.16, implementa todos os métodos necessários para a manutenção periódica do controle dos mecanismos de suspensão do robô.

Métodos ressaltados:

- void doHoming();

  O método *doHoming* executa o procedimento de *homing* em todas as juntas de suspensão do robô, conforme descrito na seção 2.3.1.

- int stop();

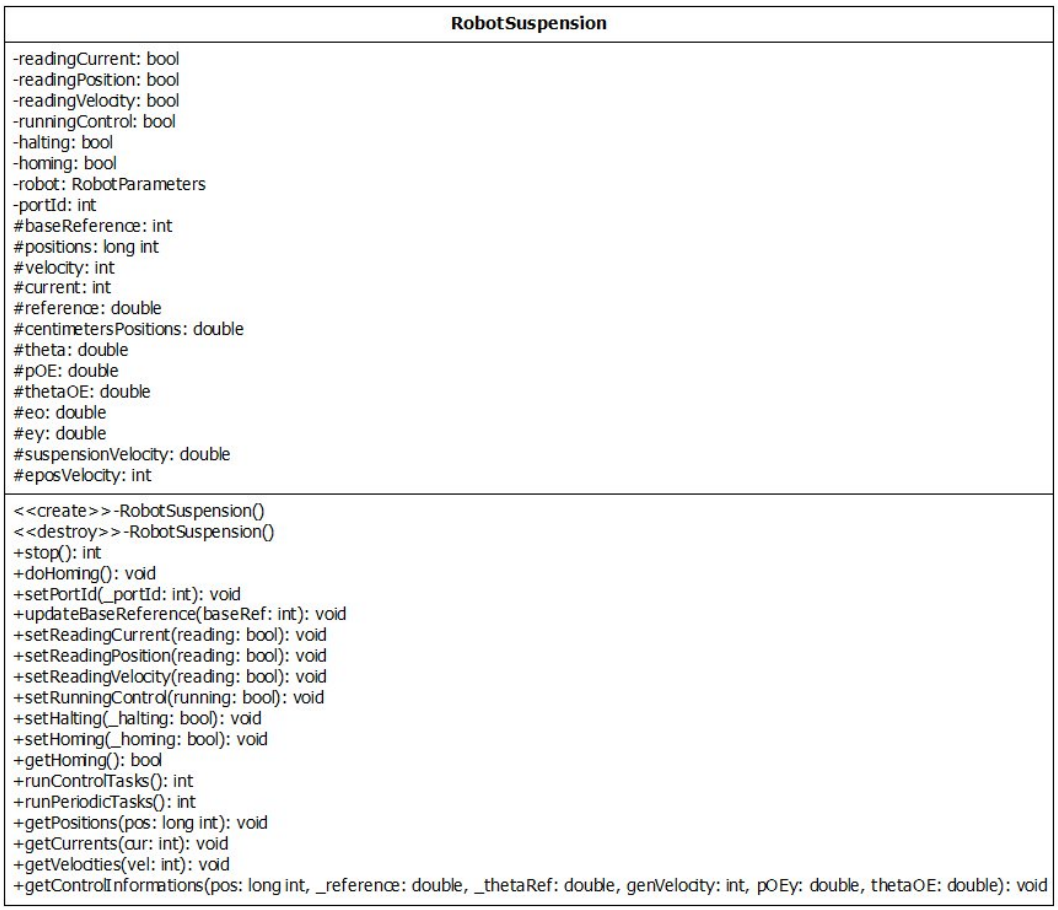

Figura 5.16: Diagrama da classe *RobotSuspension*.

O método *stop* executa a parada de todos os motores do mecanismo de suspensão do robô.

- int runPeriodicTasks();

O método *runPeriodicTasks* de maneira análoga ao da classe *Robot* executa todas as tarefas necessárias para a execução do loop de controle das suspensões do robô.

- int runControlTasks();

O método *runControlTasks* é o método responsável por todo o processamento, propriamente dito, do controle dos mecanismos de suspensão, para que sejam gerados os valores de comando para as juntas ativas do mecanismo.

5.3.1.12 RobotWheels

A classe *RobotWheels* implementa todos os métodos necessários para a manutenção periódica do controle das rodas do robô.

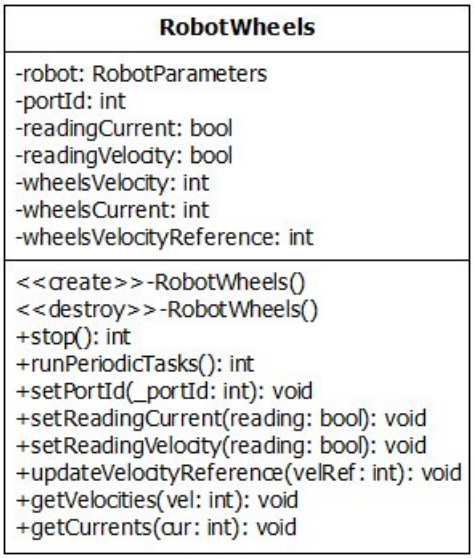

Figura 5.17: Diagrama da classe *RobotWheels*.

Métodos ressaltados:

- int stop();

  O método *stop* realiza a parada das rodas do robô.

- int runPeriodicTasks();

  O método *runPeriodicTasks* de maneira análoga aos das classes *Robot* e *RobotSuspension* realiza todas as tarefas necessárias para a execução de um loop de controle das rodas do robô.

5.3.1.13 Battery

A classe *Battery*, representada na figura 5.18, implementa todos os métodos necessários para que seja possível a leitura periódica dos dados da bateria do robô.

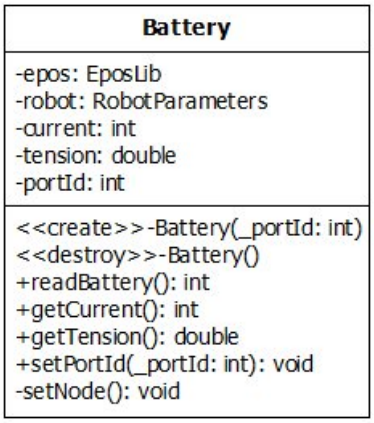

Figura 5.18: Diagrama da classe *Battery*.

5.3.1.14 Thread

A classe *Thread*, representada na figura 5.19, tem por objetivo implementar uma série de métodos comuns ao funcionamento de todas as *threads* que serão utilizadas no sistema de controle do robô.

Essa classe parte do a idéia de que embora as Threads do sistema estejam sempre em execução, elas podem se encontrar no estado *ativas* ou *inativas*.

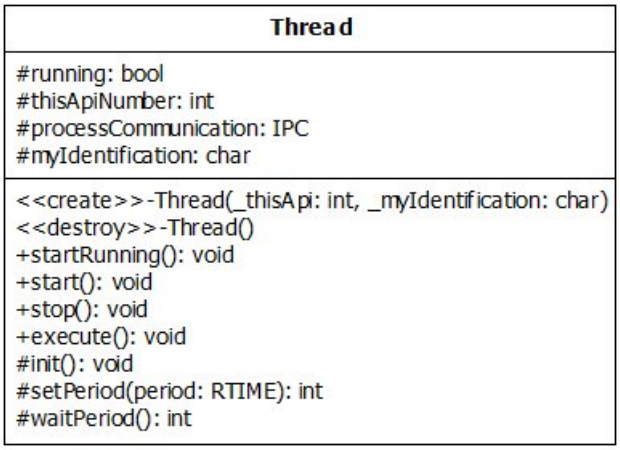

Figura 5.19: Diagrama da classe *Thread*.

Métodos ressaltados:

- void startRunning();

  Esse método deve ser chamado para iniciar a execução da Thread. A execução desse método não terá fim a não ser que alguma falha ocorra, ou o sistema tenha sua execução terminada.

- virtual void start();

  O método *start* implementa todas as tarefas necessárias de serem executadas quando a *thread* passa do estado *inativa* para o estado *ativa*.

- virtual void stop();

  O método *stop* implementa todas as tarefas necessárias de serem executadas quando a *thread* passa do estado *ativa* para o estado *inativa*.

- virtual void execute();

  O método *execute* implementa todas as tarefas periódicas que devem ser executadas quando a *thread* estiver no estado *ativa*.

5.3.1.15 XsensThread

A classe *XsensThread* implementa todos os métodos necessários para o funcionamento da *thread* que será responsável pela leitura dos dados do sensor inercial do robô.

5.3.1.16 EposThread

A classe *EposThread*, representada na figura 5.21, implementa todos os métodos necessário ao funcionamento da *thread* que será responsável pelo controle dos motores do robô, e dessa forma, deverá se comunicar com os diversos controladores no hardware do sistema.

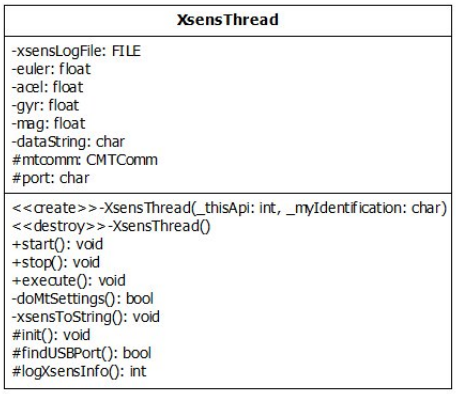

Figura 5.20: Diagrama da classe *XsensThread*.

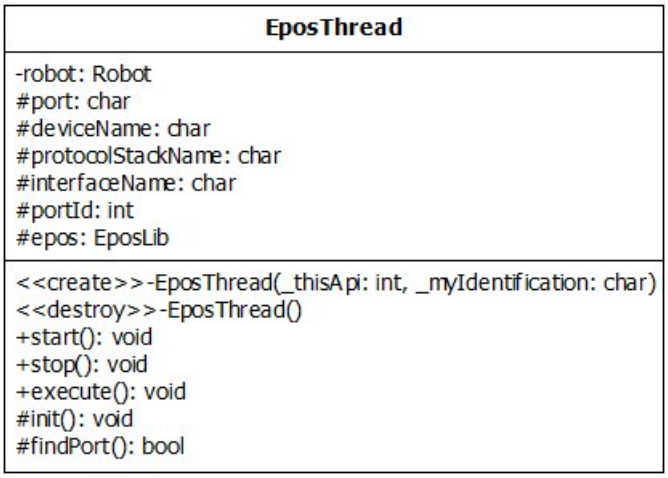

Figura 5.21: Diagrama da classe *EposThread*.

5.3.1.17 CMTComm

A classe *CMTComm* implementa todos os métodos para a comunicação com o sensor inercial Xsens. Os métodos de alto nível disponíveis são capazes de facilitar e tornar intuitiva a extração de dados deste sensor.

Essa classe não foi implementada no presente projeto. Ela representa na realidade uma biblioteca disponível junto ao sensor.

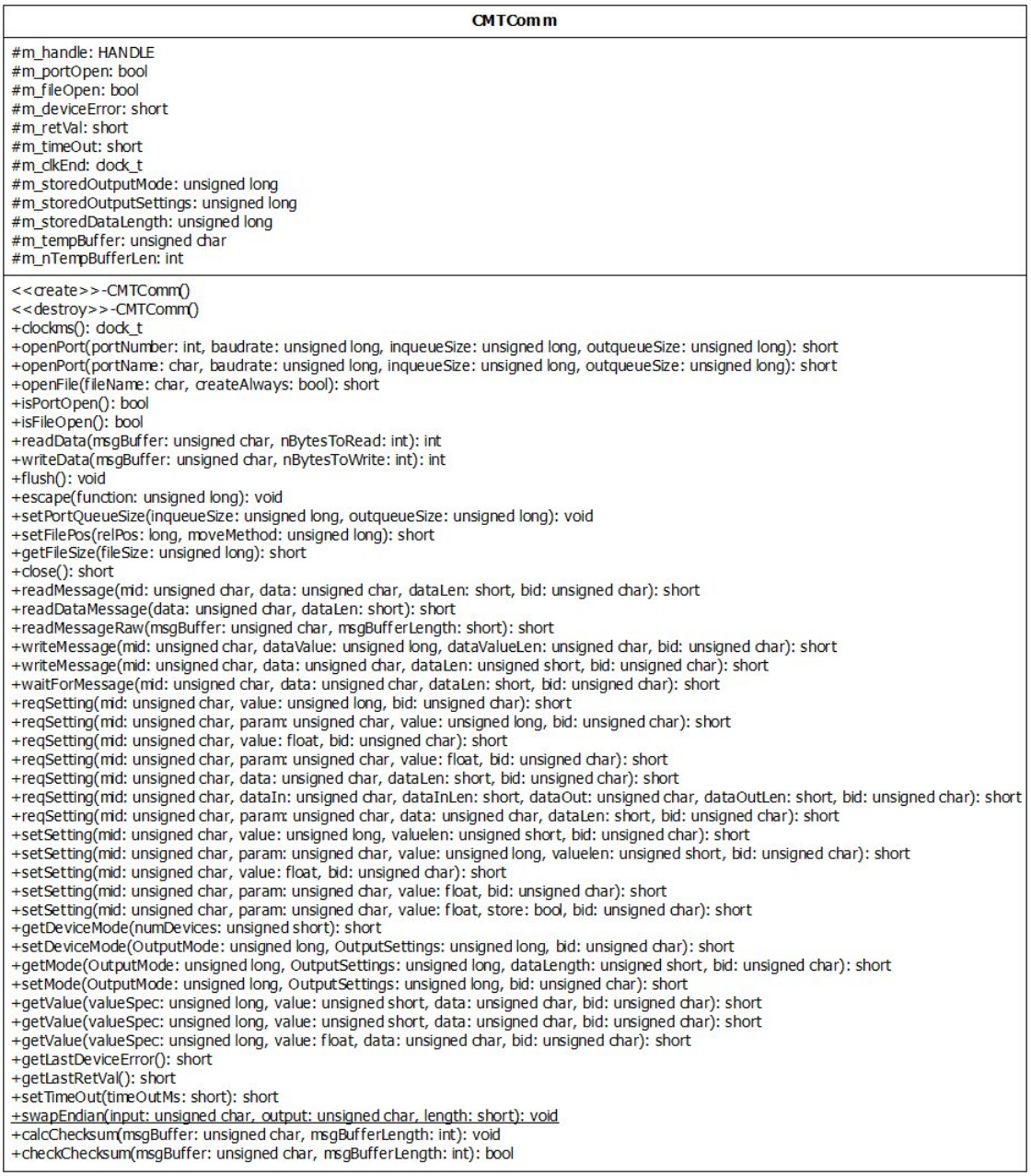

Figura 5.22: Diagrama da classe *CMTComm*.

5.3.1.18 EposRTCom

A classe *EposRTCom*, representada na figura 5.23, implementa a primeira camada de abstração para a comunicação com o hardware controlador EPOS. Os métodos implementam funções diretas para que se leiam ou escrevam determinados tipos de dados, como *integer* ou *unsigned*, em diferentes tamanhos em *bits*

O método *writeInteger32* por exemplo, escreve um inteiro de 32 bits em um endereço específico de *control word* do hardware.

A classe *EposCom* para uma simples comunicação serial com o hardware Epos já existia. Este projeto alterou a classe para *EposRTCom* introduzindo uma série de alterações que permitiram a comunicação serial através do *co-kernel* de tempo-real para que as latências fossem previsíveis e determinísticas.

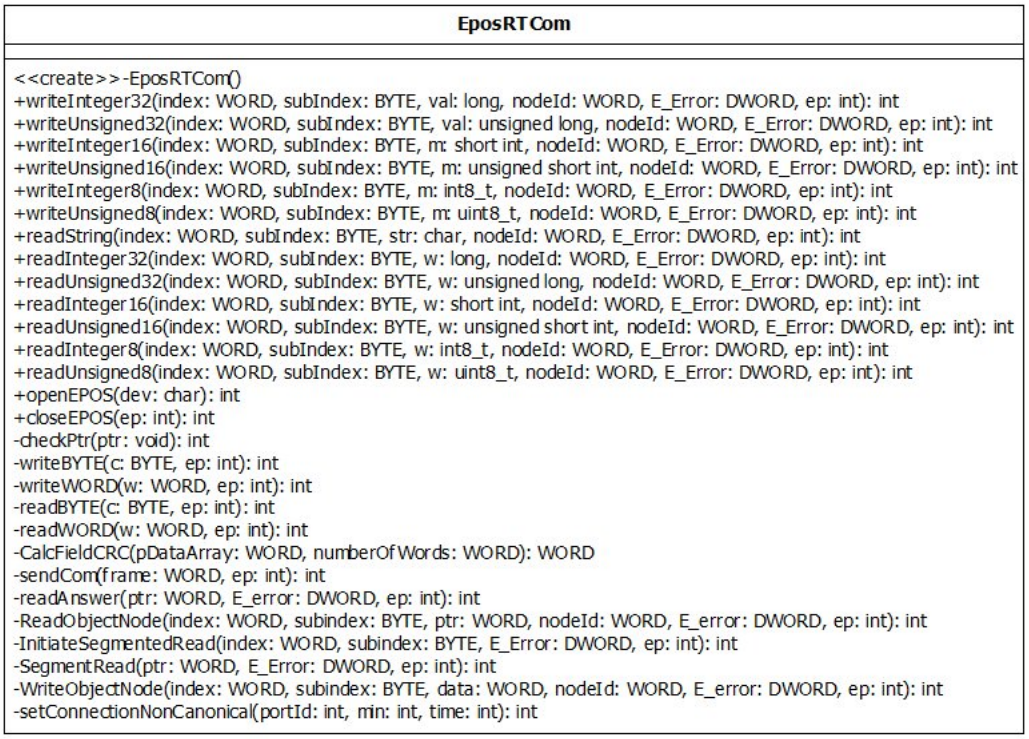

Figura 5.23: Diagrama da classe *EporRTCom*.

5.3.1.19 EposFirmWare

A classe *EposFirmWare*, representada na figura 5.24, oferece uma camada de abstração ao firmware da controladora EPOS e sua máquina de estados. Dessa forma, através dos métodos implementados é possível que seja abstraída a numeração dos objetos responsáveis por cada uma das funcionalidades do hardware.

O função da camada de abstração inserida por essa classe se torna mais compreensível quando um exemplo é observado. Suponha que seja necessário alterar a

referência de velocidade do motor para um valor qualquer. O método *VelocityModeSettingValue* deverá ser chamado, mas o método que está o chamando não precisa ter a informação de que a *control word* que precisa ser alterada tem endereço $0x206B$ e é um inteiro de 32 bits. Essas informações estão inseridas na classe.

A classe *EposFirmWare* já existia e foi apenas reutilizada nesse projeto.

5.3.1.20 EposLib

A classe *EposLib*, representada na figura 5.25, oferece uma camada de abstração, disponibilizando métodos de alto nível para o controle do hardware EPOS. Com os métodos dessa classe é possível comandar de maneira bem intuitiva o funcionamento dos motores acoplados à controladora EPOS.

Dessa forma, se por exemplo quisermos alterar a referência de velocidade do motor usaremos o simples método *VCSSetVelocityMust*.

A classe *EposLib* já existia e foi apenas reutilizada nesse projeto.

5.3.1.21 EposOperationModeDictionary

O hardware controlador dos motores possui uma série de diferentes modos de operação nos quais pode ser configurado. Para configurá-lo em um desses modos é necessário enviar via comunicação serial o comando para troca de modo de operação e o valor inteiro que identifica o modo para qual é necessário fazer a transição.

A classe *EposOperationModeDictionary*, representada na figura 5.26, implementa um dicionário para que sejam obtidos facilmente os valores inteiros que representam os modos de operação do hardware.

5.3.2 Processos

5.3.2.1 Epos

O processo *Epos* é responsável por gerenciar a comunicação através da porta serial com as controladoras dos motores embarcados, assim como rodar todos os algoritmos de controle executados no robô. Conforme discutido, esse processo só rodará

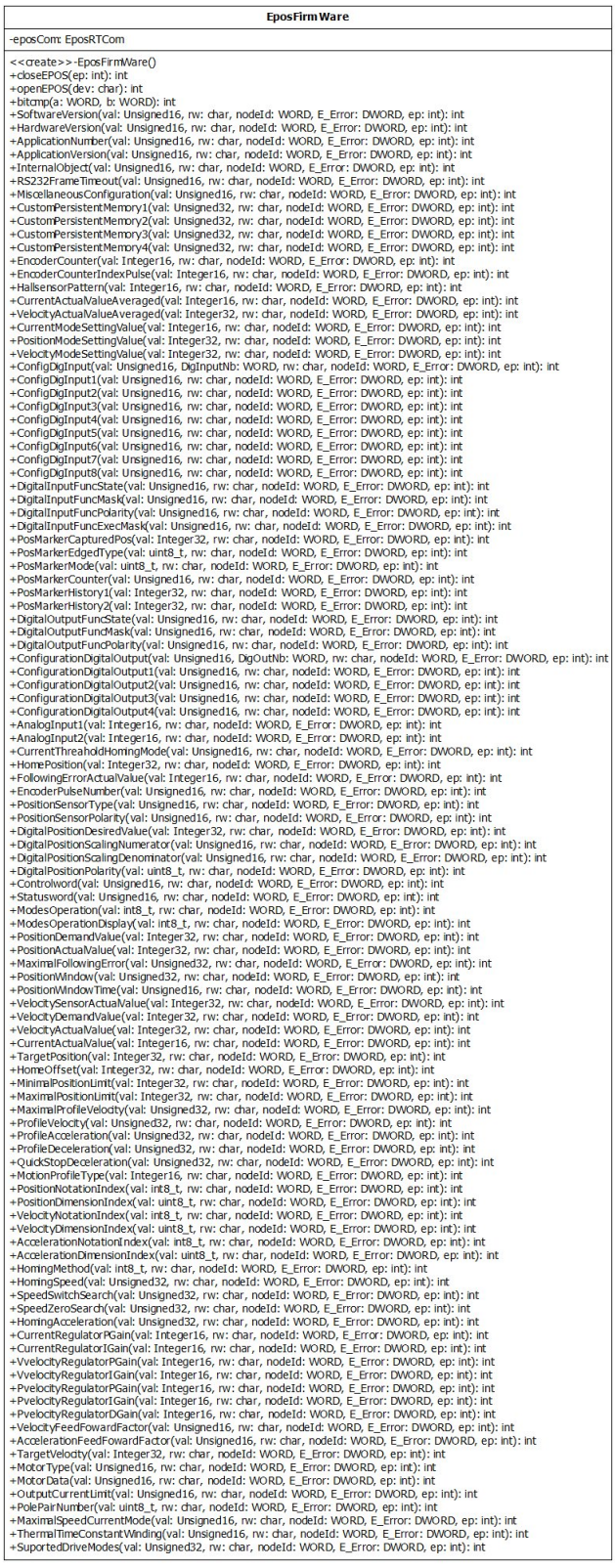

Figura 5.24: Diagrama da classe *EposFirmWare*.

Figura 5.25: Diagrama da classe *EposLib*.

chamadas de sistema ao Xenomai, de forma a não comprometer o determinismo do tempo de execução de seu loop de tarefas.

5.3.2.2 Xsens

O processo *Xsens* é responsável por gerenciar a comunicação com o sensor inercial do protótipo e fazer as leituras periódicas necessárias no mesmo.

Como o sensor inercial é conectado via USB com o PC/104, e o Xenomai da suporte apenas a comunicação serial, infelizmente esse processo terá que fazer chamadas ao sistema Linux durante sua execução, o que irá comprometer a performance de aquisição de dados desse sensor.

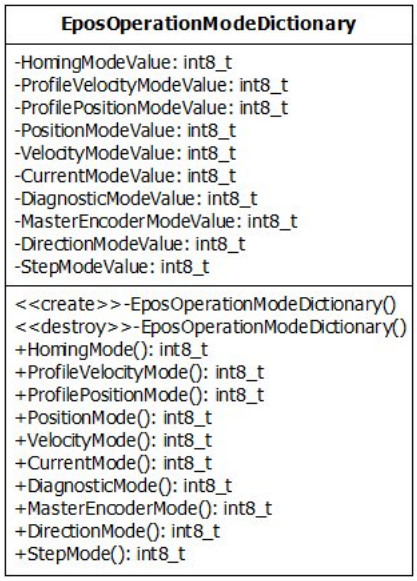

Figura 5.26: Diagrama da classe *EposOperationModeDictionary*.

Existem duas maneiras de solucionar esse problema, e essas serão enumeradas. A primeira seria desenvolver um driver de tempo-real para USB através da API RTDM do Xenomai, atividade complicada e trabalhosa que foge ao escopo do presente trabalho. A segunda solução possível seria conectar o sensor inercial ao PC104 através de comunicação serial, o que não será possível nesse trabalho devido ao fato do PUMA PC/104 utilizado ter apenas uma porta com suporte a RS232, que já é ocupada para comunicação com as controladoras dos motores.

Como no presente projeto as leituras de navegação ainda não realimentam nenhum tipo de controle, rodaremos essa tarefa Xsens no Xenomai, mas chegado o momento de comunicação faremos chamadas ao sistema Linux, perdendo um pouco da característica de tempo-real. Posteriormente esse problema deverá ser contornado com alguma das soluções apresentadas acima.

5.3.2.3 ControlSocket

O processo *ControlSocket* é responsável por gerenciar a comunicação socket TCP/IP que deve receber comandos vindos da base de operação do robô.

5.3.2.4 DataSocket

O processo *DataSocket* é responsável por gerenciar a comunicação socket TCP/IP que deve enviar todos os dados de tele-metria para a base.

5.3.2.5 GpsXsensSocket

O processo *GpsXsensSocket* é responsável por gerenciar a comunicação socket TCP/IP que deve enviar para a base todos os dados de navegação. No caso desse protótipo serão enviados os dados colhidos no sensor inercial.

5.4 Logs do Sistema

Esses arquivos terão nomes identificados pelo momento em que foram criados, que serão do tipo:

```
Ano,Mês,Dia,Hora,Minuto,Segundo,Tipo_Do_Arquivo
```

ControlSocket: Arquiva todas as mensagens de texto recebidas através da rede pelo processo *ControlSocket*.

DataSocket: Arquiva todas as mensagens de texto enviadas através da rede pelo processo *DataSocket*.

GpsXsensSocket: Arquiva todas as mensagens de texto enviadas através da rede pelo processo *GpsXsensSocket*.

WheelsCommand: Arquiva todos os comandos para as rodas do robô que foram recebidos pelo processo Epos através da estrutura de IPC do sistema.

SuspensionCommand: Arquiva todos os comandos para as suspensões do robô que foram recebidos pelo processo Epos através da estrutura de IPC do sistema.

WheelsInfo: Arquiva todos os dados coletados pelo processo Epos que dizem respeito às rodas do robô.

BatteryInfo: Arquiva todos os dados coletados pelo processo Epos que dizem respeito à bateria do robô.

SuspensionInfo: Arquiva todos os dados coletados pelo processo Epos que dizem respeito à suspensão do robô.

SuspensionControlInfo: Arquiva todos os dados referentes ao controle de suspensão, como dados lidos, resultados dos calculos das cinemáticas, referências, e comandos gerados.

XsensInfo: Arquiva todos os dados lidos pelo processo Xsens a partir do sensor inercial do protótipo.

xsens_task_loop: Arquiva todas as informações de tempo referentes ao loop Xsens: seu tempo de loop, tempo ativo e tempo em espera, para cada uma das iterações.

epos_task_loop: Arquiva todas as informações de tempo referentes ao loop Epos: seu tempo de loop, tempo ativo e tempo em espera, para cada uma das iterações.

Capítulo 6

Testes e Resultados

6.1 Introdução

Com o objetivo de validar o projeto e a implementação do sistema foi realizado um teste com o protótipo. Dessa forma o sistema de controle foi colocado em execução e seu desempenho pode ser avaliado.

O robô e sua estrutura de base foram deslocados para um gramado a céu aberto dentro do CENPES, onde o teste poderia ser realizado. O teste proposto era simples, e sua metodologia era de movimentar o robô pelo gramado assim como ajustar a altura de suas suspensões durante um longo intervalo de tempo, garantindo a não ocorrência de falhas.

Apesar do ambiente do teste ainda não ser semelhante ao imaginado para operação do robô já foi possível testá-lo em dois tipos diferentes de solo, o calçado e o gramado. Pode-se observar o robô durante os testes nas figuras 6.1, 6.2.

O teste realizado foi um sucesso, pois o protótipo atendeu a todos os comandos e não apresentou falhas, embora melhorias sejam possíveis e serão posteriormente apresentadas na seção 7.

Em fase posterior ao teste foram tratados os dados presentes nos arquivos de log do sistema, citados na seção 5.4, gerando os gráficos e dados apresentados nesse

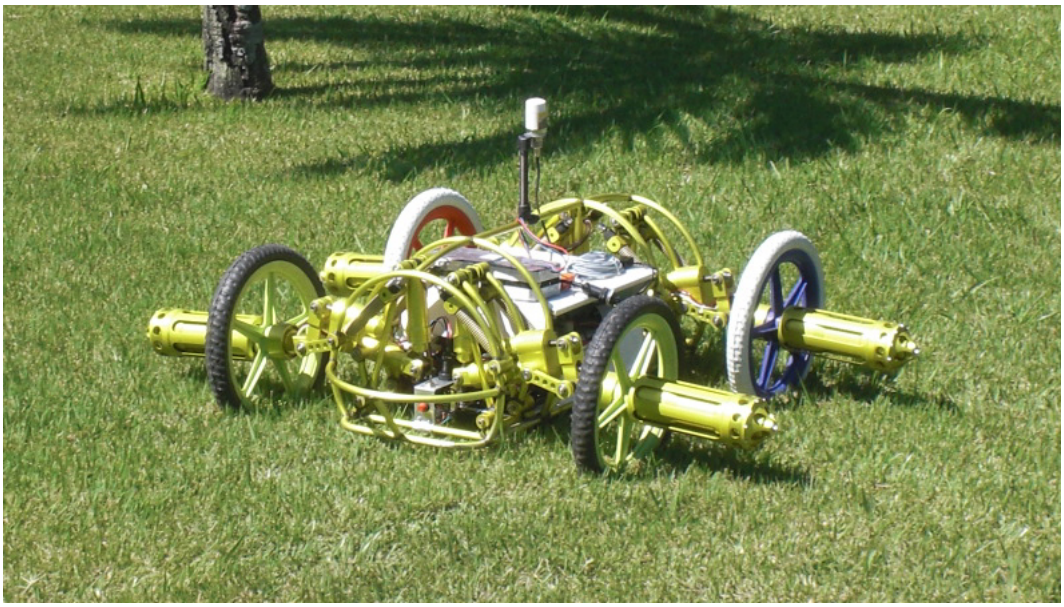

Figura 6.1: Robô durante testes no gramado do CENPES.

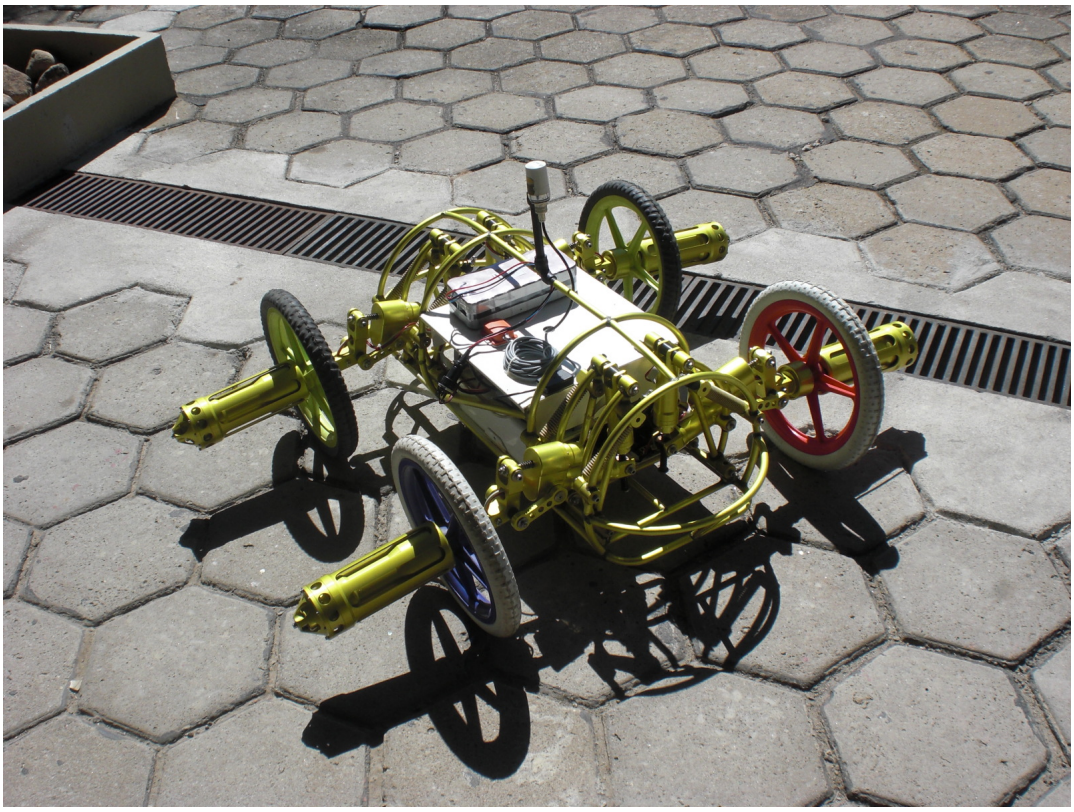

Figura 6.2: Robô durante testes em piso calçado no CENPES.

capítulo, que serão discutidos para comprovar o funcionamento e o desempenho do sistema nesse capítulo.

6.2 Tomada de dados

A primeira função a ser avaliada é a tomada dos dados necessários durante a execução do sistema.

Na figura 6.3 é apresentado o gráfico para a a tensão da bateria durante um dos momentos do teste. Na figura é possível observar um aspecto "ruidoso"no sinal lido, o que ocorre devido a influência do acionamento dos motores na tensão de saída da bateria, devido a frequente alteração na corrente fornecida pela mesma. É perceptível também, apesar do comentado aspecto, que o valo médio para a tensão da bateria está diminuindo, fato que poderia ser obviamente previsto.

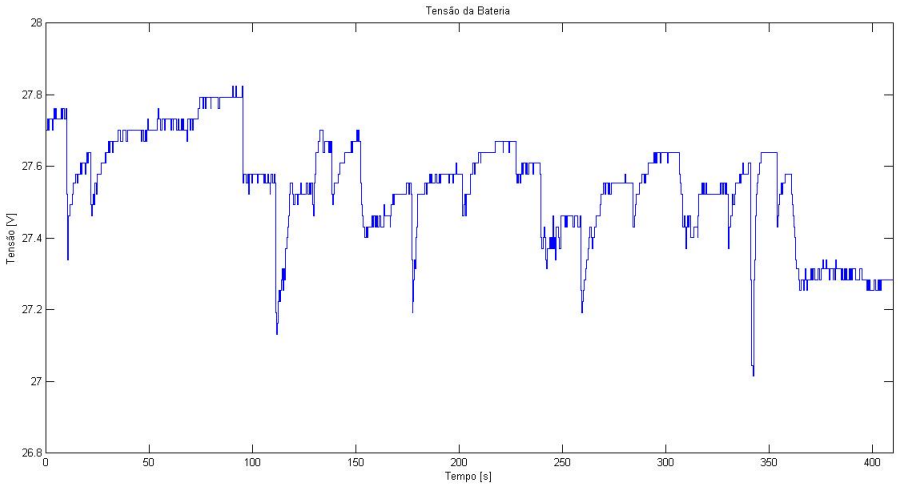

Figura 6.3: Tensão da bateria durante os testes.

A figura 6.4 apresenta a velocidade das quatro rodas do robô para um momento qualquer durante os testes, onde os comandos de velocidade estavam sendo gerados pelo software da base do robô com onde o mesmo era controlado através de um joystick.

É importante observar que todos os gráficos apresentando dados são confeccionados no padrão "sample-and-hold"para que possam ser observados as particularidades do controle digital. Obviamente esses pontos de dados coletados poderiam ser interpolados a fim de gerar uma melhor representação. A periodicidade dessas leituras

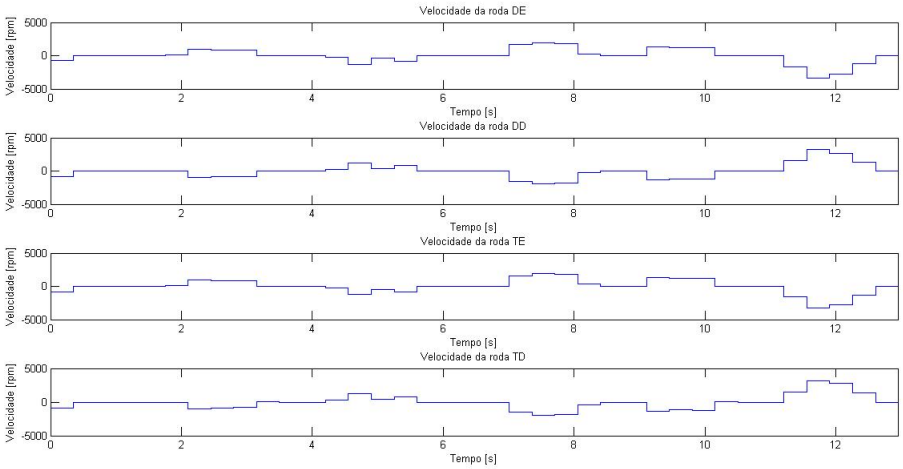

Figura 6.4: Velocidade das rodas durante os testes.

será discutida mais tarde nesse capítulo.

A figura 6.5 apresenta a corrente nas quatro rodas do robô para o mesmo período de teste apresentado na figura 6.4. É possível observar que os momentos onde existe a maior variação de velocidade são os momentos onde acontecem os maiores valores para a corrente.

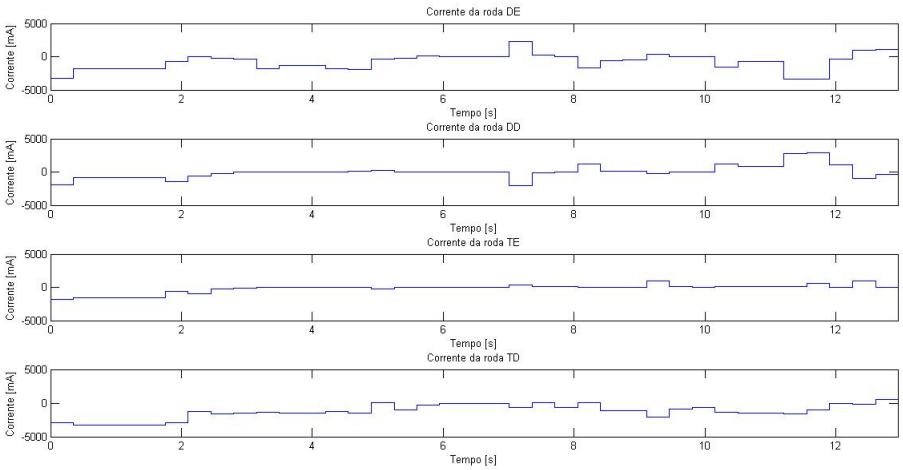

Figura 6.5: Corrente das rodas durante os testes.

A figura 6.6 apresenta a posição das oito juntas de suspensão em um determinado momento do teste. Durante esse período as juntas estavam sendo controladas sem o controle do mecanismo de suspensão. Dessa forma apenas era passada uma referência de posição para cada junta, que se locomovia até aquele determinado valor. Na próxima seção veremos os resultados apresentados para o controle do mecanismo de suspensão.

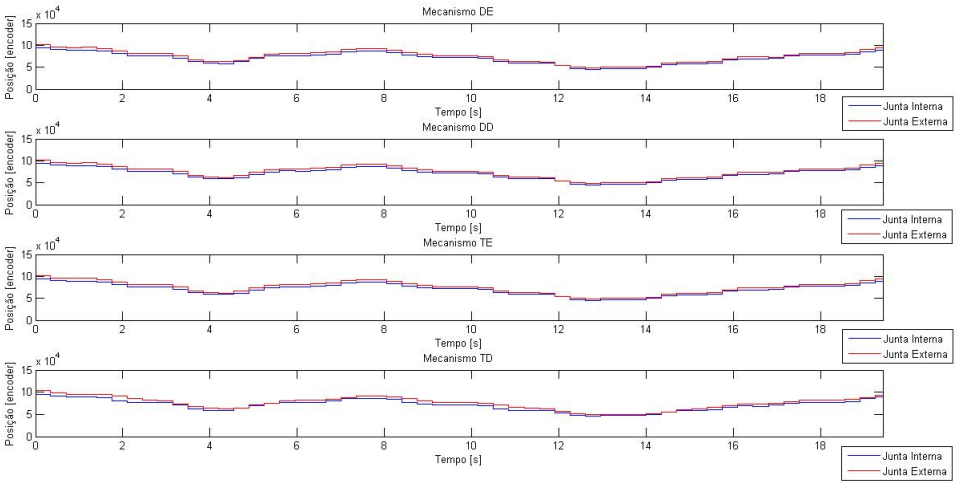

Figura 6.6: Posição das juntas de suspensão durante os testes.

6.3 Controle das Suspensões

Essa seção vai apresentar os resultados para a operação do controle dos mecanismos de suspensão. Durante o teste o controle foi acionado, e diversos valores de referência foram configurados, de forma a observar o comportamento do sistema e do protótipo.

A figura 6.7 apresenta um trecho do teste feito com o controle de suspensão ligado. No primeiro gráfico é possível observar a posição das juntas de suspensão, já convertida para centímetros. É possível observar a variação dessa posição em função do tempo.

No segundo gráfico apresentado é possível observar a referência sendo configurada para o controle e a altura convergindo para esse valor. Da mesma maneira no terceiro

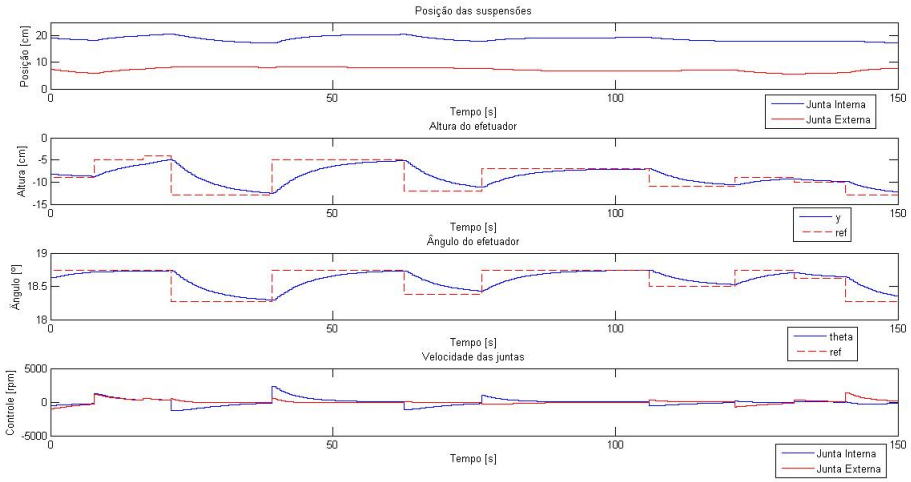

Figura 6.7: Resultados do controle do mecanismo de suspensão.

gráfico é passada uma referência e o ângulo do efetuador também converge para esse valor de referência. O último gráfico apresenta o sinal de controle (velocidade das juntas) gerado para cada instante de tempo, no decorrer do teste.

Os gráficos apresentados na figura 6.7 apresentam os resultados do controle do mecanismo de suspensão para apenas uma das suspensões, o que embora valide a implementação do controle não apresenta resultados para a posição do protótipo. É possível observar na figura 6.8, as alturas para cada uma das independentes suspensões e os ângulos lidos simultaneamente no sensor inercial do protótipo.

No primeiro trecho do gráfico, de 0 a 80s, percebemos que a mesma referência de altura é passada para os dois mecanismos dianteiros, e uma outra referência é passada para os dois mecanismos traseiros. Com isso é possível observar a variação nas leituras do ângulo de Roll do protótipo.

Em seguida, no trecho de 80s a 160s, uma mesma referência é passada para os mecanismos da esquerda, e uma outra referência é passada para os mecanismos da direita, e os efeitos podem ser observados nas leituras do ângulo de Pitch do protótipo.

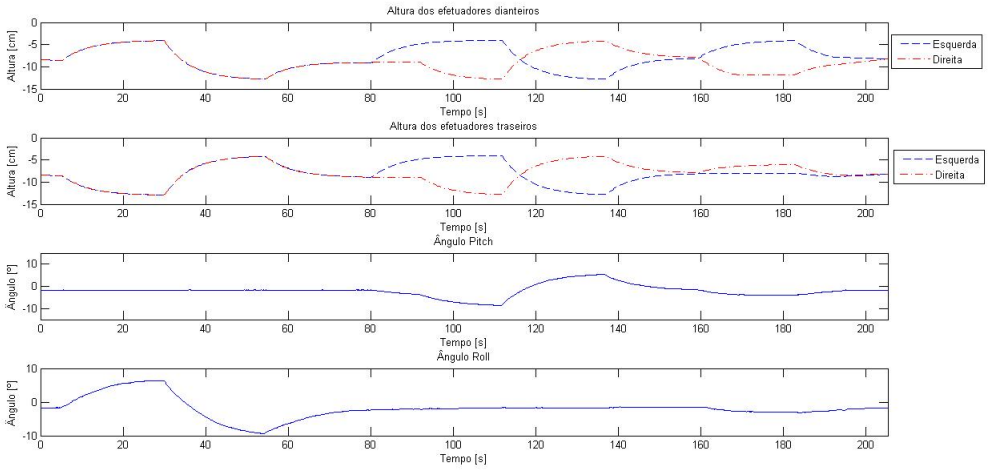

Figura 6.8: Ângulos de orientação do protótipo durante testes com o controle dos mecanismos de suspensão.

Por fim, no terceiro trecho, de 160s a 200s, uma diferente referência é gerada para cada um dos mecanismos e efeitos podem ser reparados nos ângulos de Roll e de Pitch.

Os dados apresentados comprovam que o robô, através da alteração da posição de seus mecanismos de suspensão pode alterar a maneira como interage com o ambiente, sua orientação, sua estabilidade, e até a tração entre o ponto de contato e o solo.

6.4 Análise do Tempo-Real

A parte mais importante do teste foi a medida dos parâmetros de tempo-real do sistema. É necessário uma métrica para garantir se o sistema tem ou não um bom desempenho. Os dados apresentados anteriormente provam que o sistema é capaz de realizar as tarefas propostas, mas ainda não foi provado se o uso do Xenomai como sistema operacional de tempo-real foi ou não uma vantagem.

Para avaliar a diferença de desempenho no Linux e no Xenomai os tempos referentes aos loops periódicos do sistema foram medidos durante a execução do mesmo, tanto no sistema Linux, como no sistema Xenomai.

Tabela 6.1: Medidas de tempo para o loop Epos rodando no Xenomai.

	Período de Loop	**Tempo Ativo**	**Tempo em Espera**
E[t]	349.95ms	285.63ms	64.35ms
σ	5.89ms	8.95ms	8.93ms
max(t)	451.44ms	387.17ms	116.49ms
Número de Medidas	2838		

Os tempos apresentados são o do período do loop, o tempo ativo durante o loop, e o tempo em espera durante o loop, o qual ocorre depois que o loop terminou de realizar suas tarefas periódicas e está aguardando o momento de as iniciar novamente.

Todos os valores medidos apresentam seu valor médio (E[t]), o desvio padrão associado (*sigma*), assim como o valor máximo medido.

A tabela 6.1 apresenta os tempos medidos para o loop Epos rodando no Xenomai. É possível observar que o tempo de loop médio, que foi configurado para ser de 350ms está bem próximo desse valor, e que o desvio padrão para essa medida é de cerca de 1,7%, um valor extremamente satisfatório para a variação no tempo de loop.

O pior caso medido durante os testes para execução desse mesmo loop, com o valor de 451.44ms, embora esteja 29% acima do valor determinado para o loop, ainda não é um período de tempo preocupante para que algum grande estrago ocorra, com os controles que estão rodando hoje nesse sistema.

Tabela 6.2: Medidas de tempo para o loop Epos rodando no Linux.

	Período de Loop	**Tempo Ativo**	**Tempo em Espera**
E[t]	1775.49ms	1186.08ms	605.52ms
σ	191.45ms	47.81ms	73.94ms
max(t)	2088.8ms	1465.12ms	627.71ms
Número de Medidas	98		

A tabela 6.2 apresenta os tempos medidos para o mesmo loop Epos, mas agora com as chamadas de sistema sendo realizadas ao Linux, e não ao Xenomai. O valor médio para o loop, configurado para ser de 1800ms, foi de 1775,49ms, e o desvio padrão das medidas foi de 10,78% do valor médio.

O tempo máximo para as medidas foi de 2088,8ms, cerca de 2 segundos. Esse valor já é bastante alto, uma vez que o tempo para que um comando de parada ser realizado no robô é totalmente dependente do período desse loop.

Uma outra peculiaridade interessante observada nos valores apresentados nessa tabela, é que o desvio padrão na medida do tempo de loop ativo é relativamente baixo, cerca de 4%. Isso mostra que a falta de determinismo no sistema Linux não é determinada pelo tempo em que o sistema leva para realizar as tarefas programadas, mas sim pela falta de determinismo no momento em liberar uma tarefa que está em espera e torná-la ativa.

Tabela 6.3: Medidas de tempo para o loop Xsens rodando no Xenomai.

	Período de Loop	**Tempo Ativo**	**Tempo em Espera**
E[t]	10ms	2.27ms	7.73ms
σ	1.29ms	1.83ms	1.8ms
max(t)	29.83ms	24.79ms	12.64ms
Número de Medidas	100005		

Tabela 6.4: Medidas de tempo para o loop Xsens rodando no Linux.

	Período de Loop	**Tempo Ativo**	**Tempo em Espera**
E[t]	10ms	1.25ms	8.76ms
σ	1.09ms	1.15ms	0.81ms
max(t)	95.93ms	85.66ms	11.26ms
Número de Medidas	10481		

As tabelas 6.3 e 6.4, apresentam os valores medidos para o loop Xsens, de leitura do sensor inercial, para o sistema Xenomai e Linux, respectivamente.

Conforme foi discutido no capítulo 5, todas as chamadas de comunicação quando o loop rodava no Xenomai já eram feitas ao Linux, o que nos faria prever que os tempos medidos seriam bastante semelhantes nos dois casos.

O único valor que pode ser discutido nos dados apresentados é o fato de que rodando no Linux o tempo máximo foi de cerca de 10 vezes maiores que no Xenomai. Esse valor provavelmente reflete o fato de que quando o teste do sistema foi feito no Linux, o mesmo estava gerenciando as comunicações de todos os dispositivos de hardware, o que levou a algumas situações onde um atraso maior foi gerado.

Dessa maneira avaliando todas as medidas realizadas, e apresentadas nessa seção nas últimas quatro tabelas, podemos perceber que o Xenomai apresentou vantagens em relação ao Linux nos padrões de tempo-real, como já era esperado.

Capítulo 7

Conclusões e Trabalhos Futuros

O projeto acabou por obter sucesso uma vez que o robô foi controlado e o sistema é capaz de realizar todas as tarefas designadas. Um vídeo demonstrando o protótipo em funcionamento será apresentado durante a defesa desse trabalho.

No capítulo 6 resultados de testes com o sistema foram observados mostrando que o Xenomai realmente apresenta vantagens sobre o Linux para a aplicação em questão, o desenvolvimento de sistemas de controle para robôs.

Ficou entendido também, através dos resultados apresentados no capítulo 6, que é necessário para as tarefas de tempo-real usarem durante toda sua execução chamadas ao sistema Xenomai, e não ao Linux, e os problemas listados devem ser contornados para que a tarefa *Xsens* ou qualquer outra tarefa que seja posteriormente desenvolvida para integrar ao sistema possam fazê-lo.

É importante perceber que o desenvolvimento do presente projeto representou apenas a introdução dos estudos em uma importante área de pesquisa, que pode gerar muitos frutos e utilidades para a sociedade, conforme descrito no capítulo 1. Uma série de trabalhos futuros pode ser desenvolvida a partir deste trabalho para que o Robô Ambiental Híbrido possa ser aprimorado e se torne o mais próximo possível do estado da arte das tecnologias utilizadas, e que a cada dia mais se torne uma ferramenta para pesquisa considerada de ponta.

Um dos trabalhos necessário é a instalação no PC/104 de uma placa de comunicação CAN para que seja possível se comunicar diretamente com todo o barramento de controladoras de motores, diminuindo o tempo de comunicação, de forma a permitir a diminuição do período dos controladores.

Com a comunicação utilizada através de portas CAN seria possível também utilizar a estrutura presente no xenomai chamada RT_SOCKET_CAN. Através dela seria possível utilizar a estrutura de sockets, onde diversas *tasks* podem utilizar a mesma porta física de comunicação, permitindo assim dividir cada algoritmo de controle em uma diferente tarefa.

Um outro trabalho futuro possível é a utilização da teoria de Sistemas a Eventos Discretos, como presente em [17], para a modelagem desse sistema de controle para o Robô Ambiental Híbrido.

Estudos referentes a tolerância a falha devem também ser realizados, para garantir o funcionamento integro do sistema e até mesmo algum tipo de reestruturação em caso da ocorrência de falhas.

É importante observar que o presente trabalho proporciona a infra-estrutura necessária para a implementação de controles em tempo real. Outros algoritmos de controle podem ser desenvolvidos, como por exemplo o controle de direção e de trajetória do robô, garantindo que tarefas comecem a ser feitas de maneira autônoma.

Controles que levem em conta a dinâmica do veículo podem também ser implementados. Um controle pode por exemplo ser proposto fazendo uma varredura dos obstáculos no trajeto e agendando variações no controle das suspensões em função da velocidade de translação do robô, de forma a melhorar a interação do robô com o ambiente.

Referências Bibliográficas

[1] IAGNEMMA, K., DUBOWSKY, S., *Mobile Robot In Rough Terrain*. Springer, 2004.

[2] BARRERA, J., "Mexico Uses Robot to Explore Ancient Tunnel", http://abcnews.go.com/Technology/wireStory?id=12114320, 2010, (Acesso em 20 de Fevereiro 2011).

[3] FREITAS, G. M., *Infra-estrutura para locomoção do Robô Ambiental Híbrido*, Projeto de fim de curso, Universidade Federal de Santa Catarina, 2006.

[4] UFAM, "PIATAM", http://www.piatam.ufam.edu.br, 2010, (Acesso em 15 Maio 2010).

[5] COGNITUS, "COGNITUS", http://www.cognitus.org, 2010, (Acesso em 15 Maio 2010).

[6] FIGUEIREDO, B. B. D., *Robô Ambiental Híbrido: Desenvolvimento do primeiro protótipo*, Projeto de fim de curso, Pontifícia Universidade Católica do Rio de Janeiro, 2006.

[7] MAXON, "MOTORS", http://www.maxonmotor.com, 2011, (Acesso em 10 de Maio 2011).

[8] TANENBAUM, A. S., *Modern Operating Systems*. Prentice-Hall, 2009.

[9] YAGMOUR, K., MASTERS, J., BEN-YOSSEF, G., *et al.*, *Construíndo Sistemas Linux Embarcados*, O´Reilly. Alta Books, 2009.

[10] PINHEIRO, B. C., *Sistema de Controle Tempo Real Embarcado Para Automação de Manobra de Estacionamento*. M.Sc. dissertation, Universidade Federal de Santa Catarina, 2009.

[11] YAGHMOUR, K., "Adaptive Domain Environment for Operating Systems", .

[12] XENOMAI, *Xenomai Native skin API*. 2 ed., Maio 2010.

[13] XENOMAI, *A Tour of The Native API*. Revc ed., Março 2006.

[14] OPENCV, "Open Source Computer Vision", http://opencv.willowgarage.com/, 2010, (Acesso em 26 Agosto 2010).

[15] LAPLANTE, P. A., *Real-Time System Design and Analysis*. IEEE-Press, 2004.

[16] DUC, B. M., *Real-Time Object Uniform Design Methodology with UML*. Springer, 2007.

[17] CASSANDRAS, C. G., LAFORTUNE, S., *Introduction to Discrete Event System*. Springer, 2008.

Anexo A

Instalação do Xenomai no PUMA PC104

A.1 Introdução

Esse apêndice tem por objetivo apresentar um tutorial de instalação do Linux com o pacote do framework Xenomai utilizado para que o sistema apresentado pelo presente projeto funcione. Trata-se de um tutorial passo a passo para que qualquer usuário possa fazer a sua primeira instalação, e posteriormente, com um maior entendimento do sistema possa refinar o processo para o que desejar.

A.2 Instalando o Linux

O primeiro passo, que não será explicitado nesse tutorial é fazer a instalação do sistema operacional Debian. A versão 5.0.1 é sugerida, uma vez que foi utilizada no desenvolvimento deste projeto, e seu funcionamento foi testado. Nada impede que outra versão seja utilizada.

Para a instalação do Debian recomenda-se que não seja instalada a interface gráfica, e que o pacote mínimo seja instalado, evitando que o sistema tenha seu funcionamento atrasado por utilidades desnecessárias.

O próximo passo é conectar o PC/104 a internet. Supondo o uso de servidor DHCP, não é necessário fazer nenhuma configuração especial. Caso o computador

seja conectado na internet através da rede PETROBRAS, será necessário configura-lo para utilizar o servidor proxy, com a chave e senha de acesso a rede do usuário. Os comandos a seguir devem ser utilizados:

```
# export ftp_proxy=http://<chave>:<senha>@10.2.108.27:8080
# export http_proxy=http://<chave>:<senha>@10.2.108.27:8080
```

Uma vez a configuração realizada, vamos começar a instalar todos os pacotes necessários para o sistema operacional. Uma lista extensa é apresentada, e deve ser toda executada:

```
# apt-get update
# apt-get upgrade
# apt-get install libc6
# apt-get install gcc
# apt-get install g++
# apt-get install mc
# apt-get install build-essentials
# apt-get install libst-dev
# apt-get install libpth-dev
# apt-get install binutils-dev
# apt-get install binutils-source
# apt-get install make
# apt-get install lftp
# apt-get install bzip2
# apt-get install ssh
# apt-get install libncurses-dev
# apt-get install ncurses-dev
# apt-get install initramfs-tools
# apt-get install wget
# apt-get install libavformat-dev
# apt-get install libgtk2.0-dev
# apt-get install pkg-config
# apt-get install cmake
```

```
# apt-get install libswscale-dev
# apt-get install setserial
```

Feito isso o sistema operacional básico está instalado, e podemos começar a instalação do Xenomai.

A.3 Compilando um novo Kernel e Instalando o Xenomai

Vamos compilar um novo kernel para o Linux, associado ao co-kernel do Xenomai. Para isso primeiro vamos baixar todos os arquivos necessários, através dos comandos:

```
# cd /usr/src/
# lftp
lftp:~> lftp http://www.kernel.org/pub/linux/kernel/v2.6/
lftp:~> get linux-2.6.37.tar.bz2 /usr/src/
lftp:~> lftp http://download.gna.org/xenomai/stable
lftp:~> get xenomai-2.5.6.tar.bz2 /usr/src/
lftp:~> lftp http://download.gna.org/adeos/patches/v2.6/x86
lftp:~> get adeos-ipipe-2.6.37-x86-2.9-00.patch /usr/src/
lftp:~> exit
```

Importante observar que para novas instalações podem existir versões mais recentes para os pacotes, que poderão ser utilizadas. Deve ser observado a compatibilidade entre as versões do kernel do Linux, do Adeos e do Xenomai.

O próximo passo é extrair os arquivos comprimidos que foram baixados, através dos comandos:

```
# tar -xjf linux-2.6.37.tar.bz2
# tar -xjf xenomai-2.5.6.tar.bz2
```

Vamos agora fazer o patch do kernel para receber o Adeos e o Xenomai através do comando:

```
# cd /usr/src/xenomai-2.5.6/scripts
# ./prepare-kernel.sh --linux=/usr/src/linux-2.6.37 --arch=x86
  --adeos=/usr/src/adeos-ipipe-2.6.37-x86-2.9-00.patch
```

O próximo passo a ser realizado é a configuração dos parâmetros do kernel que será compilado. Um arquivo de do fabricante do PC/104 utilizado chamado *config-2.6.28-1-versalogic-geodelx* pode ser usado como base para saber o que é necessário constar nessa nova configuração, para gerar um kernel otimizado às necessidades do PUMA PC/104. Independente disso um kernel mais genérico pode ser utilizado.

Vamos entrar na configuração do kernel através dos comandos:

```
# cd /usr/src/linux-2.6.37
# make menuconfig
```

As seguintes configurações devem ser feitas para introduzir o funcionamento do Xenomai e dos pacotes necessários para o projeto:

```
-General setup
    |--> Local version ---> -xenomai
-Processor type and features
    |--> High Resolution TImer Support ---> enable
    |--> HPET Timer Support ---> disable
    |--> Fine granularity task level IRQ time accounting ---> enable
    |--> Preemption Model ---> Preemptible Kernel (Low-Latency Desktop)
    |--> Interrupt pipeline ---> enable
    |--> Local APIC support on uniprocessors ---> disable
-Power manegement and ACPI options
    |--> Power Management support ---> disable
    -CPU Frequency sacaling
        |--> CPU Frequency sacaling ---> disable
    |--> CPU idle PM support ---> disable
-Device drivers
    |--> Misc Devices ---> enable
```

```
        |--> CS5535/CS5536 Geode MFGPT ---> module
        |--> CS5535/CS5536 high-res timer (MFGPT) events ---> module
    |--> USB support ---> enable
        |--> USB Serial Converter support ---> enable

-Real-Time sub-system
    -Drivers
        -Serial drivers
            |--> 16550A UART driver ---> module
        -CAN drivers
            |--> RT-Socket_CAN ---> module
            |--> Philips SJA1000 CAN controller ---> module
            |--> Sandard ISA controllers ---> module
            |--> Memory mapped controllers ---> module
```

Salvando então a nova configuração como *.config* podemos fechar o menu de configuração.

Os comandos a seguir devem ser utilizados para compilar o kernel, processo que deverá se estender por uma série de horas.

```
# make && make modules
# make modules_install
# make install
```

Vamos então fazer a instalação da imagem do novo kernel.

```
# cd /boot
# update-initramfs -c -k 2.6.37-xenomai
# ln -s vmlinuz-2.6.37-xenomai vmlinuz
# ln -s initrd.img-2.6.37-xenomai initrd
```

Agora os comandos necessários são para compilar e instalar o Xenomai.

```
# cd /usr/src/xenomai-2.5.6
```

```
# ./configure
# make
# make install
```

Por fim vamos adicionar o novo sistema na lista de sistemas para o boot.

```
# /sbin/update-grub
```

Realizado todo este processo, vamos reiniciar o computador e dar boot no sistema com o Xenomai integrado. Utilize o seguinte comando para testar o funcionamento do I-pipe Adeos:

```
# dmesg | grep I-pipe
```

A saída do comando acima deve ser semelhante ao seguinte:

```
I-pipe 2.9-00: pipeline enabled
I-pipe: Domain Xenomai registered
```

Vamos então testar o funcionamento do Xenomai com o seguinte comando:

```
# dmesg | grep Xenomai
```

A saída dessa vez deve ser semelhante a:

```
Xenomai: hal/i386 started
Xenomai: scheduling class idle registered
Xenomai: scheduling class rt registered
Xenomai: real-time nucleus v2.5.6 loaded
Xenomai: starting native API services
Xenomai: starting POSIX services
Xenomai: starting RTDM services
```

Realizados todos os passos acima com sucesso, a instalação do Xenomai estará completa.

A.4 Instalando a Biblioteca OpenCV

A biblioteca de visão computacional OpenCv é utilizada no presente projeto para realizar operações com matriz. Sendo assim precisamos realizar sua instalação. Vamos utilizar os comandos a seguir para baixar os arquivos necessários:

```
# mkdir /usr/src/opencv
# cd /usr/src/opencv
# wget http://sourceforge.net/projects/opencvlibrary/files
  /opencv-unix/2.2/OpenCV-2.2.0.tar.bz2
```

Vamos extrair o arquivo baixado com o seguinte comando:

```
# tar xvjf OpenCV-2.2.0.tar.bz2
```

Os seguintes comandos devem ser utilizados para a instação da biblioteca:

```
# cd OpenCV-2.2.0/
# mkdir release; cd release
# cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local
  -D CMAKE_BUILD_PYTHON_SUPPORT=ON ..
# make
# make install
# export LD_LIBRARY_PATH=/usr/src/opencv/OpenCV-2.2.0
  /release/lib:$LD_LIBRARY_PATH
# ldconfig
```

Appendix B

A novel wheel-leg parallel mechanism control for Kinematic Reconfiguration of an Environmental Hybrid Robot

Esse anexo apresenta um método de controle para mecanismos paralelos proposto e desenvolvido por Gustavo Freitas, Fernando Lizarralde e Liu Hsu, vinculados ao Programa de Engenharia Elétrica da COPPE/UFRJ.

O método é aplicado no novo protótipo do Robô Ambiental Híbrido para controlar a posição e o ângulo de orientação de suas rodas através de suspensões de 2DOF construída através de mecanismos paralelos.

A implementação do método de controle foi realizada por mim, Vitor Paranhos, e por Gustavo Freitas.

O estudo de caso é discutido baseado nos conceitos de controle cinemático e cinemáticas direta e diferencial.

Simulações e testes experimentais ilustram os pontos importantes da implementação e a eficácia do método proposto.

O artigo em anexo a seguir é o resultado do trabalho científico acima citado e foi submetido ao congresso SBAI2001 (http://www.ppgel.net.br/sbai2011/index.php), que será realizado na cidade de São João del Rei, no mês de Setembro.

A novel wheel-leg parallel mechanism control for Kinematic Reconfiguration of an Environmental Hybrid Robot

Gustavo Freitas, Fernando Lizarralde, Liu Hsu, Vitor Paranhos and Ney R. Salvi dos Reis

***Abstract*— This paper addresses a control design method for parallel mechanisms based on the kinematic constraints, rather than the constraint equations. The method is applied on a new Environmental Hybrid robot to control the position and angle of its wheels through 2DOF suspensions with actuated parallel mechanisms. This case study is discussed based on the concepts of kinematic control and forward and differential kinematics. Simulations, as well as experimental tests illustrate the implementation issues and the efficacy of the proposed design.**

I. Introduction

Mobile robots are widely used in unstructured and hazardous environments for several applications, including mining, forestry, agricultural and military activities ([1], [9]). For effective remote operation in inhospitable environments, it is necessary to consider efficient locomotion systems, capable of working in irregular and rough terrains.

This paper considers wheel-legged robots [8], which use wheels for propulsion and internal articulation to adapt their configuration to accommodate for obstacles, repositioning the center of mass and influencing the contact forces against the terrain.

Accuracy, repeatability and payload capacities are fundamental requirements for the legs mechanism considering advanced mobile robots. Parallel mechanism provide a stiff connection between the payload supported by the leg, with superior pose accuracy than serial chain mechanisms [10], [11]. The main disadvantages related to parallel structures are related to workspace limitation, difficulties in mapping the forward kinematic and complex singularity analysis [16], [13].

This paper presents a parallel mechanism control methodology based on a strategy proposed in [16], where the differential kinematic model is obtained considering the mechanism kinematic constrains from its structure equations, instead of using explicitly the constrains equations.

Then, according to the system effective degrees of freedom (DOF), a control strategy with primary and auxiliary objectives is proposed considering the mechanism differential kinematic model null space.

The described methodology is applied to control the suspensions of the Environmental Hybrid Robot (EHR) new prototype [4]. Each suspension is composed by a planar parallel mechanism with 2 DOF, allowing to control the wheel position and orientation independently.

After validating the proposed strategies through simulations, the kinematic control is implemented in the EHR, allowing to evaluate the performance of the proposed methodology through experimental results.

II. Environmental Hybrid Robot

This work is motivated by the Environmental Hybrid Robot [5], [4] from Brazilian Oil company Petrobras S.A. (Fig. 1), which has four suspensions composed by parallel mechanisms. The robot is going to help in monitoring the Amazon rain forest region, being used as a maintenance tool for the Coari-Manaus pipeline [7].

This system is able to collect data and samples and perform tasks in difficult-access areas of the forest. It is being designed to overcome obstacles and to operate in different terrains, as firm ground with vegetation, sand, swamps, marshes and water surface with vegetation.

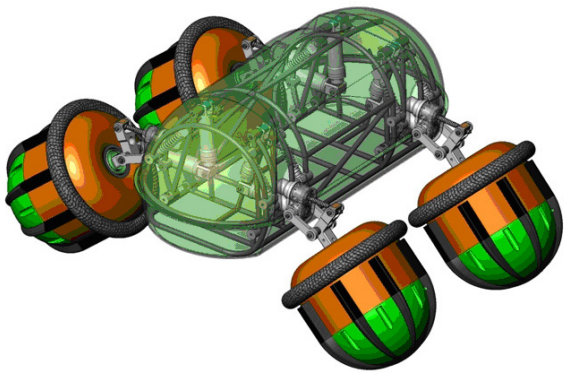

Fig. 1. Environmental Hybrid Robot new prototype with 2 DOF suspensions.

An innovative locomotion system is designed according to the encountered conditions in Amazon. The wheel-legged architecture is adopted, allowing the robot to carry load and save energy, being able to work on irregular terrains.

Each wheel is coupled to an independent suspension system, here referred as legs, composed of a spring and electric linear actuator. The suspension motors are attached to screws, composing the active prismatic joints.

The suspension parallel mechanism is designed for structural rigidity, allowing the robot to overcome obstacles and increasing wheel traction.

The original suspension developed for the EHR [4] has 1 DOF, and when the coupled motor is actuated, position and angle of the wheel in relation to the terrain are amended.

By commanding the wheels height, it is possible to modify the robot mobility, e.g. the system stability and force distribution among the legs [5], [4]. Also, the suspension angle influences the contact point with the terrain and consequently the effective radio of the spheric wheels. Thus, the suspension reconfiguration affects the relation between velocity and torque in each wheel.

The capability of influence the system stability and force distribution and also the relation between velocity and torque is desirable. However, to optimize the robot missions, the suspension should enable to independently command these parameters.

Because of that, a new mechanism with 2 DOF is being developed, allowing to decouple the height and the angle of the wheel.

G. Freitas, F. Lizarralde, L. Hsu and V. Paranhos are with the Dept. of Electrical Eng., COPPE/Federal University of Rio de Janeiro, Brazil.

N. Reis are with CENPES / Petrobras S.A., Rio de Janeiro, Brazil.

The new suspension has a larger workspace, and even allows the robot to operate up side down, as presented in Fig. 2.

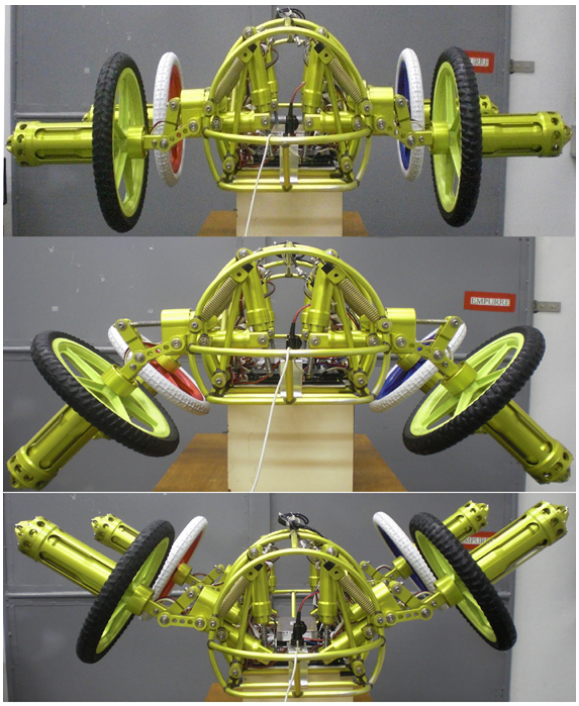

Fig. 2. EHR 2 DOF suspensions in different configurations.

The robot uses plastic bicycle $20^{"}$ wheels with solid tyres, which will be further covered by a profile as shown in Fig. 1 to increase traction and allow the EHR to buoy. These floating wheels are being designed to maximize traction considering the different operation environments.

The mobile robot weighs $35kg$ and has wheelbase of $490mm$ and track width $710mm$.

Nimh batteries supply energy for the system. A set of motors ($70W$ for each wheel and $10W$ for each suspension) and drivers (EPOs) from Maxon Motors are used (Fig. 3), allowing the robot to navigate at firm terrains with maximum speed of $1.50m/s$.

The system is teleoperated, and communication with a base computer is accomplished through $900MHz$ Ethernet radios. The embedded controls are executed by a PC/104 board with $366MHz$ processor using Linux OS. The system orientation is obtained by an MTI Xsens inertial unit.

The new Environmental Hybrid Robot prototype is still in development, and new equipments will be integrated into the system. The robot careen and floating wheels, which are under construction, must be fixed before tests in the Amazon Rain Forest.

III. Leg Forward Kinematics

The forward kinematics for a robotic system is described by the end-effector configuration specified for each chain. Here, the end-effector is represented by the contact point between wheel and terrain.

The contact point position is considered to be located in the wheel most inferior point as presented in Fig. 4. Further works will consider a contact point position model according to the floating wheels presented in Fig. 1.

Each wheel is coupled to an independent suspension system with 2 effective DOF (it can be verified using the Gluebler's formula

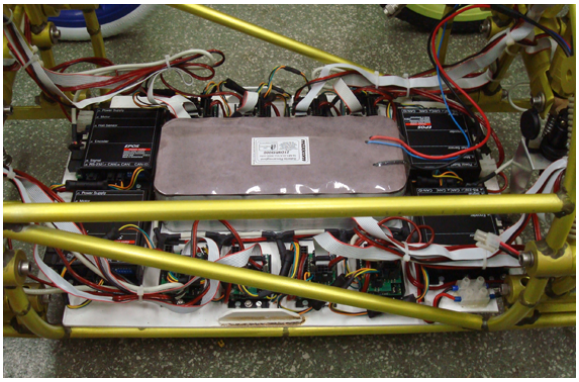

Fig. 3. EHR battery and power drivers installation.

[12]), corresponding to an underactuated planar mechanism with 2 closed kinematic chains.

The mechanism geometry is presented in Fig. 4. It is a 7 bar-link, where link 0 is fixed to the robot structure and the wheel corresponds to the link segment 6. The leg has 2 active prismatic and 6 passive revolute joints.

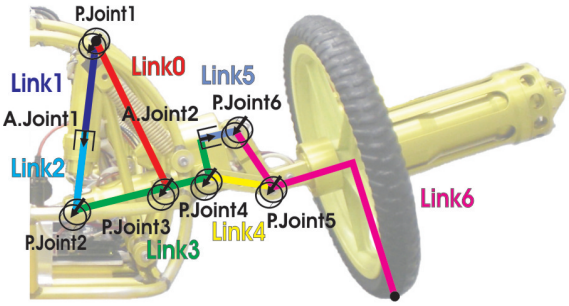

Fig. 4. EHR suspension: seven-bar-link mechanism geometry with active and passive joints.

In order to describe the leg forward kinematics, coordinate frames are attached to each link of the leg, where coordinate frame $\{x_i, y_i\}$ is attached to ith-link (see Fig. 5). The wheel-terrain contact position is defined by frame $\{x_w, y_w\}$ and $\vec{p}_{ij} \in \mathbb{R}^3$ is the vector from the i-frame origin to j-frame origin represented in base frame (frame 0). The mechanism links length are presented in table I.

The forward kinematics is described by the following *structure equation* [12]:

$$\begin{aligned} \vec{p}_{0w} &= \underbrace{\vec{p}_{02} + \vec{p}_{24} + \vec{p}_{46'} + \vec{p}_{6'w}}_{\text{chain 1}} = \underbrace{\vec{p}_{02} + \vec{p}_{24} + \vec{p}_{46} + \vec{p}_{6w}}_{\text{chain 2}} \\ &= \underbrace{\vec{p}_{03} + \vec{p}_{34} + \vec{p}_{46'} + \vec{p}_{6'w}}_{\text{chain 3}} \qquad (1) \\ \theta_{0w} &= \underbrace{\theta_1 + \theta_2 + \theta_4 + \theta_6}_{\text{chain 1}} = \underbrace{\theta_1 + \theta_2 + 2\pi + \theta_5 + \pi - \psi_6}_{\text{chain 2}} \\ &= \underbrace{\psi_0 + \theta_3 + \theta_4 + \theta_6}_{\text{chain 3}} \qquad (2) \end{aligned}$$

where θ_{0w} represents the orientation of the wheel frame with respect to base frame.

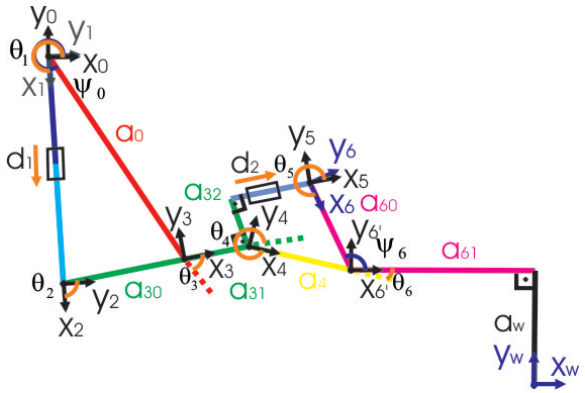

Fig. 5. EHR suspension: seven-bar-link mechanism frames.

Link #	Parameter	Value	Unit
0	a_0	185	mm
	ψ_0	117.0	deg
3	a_{30}	80	mm
	a_{31}	40	mm
	a_{32}	40	mm
4	a_4	70	mm
6	a_{60}	75	mm
	a_{61}	115	mm
	a_w	275	mm
	ψ_6	112.0	deg

TABLE I
EHR SUSPENSION MECHANISM LINKS LENGTH.

Equations (1)–(2) introduce constraints between the possible joint angles. These constrains allow the end-effector location control by specifying only the active joints variables $\theta_a = [d_1, d_2]^T$, and the other passive joints $\theta_p = [\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6]^T$ take on values such that equation (1)–(2) is satisfied.

From (1), the forward kinematics is given by:

$$\begin{aligned}\vec{p}_{0w} &= \vec{p}_{02} + \vec{p}_{24} + \vec{p}_{46'} + \vec{p}_{6'w} \\ &= d_2\ R_{01}(\theta_1)\ x + (a_{30} + a_{31})\ R_{01}(\theta_1)R_{23}(\theta_2)\ x \\ &\quad + a_4\ R_{01}(\theta_1)R_{23}(\theta_2)R_{34}(\theta_4)\ x \\ &\quad + a_{61}\ R_{01}(\theta_1)R_{23}(\theta_2)R_{34}(\theta_4)R_{46'}(\theta_6)\ x \\ &\quad - a_w\ R_{01}(\theta_1)R_{23}(\theta_2)R_{34}(\theta_4)R_{46'}(\theta_6)\ y \end{aligned} \quad (3)$$

where $a_{30}, a_{31}, a_4, a_{61}, a_w$ are given from table I, $x = [1,\ 0,\ 0]^T$, $y = [0,\ 1,\ 0]^T$ and the elementary rotation matrix $R_{ij}(\theta_i) \in SO(3)$ is the orientation of j-frame with respect to i-frame, given by:

$$R_{ij}(\theta_i) = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

Due to the parallel mechanism structure constraints, it is not trivial to obtain an analytical solution for the forward kinematics in terms of θ_a, i.e. $\vec{p}_{0w} = k_1(d_1, d_2)$, $\theta_{0w} = k_2(d_1, d_2)$.

The passive revolute joints $\theta_p \in \mathbb{R}^6$ are obtained in terms of the active joints d_1, d_2, i.e. $\theta_1 = f_1(d_1)$, $\theta_3 = f_3(d_1)$, $\theta_4 = f_4(d_1)$, $\theta_5 = f_5(d_2)$, $\theta_6 = f_6(d_2)$. These functions can be calculated through algebraic identities and geometry (by using cosine formula).

Another possibility to determine θ_p is to reduce the forward kinematic problem into appropriate subproblems whose solutions are known [12]. The passive joints θ_p can be calculated through Paden-Kahan subproblems 1 and 3 [12, pag. 99–103], which are geometrically meaningful and numerically stable.

Knowing θ_p, it is possible to obtain the contact point position $\vec{p}_{0w}$ through equation (3) and orientation (θ_{0w}) using (2).

Figure 6 illustrates suspension configurations for $d_1 \in [170, 220]mm$ and $d_2 \in [45, 105]mm$. The mechanism dexterous workspace in terms of p_{0w} and θ_{0w} is presented in Fig. 7.

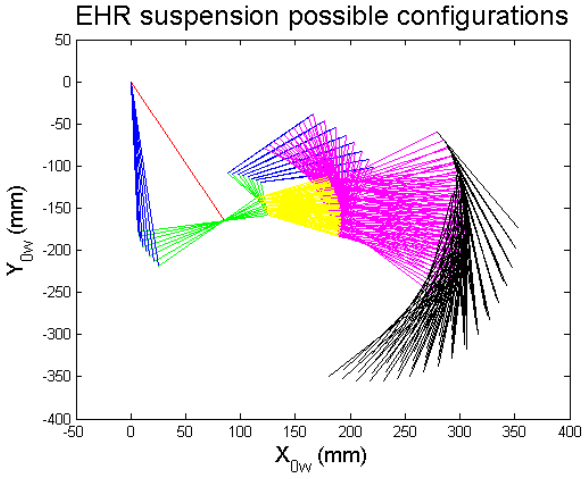

Fig. 6. EHR mechanism configurations in terms of θ_a.

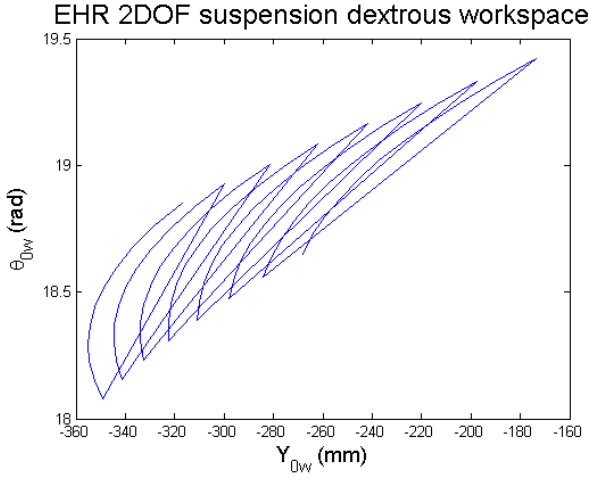

Fig. 7. EHR suspension dextrous workspace.

IV. DIFFERENTIAL KINEMATICS

The planar parallel mechanism velocity of the end-effector (wheel-terrain contact point) is related to the joints velocity by differentiating the structure equation (1). This gives a Jacobian

matrix for each chain:

$$v_w = S\ J_1 \begin{bmatrix} \dot{\theta}_1 \\ \dot{d}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_4 \\ \dot{\theta}_6 \end{bmatrix} = S\ J_2 \begin{bmatrix} \dot{\theta}_1 \\ \dot{d}_1 \\ \dot{\theta}_2 \\ \dot{d}_2 \\ \dot{\theta}_5 \end{bmatrix} = S\ J_3 \begin{bmatrix} \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_6 \end{bmatrix} \quad (5)$$

where $v_w = [\dot{p}_{0w_y}, \omega_{0w}]^T \in \mathbb{R}^2$ is the planar linear velocity in y_0 axis and angular velocity ($\omega_{0w} = \dot{\theta}_{0w}$), $S \in \mathbb{R}^{2\times 6}$ is a selection matrix considering the effective mechanism DOF:

$$S = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

and the 6-DOF Jacobian matrices are given by

$$J_1 = \begin{bmatrix} z \times \vec{p}_{0w} & R_{01}x & z \times \vec{p}_{2w} & z \times \vec{p}_{4w} & z \times \vec{p}_{6'w} \\ z & 0 & z & z & z \end{bmatrix}$$

$$J_2 = \begin{bmatrix} z \times \vec{p}_{0w} & R_{01}x & z \times \vec{p}_{2w} & R_{01}R_{23}x & z \times \vec{p}_{6w} \\ z & 0 & z & 0 & z \end{bmatrix}$$

$$J_3 = \begin{bmatrix} z \times \vec{p}_{3w} & z \times \vec{p}_{4w} & z \times \vec{p}_{6'w} \\ z & z & z \end{bmatrix}$$

where $z = [0,\ 0,\ 1]^T$, $\vec{p}_{0w}$ is given in (3) and

$$\vec{p}_{2w} = (a_{30} + a_{31})\ R_{01}(\theta_1)R_{23}(\theta_2)\ x + \vec{p}_{4w} \quad (6)$$

$$\vec{p}_{3w} = a_{31}\ R(\psi_0)R_{03}(\theta_3)\ x + \vec{p}_{4w} \quad (7)$$

$$\vec{p}_{4w} = a_4\ R_{01}(\theta_1)R_{23}(\theta_2)R_{34}(\theta_4)\ x + \vec{p}_{6'w} \quad (8)$$

$$\vec{p}_{6w} = a_{60}\ R_{01}(\theta_1)R_{23}(\theta_2)R_{36}(\theta_5)\ p_{6w} \quad (9)$$

$$p_{6w} = [a_{60}\cos\psi_6 + a_{61};\ -a_{60}\sin\psi_6 - a_w;\ 0]^T \quad (10)$$

$$\vec{p}_{6'w} = a_{61}\ R_{01}(\theta_1)R_{23}(\theta_2)R_{34}(\theta_4)R_{46'}(\theta_6)\ x - a_w\ R_{01}(\theta_1)R_{23}(\theta_2)R_{34}(\theta_4)R_{46'}(\theta_6)\ y \quad (11)$$

The manipulator Jacobian can be written in a more conventional form by stacking the Jacobians for each chain:

$$\underbrace{\begin{bmatrix} S\ J_1 & 0 & 0 \\ 0 & S\ J_2 & 0 \\ 0 & 0 & S\ J_3 \end{bmatrix}}_{J} \dot{\theta} = \underbrace{\begin{bmatrix} I \\ I \\ I \end{bmatrix}}_{A} v_w \quad (12)$$

or equivalently:

$$J\ \dot{\theta} = A\ v_w \quad (13)$$

where matrix $A \in \mathbb{R}^{6\times 2}$ is always full column rank, $J \in \mathbb{R}^{6\times 13}$ and $\dot{\theta} = [\dot{\theta}_1,\ \dot{d}_1,\ \dot{\theta}_2,\ \dot{\theta}_4,\ \dot{\theta}_6,\ \dot{\theta}_1,\ \dot{d}_1,\ \dot{\theta}_2,\ \dot{d}_2,\ \dot{\theta}_5,\ \dot{\theta}_3,\ \dot{\theta}_4,\ \dot{\theta}_6]^T$.

It is possible to observe that joints $\dot{\theta}_1, \dot{d}_1, \dot{\theta}_2, \dot{\theta}_4, \dot{\theta}_6$ are duplicated in equation (12), and because of that, J has more columns than necessary. To avoid that, it is possible to rewrite this Jacobian using column elementary operations

$$\begin{aligned} column\ 1 &= column\ 1 + column\ 6 \\ column\ 2 &= column\ 2 + column\ 7 \\ column\ 3 &= column\ 3 + column\ 8 \\ column\ 4 &= column\ 4 + column\ 12 \\ column\ 5 &= column\ 5 + column\ 13 \end{aligned}$$

and eliminating the columns related to duplicated joins (columns $6, 7, 8, 12, 13$) in order to obtain $J \in \mathbb{R}^{6\times 8}$ and $\dot{\theta} = [\dot{\theta}_1,\ \dot{d}_1,\ \dot{\theta}_2,\ \dot{\theta}_4,\ \dot{\theta}_6,\ \dot{d}_2,\ \dot{\theta}_5,\ \dot{\theta}_3]^T$.

Considering $A^\dagger \in \mathbb{R}^{2\times 6}$ the pseudo-inverse of A such that $A^\dagger A = I$, and $\widetilde{A} \in \mathbb{R}^{4\times 6}$ the annihilator matrix as $\widetilde{A}A = 0$, the differential kinematic relationship (13) can be equivalently written as [17], [14]:

$$v_w = J_t\ \dot{\theta}, \qquad J_c\ \dot{\theta} = 0 \quad (14)$$

where $J_t = A^\dagger J$ and $J_c = \tilde{A}J$.

Considering the active joints $[\dot{d}_1, \dot{d}_2]^T = \dot{\theta}_a$ and collecting all the passive joint velocities together in $\dot{\theta}_p$ and then partitioning $J_t \in \mathbb{R}^{2\times 8}$ and $J_c \in \mathbb{R}^{4\times 8}$ accordingly

$$J_t = [J_{t_a};\ J_{t_p}], \qquad J_c = [J_{c_a};\ J_{c_p}] \quad (15)$$

considering (14) and (15), one has that:

$$v_w = J_{ta}\dot{\theta}_a + J_{tp}\ \dot{\theta}_p, \qquad J_{ca}\ \dot{\theta}_a + J_{cp}\ \dot{\theta}_p = 0 \quad (16)$$

where $J_{ta} \in \mathbb{R}^{2\times 2}$, $J_{tp} \in \mathbb{R}^{2\times 6}$, $J_{ca} \in \mathbb{R}^{4\times 2}$ and $J_{cp} \in \mathbb{R}^{4\times 6}$.

If J_{cp} is full rank, the passive joints velocities can be given in terms of active joint velocity $\dot{\theta}_a$:

$$\dot{\theta}_p = -J_{c_p}^\dagger\ J_{c_a}\ \dot{\theta}_a \quad (17)$$

It is possible to verify that J_{c_p} is not singular for the considered mechanism.

Finally, the wheel velocity v_w is given in terms of the active joints velocity $\dot{d}_1, \dot{d}_2$ by:

$$v_w = \underbrace{(J_{t_a} - J_{t_p}\ J_{c_p}^\dagger\ J_{c_a})}_{J_p}\ \dot{\theta}_a \quad (18)$$

where $J_p \in \mathbb{R}^{2\times 2}$ is the leg Jacobian.

V. Kinematic Control

A kinematic control approach is proposed to change the suspension parallel mechanism posture in order to accomplish an specific task. Considering the system with 2 effective DOF, the adopted control has two objectives:

1) control the vertical distance p_{0w_y} between wheel-terrain contact point and suspension frame:

$$p_{0w_y} \to p^*_{0w_y}(t) \quad e_h = p^*_{0w_y} - p_{0w_y} \to 0 \quad (19)$$

2) regulate the wheel orientation θ_{0w} actuating on the mechanism null space velocities with respect to the primary objective:

$$\theta_{0w} \to \theta^*_{0w} \quad e_o = \theta^*_{0w} - \theta_{0w} \to 0 \quad (20)$$

The proposed scheme consists in commanding the robot joints velocity $\dot{d}_1, \dot{d}_2$, i.e. $u(t) = \dot{\theta}_a$, adjusting the contact point pose to cancel the errors (19) and (20).

The actuated planar suspension differential kinematic model (18) can be decoupled in 2 components: a linear $\dot{p}_{0w_y}$ and a angular velocity ω_{0w} as:

$$\begin{bmatrix} \dot{p}_{0w_y} \\ \dot{\theta}_{0w} \end{bmatrix} = \bar{J}_p\ \dot{\theta}_a; \qquad \bar{J}_p = \begin{bmatrix} J_{p_y} \\ J_\theta \end{bmatrix}$$

where $J_{p_y}, J_\theta \in \mathbb{R}^{1\times 2}$, thus the joints velocity can be obtained from $\dot{\theta}_a = \bar{J}_p^{-1}\ v$, where v is a cartesian control law.

From Fig. 7, it is possible to note that the suspension workspace has boundary singularities in terms of orientation. Considering $p_{0w_y} < -320mm$, the orientation singularity is achieved for $\theta^*_{0w} >= 6\pi\ rad$.

However, in these configurations, it is still possible to control p_{0w_y}. The vertical distance between suspension frame and wheel-terrain contact point influence directly the robot orientation, force distribution among legs and tip over stability as presented in [4],

[3]. Thus it corresponds to the main control objective, avoiding orientation singularities.

The adopted approach consists on given priority to control the vertical distance, considering orientation as an auxiliary control. Therefore, the mechanism is considered redundant, with 1 effective DOF and 2 active joints. The differential kinematic is given by $\dot{p}_{0w_y} = J_{p_y}\dot{\theta}_a$, and the joints velocity are given by [15], [2]:

$$\dot{\theta}_a = J_{p_y}^{\dagger}\, v + \underbrace{(I - J_{p_y}^{\dagger}\, J_{p_y})}_{\mathcal{P}}\, \dot{\theta}_{a_0} \tag{21}$$

where $col(\mathcal{P})$ spans the null space of J_{p_y}, and $\dot{\theta}_{a_0}$ is an arbitrary vector of active joints velocity. The right hand side of (21) can be interpreted as null space velocity which provides internal movements with respect to the redundant mechanism model.

Then, using a proportional law as

$$u = J_{p_y}^{\dagger} K_h\ e_h + (I - J_{p_y}^{\dagger} J_{p_y})\, \bar{u}\,, \tag{22}$$

where $\bar{u}$ is the auxiliary control signal, the position error dynamic is determined by $\dot{e}_h + K_h\, e_h = 0$, since the right hand side of (22) spans the null space of J_{p_y}. Considering $K_h > 0$ and nonsingular J_{p_y}, we have $\lim_{t\to\infty} e_h(t) = 0$.

The auxiliary control $\bar{u}$ is chosen to improve the mechanism performance during tasks execution. A typical choice is

$$\bar{u} = \bar{K}\, \left(\frac{\partial f(\theta_a)}{\partial \theta_a}\right)^T, \tag{23}$$

where $\bar{K} > 0$ is a gain factor and $f(\theta_a)$ is an objective function[1] in terms of θ_a, here chosen to cancel the orientation error: $f(\theta_a) = e_o^2$. Other possible objective functions are [15]:

- Manipulability: $f(\theta_a) = \sqrt{\det(JJ^T)}$.
- Joints limits avoidance: $f(\theta_a) = -\frac{1}{2n}\sum_{i=1}^{n}\left(\frac{\theta_{a_i} - \bar{\theta}_{a_i}}{\theta_{aM_i} - \theta_{am_i}}\right)$, $\theta_{am_i} < \theta_{a_i} < \theta_{aM_i}$, where θ_{aM_i} and θ_{am_i} are the maximum and minimum joints limits respectively, and $\bar{\theta}_{a_i}$ is the geometric average between θ_{aM_i} and θ_{am_i}.
- Obstacle avoidance: $f(\theta_a) = \min \|p(\theta_a) - p_o\|$, where p_o is the obstacle position and $p(\theta_a)$ is a point belonging tho the mechanism.

Considering $f(\theta_a) = e_o^2$, the auxiliary control is given by:

$$\bar{u} = K_o\ e_o\ \dot{e}_o = K_o\ J_\theta^T\ e_o \tag{24}$$

where $K_o > 0$ is the orientation gain.

Finally, the active joints velocity $\dot{\theta}_a$ is defined as:

$$u = \dot{\theta}_a = J_{p_y}^{\dagger} K_h\ e_h + (I - J_{p_y}^{\dagger} J_{p_y})\ K_o\ J_\theta^T\ e_o \tag{25}$$

VI. Simulations Results

This section presents simulation results considering the Environmental Robot suspension. The simulations are executed using *Matlab*.

The reference orientation is given by $\theta_{0w}^* = 6\pi\ rad$ and $p_{0w_y}^*$ is given by a series of step signals, as shown in Fig. 8.

It is possible to observe that the primary control objective (the vertical position control of the wheel-terrain contact point p_{0w_y}), $e_h \to 0$ is promptly accomplished after each step in the reference signal $p_{0w_y}^*$.

The secondary control objective which consists in keep the wheel perpendicular to the terrain such as $\theta_{0w} = 6\pi\ rad$ is accomplished if the first objective has been achieved and the mechanism still has effective DOF, i.e. if the joints are not in limit positions.

[1] $f(\cdot)$ is continuous, differentiable and convex.

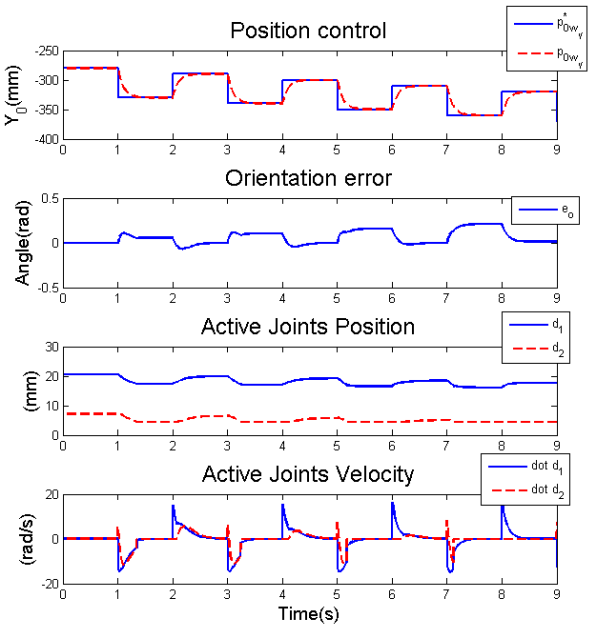

Fig. 8. Parallel mechanism kinematic control simulation results.

VII. Control Implementation and Experimental Results

This section describes the kinematic control implementation in the Environmental Robot.

The control algorithm is executed in the robot by a PC/104 single board computer featuring an AMD Geode GX500 processor with 366MHz performance and low power draw. It runs a dual kernel RTOS, Real Time Operating System, Linux (Debian 5.0.1, Kernel 2.6.34) and Xenomai 2.5.3 (http://www.xenomai.org).

The use of Xenomai as a RTOS was selected because of the possibility of running real-time tasks from the user-space [6].

Xenomai uses Adaptive Domain Environment for Operating Systems - ADEOS (http://home.gna.org/adeos), which has OS Domains approach and interrupt pipe for implementing the hardware sharing between Xenomai and Linux [18]. The interrupt pipe ensures that hardware interrupts can be handled first by Xenomai domain, which allows it to provide real-time guarantees for the system.

Xenomai has six different skins in order to provide compatibility with several different proprietary RTOS running on the GNU/Linux environment. The implementation described here uses Xenomai's Native API, a skin that is original from Xenomai and attempt to provide all real-time system calls and high integration level with Linux environment, without concerning about compatibility.

The software is implemented using C/C++ programming languages and is a simple user-space application which starts a real-time task for executing all the control steps. The algorithm for the real-time task is described bellow:

```
openLogFiles();
startCommunicationWithHardware();
setControlParameters();
while(true){
    for(allMechanisms){
        readActualPositions();
```

```
        computeForwardKinematics();
        getReferences();
        calculateErrors();
        computeVelocities();
    }
    for(allMechanisms){
        sendHardwareTheVelocities();
        logValues();
    }
}
```

For the control implementation a custom C library is used, composed by the following functions:

- *convertEncoderPointsToMillimeter()*: Convert a given active joint position $\theta_a = [d_1, d_2]^T$ in encoder points to length in millimeters;
- *forwardKinematics()*: Given the active joints position θ_a, the function calculates the mechanism forward kinematics $\vec{p}_{0w}, \theta_{0w}$;
- *suspensionControl()*: Given the joints position θ_a, θ_p and closed loop errors for position e_h and orientation e_o, the function calculates the control command $u = \dot{\theta}_a$.

The function *forwardKinematics* is obtained using *ccode* Matlab command. The mathematical expression for the forward position kinematics is declared as a symbolic expression in Matlab, and the *ccode* is used to generate a C code implementing this expression.

The *ccode* do not necessarily generates a computationally efficient code, which should be post optimized. For that, values that are often calculated during code execution are stored in variables and after reused, reducing computing mean time as shown in table II.

To implement *suspensionControl*, the *ccode* is also used, and after the code is optimized as described before. The library named OpenCV (http://opencv.willowgarage.com/wiki/) is used to execute the matrix inversion J_{cp}^{-1}. It provides a function for inverting matrices either with singular value decomposition method (SVD) or Gaussian elimination with optimal pivot element chosen method (LU). The computing mean time for *suspensionControl* using both methods with optimization or not is shown in table III.

Algorithm	E[t]	σ
Non-Optimized *forwardKinematics*	165.79μs	74.72μs
Optimized *forwardKinematics*	35.19μs	6.50μs

TABLE II
forwardKinematics COMPUTING MEAN TIME E[T] AND ASSOCIATED STANDARD DEVIATION σ.

Computing mean times shown in tables II and III only considers the processing time, and do not include communication with the robot hardware. Table II shows that *forwardKinematics* is reduced to 20% of it's original computing time. Table III shows that *suspensionControl* computing time is reduced to 60% of it's original value and also shows that LU inversion for this application is $\sim 4\times$ faster than SVD.

In order to validate the kinematic control implementation, the same reference values employed during simulations (Fig. 8) where used to control one EHR suspension. The results are presented in Fig. 9.

Algorithm	E[t]	σ
Non-Optimized with LU inversion	353.60μs	99.79μs
Non-Optimized with SVD inversion	1074μs	158.93μs
Optimized with LU inversion	219.07μs	65.05μs
Optimized with SVD inversion	942.57μs	175.61μs

TABLE III
suspensionControl COMPUTING MEAN TIME E[T] AND ASSOCIATED STANDARD DEVIATION σ.

Similar to simulations, the primary objective, to control the vertical position of the wheel-terrain contact point p_{0w_y} is promptly accomplished, and $e_h \rightarrow 0$ after each change in the reference signal $p^*_{0w_y}$.

The orientation error e_o is also decreased for each reference posture, and the secondary control objective is eventually achieved when the active joints are not in limit positions.

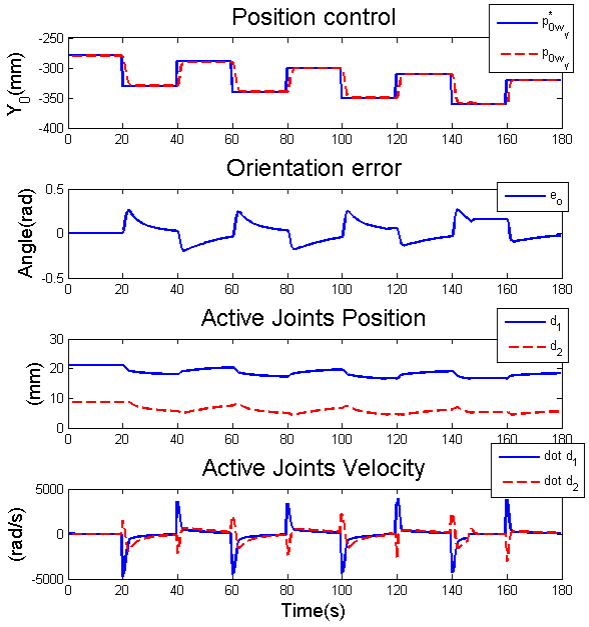

Fig. 9. EHR suspension kinematic control experimental results.

VIII. Conclusions and Future Works

This paper presents a control methodology for parallel mechanisms, based on a modeling procedure proposed in literature which consider the mechanism kinematic constrains from its structure equations, instead of explicitly use the constrains equations. A kinematic control is then proposed considering the mechanism DOF to improve primary and auxiliary objectives.

The presented methodology is demonstrated considering the parallel 2 DOF suspension mechanism of the Environmental Hybrid Robot new prototype. The mechanism forward and differential kinematics are obtained. Then, a strategy is presented to control first the vertical distance between wheel and base frame and then the wheel orientation.

Simulations and experimental results are presented considering constant orientation reference θ^*_{0w} and a sequence of steps as vertical position reference $p^*_{0w_y}$. The control implementation in the robot is described, presenting details related to real time computation, and tests are executed with the EHR.

Future works will consist on commanding the combined suspensions in order to improve the robot mobility, i.e. traction and stability, as proposed in [4], [3].

The Environmental Hybrid Robot new prototype is still in development. Some preliminary tests have being accomplished in laboratory and field, and are presented in the accompanying video and in Fig. 10. The robot complete functionality will be achieved when the floating wheels are integrated into the system.

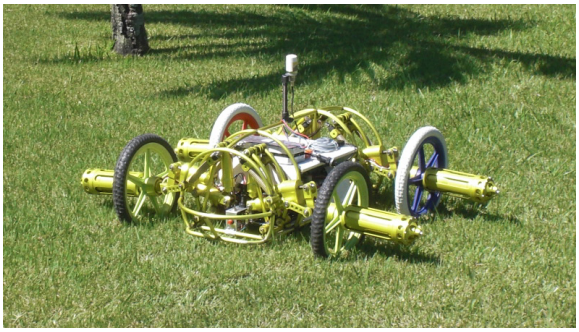

Fig. 10. EHR during preliminary field tests.

REFERENCES

[1] A. D. Calderon and A. Kelly, "On-line stability margin and attitude estimation for dynamic articulating mobile robots," *The Int. Journal of Robotics Research*, vol. 24, no. 10, pp. 845–866, October 2005.

[2] S. Chiaverini, G. Oriolo, and I. D. Walker, "Kinematically redundant manipulators," in *Springer Handbook of Robotics*, 1st ed., B. Siciliano and O. Khatib, Eds. Springer-Verlag Ltd., 2008, pp. 245–268.

[3] G. Freitas, G. Gleizer, F. Lizarralde, and L. Hsu, "Multi-objective optimization for kinematic reconfiguration of mobile robots," in *Proc. of the 6th annual IEEE Conf. on Automation Science and Eng.*, Toronto, August 2010, pp. 686–691.

[4] G. Freitas, G. Gleizer, F. Lizarralde, L. Hsu, and N. R. S. dos Reis, "Kinematic reconfigurability control for an environmental mobile robot operating in the amazon rain forest," *Journal of Field Robotics*, vol. 27, no. 2, pp. 197–216, March–April 2010.

[5] G. Freitas, F. Lizarralde, L. Hsu, and N. R. S. dos Reis, "Kinematic reconfigurability of mobile robot on irregular terrains," in *Proceedings of IEEE International Conference on Robotics & Automation*, Kobe, May 2009, pp. 1340–1345.

[6] P. Gerum, "Xenomai: Implementing a RTOS emulation framework on GNU/Linux," www.xenomai.org/documentation/branches/v2.3.x/pdf/xenomai.pdf, 2004.

[7] E. Guizzo, "Dream jobs 2008 - Ney Robinson Salvi dos Reis: Into the wild," *IEEE Spectrum Magazine*, vol. 45, no. 2, pp. 33–34, February 2008.

[8] K. Iagnemma and S. Dubowsky, *Estimation, Motion Planning, and Control with application to Planetary Rovers*. Berlin: Springer-Verlag, 2004.

[9] C. Lee, S. Kim, S. Kang, M. Kim, and Y. Kwak, "Double-track mobile robot for hazardous environment applications," *Advanced Robotics*, vol. 17, no. 5, pp. 447–459, December 2003.

[10] J.-P. Merlet, "Parallel manipulators: state of the art and perspectives," *Advanced Robotics*, vol. 8, no. 6, pp. 589–596, 1993.

[11] J.-P. Merlet and C. Gosselin, "Parallel mechanisms and robots," in *Springer Handbook of Robotics*, 1st ed., B. Siciliano and O. Khatib, Eds. Springer-Verlag Ltd., 2008, pp. 269–285.

[12] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC, 1993.

[13] J. F. O'Brien, F. Jafari, and J. T. Wen, "Determination of unstable singularities in parallel robots with N-arms," *IEEE Transactions on Robotics*, vol. 22, no. 1, pp. 160–167, 2006.

[14] ——, "Determination of unstable singularities in parallel robots with N-arms," *IEEE Transactions on Robotics*, vol. 22, no. 1, pp. 160–167, February 2006.

[15] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*, 2nd ed. Springer-Verlag Ltd., 2000.

[16] J. T. Wen and J. F. O'Brien, "Singularities in three-legged platform-type parallel mechanisms," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 4, pp. 720–727, 2003.

[17] ——, "Singularities in three-legged platform-type parallel mechanisms," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 4, 720–722 2003.

[18] K. Yaghmour, "Adaptive domain environment for operating systems," http://www.opersys.com/ftp/pub/Adeos/adeos.pdf, 2001.