UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

MARCUS VINICIUS DUARTE VELOSO

SOM4R: UM *MIDDLEWARE* PARA APLICAÇÕES ROBÓTICAS BASEADO NA ARQUITETURA ORIENTADA A RECURSOS.

FORTALEZA
2014

SOM4R: UM *MIDDLEWARE* PARA APLICAÇÕES ROBÓTICAS BASEADO NA ARQUITETURA ORIENTADA A RECURSOS.

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará, como parte dos requisitos para obtenção do grau de Doutor em Engenharia de Teleinformática.

Orientador: Prof. Dr. José Tarcísio Costa Filho
Co-Orientador: Prof. Dr. Guilherme de Alencar Barreto

Por

MARCUS VINICIUS DUARTE VELOSO

Fevereiro, 2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Pós-Graduação em Engenharia - BPGE

V555s Veloso, Marcus Vinicius Duarte.
SOM4R: um Middleware para aplicações robóticas baseado na arquitetura orientada a recursos / Marcus Vinicius Duarte Veloso. – 2014.
125 f. : il. color. , enc. ; 30 cm.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2014.
Área de concentração: Sinais e Sistemas.
Orientação: Prof. Dr. José Tarcísio Costa Filho.
Coorientação: Prof. Dr. Guilherme de Alencar Barreto.

1. Teleinformática. 2. Robótica. 3. Processamento distribuído. I. Título.

CDD 621.38

MARCUS VINICIUS DUARTE VELOSO

SOM4R: UM MIDDLEWARE PARA APLICAÇÕES ROBÓTICAS BASEADO NA ARQUITETURA ORIENTADA A RECURSOS

Tese submetida à Coordenação do Curso de Pós-Graduação em Engenharia de Teleinformática, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Doutor em Engenharia de Teleinformática, área de concentração Sinais e Sistemas.

Aprovada em 06/02/2014.

BANCA EXAMINADORA

Prof. Dr. José Tarcisio Costa Filho (Orientador)
Universidade Federal do Ceará - UFC

Prof. Dr. Guilherme de Alencar Barreto (PPGETI/UFC)
Universidade Federal do Ceará - UFC

Prof. Dr. José Marques Soares
Universidade Federal do Ceará - UFC

Prof. Dr. Luiz Marcos Garcia Gonçalves
Universidade Federal do Rio Grande do Norte - UFRN

Prof. Dr. Adrião Duarte Dória Neto
Universidade Federal do Rio Grande do Norte - UFRN

Dedico este trabalho para as pessoas mais fortes que conheço nessa vida até hoje, que são meu pai Veloso, meu irmão Marcus Flávio, minha filha Rafaela e minha esposa Samia.

AGRADECIMENTOS

A Deus, acima de tudo. Porque ”tudo é do Pai”.

A toda minha família, em especial aos meus pais, Veloso e Marlene, a minha esposa, filhos e afilhados, Samia, Rafaela, João Pedro, Mariana, Maria Clara e Vinicius, e aos meus avós paternos e maternos, Veloso e Mariinha, e Milton e Maria Alice, pelo amor e carinho incondicionais que tenho recebido deles ao longo da vida.

Ao Prof. Dr. José Tarcisio Costa Filho pela orientação, confiança, incentivo, apoio e essencial ajuda no desenvolvimento desse trabalho. Agradeço também ao professor pelo ambiente de trabalho agradável, alegre e produtivo, que marcou todas as etapas desse trabalho.

Ao Prof. Dr. Guilherme de Alencar Barreto pela orientação, apoio e incentivo recebidos.

Ao Prof. Dr. José Marques Soares pelo apoio e incentivo recebidos.

Ao Engenheiro Mecânico Rafael Vasconcelos Cordeiro pela parceria no desenvolvimento e construção do robô e pelas discussões sobre robótica. Agradeço também ao Rafael pelo ambiente de trabalho agradável, alegre e produtivo, que marcou todas as etapas do nosso trabalho.

Ao Dr. César Cals Neto pelo apoio e incentivo que me permitiram realizar esse trabalho em paralelo com o trabalho desenvolvido sob sua orientação na FIEC.

Ao aluno de doutorado MSc. Andre Luiz Carneiro de Araujo pelo apoio e incentivo recebidos. Agradeço também ao professor pela disponibilização do laboratório do ITTI no IFCE.

À aluna de doutorado MSc. Ananda Lima Freire pelos trabalhos publicados em parceria e pelas discussões sobre redes neurais e robótica.

Ao Prof. Dr. Josué Mendes Filho pela orientação e apoio recebidos desde a graduação e mestrado. Agradeço também ao professor pela disponibilização da oficina do Departamento de Física da UFC.

Ao Prof. Dr. Carlos André Dias Bezerra pela confiança, incentivo e apoio recebidos.

À MSc. Rejane Cavalcante Sá pela ajuda na confecção das placas de circuito impresso no ITTI/IFCE.

Ao Sr. Israel Rodrigues Carvalho pela ajuda com as linguagens Php, Javascript e Flash/Flex.

Aos Amigos, Professores e Funcionários do Departamento de Engenharia de Teleinformática (DETI) da UFC que de alguma forma me apoiaram nesse trabalho.

“O ótimo é inimigo do bom.“
Voltaire

RESUMO

Middleware é a camada de *software*, situada entre o sistema operacional e a camada de aplicações ou entre camadas de aplicações, que fornece uma infraestrutura para integração de programas aplicativos e dados em sistema de processamento distribuído. Nesta tese propomos uma nova camada de *software* (*Middleware*) para integração e compartilhamento inteligente dos recursos (sensores, atuadores e/ou serviços) robóticos identificados por URIs (*Uniform Resource Identifiers*), empregando a rede TCP/IP, utilizando protocolos com menores restrições em *firewall*, uma interface de interação humano-máquina (IHM) implementada através de um portal web e uma linguagem de descrição dos recursos que torna os dados mais portáveis e interoperáveis entre diferentes tipos de computadores, sistemas operacionais e linguagens de programação. O *middleware* proposto facilita a computação interativa de múltiplos aplicativos interconectados com a finalidade de criar uma aplicação maior, geralmente distribuída sobre uma rede de computadores composta de vários tipos heterogêneos de *hardware* e *software*. Com este modelo de *middleware*, é possível garantir segurança de acesso aos recursos, abstrair a diversidade do *hardware* robótico, reutilizar a infraestrutura de *software* para robôs entre múltiplos esforços de pesquisa, reduzir o acoplamento entre os múltiplos aplicativos, estimular a portabilidade do código e suportar escalabilidade da arquitetura.

Palavras-chaves: Middleware – Robótica Móvel – Processamento Distribuído

ABSTRACT

Middleware is the software layer situated between the operating system and applications layer or between layers of applications, which provides an infrastructure for integrating applications and data in a distributed processing system. In this thesis we propose a new software layer (middleware) for integration and intelligent sharing of robotic resources (sensors, actuators and / or services) identified by URIs (Uniform Resource Identifiers), using the TCP/IP network, employing protocols with minor firewall restrictions and a resource description language that makes data more portable and interoperable between different types of computers, operating systems and programming languages. The proposed middleware facilitates interactive computing of multiple interconnected applications with the purpose to create a larger application, usually distributed over a computer network consisting of various kinds of heterogeneous hardware and software. With this model of middleware, it is possible to ensure security of access to resources, abstracting the diversity of robotic hardware, to reuse the infrastructure of software for robots between multiple search efforts, reduce the coupling between multiple applications, encourage code portability and support scalability of the architecture.

Keywords: Middleware – Mobile Robotics – Distributed Processing

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

BSD - Berkeley Software Distribution

BPMN - Business Process Model and Notation

CI - Circuito Integrado

CNC - Controle Numérico Computadorizado

CPU - Central Processing Unit

dsPIC - Programmable Controller Interface with Digital Signal processing

HMI - Human-Machine Interface

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

IDL - Interface Description Language

IMU - Inertial Measurement Unit

PIC - Programmable Controller Interface

PTZ - Pan, Tilt e Zoom

RDF - Resource Description Framework

REST - Representational State Transfer

RGB - Red, Green and Blue

RMR - Robô Móvel com Rodas

ROA - Resource-Oriented Architecture

RPC - Remote Procedure Call

SO - Sistema Operacional

SOA - Service-Oriented Architecture

SOAP - Simple Object Access Protocol

STM - Short Term Memory

TCP - Transmition Control Protocol

TCP/IP - Transmission Control Protocol / Internet Protocol

TTS - Text To Speech

UDP - User Datagram Protocol

XML - Extensible Markup Language

CAPÍTULO 1

1 - INTRODUÇÃO

1.1 Introdução

Middleware é a camada de *software*, situada entre o sistema operacional e a camada de aplicações ou entre camadas de aplicações, que fornece uma infraestrutura para integração de programas aplicativos e dados em sistema de processamento distribuído [1]. O *middleware* facilita a computação interativa de múltiplos aplicativos interconectados com a finalidade de criar uma aplicação maior, geralmente distribuída sobre uma rede de computadores composta de vários tipos heterogêneos de *hardware* e *software*. Tal sistema de processamento distribuído é altamente escalável, possibilitando a prestação de serviços em tempo real (e.g. robótica). O desenvolvimento e implantação de um *middleware* sobre uma rede TCP/IP é bastante atraente porque ele pode integrar as funções de alto e baixo nível, tais como: percepção sensorial, controle motor, interação com o meio (evitar colisões), memória, representação espacial, planejamento de tarefas (trajetórias, mapas), aprendizado e adaptação ao meio, navegação, sensoriamento e interação humano-robô e robô-robô.

Cabe salientar que, apesar dos avanços em sistemas robóticos nos últimos anos, o compartilhamento e integração dos recursos de um ou mais robôs através da rede TCP/IP de forma transparente ainda é um grande desafio. Tal constatação resulta do fato de que recursos tais como: *laser*, sonar, câmera, acelerômetro, giroscópio, entre outros, possuem diferentes formatos de dados, *drivers* e APIs (interface de programação de aplicativos) de comunicação, onde grande parte dessas APIs são proprietárias (código fechado), dificultando o processo de depuração em todos os níveis da pilha de *software*, ou são desenvolvidas para um sistema operacional específico, reduzindo as possibilidades de integração dos recursos.

Nesta tese, propomos uma nova camada de *software* (*Middleware*), denominado SOM4R (*Simple and Open Middleware for Robotics*), para integração e compartilhamento inteligente dos recursos (sensores, atuadores e/ou serviços) robóticos identificados por URIs (*Uniform Resource Identifiers*), empregando a rede TCP/IP, utilizando protocolos com menores restrições em *firewalls* e uma linguagem de descrição dos recursos que pode ser estendida para expressar proposições sobre diversos assuntos. Com este modelo de *middleware*, é possível garantir segurança de acesso aos recursos, abstrair a diversidade do *hardware* robótico, reutilizar a infraestrutura de *software* para robôs entre múltiplos esforços de pesquisa, reduzir o acoplamento entre os múltiplos aplicativos, estimular a portabilidade do código e suportar escalabilidade da arquitetura.

1.2 Motivação

A utilização de um *middleware* permite que os desenvolvedores de aplicações não se envolvam com os detalhes relativos a cada fonte de dados e com a complexidade dos ambientes computacionais. Por outro lado, para o desenvolvimento e a manutenção de sistemas de *software*, como no caso dos sistemas *middleware*, devem ser utilizadas metodologias que facilitem o projeto e que possibilitem a redução do tempo de implementação e de manutenção. Uma alternativa importante é a possibilidade de reuso dos projetos e dos códigos já existentes. Este procedimento se torna necessário devido às rápidas mudanças tecnológicas e às constantes alterações nos requisitos das aplicações em robótica, o que implica na necessidade de revisões periódicas dos projetos, afim de evitar que o *middleware* se torne obsoleto no curto prazo. Um outro aspecto significativo resulta de uma maior prudência financeira em novos investimentos por parte das empresas da área de robótica, visto que os custos de desenvolvimento de novas aplicações e de manutenção devem ser reduzidos.

Com base nas deficiências (vide capítulo 3) percebidas dos projetos disponíveis atualmente, muita pesquisa [9,13,14] tem sido realizada com objetivo de se estabelecer critérios para a avaliação e comparação das características distintas dos projetos no que diz respeito à integração de recursos de robótica através da rede. Neste sentido, tais pesquisas tem possibilitado identificar limitações e sugerir melhorias nos projetos.

Um trabalho de pesquisa realizado por Kramer e Scheutz [9] sobre ambientes de desenvolvimento de robôs (RDEs - *Robotic Development Environments*) descreve, avalia e compara nove projetos *open source*, a saber: TeamBots, ARIA, Player/Stage, Pyro, CARMEN, MissionLab, ADE, Miro, e MARIE. Esses autores sugerem potenciais áreas de melhoria para os mantenedores desses ambientes (RDEs) com base em recursos encontrados em sistemas multiagentes (MAS - *Multi-Agent System*).

Mohamed et al. [13] apresenta uma breve revisão de *middleware* para robôs em rede (*Networked Robot Middleware*). Eles examinam as limitações atuais de dez plataformas de *middleware*, a saber: Miro, RT, UPnP Robot, Player/Stage, PEIS Kernel, MARIE, RSCA, AWARE, Sensory Data Processing e Middleware Layer for Incorporation, e identificam várias questões em aberto que precisam ser abordadas, a saber: segurança, recursos avançados de integração, descoberta automática de recursos, e abstrações de alto nível de *hardware* e *software*.

Elkady e Sobh [14] apresentam um levantamento dos projetos de *middleware* para robótica (RM - *Robotics Middleware*). Os autores discutem a arquitetura e algumas características importantes de dezenove plataformas de *middleware* para robôs, a saber: Orocos, Pyro, Player, Orca, Miro, OpenRT Maist, ASEBA, MARIE, RSCA, MRDS, OPROS, CLARAty,

ROS, SmartSoft, ERSP, Webots e RoboFrame. Com base nas características identificadas no levantamento dessa pesquisa [14], Elkady et al. propôs o *middleware* robótico RISCWare [34].

O desenvolvimento de uma arquitetura de *software* para robótica capaz de integrar e compartilhar os vários componentes heterogêneos (*software* e *hardware*) de um robô, assim bem como, os recursos de vários robôs interligados em rede, como é aqui o caso, tem sido um grande desafio para os pesquisadores [2,3,4,7,8,9,10,11,13,14,32,33,34].

As principais dificuldades na formação dessa complexa arquitetura de *software* envolvem, basicamente, os problemas de abstração do *hardware*, comunicação e segurança. Estas questões encontram-se descritas em mais detalhes nos parágrafos seguintes.

1.2.1 Abstração do *Hardware*

Em geral, os vários projetos de robôs apresentam concepções distintas com relação a estrutura e organização física (mecânica e eletrônica), como por exemplo, as arquiteturas de *hardware*, as configurações de sensores, as especificações de *drivers* e de APIs de comunicação, entre outros aspectos construtivos de robôs.

Um desafio comum aos pesquisadores da área de robótica é a reutilização de módulos de *software* desenvolvidos pelos vários projetos de robôs, uma vez que, na maioria dos projetos, não há compatibilidade entre arquiteturas de *hardware* e sistemas operacionais empregados pelos robôs. Como consequência, algumas vezes o *software* é inteiramente reprojetado e construído para cada novo robô [3]. Neste sentido, uma arquitetura de *software* aberto é fundamental para oferecer suporte ao *hardware* de muitos robôs. O acesso ao código fonte completo é bastante significativo para facilitar o processo de encontrar e corrigir defeitos num aplicativo de *software* (depuração) em todos os níveis da pilha de *software*. Um *middleware* comum a vários robôs não só define como os componentes podem interagir uns com os outros por meios de comunicação e mecanismos de sincronização, mas também fornece infraestrutura e funcionalidade para a construção do sistema. Isso estimula outros pesquisadores a utilizar os aplicativos disponíveis, em vez de desenvolver o mesmo aplicativo para cada robô, induzindo o desenvolvimento de uma biblioteca padrão.

Outro aspecto importante a ser considerado é o alto grau de acoplamento e interdependência entre os módulos de *software* e o *hardware*, ou entre os módulos, cujo impacto é reduzir o nível de coesão e a confiabilidade do sistema de *software*, pois a falha de um módulo possivelmente causará falhas em outros módulos do sistema. Acoplamentos muito fortes limitam a modularidade e dificultam o desenvolvimento e a manutenção de *software* [4]. Uma arquitetura de *software* modular permite a mudança de um módulo de forma independente dos demais, tornando-se mais flexível para suportar novos dispositivos de *hardware* [7,13].

1.2.2 Comunicação

A disponibilidade do sinal e a reduzida *largura de banda* de algumas redes sem fio (e.g. 3G, Wi-Fi 802.11b) podem desacelerar e até reiniciar os canais de comunicação entre os processos (programas ou módulos em execução) distribuídos em computadores disponíveis na rede para atender a heterogeneidade das tarefas a serem realizadas. Em alguns casos, isso pode ocasionar a degradação do tempo de resposta devido a comunicação, comprometendo a solução de integração de sistemas. O tempo de resposta das aplicações está também relacionado com as tecnologias utilizadas para comunicação entre processos, a saber: CORBA (*Common Object Request Broker Architecture*), *Socket* TCP (*Transmission Control Protocol*) ou UDP (*User Datagram Protocol*), Java RMI (*Remote Method Invocation*), XML-RPC (*Remote Procedure Call*), *Web Services*, SOAP (*Simple Object Access Protocol*), REST (*REpresentational State Transfer*), entre outras. Em [5], por exemplo, é mostrado que a tecnologia CORBA tem apresentado um tempo de resposta de 7 até 400 vezes menor que SOAP. O RPC é outra tecnologia de comunicação entre processos muito popular para a implementação do modelo cliente-servidor em computação distribuída. Contudo, estudos mostram que REST (vide capítulo Trabalhos Correlatos e Fundamentação) é melhor que RPC sob as perspectivas de escalabilidade, acoplamento e segurança [6].

Os acessos desordenados aos recursos compartilhados pelos processos concorrentes geram um comportamento imprevisível do sistema de computação distribuída. A gestão da concorrência entre dois ou mais processos que disputam acesso ao mesmo recurso na rede é uma fonte de dificuldade na fase de desenvolvimento de *software*. Há algumas metodologias distintas para a implementação de mecanismos de gestão de recursos compartilhados. Em [7], um projeto associa um *buffer* de comando e um *buffer* de dados para cada dispositivo, mas como o servidor não implementa qualquer bloqueio do dispositivo, os clientes devem implementar seu próprio mecanismo de arbitragem. Em [4], um projeto implementa mecanismos de sincronização (*no-wait*, *wait-after* e *wait-before*) no lado servidor e de comportamento (*triple-buffer*, *double-buffer* e *single buffer*) no lado cliente, mas esses mecanismos geram a possibilidade de que as atualizações do recurso ou sejam perdidas, pelo fato do servidor nunca enviá-las, ou sejam descartadas no lado cliente, gerando um tráfego de dados desnecessário na rede.

1.2.3 Segurança

Em geral, um processo sendo executado em uma unidade de computação embarcada em um robô pode prescindir de mecanismos de segurança de rede. No entanto, a integração com a internet sempre levanta preocupações com relação à segurança. É necessário, portanto, ter mecanismos de segurança para garantir que somente os robôs autorizados pelo *middleware* possam se comunicar entre si e que somente os usuários autenticados nele possam acessar ou controlar os robôs conectados na rede.

Para facilitar a operação em “tempo real” do *middleware* robótico é importante minimizar a sobrecarga de rede devido à segurança (por exemplo, em caso de *handshaking*). Por esta razão, o *middleware* pode ser deliberadamente projetado para não usar segurança, mas isso restringe sua área de atuação para uma rede isolada ou protegida por *firewall* [4]. No entanto, como mostrado neste trabalho, é possível garantir segurança de rede integrada a internet sem comprometer a operação em "tempo real" do *middleware.*

1.3 Objetivos

O principal objetivo desta tese é propor uma nova camada de *software* (*Middleware*) para integração e compartilhamento inteligente de dispositivos robóticos via rede de computação distribuída considerando os seguintes aspectos filosóficos: comunicação ponto a ponto, múltiplas linguagens de programação, codificação modular onde praticamente toda a complexidade reside em bibliotecas externas ao *middleware*, segurança de rede integrada a internet, *software* livre com código aberto, abstração da diversidade do *hardware* robótico, reutilização da infraestrutura de *software* para robôs entre diversos esforços de pesquisa, redução do acoplamento entre os múltiplos aplicativos, portabilidade do código e escalabilidade da arquitetura.

1.3.1 Objetivos Específicos

A especificação de um *middleware* flexível que permita o desenvolvimento em paralelo dos serviços e dos componentes, possibilitando sua composição futura através das interfaces definidas;

Desenvolvimento de um mecanismo de arbitragem inspirado nos conceitos da arquitetura de subsunção que permita a seleção e a integração de um conjunto variável e adequado de serviços e de componentes demandados pelos projetos de robôs;

Propor o emprego de recursos baseados nos serviços tradicionais de internet, adaptando-os ao *middleware* para a integração de componentes e serviços demandados pelos projetos de robôs;

Utilização do estilo arquitetônico REST para especificação dos componentes, possibilitando maior flexibilidade, facilidade de customização para uma aplicação específica e reuso;

Desenvolvimento, implementação e experimentação de componentes e serviços, que podem ser compartilhados com outras instituições, possibilitando o uso de outros componentes já

disponíveis dentro do projeto SOM4R;

Desenvolvimento de um robô móvel autônomo para aplicações *indoor* com capacidade de detectar obstáculos e com flexibilidade tanto na sua estrutura mecânica, quanto eletrônica, apresentando resultados confiáveis para o teste do *middleware* proposto, com qualidade e baixo custo.

1.4 Organização da Tese

O restante deste trabalho está organizado da seguinte forma:

- No capítulo 2, são apresentados e discutidos os trabalhos mais importantes existentes na literatura sobre *middleware* para robótica relacionados com esta pesquisa. Adicionalmente, são apresentados os fundamentos relacionados a arquitetura de *software* para configuração de *middleware*.

- No capítulo 3, são discutidas a especificação e a proposta do *middleware* SOM4R, descrevendo seus componentes e recursos de forma integrada e uniforme. Adicionalmente é apresentada uma discussão comparativa com os trabalhos relacionados citados no capítulo anterior.

- No capítulo 4, são apresentados os aplicativos desenvolvidos, a interface HMI e os resultados dos experimentos realizados. Adicionalmente, é descrito o robô móvel proposto e desenvolvido para a realização dos testes do SOM4R.

- Finalmente, o capítulo 5 apresenta uma síntese do trabalho realizado, suas principais contribuições e a indicação de alguns trabalhos futuros para a continuidade desta pesquisa.

Em seguida são listadas as referências bibliográficas utilizadas e os referidos apêndices.

1.5 Produção Científica

a) Artigos

Durante o desenvolvimento deste trabalho tivemos como produção científica os seguintes artigos:

- FREIRE, Ananda Lima ; BARRETO, G. A. ; VELOSO, M ; VARELA, A. T. . Short-Term Memory Mechanisms in Neural Network Learning of Robot Navigation Tasks: A

Case Study. In: 6th Latin American Robotics Symposium (LARS2009), 2009, Valparaíso. Annals of VI Latin American Robotics Symposium, 2009.

- FREIRE, Ananda Lima ; BARRETO, G. A. ; VELOSO, M ; VARELA, A. T. . A Dimensão Temporal no Aprendizado de Tarefas de Navegação de Robôs Móveis Usando Redes Neurais Artificiais. In: IX Congresso Brasileiro de Redes Neurais / Inteligência Computacional, 2009, Ouro Preto. Anais do IX Congresso Brasileiro de Redes Neurais / Inteligência Computacional, 2009, 2009.

- FREIRE, Ananda Lima ; BARRETO, G. A. ; VELOSO, M ; VARELA, A. T. . Comparação de Algoritmos de Classificação de Padrões em Tarefas de Navegação de Robôs Móveis. In: IX Simpósio Brasileiro de Automação Inteligente (SBAI 2009), 2009, Brasilia. Anais do IX Simpósio Brasileiro de Automação Inteligente, 2009.

- FILHO, José Tarcísio C.; Veloso, Marcus V.D.; CORDEIRO, Rafael V. Sistema Dedicado de Aquisição de Dados para Automação do Processo de Produção de Biodiesel. Anais do III CONGRESSO DA REDE BRASILEIRA DE TECNOLOGIA DE BIODIESE, Brasília, DF, 2009

- CORDEIRO, Rafael V.; FILHO, José Tarcísio C.; Veloso, Marcus V.D. , Formação de Engenheiro de Automação via Projeto de P&D Demandado pela Indústria de Biodiesel. Anais do XXXVIII CONGRESSO BRASILEIRO DE EDUCAÇÃO EM ENGENHARIA. Fortaleza, CE,2010.

- SOM4R: A Middleware for Robotic Applications based on the Resource-Oriented Architecture - Marcus V. D. Veloso, José T. Costa Filho, Guilherme A. Barreto, a ser submetido.

b) Patente

- Patente do robô depositada no Instituto Nacional da Propriedade Industrial (INPI) sob o número PI 1106417-0 e publicada inicialmente na Revista de Propriedade Industrial (RPI) número 2171 de 14/08/2012. Data do depósito: 11/10/2011. Depositantes: Fundação Núcleo de Tecnologia Industrial do Ceará - NUTEC (BR/CE), Universidade Federal do Ceará - UFC (BR/CE). Inventores: Marcus Vinicius Duarte Veloso, Rafael Vasconcelos Cordeiro, José Tarcísio Costa Filho. Título: ROBÔ AUTÔNOMO PARA VIGILÂNCIA E PROCESSO DE PRODUÇÃO DE ROBÔ AUTÔNOMO PARA VIGILÂNCIA.

c) Projetos

- Título: “PROJETO DE INFRA-ESTRUTURA COMPLEMENTAR DO LABORATÓRIO DE ROBÓTICA DO CENTRO DE REFERÊNCIA EM AUTOMAÇÃO E ROBÓTICA - CENTAURO”. Parceiros: NUTEC, UFC e IFCE. Fonte de Financiamento: MAPP (Monitoramento de Ações e Projetos Prioritários) - Governo do Estado do Ceará. Período: 2014/2016. Coordenador: Dr. José Tarcisio Costa Filho.

CAPÍTULO 2

2 - TRABALHOS CORRELATOS E FUNDAMENTOS

2.1 Trabalhos Correlatos

Com o ritmo acelerado da evolução dos dispositivos móveis (*smartphones, tablets, ultrabooks*) tem sido observado o surgimento de um novo paradigma na construção dos robôs, a saber: robôs construídos utilizando dispositivos móveis de uso geral (e.g. *netbook, notebooks, tablets* ou *smartphones*). Diferentemente dos robôs construídos usando *hardware* de propósito específico (e.g. PIC - *Programmable Controller Interface*, dsPIC), cujos recursos computacionais são limitados, o novo paradigma tem a vantagem de empregar dispositivos móveis que atualmente possuem muito mais poder computacional do que os de propósito específico, agregando ao robô a capacidade de processamento paralelo, várias opções de rede sem fio com conexão de banda larga e a disponibilidade de diversas plataformas (Sistemas Operacionais e Linguagens) de programação.

Este novo paradigma na construção dos robôs facilitou consideravelmente o desenvolvimento de aplicações que requerem integração dos recursos de um ou mais robôs em rede. Na verdade, a integração de recursos robóticos através da rede está dentro do núcleo de vários projetos de código aberto e fechado disponíveis na comunidade robótica. Entre os projetos de código fechado, podemos citar o *Microsoft Robotics Developer Studio* (MRDS) [33], que está disponível gratuitamente para uso mas não se tem acesso ao código fonte. Entre os de código aberto podemos citar o Orocos (*Open Robot Control Software*) [32], cujo foco é o controle de robôs industriais.

Neste trabalho, estamos particularmente interessados em projetos de código aberto que podem ser usados para uma variedade de aplicações robóticas, além de robótica industrial, tais como robótica de serviços, robótica de campo, robótica humanoide e robótica espacial. Neste sentido, destacam-se os seguintes projetos de código aberto que são flexíveis o suficiente para serem usados em qualquer uma dessas aplicações robóticas: *i)* Player [7]; *ii)* CLARAty (*Coupled Layer Architecture for Robotic Autonomy*) [3,10,11]; *iii)* CARMEN (*Carnegie Mellon Navigation*) [2]; *iv)* YARP (*Yet Another Robot Platform*) [4]; e *v)* ROS (*Robot Operating System*) [8].

2.1.1 Projetos *Open Source*

A seguir, descrevemos brevemente os projetos *open source* de integração de recursos robóticos em rede citados. As Figuras 2.1, 2.2, 2.3, 2.4 e 2.5 mostram alguns dos robôs compatíveis com esses projetos.

_i_) Player

O Player é um servidor que fornece acesso de rede transparente para uma variedade de sensores e atuadores frequentemente embarcados em robôs, e que também oferece duas ferramentas de simulação de ambientes, o Stage (2D) e o Gazebo (3D) [7]. Desenvolvido pela University of Southern California. É um dos projetos pioneiros e, provavelmente, a interface de controle de robô mais utilizada no mundo. É compatível com diversos robôs de pesquisa (e.g. Pioneer II family, iRobot) (vide Fig.2.1). Suas principais características são:

- A utilização da tecnologia *socket* (vide seção 2.2.3) permite que os clientes sejam desenvolvidos em diversas plataformas de sistema operacional e linguagens de programação com suporte a *sockets*;
- Os clientes podem se conectar a múltiplos servidores e os servidores podem aceitar conexões de vários clientes;
- Os clientes (e.g. controladores de robô, processadores de dados de sensores, entre outros) precisam se subscrever para um conjunto de dispositivos e especificar a frequência com que os dados devem chegar;
- Cada dispositivo tem associado a ele um *buffer* de comando e um *buffer* de dados;
- Por padrão, os clientes recebem dados a 10Hz, mas podem reduzir a taxa de dados para 5 Hz ou aumentar para 30Hz. O cliente a 30Hz e o cliente a 5 Hz podem ser conectados simultaneamente ao mesmo dispositivo;
- O Player não implementa qualquer mecanismo de bloqueio do dispositivo, cada novo comando irá substituir o antigo;
- O Player é essencialmente um protocolo.

Figura 2.1: Robô iRobot

***ii*) CLARAty (*Coupled Layer Architecture for Robotic Autonomy*)**

O CLARAty é um *framework* integrado para desenvolvimento de *software* robótico reutilizável [3,10,11]. Desenvolvido pelo Jet Propulsion Laboratory, Ames Research Center, Carnegie Mellon University e University of Minnesota, com apoio da NASA através do *Mars Technology Program*. Plataforma utilizada como ambiente de integração para a evolução dos rovers das séries Rocky e FIDO (vide Fig.2.2), precursores dos robôs Spirit e Opportunity (Mars Exploration Rovers). Suas principais características são:

- Arquitetura em duas camadas (*Functional layer*, *Decision layer*);
- A camada funcional é uma interface para todo o *hardware* do sistema e as suas funcionalidades, através da qual a camada de decisão usa o sistema robótico.
- A camada de decisão divide objetivos de alto nível em objetivos menores, os organiza no tempo em função das restrições conhecidas e do estado do sistema, e acessa as funcionalidades adequadas da camada funcional para alcançá-los.

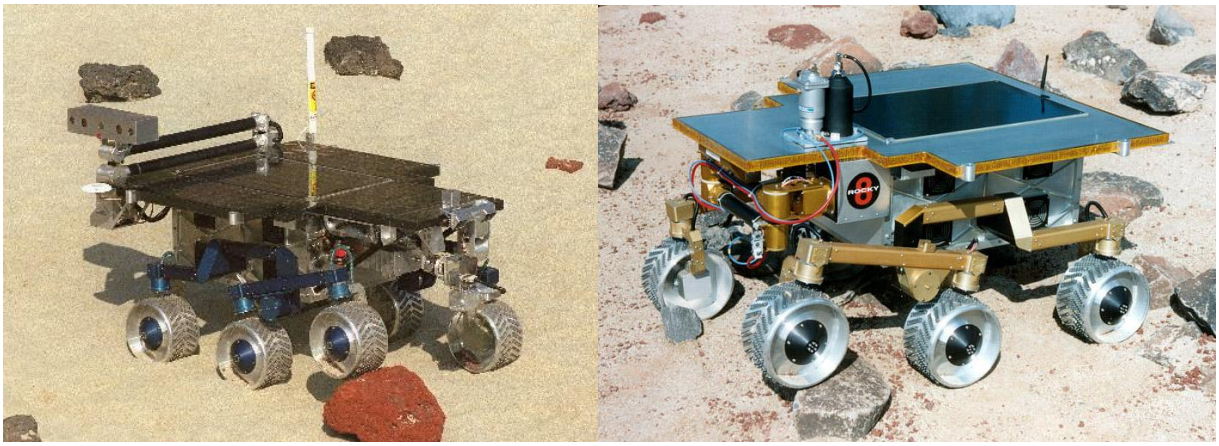

Figura 2.2: Robôs FIDO

***iii*) CARMEN (*Carnegie Mellon Navigation*)**

O CARMEN é uma coleção de ferramentas de *software* para controle de robôs móveis e sensores, projetado para fornecer primitivas básicas de navegação [2]. Desenvolvido pela Carnegie Mellon University. Foi patrocinado pelo Programa DARPA MARS (*Mobile Autonomous Robot Software*). Plataforma robótica com suporte a diversos robôs (e.g. iRobot, Pioneer I e II) (vide Fig.2.3). Suas principais características são:

- É um *software* modular organizado em uma arquitetura de três camadas;
- A camada base é responsável pelo gerenciamento de *hardware* e comunicação, e pela detecção de colisão;
- A camada intermediária implementa primitivas de navegação, incluindo localização, rastreamento dinâmico de objetos, e planejamento de trajetória;
- A camada superior é reservada para as tarefas em nível de usuário empregando primitivas da segunda camada;
- Módulos diferentes que executam a mesma função (por exemplo, os módulos para cada tipo de base) devem convergir para uma única interface abstrata;
- Utiliza o paradigma de *software* conhecido como o MVC (*Model-View-Controller*).

Figura 2.3: Robô Pioneer

***iv*) YARP (*Yet Another Robot Platform*)**

O YARP é um conjunto de bibliotecas, protocolos e ferramentas de *software* para ajudar a organizar a comunicação entre os sensores, processadores e atuadores [4]. Desenvolvido pelo Italian Institute of Technology (IIT). É a plataforma robótica utilizada pelo robô humanoide europeu iCub [41] (vide Fig.2.4), robô adotado por dezenas de laboratórios ao redor do mundo para pesquisas em cognição humana e inteligência artificial. Suas principais características são:

- É uma biblioteca simples, ligada ao código do cliente através do instanciamento das classes apropriadas;
- Utiliza o ACE, uma biblioteca de código aberto que fornece uma estrutura para programação concorrente através de diversos sistemas operacionais;
- Segue o padrão Observador (*Observer*), padrão de projeto de *software* que define uma dependência um-para-muitos entre objetos de modo que quando um objeto muda o estado, todos seus dependentes são notificados e atualizados automaticamente;
- O servidor pode atualizar o observável de três formas (mecanismos): *no-wait*, *wait-after* e *wait-before*. No lado do cliente, as opções de *buffer* são: *single-buffer*, *double-buffer* e *triple-buffer*;
- Uma porta é um objeto ativo que faz o gerenciamento de múltiplas conexões para uma determinada unidade de dados, podendo ser unicamente de entrada ou de saída;
- As portas são localizadas na rede por nomes simbólicos que são gerenciados por um servidor de nomes. Cada porta deve se registrar junto ao servidor de nomes. O tipo de protocolo é definido pela porta de saída;
- Canais de serviço são criados temporariamente para realizar o "aperto de mão" (*handshaking*) entre as portas;
- A comunicação é totalmente assíncrona.

Figura 2.4: Robô iCub

v) ROS (*Robot Operating System*)

O ROS é um *framework* para robótica que fornece ferramentas básicas de *softwar*e com mais de duas mil bibliotecas para ajudar desenvolvedores a criar aplicativos modulares para robôs [8]. Desenvolvido pela universidade de Stanford. Plataforma robótica utilizada por dezenas de robôs e manipuladores móveis, veículos autônomos e robôs humanoides, entre eles o NAO

(Vide Fig.2.5, a esquerda) e o Robonaut-2 (R2) da NASA (vide Fig.2.5, a direita), enviado para estação espacial internacional (ISS). Apesar do nome, ROS não é um sistema operacional, é um *framework* flexível que visa simplificar a tarefa de criar *software* para uma ampla variedade de plataformas robóticas, utilizando uma coleção de ferramentas, bibliotecas e convenções. Suas principais características são:

- Os conceitos fundamentais da implementação ROS são: *nodes* (módulos de *software*), mensagens, tópicos e serviços;
- O *node* envia uma mensagem ao publicá-la em um determinado tópico, que é simplesmente uma string (e.g. “odometria" ou "mapa"). Um *node* que está interessado em um certo tipo de dado vai assinar o tópico apropriado.
- Os serviços são definidos por um nome (uma *string*) e um par de mensagens estritamente tipadas, uma para a requisição e outra para a resposta.
- A negociação e configuração da conexão ponto-a-ponto ocorre em XML-RPC, um protocolo de comunicação com implementações na maioria das principais linguagens.
- Utiliza uma linguagem de definição de interface (IDL - *Interface Description Language*) para descrever as mensagens enviadas entre os módulos.
- É um projeto de *microkernel*, onde um grande número de pequenas ferramentas (e.g. rospack, rxgraph) são usadas para criar e executar os vários componentes ROS.
- Reutiliza código de outros projetos open source, tais como: simulador de ambientes do projeto Player, algoritmos de visão do OpenCV, e algoritmos de planejamento do OpenRAVE [7,8,21];

Figura 2.5: Robôs NAO (esquerda) e Robonaut-2 (direita)

2.2 Fundamentos

Nesta seção, apresentamos as principais tecnologias e conceitos utilizadas nesse trabalho, tais como: recursos, *framework* RDF (*Resource Description Framework*), protocolo TCP/IP (HTTP), tecnologia *Socket*, serviços web, arquitetura orientada a serviços (SOA), arquitetura orientada a recursos (ROA), estilo arquitetônico REST (*Representational State Transfer*) e *middleware*.

2.2.1 Recursos

O conceito de recurso evoluiu ao longo da história da web a partir da noção inicial de documentos estáticos endereçáveis ou arquivos para uma definição mais genérica e abstrata. Na especificação RFC3986 [43] do W3C (*World Wide Web Consortium*) o termo "recurso" é usado em um sentido geral para o que pode ser identificado por um URI (*Uniform Resource Identifier*). Entre os exemplos de recursos podemos citar: um documento eletrônico, uma imagem, uma fonte de informação com um propósito consistente, ou uma coleção de outros recursos. Os identificadores URIs têm um alcance global e são interpretados de forma consistente independentemente de contexto. No caso dos recursos acessíveis através da Internet, essa semântica do mapeamento (URI) é a única característica que é necessariamente estática, uma vez que essa semântica é o que distingue um recurso de outro. Cabe salientar que conceitos abstratos também podem ser recursos, como por exemplo: os operadores e operandos de uma equação matemática, os tipos de relacionamento ("pai", "empregado", dentre outros), ou valores numéricos (zero, um, infinito, dentre outros).

O crescente volume de recursos disponibilizados na Web, onde a maior parte dessa informação está disponível de uma forma fracamente estruturada, por exemplo, texto, áudio e vídeo, tem demandado por sistemas capazes de reconhecer e tratar estes recursos de forma mais significativa e seletiva quanto ao seu contexto. Nesse sentido, é importante ter cada tipo de informação devidamente identificada, de forma que permita a busca, extração, manutenção e mineração de informações [17,44]. XML é uma metalinguagem universal para a definição de marcação. Ela fornece uma estrutura uniforme e um conjunto de ferramentas (e.g. *parsers*) para o intercâmbio de dados e metadados entre os aplicativos. Contudo, XML não fornece qualquer meio para expressar a semântica (significado) dos dados [55].

2.2.2 RDF (*Resource Description Framework*)

RDF é um *framework* concebido para descrever recursos e representar informação na web, é uma linguagem criada para expressar proposições usando vocabulários formais especificados. Foi projetado para ser lido e entendido por computadores, com ênfase em generalidade e precisão para expressar proposições sobre qualquer assunto.

RDF é a linguagem básica da Web Semântica [17,44,55] ou Web dos Dados, que é um estágio evolutivo da web em que o *software* pode procurar, extrair, armazenar, trocar e utilizar informações "legível por máquina" distribuídas por toda a web. A Web Semântica incorpora significado às informações da web, proporcionando um ambiente onde máquinas e usuários trabalham em conjunto. RDF permite vincular um recurso a outros recursos na web, assim como HTML permite vincular uma página a outras páginas na web. Publicar os recursos na web usando RDF torna os dados mais portáveis e interoperáveis entre diferentes tipos de computadores e diferentes tipos de sistemas operacionais e linguagens de programação [17], além de permitir sua utilização em contextos de Web Semântica.

RDF é essencialmente um modelo de dados [55]. Os objetos RDF podem ser definidos como instâncias de uma ou mais classes. É possível estender o RDF usando RDFSchema [44]. O RDFSchema define o vocabulário usado em modelos de dados RDF. Isso permite a criação e a descrição de classes, subclasses e domínios. É possível definir novos vocabulários (atributos) para descrever recursos específicos, definir os tipos de objetos aos quais se aplicam esses atributos e os valores que eles podem ter, proporcionando um mecanismo de tipagem básica, e descrever as relações entre os objetos. Na Fig. 2.6 é apresentado o emprego do RDF para tipagem, ilustrando um estado representativo do recurso TTS (Text To Speech), detalhado no capítulo 3.

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
      "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
      <rdf:Description  rdf:ID="tts">
            <textToSpeech  rdf:datatype="&xsd;string">
                  Hello World</textToSpeech>
            <rdfs:id  rdf:datatype="&xsd;integer">1</rdfs:id>
      </rdf:Description>
</rdf:RDF>
```

```
<< TTS interface >>
+ id: integer
+ textToSpeech: string
```

Figura 2.6: Interface TTS. Exemplo de descrição do estado representativo de um recurso usando a sintaxe RDF/XML.

2.2.3 TCP/IP (*Transmition Control Protocol* / *Internet Protocol*)

O TCP/IP é um conjunto de protocolos de comunicação entre computadores em rede que trabalham em conjunto para o fornecimento de serviços. A pilha de estruturas de dados TCP/IP pode ser vista como um modelo de camadas, onde cada camada é responsável por um grupo de

tarefas e por fornecer um conjunto de serviços para o protocolo da camada superior. O modelo inicial da pilha TCP/IP é dividido em quatro camadas, a saber, camada de aplicações (protocolos HTTP, FTP, SMTP, RPC, SSH, entre outros), camada de transporte (protocolos TCP, UDP, entre outros), camada de rede (protocolos IP, ARP, ICM, entre outros), e a camada de interface física (protocolos Ethernet, Frame Relay, Token Ring, ATM, entre outros), conforme a Fig. 2.7. As camadas mais altas da pilha TCP/IP tratam com dados mais abstratos, deixando as tarefas de menor nível de abstração para os protocolos de camadas mais baixas, permitindo elas forneçam serviços que as camadas mais baixas não podem fornecer.

O protocolo HTTP (*Hypertext Transfer Protocol*) pertence a camada mais alta (camada de aplicações) da pilha TCP/IP. É um protocolo para sistemas distribuídos e colaborativos que apresenta menos restrições em *firewall*, possui uma elevada compatibilidade entre plataformas diferentes (linguagens de programação e sistemas operacionais), e está presente em muitos dispositivos embarcados (e.g. microcontroladores PIC com módulo *Ethernet*).

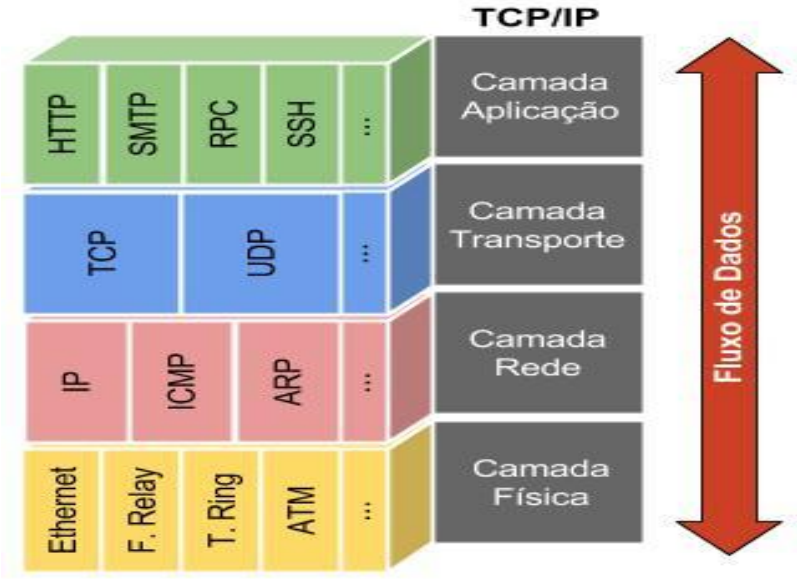

Figura 2.7: Pilha de protocolos TCP/IP.

O protocolo HTTP utiliza o modelo cliente-servidor [16], como a maioria dos protocolos de rede, baseando-se no paradigma de requisição e resposta. Para que o protocolo HTTP consiga transferir seus dados pela rede, é necessário que os protocolos TCP e IP tornem possível a conexão entre clientes e servidores através da tecnologia *Socket* de rede (*socket* TCP no caso do HTTP). Um *Socket* de rede é a extremidade de um fluxo de comunicação entre processos através de uma rede de computadores. Um *socket* é um conceito ou abstração computacional que surgiu originalmente no sistema operacional Unix BSD (*Berkeley Software Distribution*). Uma API de *sockets* é uma interface de programação de aplicativos que permite que os programas controlem e usem *sockets*. A API *Socket* é normalmente fornecida pelo sistema operacional. Um *socket* se caracteriza por uma única combinação composta por: *i*) endereço IP local e o número da porta,

ii) protocolo de transporte (e.g. TCP, UDP, ou outros), e *iii*) endereço do *socket* remoto - apenas para *sockets* TCP (*established*), uma vez que um servidor TCP pode servir vários clientes ao mesmo tempo.

No protocolo HTTP, o tipo de Identificador Uniforme de Recurso (URI) utilizado é chamado de URL (*Uniform Resource Locater*), composto pela identificação do protocolo, pelo endereço do servidor e pelo caminho e nome do recurso requisitado (protocolo://servidor:porta/caminho/recurso).

No processo de comunicação HTTP um programa requisitante (cliente) estabelece uma conexão com um outro programa receptor (servidor) e lhe envia uma requisição, contendo a URI, a versão do protocolo, uma mensagem MIME [1] contendo os modificadores da requisição, informações sobre o cliente e, possivelmente, um conteúdo no corpo da mensagem. O servidor responde com uma linha de status incluindo sua versão de protocolo e os códigos de erro informando o resultado da operação, seguida pelas informações do servidor, metainformações da entidade e um possível conteúdo no corpo da mensagem. Após o envio da resposta pelo servidor, encerra-se a conexão estabelecida.

O HTTP emprega um sistema de classificação da solicitação de mensagem (interfaces uniformes) para indicar a ação desejada para ser executada no recurso identificado (e.g. GET, POST, PUT e DELETE). As interfaces uniformes do protocolo HTTP podem ser usadas para formar um CRUD de dados. O que este recurso representa, sejam dados pré-existentes ou dados que são gerados de forma dinâmica, depende da implementação do servidor. A especificação HTTP/1.0 definiu os métodos GET, POST e HEAD, e a especificação HTTP/1.1 adicionou os métodos OPTIONS, PUT, DELETE, TRACE e CONNECT.

2.2.4 Arquitetura Orientada a Serviço (SOA)

A arquitetura SOA (*Service Oriented Architecture*) [45,53,54,59] representa uma nova geração de plataforma de computação distribuída. Ela promove a integração e orquestração de processos por meio de serviços, e permite que novas aplicações sejam criadas através da combinação de diversos serviços. Os princípios fundamentais de SOA são:

- A disponibilização das funcionalidades implementadas pelas aplicações na forma de serviços autônomos;
- O baixo acoplamento (grau de dependência) entre o serviço e o cliente;
- Os serviços permitem o acesso as suas funcionalidades ou capacidades através de contratos bem definidos, a partir de qualquer processo ou sistema.

[1] MIME é o padrão utilizado para codificar dados em formato de textos ASCII para serem transmitidos pela Internet.

Na arquitetura SOA, o ciclo de vida de um serviço é independente da interação com o cliente. A interação é do tipo *stateless*, onde cada solicitação é tratada pelo serviço como uma operação independente, que não está relacionada a qualquer solicitação anterior. O cliente envia uma mensagem descrevendo todas as informações necessárias a cumprir a solicitação e o serviço responde com uma mensagem descrevendo completamente a resposta. Após o servidor entregar a resposta requisitada por um cliente, ele "esquece" daquele cliente.

2.2.5 Serviços Web

O serviço web [45,46,47] é uma tecnologia importante para a implementação da arquitetura SOA. Pela definição do W3C (*World Wide Web Consortium*), comunidade internacional que desenvolve padrões abertos para garantir o crescimento a longo prazo da Web, um serviço web [48] é um sistema de *software* identificado por um URI (*Uniform Resource Identifier*) [43], cujas interfaces públicas são definidas e descritas usando XML (*Extensible Markup Language*) [49]. Uma característica importante do serviço web é que a interface XML é independente da plataforma e da linguagem de implementação do aplicativo, de forma que alterações no serviço causam impacto mínimo no restante dos códigos de um sistema de computação distribuída [45].

Os serviços web oferecem uma interface padrão XML que define e transfere o estado representativo do recurso entre o serviço e o cliente. No caso de um dispositivo sensor e/ou atuador, o serviço web provedor do recurso contém toda lógica de controle sobre esse dispositivo, envolvendo a conexão, configuração, leitura e envio de dados, permitindo suportar uma comunicação bidirecional.

Os serviços web podem ser implementados em diferentes linguagens, de acordo com as necessidades de performance e acessibilidade de cada recurso. Particularmente, nesse trabalho, empregamos as linguagens Python, C/C++, Java, Javascript, Php e Flash/Flex.

2.2.6 Arquitetura ROA (*Resource Oriented Architecture*)

A arquitetura orientada a recursos é um conjunto de restrições adicionada a arquitetura orientada a serviços (SOA) [15]. Os recursos também têm restrições declaradas, ou seja, eles têm um ciclo de vida independente, uma referência mundial única, e um estilo de interação baseado em troca de mensagens *stateless*. Além disso, os recursos têm um sistema de classificação da solicitação de mensagem, as interfaces uniformes, que no caso do protocolo HTTP são os métodos GET, POST, PUT, DELETE, entre outros. A arquitetura ROA forma um conjunto específico de orientações para uma implementação do estilo arquitetônico REST. O REST é uma manifestação da orientação a recursos (ROA) [15].

2.2.7 REST (*REpresentational State Transfer*)

O estilo arquitetônico REST foi desenvolvido como um modelo abstrato da arquitetura Web moderna para orientar a reformulação do protocolo HTTP (*Hypertext Transfer Protocol*), através da identificação de falhas na arquitetura preexistente, e para a definição do identificador uniforme de recursos (URI - *Uniform Resource Identifier*) [16]. O termo REST foi introduzido e definido na tese de doutorado de Fielding [42] e representa um novo estilo arquitetônico de *software* para sistemas distribuídos.

O REST fornece um conjunto de restrições arquitetônicas que, quando aplicadas como um todo, enfatizam a escalabilidade das interações entre os componentes e a generalidade de interfaces [16]. É um conjunto coordenado de restrições arquiteturais que tenta minimizar a latência e a comunicação em rede e maximizar a independência e a escalabilidade das implementações de componentes, através da colocação de restrições semânticas no conector. Ele permite a reutilização de interações, a substituibilidade dinâmica de componentes e o processamento de ações por parte de intermediários, atendendo assim as necessidades de um sistema hipermídia distribuído em escala de Internet [42].

REST utiliza um URI para identificar um recurso em particular envolvido numa interação entre componentes. Os conectores REST (e.g. cliente, servidor) fornecem uma interface genérica para acessar e manipular o conjunto de valores de um recurso, independentemente do tipo de *software* que está realizando a interação. Componentes REST executam ações em um recurso usando uma representação para capturar o estado atual ou previsto do recurso, e transferir esta representação entre os componentes. A representação consiste de dados, metadados descrevendo os dados e até metadados para descrever os metadados. O metadado está na forma de pares de nome e valor, em que o nome corresponde a uma norma que define a estrutura e semântica do valor. Um componente pode incluir tanto conectores cliente como servidor [16].

Todas as interações REST são *stateless*. Isto é, cada solicitação contém todas as informações necessárias para um conector compreender a solicitação, independente de qualquer solicitação anterior. Isso elimina a necessidade dos conectores de manter o estado do aplicativo entre as solicitações, reduzindo o consumo de recursos físicos e melhorando a escalabilidade.

O estilo REST impõe uma restrição de interface uniforme [16]. Numa chamada de serviço web do tipo REST usando o protocolo HTTP, a ação a ser realizada pelo serviço é especificada no campo *Method* do cabeçalho HTTP, ele determina o que o servidor deve fazer com o URL fornecido no momento da requisição de um recurso, ou seja, as informações que restringem o escopo da ação que está ocorrendo sempre são armazenadas no corpo da mensagem XML. Dado um URL arbitrário, qualquer destinatário desse URI sabe que pode executar um HTTP GET para recuperar uma representação desse recurso, e que se pode criar, atualizar e excluir esse recurso usando os métodos HTTP POST, HTTP PUT e HTTP DELETE,

respectivamente.

Toda transação que transparece na web é baseada na lógica de representação, no sentido em que o recurso não chega a ser manipulado diretamente pelo aplicativo cliente, o que acontece na prática é que várias representações do recurso são preparadas pelo servidor e enviadas para consumo dos clientes, ou são enviadas pelo cliente e processadas pelo servidor, que faz a manipulação direta do recurso. Isto permite um isolamento maior entre os serviços e os clientes. A chamada de um serviço web REST é realizada de forma independente da tecnologia empregada na comunicação cliente-servidor.

2.2.7 *Middleware*

O *middleware* [1,20,23,60] (mediador) é uma camada de *software* que reside entre o sistema operacional (e APIs) de rede do sistema de computação distribuída e a camada de aplicação, afim de facilitar o desenvolvimento de aplicativos na camada de *software*, permitindo a comunicação entre as aplicações distribuídas. Os mediadores evoluíram a partir de *softwares* simples, que abstraem detalhes da rede para as aplicações distribuídas, para *softwares* sofisticados que lidam com muitas funcionalidades importantes para as aplicações, com relação a distribuição, heterogeneidade e mobilidade [20].

Sistemas distribuídos exigem que processos em execução em diferentes espaços de endereçamento de memória, potencialmente em diferentes servidores, sejam capazes de se comunicar. O papel essencial do *middleware* é gerenciar a complexidade e heterogeneidade das infraestruturas distribuídas e, assim, proporcionar um ambiente de programação mais simples para os desenvolvedores de aplicativos distribuídos [23]. Ele fornece um conjunto de serviços a serem chamados pelos aplicativos para utilizar o *middleware*.

Comentário Final

Os projetos Player, CLARAty, CARMEN, YARP e ROS não têm em absoluto a pretensão de ser uma solução abrangente para todas as necessidades dos desenvolvedores de aplicativos para robôs, dada a diversidade de tarefas e metas na área da robótica. As pesquisas [9,13,14] têm possibilitado identificar limitações e sugerir melhorias nesses projetos. Dessa forma, podemos constatar que o compartilhamento e integração dos recursos de um ou mais robôs através da rede permanece um grande desafio.

Uma alternativa importante é a possibilidade de reuso dos projetos e dos códigos já existentes. Nesse sentido, nossa abordagem foi procurar criar um *middleware* que pode ser integrado ao “ambiente” (*workspace*) dos demais projetos, não necessariamente competindo

diretamente com eles, mas também agregando funcionalidades (e.g. segurança, modularidade e escalabilidade) que faltam a um ou a outro projeto (vide capítulo 3, seção 3.2.6).

Nesse capítulo, estão apresentados e discutidos os trabalhos mais importantes existentes na literatura sobre *middleware* para robótica relacionados com esta pesquisa. Adicionalmente, estão apresentados os fundamentos relacionados a arquitetura de *software* para configuração de *middleware*. No capítulo seguinte, uma descrição detalhada do *middleware* proposto e as suas características são apresentadas e discutidas no contexto de tecnologias alternativas existentes.

CAPÍTULO 3

3 - SOM4R - SIMPLE and OPEN MIDDLEWARE FOR ROBOTICS

Neste capítulo, apresentamos os conceitos preliminares, a especificação e implementação do *middleware* SOM4R. Também são apresentados gráficos detalhados que descrevem o fluxo de informações entre os processos, módulos e recursos para facilitar a compreensão da abordagem proposta. No final fazemos uma discussão mais aprofundada comparando características comuns dos projetos para integração de recursos robóticos em rede citados no Capítulo 2 (Trabalhos Correlatos e Fundamentos).

3.1 Conceitos Preliminares

O *middleware* proposto é uma camada de serviços que faz interface com a camada base de sensores e atuadores e com a camada superior de aplicativos de usuários, conforme mostrado na Fig. 3.1 a ser detalhada no decorrer desse capítulo. A camada base compreende os dispositivos sensores e atuadores conectados diretamente ao computador ou microcontrolador do robô e reconhecidos pelos respectivos *drivers* do sistema operacional ou *firmware*. A camada superior é composta pelos aplicativos de *software* que são responsáveis por funções tais como: comando de voz, detecção de obstáculos, sistema de navegação, identificação de pessoas, entre outras funções utilizadas pelo usuário através de uma Interface Humano-Máquina baseada na Web. Esses aplicativos devem interagir com os serviços do *middleware*, a fim de utilizar os recursos dos robôs para atingir seus objetivos e realizar as funções sob suas responsabilidades.

Mais especificamente, o *middleware* SOM4R é formado por um conjunto de serviços web identificados por URIs e organizados de forma modular, seguindo a arquitetura orientada a recursos (ROA - *Resource Oriented Architecture*) [15] para implementar uma estrutura de *software* para robôs autônomos. Assim, mudanças num serviço web causam impacto mínimo sobre os outros serviços. Neste sentido, o SOM4R apresenta interfaces para todos os sistemas de *hardware* e suas funcionalidades, de forma independente da plataforma de computação distribuída e da linguagem de implementação dos seus serviços web.

Os serviços web do *middleware* especificam as interfaces de abstração, definidas e descritas utilizando linguagem XML (*Extensible Markup Language*), para acessar os recursos do(s) robô(s), tais como sensores, atuadores e serviços, e para transferir o estado representativo

(REST) desses recursos em múltiplos níveis de granularidade dos serviços, empregando um sistema de classificação da solicitação de mensagem (interfaces uniformes). As interfaces de abstração definidas para todos os serviços web implementados estão detalhadas adiante na seção 3.2. Os serviços web são responsáveis por disponibilizar os recursos na rede através da plataforma web usando o protocolo HTTP.

No *middleware* proposto, qualquer informação ou conceito que possa ser "nomeado" por um substantivo (*noun*) se qualifica como um recurso em potencial. Um recurso é um mapeamento conceitual para um conjunto de entidades [42]. O recurso, identificado por um URI, é a informação proveniente do sensor, é a informação enviada ao atuador, como também é a informação de um serviço web. Como exemplo, um recurso pode ser a informação do estado de um dispositivo robótico (e.g. webcam, laser, gps, *hardware* sensor/atuador, entre outros), ou a informação resultante do processamento dos recursos fornecidos por outros serviços (e.g. comando de voz, detecção de faces, detecção de marcadores). É possível encontrar vários *nouns*, de granularidades diferentes, de acordo com o cenário de cada aplicação cliente. Granularidade diz respeito ao nível de detalhe ou de resumo contido nas unidades de dados existentes (recursos). Por exemplo, o equipamento Kinect possui dados de diversos sensores, tais como acelerômetros, câmeras e microfones. Podemos modelar os dados desses sensores como um único recurso (granularidade grossa) ou como diversos recursos de granularidade fina, e neste caso, um recurso representa apenas a informação de um sensor.

No SOM4R, há a garantia de que a troca de dados entre cliente e servidor é feita de forma atômica, sem acoplamento entre os aplicativos de *software* e o *middleware*. Cabe salientar que REST tem se tornado uma nova alternativa ao estilo RPC-SOAP para a arquitetura de serviços web por fazer uso completo dos recursos da web. Em [12], é realizada uma bateria de testes para medir o desempenho dos serviços web REST sendo demonstrado que ele é mais adequado para realizar a integração de dados distribuídos em escala web. Para implementar o estilo arquitetônico REST no SOM4R, usamos as interfaces uniformes do protocolo HTTP para formar um CRUD de dados (*Create*, *Read*, *Update*, *Delete*), onde o método GET é usado para recuperar uma representação de um recurso ou realizar uma consulta, o POST é empregado para criar um recurso novo, nomeado de forma dinâmica, o PUT é utilizado para atualizar um recurso já existente, e o DELETE é usado para remover um recurso.

No SOM4R, o estado representativo de um recurso é transferido apenas sob demanda, onde a frequência é determinada pelo cliente em tempo de execução de acordo com as suas necessidades ou capacidade de processamento, reduzindo o tráfego de dados no sistema de computação distribuída.

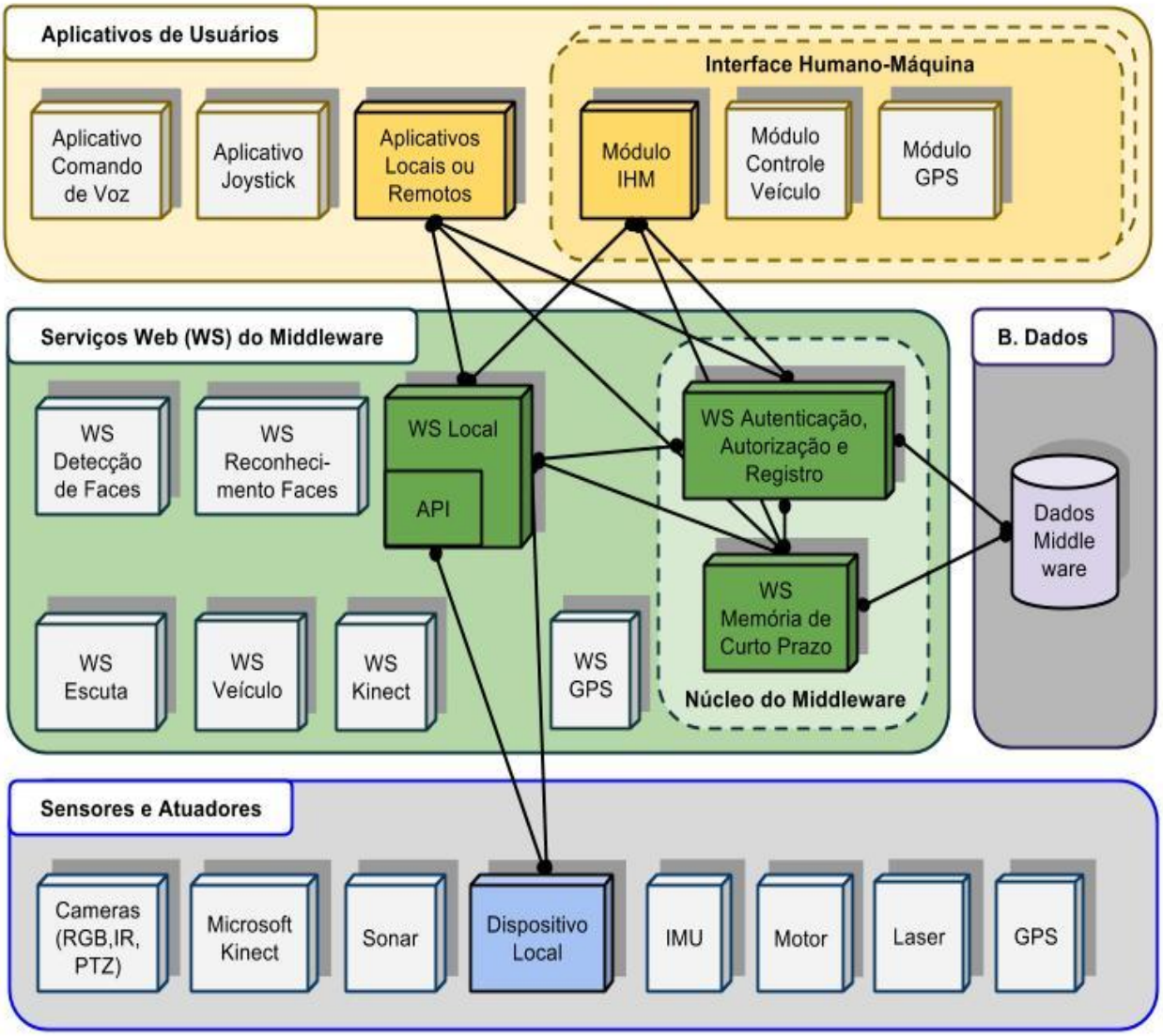

Figura 3.1: Modelo genérico do fluxo de dados e de controle entre as aplicações, os serviços web, o núcleo do *middleware* e o servidor de banco de dados.

3.2 Implementação

Nesta seção, descrevemos em mais detalhes a implementação dos serviços web do núcleo do *middleware* SOM4R e a segurança. Apresentamos também os serviços adicionais construídos especificamente para um robô móvel com rodas autônomo (apresentado no Capítulo 4), utilizando o *middleware* proposto, a fim de integrar as seguintes funcionalidades preliminares: comando de voz, detecção de face, detecção de marcadores (*landmarks*), locomoção do veículo, TTS (*Text To Speech*), posicionamento GPS (*Global Positioning System*), monitor de bateria, detecção de obstáculos em 3D. A ênfase é dada à estrutura de controle, suporte para robótica de enxame, integração com outro *middleware*, e funcionalidades do núcleo e dos serviços

adicionais. Uma discussão é apresentada a seguir.

Todos os códigos do projeto foram implementados na forma de *software open source* e estão disponíveis no site do SOM4R (**http://som4r.net**).

3.2.1 Segurança

Para suportar um nível básico de segurança de acesso aos serviços web integrantes do *middleware* proposto, utiliza-se uma abordagem inspirada no protocolo de segurança OAuth 2.0 [57] e baseada no método de autenticação de acesso HTTP *Digest* (RFC2617) [18,19]. A motivação para esta abordagem resulta dos seguintes fatos: *i)* HTTP *Digest* é o método de autenticação usado pelos servidores HTTP para validar a autenticação dos clientes, e *ii)* OAuth, atualmente adotado pelas grandes empresas de TI, é um protocolo aberto, simples e padrão, que permite a autorização segura entre aplicações web e *desktop*.

OAuth 2.0 define quatro papeis no processo de autenticação e autorização: a) *resource owner*, b) *authorization server*, c) *resource server* e d) *client*. Neste contexto, o núcleo do SOM4R realiza funções equivalentes aos papéis de: *i) resource owner* - responsável pela permissão de acesso a um recurso protegido; e *ii) authorization server* - emite o *access token*[2] [57] para os clientes após a autenticação e autorização do *resource owner*. Do mesmo modo, os outros serviços web do *middleware* realizam função semelhante ao papel de *resource server* que permite acesso ao recurso protegido através de requisições com *token*, enquanto que os aplicativos realizam o papel de *client* fazendo as solicitações de recursos em nome do *resource owner* e com a sua autorização (*token*).

O SOM4R utiliza dispositivos de segurança com criptografia MD5 (HTTP *Digest* e *access tokens*) cujo algoritmo de *hash* unidirecional de 128 bits é descrito na RFC1321, permitindo a utilização segura do *middleware* através da Internet. Desse modo, o cliente precisa estar autenticado e autorizado no SOM4R para ter acesso aos recursos protegidos pelo *middleware* , conforme os passos descritos (Figuras 3.2 e 3.3) utilizando a notação *Business Process Modeling Notation*[3] (BPMN) [39]. Para aumentar a segurança, os aplicativos clientes podem realizar uma conexão segura (HTTPS) com os serviços do *middleware*, pois a incapacidade do cliente em confirmar a identidade do servidor é uma fraqueza do método de autenticação HTTP *Digest*.

[2] *Access tokens* são credenciais utilizadas para acessar os recursos protegidos.

[3] Um padrão para modelagem de processos de negócio que fornece uma notação gráfica para a especificação de processos. BPMN é baseado em uma técnica de fluxograma semelhante aos diagramas de atividades UML (*Unified Modeling Language*) e também é muito utilizada na engenharia de *software*.

Para o processo de autenticação, ilustrado na Fig. 3.2, o cliente precisa realizar uma requisição indicada pelo passo 1 e informar a resposta adequada ao desafio, conforme o passo 3, enviado pelo SOM4R no passo anterior. A resposta ao desafio é calculada com base no *nonce*, número aleatório gerado pelo *middleware*, no *realm* (domínio ou descrição do computador ou sistema que está sendo acessado) e nos dados usuário e senha, conhecidos apenas pelo cliente e pelo servidor [18,19]. Quando o cliente informar a resposta adequada, conforme o passo 4, ele vai receber uma autorização de acesso (*token*) do SOM4R indicada pelo passo 5.

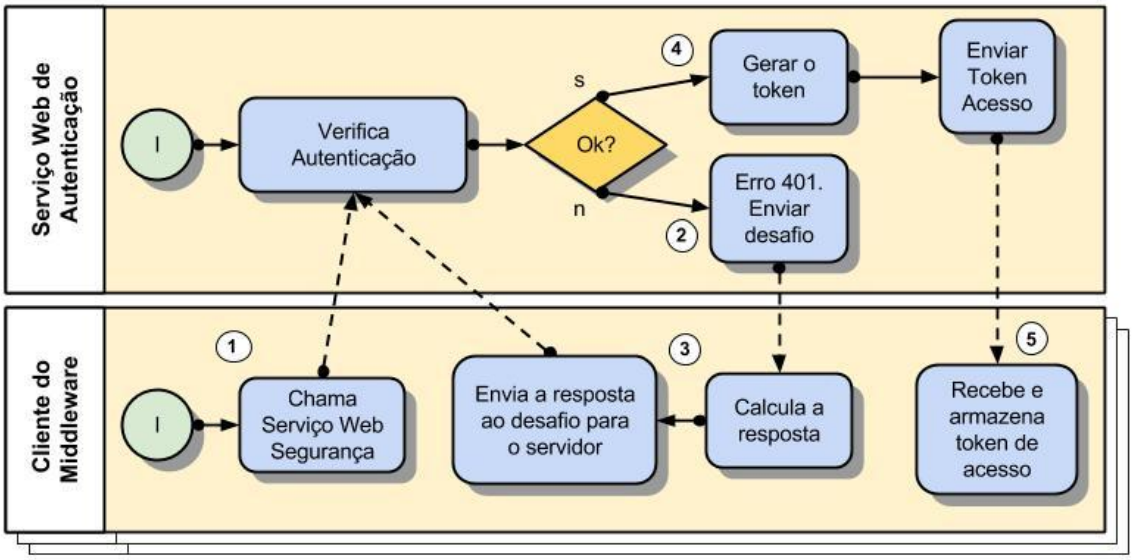

Figura 3.2: Modelagem BPMN do processo de autenticação do SOM4R.

O cliente precisa estar autorizado no SOM4R, além da autenticação HTTP *Digest* já discutida, para ter acesso aos recursos protegidos pelo *middleware*. A Fig. 3.3 descreve o processo de autorização usando a notação BPMN. O processo de autorização do SOM4R inicia quando um serviço web do *middleware* recebe uma requisição feita por um cliente conforme indicado pelo passo 1. O serviço web faz uma requisição de autorização de acesso indicada pelo passo 2, contendo os *tokens* do cliente e do *web service*, para o serviço de autorização do *middleware*. Esse serviço é responsável pela validação dos *tokens* e verificação de autorização de acesso aos recursos entre serviços e aplicações, retornando ao serviço web requisitante, conforme o passo 3, o tempo de validade (*timeout*) do *tokens* recebidos. Desse modo, quando o *middleware* responder com a permissão de acesso (ou seja, *timeout* > 0), o serviço web pode fornecer, conforme o passo 4, o recurso solicitado pelo cliente (serviço ou aplicativo). Caso contrário (*timeout* = 0), o serviço web vai responder ao cliente com o erro HTTP 401 (*Unauthorized*).

Cabe salientar que o SOM4R emprega *access tokens* com um prazo de validade (*timeout*) que permitem a configuração dinâmica da comunicação entre processos em tempo de execução, proporcionando segurança na identificação dos serviços e tornando os fluxos de dados e de controle mais seguros e configuráveis.

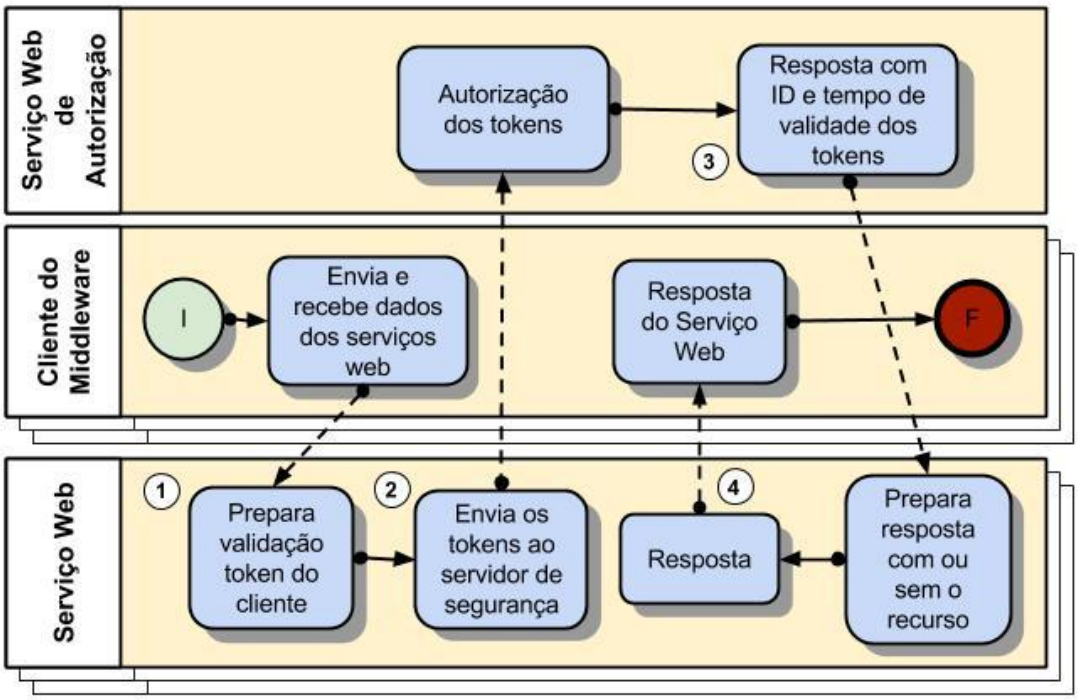

Figura 3.3: Modelagem BPMN do processo de autorização do SOM4R.

3.2.2 Núcleo do *Middleware* (*Core*)

Os serviços web do núcleo do SOM4R são responsáveis pelo registro, autenticação e autorização (segurança) e memória de curto prazo do *middleware* proposto, a saber:

i) Serviço Web de Autenticação e Registro

No SOM4R, este serviço é responsável pela autenticação e segurança do *middleware*, controlando o acesso aos recursos empregando o método HTTP *Digest Authentication* [18] e utilizando *access tokens* [19]. A fim de obter o acesso aos recursos protegidos pelo ambiente do *middleware* proposto, todos os serviços e aplicações têm que iniciar a conexão e o processo de autenticação e registro com este serviço, conforme detalhado anteriormente e ilustrado na Fig. 3.2.

A topologia de rede ponto a ponto requer algum tipo de mecanismo de pesquisa para permitir que os processos encontrem uns aos outros em tempo de execução [8]. Nesse sentido, o serviço de registro mantém uma lista atualizada dos serviços e aplicativos autenticados pelo SOM4R com seus respectivos endereços.

***ii)* Serviço Web de Autorização**

Responsável por validar a comunicação entre processos. Todas as comunicações entre as aplicações e serviços conectados ao *middleware* proposto são validadas usando este serviço de autorização, conforme detalhado anteriormente na seção 3.2.1 e ilustrado na Fig. 3.3. Os serviços web do SOM4R podem manter em memória o tempo (*timeout*) de validade do *token* do cliente, reduzindo o tráfego de dados através da rede.

***iii)* Serviço Web de Memória de Curto Prazo**

Responsável por manter um histórico em banco de dados dos recursos e das atividades e eventos recentes relacionados aos módulos do *middleware* que pode ser embarcado num robô. Este serviço também permite a consulta de atividades e eventos anteriores para uso na tomada de decisão dos aplicativos e serviços do *middleware* proposto. Por exemplo, o aplicativo "Saudando uma Pessoa", descrito na seção 4.2, faz uma busca na memória de curto prazo para evitar saudar a mesma pessoa mais de uma vez num certo intervalo de tempo. Em outro exemplo, as atividades do aplicativo "Olhe para Mim" (vide seção 4.2) estão associadas ao movimento realizado pelo robô no último minuto. Ou seja, a memória de curto prazo permite simplificar os módulos de *software*, uma vez que eles não precisam manter uma estrutura de dados interna para memorizar as suas próprias atividades ou as atividades dos outros módulos.

Nesse trabalho estabelecemos empiricamente, para o robô considerado, que a memória de curto prazo só mantém registros ocorridos nas últimas 24 horas, evitando comprometer a capacidade de armazenamento e a velocidade das consultas.

3.2.3 Serviços Adicionais

Nesta seção, vamos descrever brevemente alguns serviços adicionais que foram implementados no SOM4R para permitir o acesso dos clientes aos vários recursos de um robô móvel com rodas (mais sobre o robô no Capítulo 4).

Uma característica importante do *middleware* proposto é a sua capacidade de incorporar facilmente várias bibliotecas de código aberto amplamente utilizadas para tarefas ou sensores específicos, como por exemplo OpenCV [21], ARToolKit [23,50,51], OpenKinect [24] e PyFaces [35]. Alguns serviços descritos a seguir foram desenvolvidos utilizando tais bibliotecas.

i) Serviço Web do Veículo (robótico)

Responsável pela comunicação direta com o *firmware* que controla os motores de um veículo robótico conectado através de uma porta USB, proporcionando a leitura do estado atual do recurso (velocidade, direção e sentido) e o recebimento de comandos que são enviados para o *hardware* controlador dos motores do sistema de propulsão (Fig. 3.4). No robô considerado, a comunicação entre *firmware* e serviço web utiliza a interface USB *Human Interface Device Class* (HID) (vide apêndice A-3.8). A vantagem de usar a interface HID é que, para a maioria das necessidades, o suporte existente para dispositivos HID geralmente pode ser adaptado muito mais rápido do que ter que criar um protocolo completamente novo, e a maioria dos sistemas operacionais já reconhece dispositivos padrão HID sem precisar de um *driver* especializado (e.g. mouse, webcam).

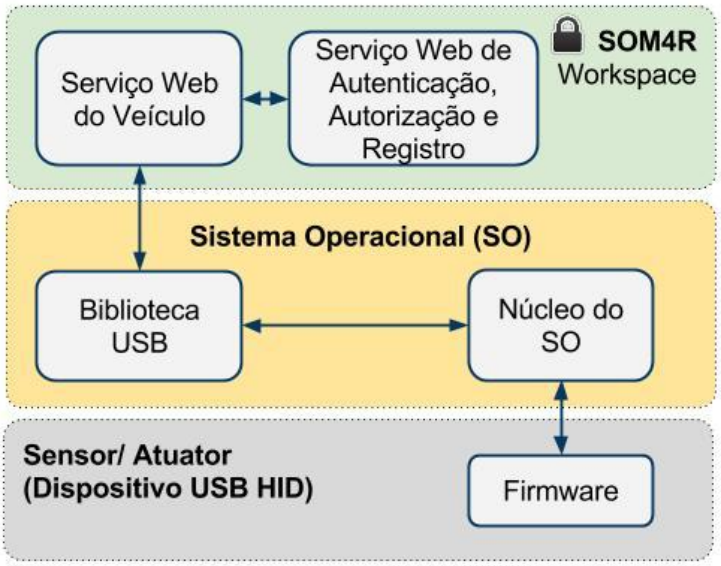

Figura 3.4: Processo de comunicação entre o Serviço Web do Veículo e o *Firmware* que controla os motores.

A interface de abstração definida para o serviço web do veículo robótico é apresentada na Fig. 3.5. Nesse caso, o atributo direção (*direction*) é o ângulo entre a posição frontal do robô e a direção do deslocamento, e também é informada a velocidade (*velocity*) como um percentual inteiro entre 0 e 100, entre outros dados.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
        'http://www.w3.org/2001/XMLSchema#'>]>
<rdf:RDF
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
        xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
        <rdf:Description rdf:ID='vehicle'>
                <rdfs:id  rdf:datatype="&xsd;integer">1</rdfs:id>
                <direction   rdf:datatype="&xsd;integer">2</direction>
                <velocity   rdf:datatype="&xsd;integer">60</velocity>
                <time   rdf:datatype="&xsd;integer">0</time>
                <status   rdf:datatype="&xsd;string">ready</status>
        </rdf:Description>
</rdf:RDF>
```

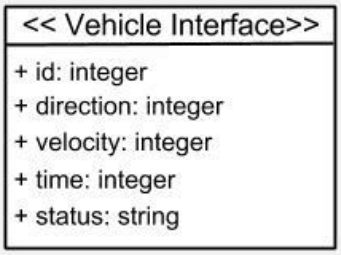

Figura 3.5: Interface do Veículo Robótico.

***ii)* Serviço Web de Reconhecimento de Voz**

Responsável pelo reconhecimento de comandos de voz humana, utilizando o microfone do computador do robô onde o serviço está ativo, e pela disponibilização do resultado para outros serviços e aplicações. Este serviço utiliza a biblioteca CMU Sphinx [29], um conjunto de ferramentas de código aberto para o reconhecimento de voz desenvolvido na Universidade de Carnegie Mellon. Essa implementação de *software* foi baseada em exemplos do projeto CMU Pocketsphinx.

A interface de abstração definida para o serviço web de reconhecimento de voz é apresentada na Fig.3.6. Nesse caso, o atributo ‘uttid’ é um sequencial inicializado quando o serviço é iniciado, o ‘parcial’ é a palavra ou frase que está em processo de reconhecimento, e o ‘result’ é a última palavra ou frase que foi reconhecida pelo serviço.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
        'http://www.w3.org/2001/XMLSchema#'>]>
<rdf:RDF
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
        xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
        <rdf:Description rdf:ID='listen'>
                <uttid  rdf:datatype="&xsd;integer">000002030</uttid>
                <partial   rdf:datatype="&xsd;string"></partial>
                <result   rdf:datatype="&xsd;string">YES</result>
        </rdf:Description>
</rdf:RDF>
```

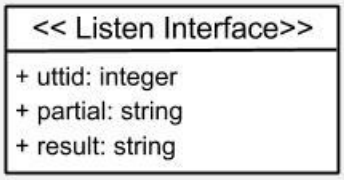

Figura 3.6: Interface Listen.

iii) Serviço Web para TTS (*Text To Speech*)

Responsável por receber uma palavra ou frase e enviá-la para o sintetizador de voz do computador do robô onde o serviço web TTS está instalado, permitindo que o robô tenha uma interface de voz com o usuário. Este serviço utiliza o *software* espeak [30], um *software* sintetizador de voz compacto de código aberto com suporte para mais de 30 idiomas.

A interface de abstração definida para o serviço web para TTS é apresentada na Fig.3.7. Nesse caso, o atributo 'textToSpeech' é a palavra ou frase que o serviço deve enviar ao sintetizador de voz.

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
        "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
        <rdf:Description  rdf:ID="tts">
                <textToSpeech  rdf:datatype="&xsd;string">
                        Hello World</textToSpeech>
                <rdfs:id  rdf:datatype="&xsd;integer">1</rdfs:id>
        </rdf:Description>
</rdf:RDF>
```

```
<< TTS interface >>
+ id: integer
+ textToSpeech: string
```

Figura 3.7: Interface Text To Speech.

iv) Serviço Web para GPS (*Global Positioning System*)

Responsável pela leitura da posição GPS do robô, diretamente do *hardware* ou através de APIs, e pela disponibilização do resultado para os outros serviços e aplicações. Para o robô considerado, o dispositivo GPS está conectado ao *hardware* Arduino que está conectado ao notebook por um adaptador USB-to-Serial Com Port. Nesse caso, a comunicação entre o *firmware* e serviço web utiliza uma interface serial padrão RS232.

A interface de abstração definida para o serviço web para o GPS é apresentada na Fig.3.8. Nesse caso, os atributos latitude, longitude, altitude, velocidade (*velocity*) e ângulo (*angle*) são retornados pelo *hardware* GPS no momento da leitura, e o '*id*' é um sequencial.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
        'http://www.w3.org/2001/XMLSchema#'>]>
<rdf:RDF
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
        xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
        <rdf:Description rdf:ID='gps'>
                <rdfs:id  rdf:datatype="&xsd;integer">1</rdfs:id>
                <latitude   rdf:datatype="&xsd;float">-3.75</latitude>
                <longitude   rdf:datatype="&xsd;float">-38.53</longitude>
                <altitude    rdf:datatype="&xsd;float">0</altitude>
                <velocity  rdf:datatype="&xsd;decimal">0</velocity>
                <angle   rdf:datatype="&xsd;float">0</angle>
        </rdf:Description>
</rdf:RDF>
```

<< GPS Interface>>
+ id: integer + latitude: float + longitude: float + altitude: float + velocity: decimal + angle: float

Figura 3.8: Interface GPS.

***v)* Serviço Web de Detecção de Faces**

Responsável pela detecção de faces usando a biblioteca de visão computacional OpenCV [21], fornecendo os resultados para outros serviços e aplicações. Esta implementação de *software* foi baseada em exemplos de código do projeto OpenCV e utiliza classificadores obtidos a partir da biblioteca OpenCV (e.g. haarcascade_frontalface_alt.xml).

A interface de abstração definida para o serviço web de detecção de faces é apresentada na Fig.3.9. Nesse caso, os atributos ‘image_full’, ‘*screen_width*’, ‘*screen_height*’ se referem ao nome do arquivo, largura e altura da imagem total da câmera, o ‘*response_time*’ é o tempo decorrido para a detecção da(s) face(s) na imagem, o ‘*num_faces*’ é a quantidade de faces detectadas, e o objeto ‘*faces*’ informa o nome do arquivo, posição relativa, largura e altura de cada imagem de face detectada, entre outros dados.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
        'http://www.w3.org/2001/XMLSchema#'>]>
<rdf:RDF
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
        xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
        <rdf:Description rdf:ID='face_detection'>
                <rdfs:id  rdf:datatype="&xsd;integer">
                        13774656917466</rdfs:id>
                <screen_width  rdf:datatype="&xsd;integer">
                        640</screen_width>
                <screen_height  rdf:datatype="&xsd;integer">480</screen_height>
                <image_full  rdf:datatype="&xsd;string">
                        13774656917466_full.png</image_full>
                <response_time  rdf:datatype="&xsd;integer">937</response_time>
                <num_faces  rdf:datatype="&xsd;integer">1</num_faces>
                <faces>
                        <face0_y  rdf:datatype="&xsd;integer">152</face0_y>
                        <face0_x  rdf:datatype="&xsd;integer">373</face0_x>
                        <face0_sy  rdf:datatype="&xsd;integer">70</face0_sy>
                        <face0_sx  rdf:datatype="&xsd;integer">59</face0_sx>
                        <face0_filename  rdf:datatype="&xsd;string">
                                /home/centauro/robot_/faces/13774656917466_face0.png
                        </face0_filename>
                </faces>
        </rdf:Description>
</rdf:RDF>
```

<< Face Detection interface >>
+ id: integer
+ screen_width: integer
+ screen_height: integer
+ image_full: string
+ response_time: integer
+ num_faces: integer
+ faces: object

Figura 3.9: Interface Face Detection.

***vi)* Serviço Web de Identificação de Faces**

Responsável pela identificação de pessoas pela face. Ele utiliza as bibliotecas do projeto PyFaces [35], um sistema de reconhecimento facial que usa o algoritmo *eigenfaces* [36]. Essa implementação de *software* foi baseada nos exemplos do projeto PyFaces [35].

***vii)* Serviço Web de Detecção de Marcadores**

Responsável pela detecção de marcadores (*landmarks*) e pela disponibilização de informações sobre a localização (x,y,z) do marcador em relação ao robô, utilizando a API ARToolKit [23], uma biblioteca de *software* para a construção de aplicações de realidade aumentada - *Augmented Reality* [50,51]. No SOM4R este serviço pode ser usado para orientar a navegação. Por exemplo, o aplicativo de recarga (vide Capítulo 4, seção 4.2) detecta quando a bateria atinge um nível baixo de carga e então conduz o robô até o marcador da base de carga.

Neste serviço, a trajetória do deslocamento foi calculada utilizando lógica Fuzzy [56].

A interface de abstração definida para o serviço web de detecção de marcadores é apresentada na Fig.3.10. Nesse caso, o atributo '*id_landmark*' informa o código do marcador encontrado, o '*marker_x*' e '*marker_y*' informam a posição relativa do marcador na imagem total da câmera, e também é informada a posição relativa (*x,y,z*) do marcador em relação a câmera, entre outros dados.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
      'http://www.w3.org/2001/XMLSchema#'>]>
<rdf:RDF
      xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
      xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
      <rdf:Description rdf:ID='landmark'>
            <rdfs:id  rdf:datatype="&xsd;integer">1376</rdfs:id>
            <x  rdf:datatype="&xsd;decimal">3.81</x>
            <y  rdf:datatype="&xsd;decimal">-52.78</y>
            <z  rdf:datatype="&xsd;decimal">-10.77</z>
            <size_x  rdf:datatype="&xsd;integer">1280</size_x>
            <size_y  rdf:datatype="&xsd;integer">800</size_y>
            <id_landmark  rdf:datatype="&xsd;integer">
                  10</id_landmark>
            <marker_x  rdf:datatype="&xsd;decimal">675.67</marker_x>
            <marker_y  rdf:datatype="&xsd;decimal">442.07</marker_y>
            <dtime  rdf:datatype="&xsd;datetime">2013-06-22 11:31:18</dtime>
      </rdf:Description>
</rdf:RDF>
```

```
<< Landmark interface >>
+ id: integer
+ x: decimal
+ y: decimal
+ z: decimal
+ size_x: integer
+ size_y: integer
+ id_landmark: integer
+ marker_x: decimal
+ marker_y: decimal
+ dtime: datetime
```

Figura 3.10: Interface Landmark.

***viii)* Serviço Web para o sensor Kinect da Microsoft**

Responsável pela disponibilização dos dados do sensor *Kinect* (câmeras de luz visível e de infravermelho, laser, acelerômetro e microfones), fornecendo imagens e informações de profundidade (matriz de distâncias dos objetos próximos entre 0.5m até 3m em 3D) utilizando a biblioteca OpenKinect [24]. No SOM4R, essas imagens são usadas para detecção de faces, detecção de obstáculos próximos ao robô e para evitar colisão durante a navegação, podendo também pode ser usado para mapear o ambiente do robô.

A interface de abstração definida para o serviço web de detecção de marcadores é apresentada na Fig.3.11. Nesse caso, os atributos '*row_closest*' e '*col_closest*' informam a posição relativa do ponto mais próximo em relação a imagem da câmera (2D), e também o '*depth_closest_cm*' é a distância aproximada desse ponto até a câmera (em cm), entre outros

dados (vide Capítulo 4, secão 4.2).

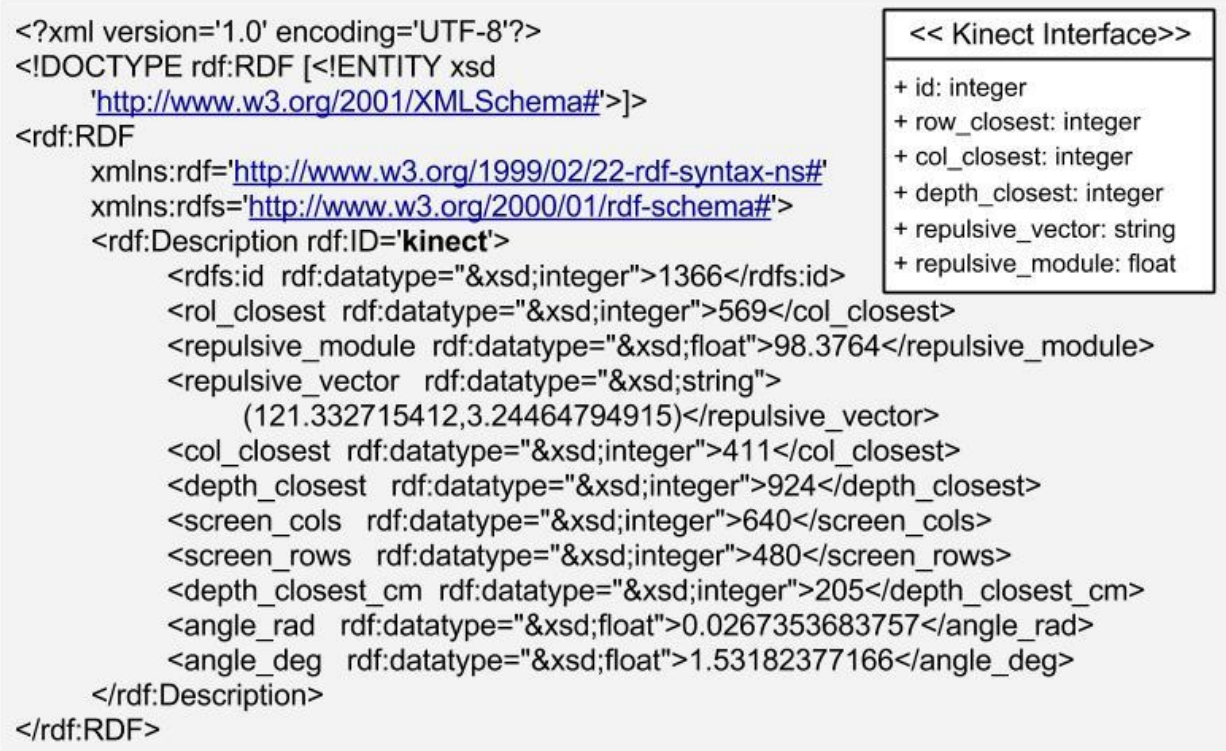

Figura 3.11: Interface Kinect.

***ix)* Serviço Web para a Bateria**

Responsável pela comunicação direta com o *firmware* que faz a leitura da carga da bateria do robô utilizando a interface USB HID, proporcionando a leitura do estado atual da carga para outros serviços e aplicativos.

A interface de abstração definida para o serviço web para a bateria é apresentada na Fig.3.12. Nesse caso, o atributo ‘id’ é um sequencial e o ‘level_dc’ é o valor da voltagem (em volts).

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
        'http://www.w3.org/2001/XMLSchema#'>]>
<rdf:RDF
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
        xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
        <rdf:Description rdf:ID='battery'>
                <rdfs:id  rdf:datatype="&xsd;integer">1</rdfs:id>
                <level_dc   rdf:datatype="&xsd;decimal">12.75</level_dc>
        </rdf:Description>
</rdf:RDF>
```

```
<< Battery Interface>>
+ id: integer
+ level_dc: decimal
```

Figura 3.12: Interface *Battery*.

3.2.4 Estrutura de Controle

A arquitetura de subsunção foi introduzida por Brooks [25,26], visando a implementação de uma arquitetura geral de controle de alto nível para robôs autônomos. Com base nos conceitos da arquitetura de subsunção, os serviços web do *middleware* suportam métodos de supressão e inibição de comportamentos. Isto é, é possível suprimir a entrada de um serviço web durante um curto período de tempo (50 ~ 200 ms), durante o qual este serviço só aceita comandos enviados pelo cliente que o suprimiu, ignorando todos os outros comandos de clientes. Também é possível inibir a saída de um serviço web durante um curto período de tempo ou manter temporariamente inalterados os valores de entrada e/ou de saída destes serviços.

Tal abordagem permite a implementação de uma arquitetura de controle flexível, adaptando o comportamento dos serviços e aplicativos ao contexto dinâmico do ambiente do robô. Por exemplo: quando um robô tem um serviço ou aplicativo (e.g. *Runaway*) em execução para evitar obstáculos, mas tem um outro aplicativo (*App-Push*) local ou remoto cuja missão é empurrar um objeto, o aplicativo *App-Push* pode agir de duas formas para garantir o movimento deste robô na direção do objeto: *i)* inibir a saída do *Runaway* que evita obstáculos; ou *ii)* suprimir a entrada do serviço web do veículo robótico, para então empurrar o objeto até atingir seu objetivo. Contudo, a opção de inibir a saída do *Runaway* não impede que outros aplicativos ou serviços enviem comandos de movimentação. Enquanto que, ao suprimir a entrada do serviço web do veículo, somente os comandos de movimentação do *App-Push* serão considerados.

A interface de abstração definida para a estrutura de controle é apresentada na Fig.3.13. Nesse caso, o atributo '*id*' é um sequencial, o '*supressed_by*' informa o nome do serviço ou aplicativo que está suprimindo a entrada do serviço, e também o '*supressed_timestamp_ms*' informa o momento da supressão, entre outros dados.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
        'http://www.w3.org/2001/XMLSchema#'>]>
<rdf:RDF
        xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
        xmlns:rdfs='http://www.w3.org/2000/01/rdf-schema#'>
        <rdf:Description rdf:ID='subsumption'>
                <rdfs:id  rdf:datatype="&xsd;integer">1234</rdfs:id>
                <supressed_by   rdf:datatype="&xsd;string">
                        runaway</supressed_by>
                <supressed_timestamp_ms   rdf:datatype="&xsd;float">
                        1308753078299</supressed_timestamp_ms>
                <supressed_post_from  rdf:datatype="&xsd;string">
                        joystick_app</supressed_post_from>
                <supressed_post_id   rdf:datatype="&xsd;integer">
                        85267</supressed_post_id>
        </rdf:Description>
</rdf:RDF>
```

<< Subsumption Interface>>

+ id: integer
+ supressed_by: string
+ supressed_timestamp_ms: string
+ supressed_post_from: string
+ supressed_post_id: integer

Figura 3.13: Interface *Subsumption*.

3.2.5 Suporte para Robótica de Enxame (*Swarm Robotics*) e dispositivos com recursos limitados

A arquitetura do SOM4R permite integração entre os processos de *software* que operam embarcados no *hardware* robótico (*firmwares*) e aqueles que operam em computadores através de uma rede (vide seção 4.2). Isto é de particular interesse para os robôs com recursos computacionais limitados (e.g. microcontroladores), pois permite que eles se beneficiem dos recursos e serviços da computação em nuvem [27].

O SOM4R permite que os robôs se comuniquem através da rede visando a formação de enxames (*swarm robotics*). Por exemplo, vamos supor um conjunto de quadrirotores[4] onde cada um é controlado pelo *firmware* embarcado em seu microcontrolador com suporte nativo as tecnologias Ethernet e Wi-Fi (e.g. MicroChip PIC18F97J60 com módulo Wi-Fi MRF24WB). Os serviços web de autenticação e autorização do SOM4R permitem a um determinado dispositivo quadrirotor robótico proporcionar o acesso seguro aos seus recursos através da rede, bem como consumir os recursos compartilhados pelos outros robôs conectados ao *realm* do SOM4R. O *middleware* permite que o enxame de quadrirotores seja monitorado e controlado, ou por um dos dispositivos, ou remotamente por um ou vários processos de *software* distribuídos ao longo da rede.

[4] Aeronaves impulsionadas por quatro motores.

O *middleware* permite que os dispositivos de *hardware* baseados em microcontroladores (e.g. MicroChip PIC18F2550), com sensores e/ou atuadores, sejam conectados diretamente uns aos outros de forma segura em um modelo de serviço baseado em rede ponto-a-ponto[5] através da rede sem fio e/ou internet. Tal modelo permite, por exemplo, a gestão integrada de diversos dispositivos (recursos) de uma habitação com o objetivo de controlar a iluminação, climatização e segurança.

3.2.6 Integração com outro *Middleware*

A arquitetura orientada a serviços do SOM4R permite fácil integração com os módulos já desenvolvidos para outro *middleware*, como ROS, Yarp e Player. Por exemplo, a Fig. 3.14 mostra um cenário de integração entre SOM4R e ROS usando a API Rosbridge. Neste caso, temos um serviço web (Rosbridge-WS) (centro) que vai traduzir o formato de mensagem RDF/XML, enviada pelo SOM4R-WS (esquerda, embaixo), para o protocolo Rosbridge (baseado em JSON - *JavaScript Object Notation*). O Rosbridge-WS usa a implementação da API Rosbridge (rosbridge_library e/ou rosapi e/ou rosbridge_server) para montar a *string* JSON equivalente a mensagem RDF/XML do SOM4R-WS e enviar os comandos para os nós e serviços do ROS (direita) e vice-versa.

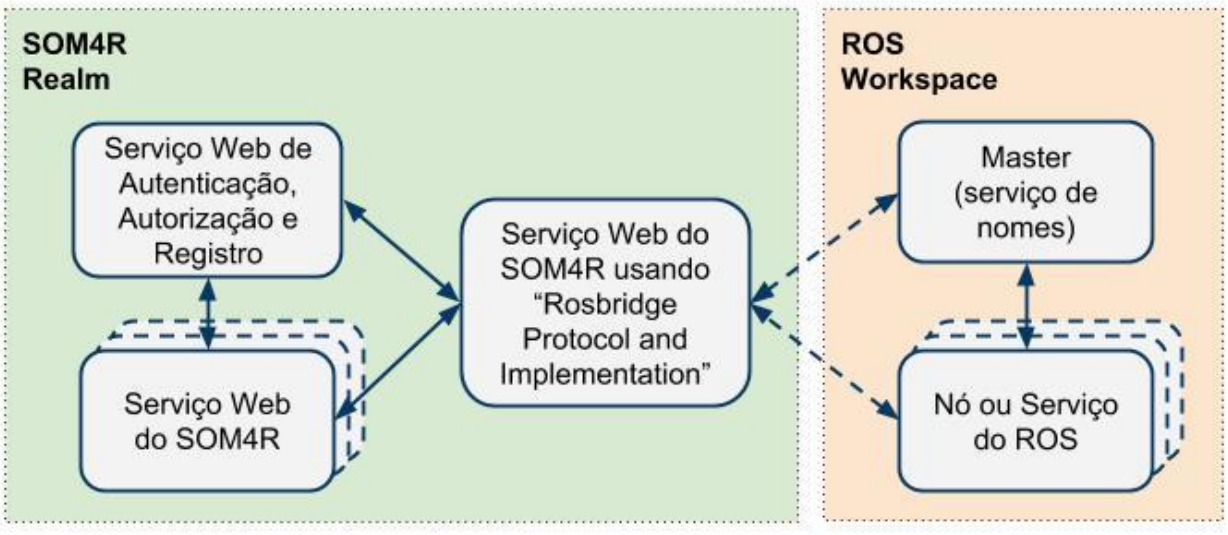

Figura 3.14: Cenário de integração com o ROS. O SOM4R se comunica com o ROS utilizando a API Rosbrigde, permitindo o reuso do *software* desenvolvido usando a API do ROS.

[5] Cada computador da rede pode funcionar como um cliente ou servidor para os outros computadores da rede, permitindo o acesso a diferentes recursos compartilhados sem a necessidade de um servidor central.

3.3 Discussão

A necessidade de estabelecer um conjunto adequado de critérios que servem de base para a comparação entre os projetos de robótica [2,3,4,7,8,10,11,32,33,34] é um ponto crítico comum a todas as pesquisas anteriores sobre esta questão [9,13,14].

Os estudos citados no capítulo 1 [9,13,14], também fornecem informações relevantes para a seleção dos critérios de avaliação e comparação entre os projetos. Por exemplo, Kramer e Scheutz [9] propõem quatro critérios para comparação de RDEs: especificação, plataforma de apoio, infraestrutura e implementação. Mohamed et al. [13] propõem seis critérios para avaliar as características de *middleware* para robôs em rede: modelo de comunicação (standard/nonstandard), interoperabilidade, suporte a descoberta, configuração e integração automáticas, tipos de serviços do *middleware* (específicos / expansíveis), serviços de comunicação, componentes embarcados e suporte a dispositivos com recursos limitados.

Elkady e Sobh [14] propõem nove critérios de avaliação: arquitetura, ambiente de simulação, padrões e tecnologias, suporte para ambiente distribuído, segurança de acesso para os módulos, detecção de falhas e recuperação, capacidades de coordenação e comportamento em tempo real, código aberto e conexão dinâmica.

Em [34], Elkady et al. propõem uma abordagem estruturada para projetos modulares nos ambientes de robótica e de automação, chamado RISCWare. Eles implementam um aplicativo para robôs ("cumprimentando uma pessoa") para avaliar o *framework* RISCWare com relação a aplicabilidade. A fim de avaliar o desempenho do *framework*, uma série de testes de *stress* foram realizados usando diferentes tamanhos de mensagem para medir a latência de pacotes de dados de ponta-a-ponta.

Contudo, vale ressaltar que cada conjunto de critérios de avaliação utilizados para comparar prós e contras de projetos de robótica está diretamente correlacionado com o cenário, pois um determinado critério pode ser relevante para uma necessidade ou contexto específico [9]. Entre a miríade de atributos disponíveis, escolhemos os seguintes: as tecnologias, arquiteturas, protocolos e camadas de comunicação, as interações cliente-servidor, o formato utilizado na descrição da mensagem, os sistemas operacionais e linguagens de programação suportadas, e a data da última atualização.

Na Tabela 3.1, listamos uma série de atributos importantes de cinco *middleware* existentes para a área de robótica, a fim de comparar as suas funcionalidades com as fornecidas pelo SOM4R. A seguir, destacamos as principais diferenças entre as abordagens do SOM4R e dos projetos citados.

Características / Projetos	Player	CLARAty	CARMEN	YARP	ROS	SOM4R
Arquitetura / Design de Software	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor	SOA	ROA (REST)
Protocolo(s) de Comunicação	TCP (Socket)	TCP and UDP (Sockets)	IPC (TCP/IP sockets)	TCP,UDP, Multicast and QNet	XML-RPC	HTTP
Camada da Pilha TCP/IP	Transporte	Transporte	Transporte	Transporte	Aplicação	Aplicação
Formato usado para descrição da mensagem	Protocolo Próprio	Protocolo Próprio	Protocolo Próprio	Binário	IDL	RDF/XML
Sistema Operacional	Linux, Solaris and Windows	Linux, Solaris, Mac OSX and Cygwin	Linux	Windows, Linux, QNX 6 and Mac OSX	Linux and Windows (partial functions)	Linux and Android (partial functions)
Linguagens de Programação Suportadas	C/C++, Java, Matlab, Python, Perl, Tcl/Tk	C++	C++, Java	C++, Python, Matlab, Java, Tcl, Lisp, Ruby	C++, Python, Octave, Lisp	C/C++, C#, Python, PHP, Java, Ruby, Flash/Flex
Segurança do Middleware	Nenhuma	Nenhuma	Nenhuma	Nenhuma	Possível	Inerente
Última Atualização	2010	2007	2008	2013	2013	2013

Tabela 3.1: Características comuns dos projetos de integração de recursos robóticos em rede para fins de comparação com o projeto SOM4R.

Com relação a comunicação, o projeto SOM4R utiliza o modelo cliente-servidor, baseando-se no paradigma de requisição e resposta (*request-response*) [16], onde o estado de um recurso é transferido apenas sob demanda do cliente através da implantação de um *web service* servidor. O processo cliente tem sempre a iniciativa de estabelecer comunicação com o processo servidor e enviar ou solicitar uma transferência de estado representativo do recurso (REST). Este paradigma apresenta a vantagem de reduzir o tráfego de dados ao explicitamente requisitado aos servidores, de acordo com a necessidade e capacidade de processamento de cada cliente em tempo de execução, além de contribuir para simplificar o desenvolvimento do *software* servidor. Além disso, em [12], é demonstrado que o desempenho dos serviços REST é mais adequado para realizar a integração de dados distribuídos em escala web. Portanto, não há necessidade do cliente especificar a frequência com que os dados devem ser enviados pelo servidor como é o caso do Player (10Hz *default*), nem de configurar mecanismos de sincronização e *buffer* como pode ser o caso do YARP, e nem de subscrever para tópicos publicados no servidor (*topic-based publish-subscribe model*) como é o caso do ROS, pois tais mecanismos de envio de dados entre o cliente e o servidor podem gerar um tráfego de dados desnecessário na rede (vide Capítulo 1, seção 1.2.2).

Quanto ao protocolo de comunicação, o projeto SOM4R emprega, em relação aos projetos Player, CLARAty, CARMEN, YARP e ROS, o protocolo HTTP para sistemas distribuídos e colaborativos. A tecnologia *socket* (vide Capítulo 2, seção 2.2.3) é utilizada direta ou indiretamente por todos os projetos citados, uma vez que os mecanismos de comunicação IPC e os protocolos XML-RPC e HTTP são todos baseados em *sockets*. O uso de protocolos da camada de aplicações da pilha TCP/IP, XML-RPC ou HTTP, abstrai a complexidade inerente a comunicação via *socket*. O XML-RPC é um protocolo de chamada de procedimento remoto (RPC) que utiliza XML para codificar suas chamadas e HTTP como um mecanismo de transporte. Entretanto, no SOM4R, optou-se pelo protocolo HTTP pelas seguintes razões: a) apresentar menos restrições em *firewall* o que facilita muito a interoperabilidade entre diferentes domínios na rede, b) possuir uma elevada compatibilidade entre plataformas diferentes através de bibliotecas nativas disponíveis em diversas linguagens de programação e sistemas operacionais, o que facilita a integração, reduzindo significativamente a complexidade da comunicação entre processos, e c) estar presente em muitos dispositivos embarcados (e.g. microcontroladores PIC com módulo *Ethernet*), seguindo a tendência atual do emprego da rede *Ethernet* em inúmeros equipamentos disponíveis de diversos fabricantes.

No SOM4R, as mensagens trocadas entre os processos de *software* são formatadas no padrão RDF/XML e descrevem o estado representativo dos recursos (REST) de forma simples conforme ilustrado no Capítulo 2, seção 2.2. Já os projetos Player, CLARAty e CARMEN usam formatos próprios de mensagens, enquanto o YARP emprega um formato binário e o ROS usa *Interface Description Language* (IDL) para descrever o conteúdo das mensagens trocadas entre os módulos de *software*. IDL descreve as interfaces de um modo independente da linguagem de programação e do sistema operacional. Contudo, no SOM4R, optou-se pela descrição dos

recursos usando RDF por tornar os dados mais portáveis e interoperáveis entre diferentes tipos de computadores e diferentes sistemas operacionais e linguagens de programação. Além disso, RDF proporciona um mecanismo de tipagem básica dos atributos desses recursos. Cabe salientar, que no estágio evolutivo de contextos web, descrever recursos e representar informação empregando a linguagem RDF é bastante atraente e promissor, permitindo que esses recursos sejam ligados entre si, integrados e reutilizados [17].

A escolha da rede TCP/IP é um consenso geral entre todos os projetos citados. No SOM4R, optou-se por TCP/IP pelas seguintes razões: a) a pilha TCP/IP é uma coleção de diversos tipos de protocolos de comunicação que trabalham em conjunto para realizar a comunicação em rede, recomendável tanto para redes pequenas quanto para redes gigantescas como a Internet, b) é compatível com uma grande variedade de *hardware*, e c) é incluído nas versões dos principais sistemas operacionais, tais como Unix, Linux, Windows (Microsoft), Android (Google), MacOS e iOS (Apple).

Comentário Final

Nesse capítulo, estão discutidas a especificação e a proposta do *middleware* SOM4R, descrevendo seus componentes e recursos de forma integrada e uniforme. Adicionalmente está apresentada uma discussão comparativa com os trabalhos relacionados, citados no capítulo anterior. No próximo capítulo, apresentamos resultados sobre a avaliação de desempenho do SOM4R num contexto específico de vigilância, monitoramento e controle remoto, navegação autônoma e semi-autônoma. Adicionalmente, são apresentadas a interface HMI e a descrição do robô móvel desenvolvido.

CAPÍTULO 4

4 - RESULTADOS EXPERIMENTAIS

Para avaliar o desempenho do *middleware* proposto, SOM4R, realizamos experimentos com as aplicações desenvolvidas, construídas especificamente para um robô móvel com rodas (RMR) autônomo, integrando funcionalidades de comando de voz, desvio de obstáculos, “saudando uma pessoa”, “olhe para mim”, recarga automática da bateria, e controle remoto da movimentação do veículo. O robô com rodas apresentado na Fig. 4.1 foi projetado e construído ao longo do desenvolvimento da pesquisa em prol da validação experimental do SOM4R. A justificativa para a construção de um robô móvel com rodas específico, em vez de usar um robô disponível no mercado, surgiu a partir da necessidade de testar as hipóteses por trás da proposta do SOM4R, especialmente a abstração de *hardware* e *software*, para um projeto completamente novo de *hardware* robótico. Como prova de conceito, alguns experimentos foram repetidos em um robô Scitos-G3 (vide seção 4.2), disponível no nosso laboratório, para testar a hipótese da abstração de *hardware* e avaliar a aplicabilidade do SOM4R em diferentes plataformas robóticas.

4.1 Plataforma de *Hardware* Desenvolvida

Nessa seção, descrevemos sucintamente o desenvolvimento do *hardware* de um robô móvel com rodas (RMR) utilizado como plataforma para pesquisa e desenvolvimento do *middleware* SOM4R. A plataforma robótica (RMR) considerada, foi projetada e construída dentro das atividades da tese para a validação experimental desta pesquisa.

A patente deste robô foi depositada no Instituto Nacional da Propriedade Industrial (INPI) sob o número PI 1106417-0 (publicada inicialmente na RPI 2171 de 14/08/2012). Devido a este fato, neste trabalho não é permitida a revelação de detalhes mais internos do robô.

Aspectos da operação desta plataforma robótica estão disponíveis no site **http://www.som4r.net**, servindo como complemento às informações adicionais a este texto.

***i*) Características do Projeto**

Foi realizado estudo sobre os diversos tipos de robôs analisando as diversas configurações para robôs móveis com rodas (vide apêndice A). Desta análise, resultou a escolha de um sistema de tração diferencial para compor o sistema de locomoção do robô (vide Fig. 4.2) em desenvolvimento. Também foram definidas algumas características iniciais do projeto do robô a ser construído, como por exemplo, que o robô seria utilizado para aplicações *indoor* do tipo: vigilância, monitoramento e controle remoto. Estas características serviram de restrições para guiar o planejamento das atividades de desenvolvimento da plataforma de *hardware*. Estas restrições são:

- Operação em ambientes semi-estruturados;
- Peso máximo do robô de 20 kg, incluindo circuitos eletrônicos, baterias, estrutura mecânica, *notebook* de controle embarcado, motores, rodas e revestimento exterior de proteção;
- As dimensões devem permitir ao robô passar por portas. Considerando uma porta de largura de 70 cm, o robô não deve possuir mais de 50 cm de largura, pois se faz necessário ter uma margem de segurança para evitar colisões;
- Autonomia das baterias para tempo de operação mínima de 6 (seis) horas;
- Velocidade máxima de 1 m/s com aceleração suave, de no máximo 0,5 m/s^2;
- Altura do chassi em relação ao solo de 3 cm com diâmetro das rodas mínimo de 12,7 cm (5 in), para possibilitar ao robô subir pequenos batentes e transpor pequenas irregularidades no solo (valas);
- A comunicação entre o *notebook* de controle embarcado e o *hardware* robótico desenvolvido utiliza o protocolo USB 2.0 (*Universal Serial Bus*), segundo os conceitos de *Plug-and-Play* (PnP) e *Hot Swapping.* Tais dispositivos utilizam o tipo de transferência de dados de interrupção (*Interrupt transfer*) e são da classe HID (*Human Interface Device*). Diversas plataformas de sistema operacional incluem suporte a classe HID em nível de *kernel*, dispensando a necessidade de um *driver* específico (vide Apêndice A). A porta USB também fornece energia para os dispositivos robóticos de baixo consumo, evitando a necessidade de fontes externas.

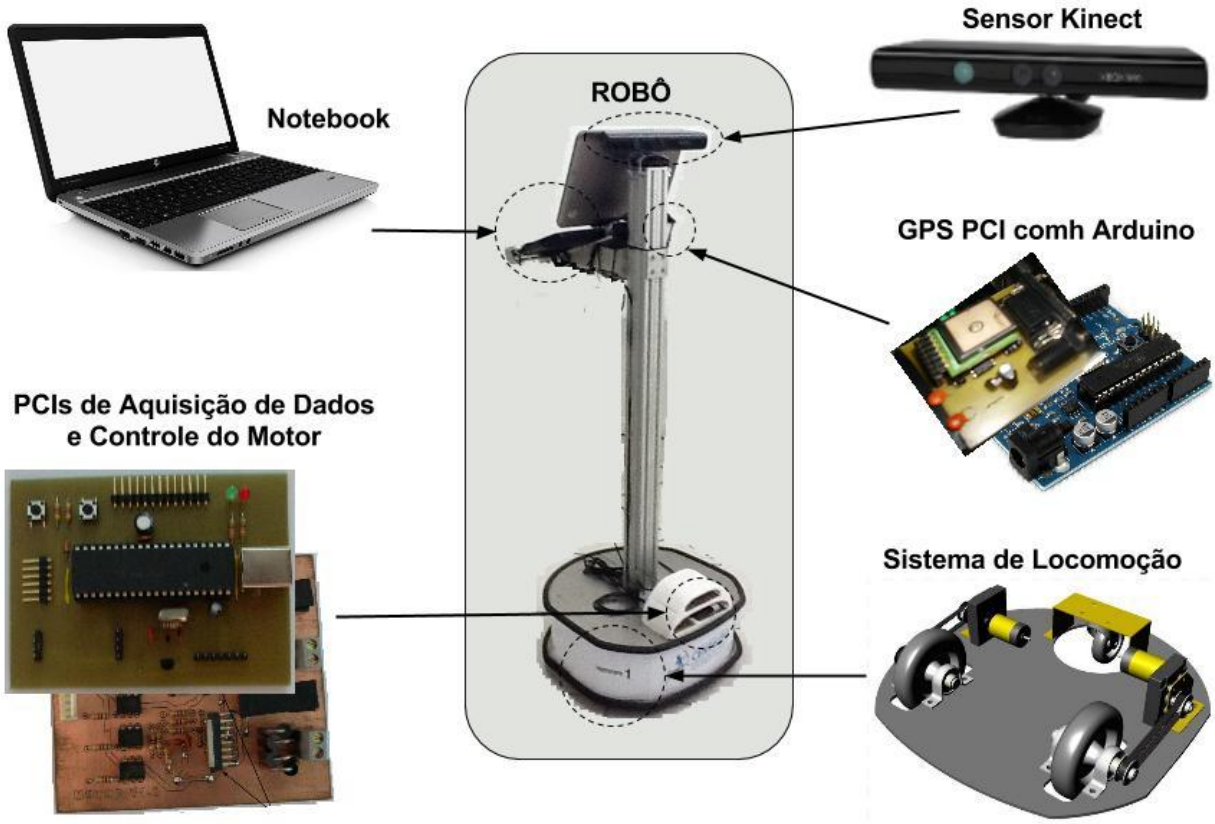

Figura 4.1 - Robô móvel com rodas construído como plataforma para a investigação, desenvolvimento e validação experimental do *middleware* proposto.

ii) Design da Estrutura Mecânica e Sistema de Locomoção

Observando outros robôs desenvolvidos anteriormente no nosso laboratório, percebeu-se que a ausência de rigidez e durabilidade do conjunto mecânico fazia com que as leituras dos sensores fossem alteradas pela vibração excessiva, resultando num comportamento diferente daquele desejado pelo seu sistema de controle. Assim, a preocupação inicial foi a de desenvolver uma estrutura mecânica que fosse rígida e confiável do ponto de vista estrutural.

Escolheu-se o sistema de tração diferencial com duas rodas fixas independentes e uma roda de apoio para compor o sistema de locomoção para o robô (Fig. 4.2), composto basicamente dos seguintes elementos: mancais de apoio, correias e polias sincronizadoras, motores elétricos e circuito eletrônico de acionamento e controle.

O sistema de locomoção consiste numa base móvel com três rodas, duas fixas e uma roda de apoio. As rodas fixas são acionadas independentemente através de dois motores de corrente contínua, acoplados no eixo de cada roda. O acoplamento entre os eixos dos motores e os eixos das rodas é realizado mediante o uso de correias sincronizadoras. A Fig. 4.2 mostra o arranjo

espacial utilizado para todos os elementos do sistema de locomoção.

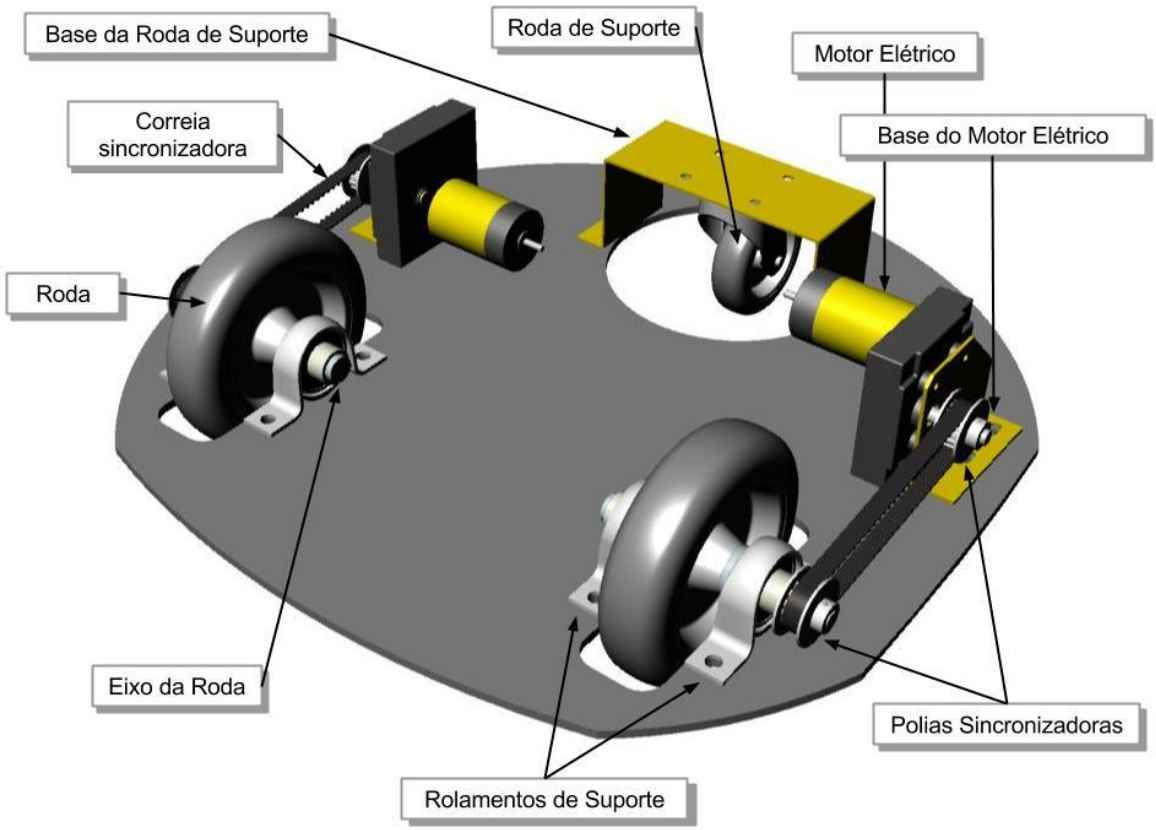

Figura 4.2: Base do robô mostrando todos os elementos utilizados no sistema de locomoção.

Todas as peças foram fabricadas em oficinas de usinagem tradicional e algumas através de Comando Numérico Computadorizado. A fabricação mecânica do chassi foi realizada através de corte a jato de água por Comando Numérico Computadorizado (CNC), em empresa especializada, MAEMFE. Este procedimento se deveu ao fato de que o formato escolhido para o chassi possuía muitas curvas e sua fabricação em oficinas mecânicas tradicionais seria dificultada. Todas as outras peças foram fabricadas e usinadas em duas oficinas mecânicas dentro da Universidade Federal do Ceará – a oficina do Departamento de Engenharia Mecânica e Produção e a oficina do Departamento de Física – pois acreditou-se que sua geometria não requeria fabricação computadorizada, que resultaria em aumento de custos. O robô foi montado, resultando num protótipo rígido e robusto.

O sistema de propulsão foi realizado com base nos motores de corrente contínua (CC) com caixa de redução acoplada, com tensão nominal de alimentação de 12V. Correias sincronizadoras são geralmente utilizadas quando se deseja transmitir potência entre eixos e para garantir que a polia acionada esteja em movimento com razão de velocidade constante em

relação à polia motora. A versatilidade destas correias vem substituindo a transmissão por correntes, pois são silenciosas, não necessitam de lubrificação e podem atingir velocidades maiores do que as correntes. No nosso caso, os esforços mecânicos são pequenos e não foram considerados relevantes para a escolha dos elementos do sistema de transmissão. Os aspectos restritivos estavam concentrados nas configurações geométricas e espaciais, pois o espaço é limitado, com distância entre eixos de 15 cm, e diâmetros dos eixos movido e motor de 8 mm. A correia escolhida foi o modelo 140XL037, da empresa Dina.

iii) Sistema de Aquisição de Dados e Controle

Foram fabricadas duas Placas de Circuito Impresso (PCI), de acionamento dos motores (vide Fig. 4.3) e do Sistema de Aquisição de Dados (vide Fig. 4.4), no Instituto de Telemática - ITTI do Instituto Federal do Ceará - IFCE, através de máquina operatriz via CNC (Controle Numérico Computadorizado). A alta qualidade da fabricação refletiu-se na confiabilidade e facilidade na montagem dos circuitos, permitindo comunicação rápida e direta entre o *notebook* de controle embarcado e os motores elétricos através da porta USB 2.0.

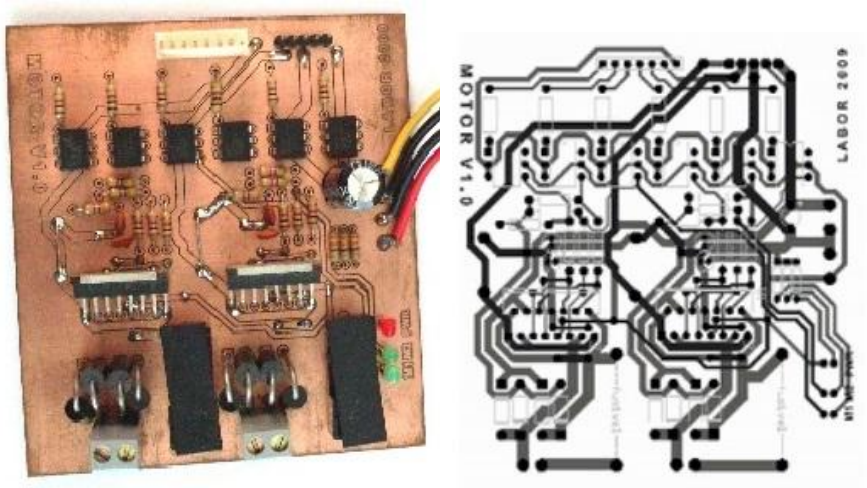

Figura 4.3: PCI do circuito eletrônico de acionamento dos motores.

A placa do sistema de aquisição de dados (vide Fig. 4.4) foi desenvolvida utilizando o microcontrolador Microchip PIC18F4550, de 16 bits, interface USB 2.0 (12Mbit/s) que pode trabalhar em até 12 MIPS (milhões de instruções por segundo). O circuito demonstrou funcionamento satisfatório, sendo testado com sistemas operacionais Microsoft Windows e com distribuições de linux, tais como Ubuntu e Fedora, realizando comunicação confiável e segura via uma interface HID com pacotes de dados de 8 (oito) bytes (vide Apêndice A). Foram realizados testes onde o *software* embarcado no microcontrolador (*firmware*) contava a quantidade de ciclos de processamento executados e enviava esses dados quando solicitado pelo *software* no *notebook host*. Verificamos que não houve perda de dados nem erro nos pacotes de transmissão. O resultado também mostrou que a velocidade do ciclo de processamento de dados

do PIC18F4550 foi em média 20 vezes superior à capacidade de leitura de dados pelo *software* executando no *notebook* usando a porta USB 2.0.

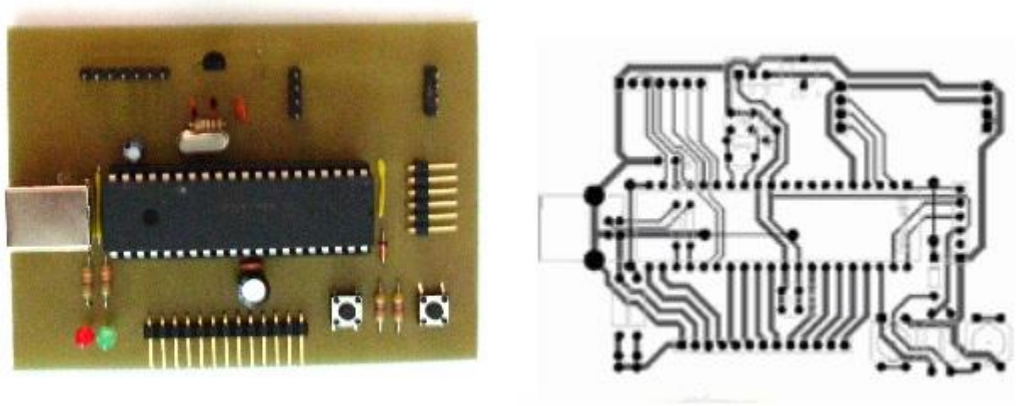

Figura 4.4: PCI do Sistema de Aquisição de Dados.

***iv*) Computador de Bordo**

O *middleware* proposto foi instalado no computador pessoal do robô acima mencionado, ou seja, um *notebook* com processador Intel (R) Core 2 Duo (TM) SU7300 de 1,3 GHz, placa de vídeo GMA 4500MHD, memória RAM de 4GB DDR3, disco rígido 320GB 7200rpm SATA, tela LED 12,1", USB 2.0, *touchscreen* e conectado através da rede Wi-Fi (802.11g 54Mbps). O sensor MS Kinect, a Placa de Circuito Impresso (PCI) de controle do veículo robótico, e a placa Arduino com o sensor GPS integrado (via adaptador USB-Serial db9) foram conectadas nas portas USB do *notebook*. Todos os testes experimentais foram realizados com esse *notebook* rodando o sistema operacional Linux (Ubuntu 11.04 32 bits).

***v*) Avaliação**

Observamos que o robô tem apresentado funcionamento satisfatório, podendo desde então servir como plataforma para pesquisa e desenvolvimento do *middleware* proposto. A rigidez das partes fixas e móveis do RMR foi verificada, com boas características de durabilidade e confiabilidade. Seus elementos utilizados, com exceção dos motores elétricos, foram todos superdimensionados, e devem apresentar longa vida útil.

Muito trabalho ainda deve ser feito, no intuito de trazer melhorias ao projeto de *hardware* do robô. Por exemplo, na placa controladora dos motores (*driver*), o Circuito Integrado (CI) L298 (vide Fig. 4.3) não se apresentou adequado para o projeto, pois apresentou queda de tensão elevada, em torno de 15%, interferindo diretamente no desempenho do motor e em sua potência mecânica disponível, resultando em uma velocidade máxima de aproximadamente 0,2 m/s.

4.2 Operação e Performance dos Aplicativos Desenvolvidos

No SOM4R, um único aplicativo pode monitorar e/ou controlar o comportamento de um conjunto de robôs. Já os diferentes aspectos (por exemplo, postura, percepção e atitudes) do comportamento de um robô podem ser controlados por vários aplicativos, executando diretamente no computador deste robô ou em outros computadores e dispositivos móveis distribuídos através da rede.

Os aplicativos e serviços web, interligados através do "território de autenticação" do SOM4R, são desenvolvidos utilizando a biblioteca HTTP nativa, disponível em diversas linguagens de programação e em diferentes plataformas de sistemas operacionais (e.g. Linux, Android, MS Windows). Isso permite maior liberdade na escolha do ambiente (*software* e *hardware*) mais adequado para o desenvolvimento e implementação de cada tarefa robótica.

No método de autenticação *HTTP Digest*, *Realm* é uma diretiva que define um "território de autenticação", tipicamente uma descrição do sistema ou do computador que está sendo acessado. A utilização dos *realms* permite que as aplicações e os serviços web, tanto locais como remotos, possam ser particionados em um conjunto de espaços de proteção (e.g. quadrotors@lab1.som4r.net, mobile_robots@lab2.som4r.net).

O SOM4R permite a integração entre aplicações e serviços web executando em diversos robôs e autenticados em diferentes *realms* através da rede de computadores. Na Fig.4.5, os cinco robôs (na cor branca) estão distribuídos em dois *realms* (cinza escuro) interligados através da internet (cinza claro). Não há restrição para a localização física dos robôs. Os aplicativos e serviços (na cor amarela) que estão executando nestes robôs podem participar de um ou mais *realms*, interagindo simultaneamente com diversos robôs.

Nós descrevemos nos parágrafos seguintes seis aplicativos que são úteis para a implementação de tarefas robóticas.

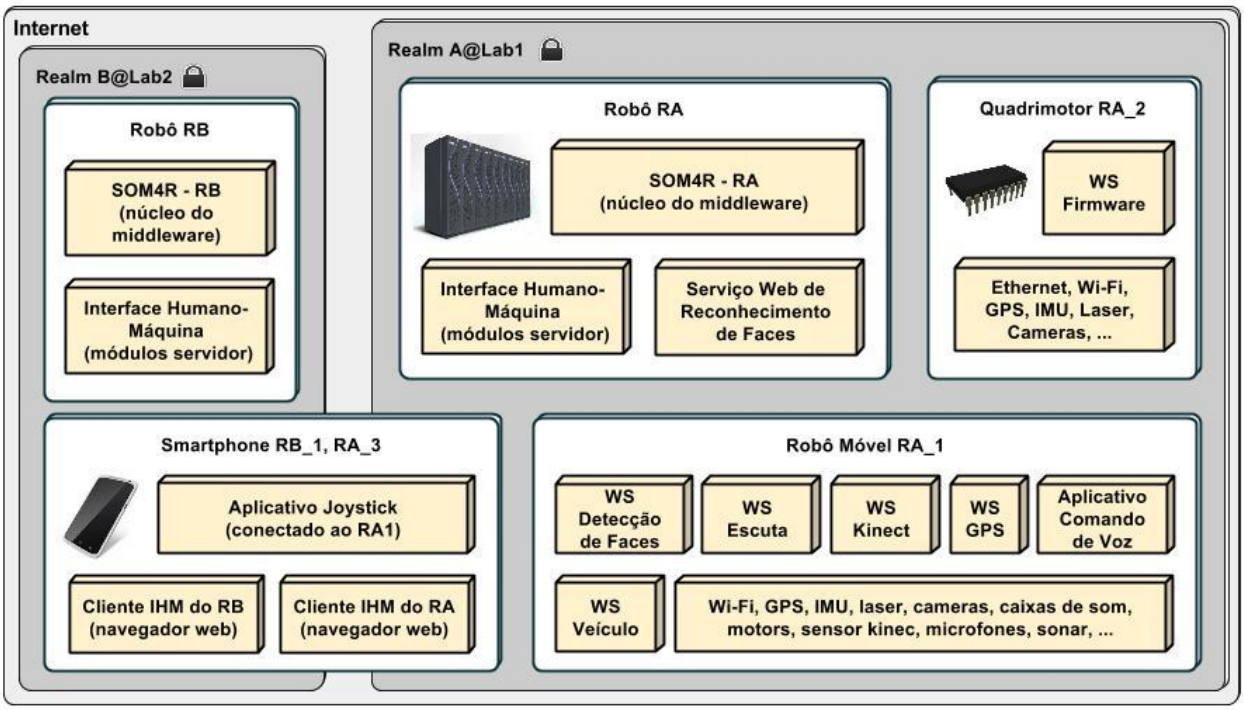

Figura 4.5 - Cenário de integração entre aplicações e serviços web executando em diversos robôs, integrando microcontroladores (RA_2), dispositivos móveis (RB_1) e supercomputadores (RA).

i) Aplicativo *Joystick*

Responsável pelo controle da movimentação do robô utilizando os sensores de movimento (e.g. acelerômetro triaxial) de um dispositivo móvel, tipo *smartphone* ou *tablet*, equipado com o sistema operacional (SO) Android a partir da versão 2.2. A modelagem BPMN do processo executado pela aplicação e pelos serviços envolvidos é apresentada na Fig. 4.6. A leitura do acelerômetro no passo 1 é transformada em comando no passo 2 e enviado pela rede sem fio (Wi-Fi ou 3G) para o serviço de movimentação do robô (serviço web do Veículo) no passo 3. O serviço web do Veículo executa o comando como (passo 4), envia um registro do comando executado para o serviço web STM (passo 5) e retorna o *status* do veículo para o aplicativo. Depois, o aplicativo *Joystick* envia um registro da ação tomada para o serviço web STM (passo 6) e espera alguns milissegundos (passo 7) antes de reiniciar o ciclo. Esta implementação foi desenvolvida utilizando a linguagem Java para o sistema operacional Android.

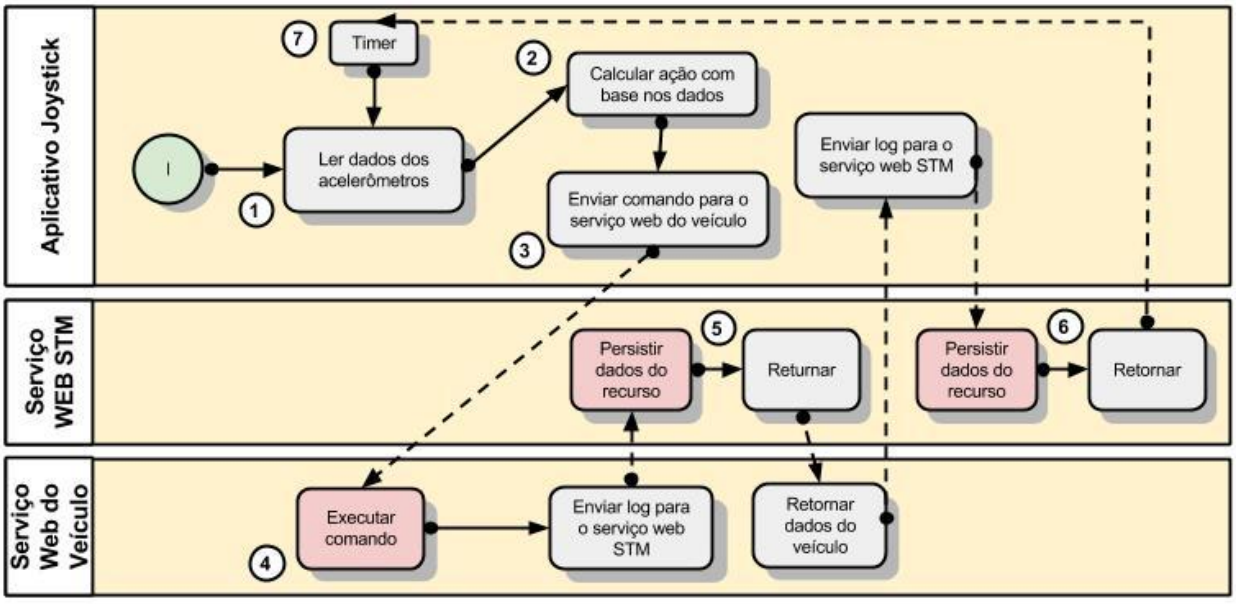

Figura 4.6: Modelagem BPMN do aplicativo *Joystick.*

Os testes de desempenho do aplicativo, ilustrados na Fig.4.7, foram realizados usando um *smartphone*, com processador *single-core* de 600 MHz e 170 MB de memória interna. O tempo médio de resposta do aplicativo conectado ao *middleware* através da rede Wi-Fi protegida (WPA2 com TKIP + AES, até 54Mbps) foi de cerca de 300ms, o que pode ser considerado bastante aceitável para aplicações de controle remoto de veículos robóticos (http://www.youtube.com/watch?v= G2iMuNAkWkE).

Testes adicionais de acesso ao robô foram realizados com este *smartphone* utilizando a rede de telefonia celular 3G. Neste caso, devido à baixa velocidade destas redes (medida em cerca de 512 Kbps) no Brasil, o tempo de resposta teve um atraso de até sete segundos, comprometendo o controle remoto do movimento do veículo. Isto é, uma redução da velocidade da rede sem fios da ordem de 108:1 resultou no aumento do tempo de resposta da ordem de 1:23. No entanto, o cenário atual de evolução no poder de processamento paralelo dos dispositivos móveis, e na velocidade de transmissão de redes de banda larga (e.g. WiMAX, 4G) e tecnologias de Wi-Fi mais rápidas [22] (e.g. IEEE802.11ac, IEEE802.11ad), certamente levará a resultados de desempenho melhores.

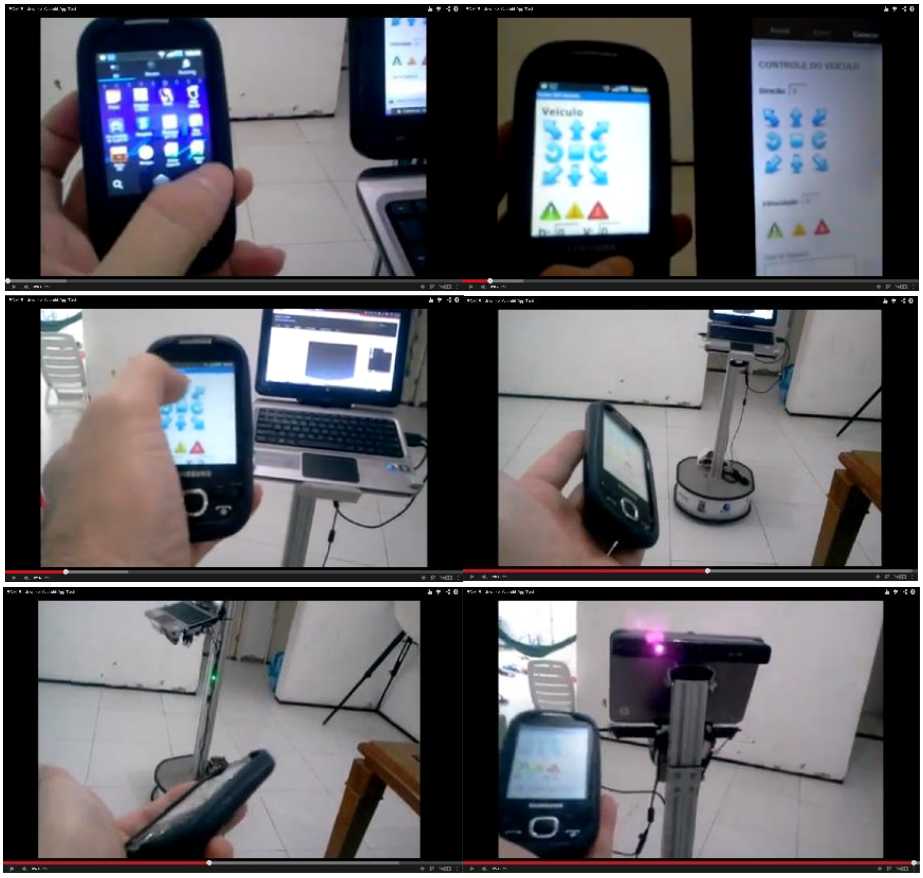

Figura 4.7: Testes do aplicativo *Joystick*.

ii) Aplicativo *Runaway*

Responsável pela função de desvio de obstáculos. Esta aplicação utiliza o serviço web para o Kinect e o serviço web para o Veículo. Na Fig. 4.8, no passo 1 ela monitora constantemente os obstáculos detectados pelo serviço web para o sensor Kinect e considera que cada ponto vizinho tem uma "força repulsiva" proporcional ao inverso do quadrado da distância, conforme o passo 2 (vide Apêndice A-3.9). Neste trabalho, seguimos a implementação de Brooks [25] para o desenvolvimento da nossa aplicação *Runaway*. Quando o módulo dessa "força resultante" excede um determinado limiar definido empiricamente (passo 3), a aplicação assume o comando do robô utilizando a supressão da entrada do serviço web do Veículo. Se o robô está em movimento, o comando STOP é enviado para o robô conforme o passo 4. Depois disso, os comandos girar e mover são executados de forma a evitar o obstáculo, com base na direção e sentido do vetor "força resultante" (passo 5). Em seguida ela registra as ações realizadas e reinicia o ciclo de monitoramento de obstáculos.

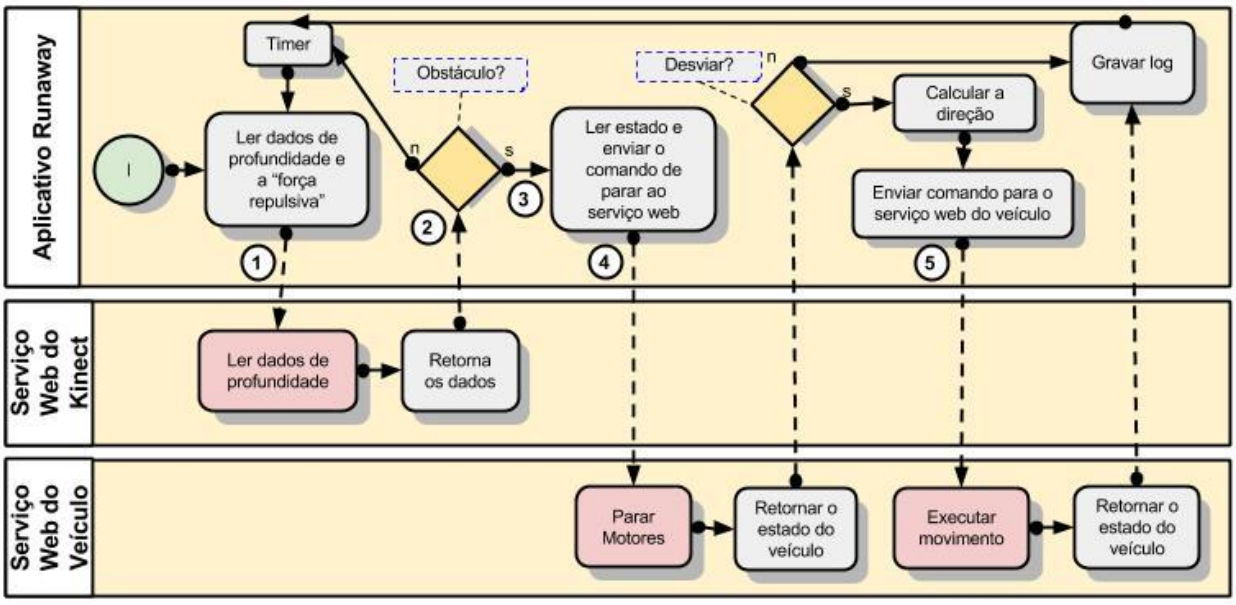

Figura 4.8: Modelagem BPMN do aplicativo *Runaway*.

Os testes do aplicativo *Runaway* apresentaram resultados compatíveis com o objetivo proposto de evitar obstáculos. Por exemplo, o tempo médio para a leitura dos dados de profundidade do sensor Kinect e para o cálculo da "força repulsiva" ficou em torno de 100ms. Foram realizados os dois tipos de experimentos com o aplicativo. No primeiro, o movimento do robô estava sendo controlado pelo aplicativo *Joystick*. Durante a movimentação, quando a "força repulsiva" indicava a presença de um objeto muito próximo, o *Runaway* tomava o controle suprimindo a entrada do serviço web do veículo, parando e posicionando o robô na direção para desviar do obstáculo, depois o serviço web do veículo voltava a aceitar os comandos do aplicativo *Joystick*. No segundo experimento, o *Runaway* foi configurado com um comportamento diferente. O movimento do robô estava sendo controlado pelo módulo web através da IHM e, durante a movimentação, quando detectava a presença de um objeto muito próximo, o *Runaway* fazia a leitura da situação atual do veículo robótico (velocidade e direção) e suprimia a entrada do serviço web do veículo, parando e posicionando o robô na direção para desviar do obstáculo, para depois enviar ao serviço web do veículo o comando de movimentação lido anteriormente à subsunção. Como resultado, ao ser enviado na direção de uma parede, o robô assumia o comportamento de seguir paredes até que fosse enviado o comando para parar, permitindo realizar a ronda automática em um ambiente fechado com um único comando de movimentação.

A Fig.4.9 mostra o tempo necessário para a leitura dos dados de profundidade do sensor Kinect e para o cálculo da "força repulsiva" sob a forma de *boxplot*, que é um diagrama que fornece uma representação visual para a distribuição dos dados, destacando onde a maioria dos valores se encontram e aqueles valores que diferem da norma, chamados de *outliers*. Foram utilizadas 7000 medições coletadas num intervalo de 16 minutos durante a realização dos testes

descritos anteriormente. O tempo mínimo foi de 27ms e o máximo de 509ms, o primeiro quartil ficou em 61ms, a mediana ficou em 83ms, o terceiro quartil foi de 134ms e a amplitude inter-quartil foi de 73ms. O tempo máximo foi registrado no momento em que o uso dos dois núcleos (CPUs) do processador estava praticamente em 100%, sendo considerado como ponto fora da curva (*outliers*).

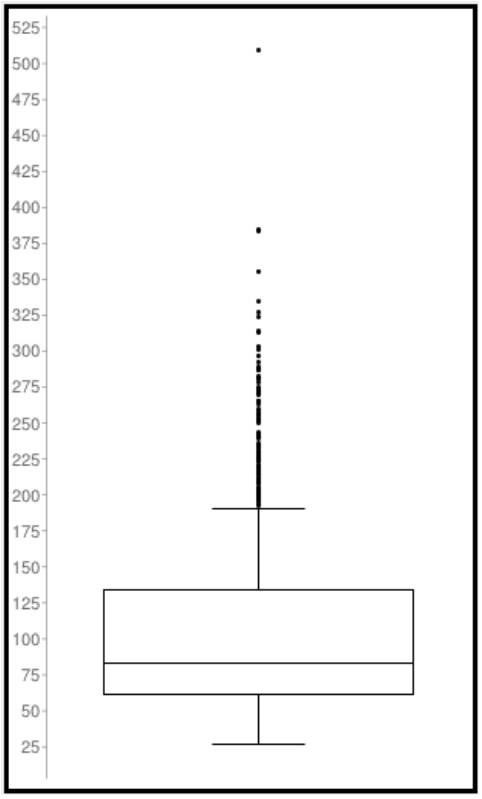

Figura 4.9: *Boxplot* - tempo de detecção de obstáculos e cálculo da “força repulsiva” (ms).

iii) **Aplicativo “Comando de Voz”**

Responsável pela função de ativar, executar e desativar o comando de voz. A modelagem BPMN do processo executado pela aplicação é apresentada na Fig. 4.10. Ela monitora o serviço web de reconhecimento de voz humana indicado pelo passo 1. Quando a resposta do serviço web é uma palavra ou sentença previamente determinada como um “comando”, conforme o passo 2, a aplicação seleciona a ação a ser executada. Se o comando for para ativar ou desativar o comando de voz, ele envia uma frase sobre o estado atual da funcionalidade de comando de voz (ativada ou desativada) ao serviço web para TTS, que emite o som da(s) palavra(s) através do sintetizador de som do computador, conforme o passo 3. Se o comando for para movimentação do veículo robótico, definidos como *right*, *left*, *ahead*, *back*, *faster*, *slower* e *stop*, a ação equivalente a esse comando é realizada, conforme o passo 4. Depois de tudo, os comandos recebidos e os eventos que eles desencadearam são registrados pela aplicação utilizando o serviço web de STM (*Short*

Term Memory), conforme o passo 5, e depois retorna ao início do ciclo de monitoramento.

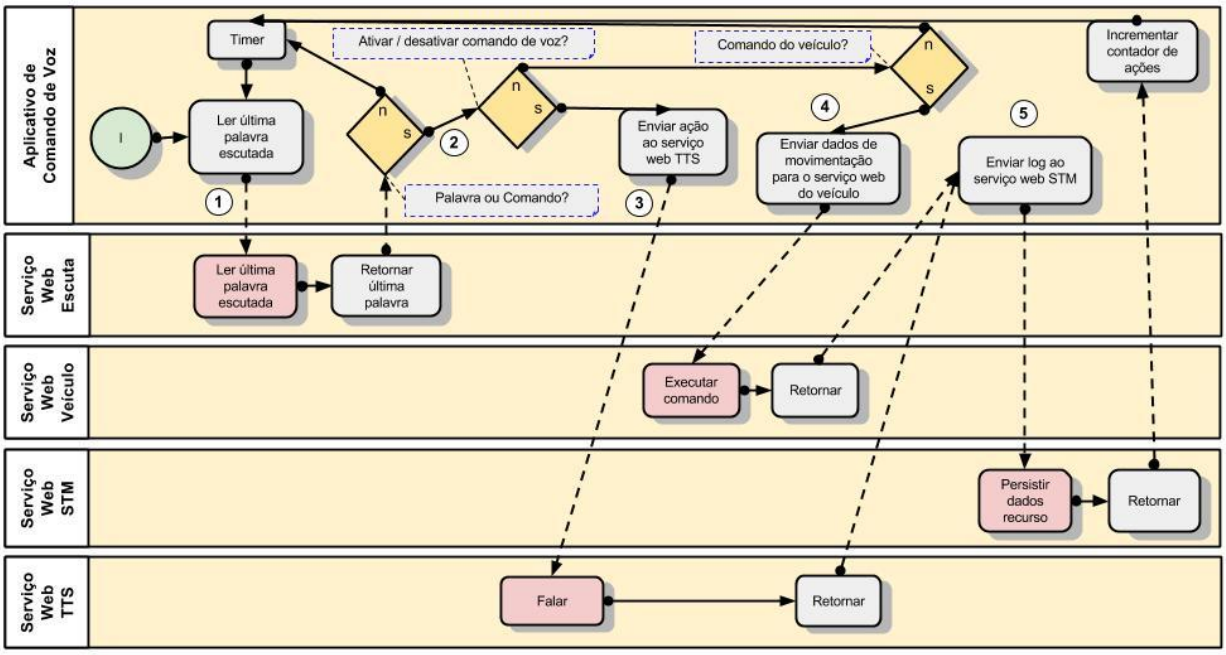

Figura 4.10: Modelagem BPMN do aplicativo para comando de voz.

Os testes de desempenho, realizados no robô considerado e também no robô Scitos-G5, mostraram um tempo de resposta abaixo de 500 ms, considerado satisfatório dentro do contexto definido para esse projeto (funções de vigilância, monitoramento e controle remoto, navegação autônoma e semi-autônoma). Contudo, a biblioteca CMU Sphinx não tem suporte para o Português Brasileiro, por isso os “comandos” utilizados estão em Inglês. Os testes mostraram consideráveis taxas de falso negativo, com o não reconhecimento de algumas palavras, levando a necessidade de repetição do comando verbal, bem como pequenas taxas de falso positivo, com o reconhecimento de outra palavra foneticamente parecida. Tudo isso leva à necessidade de uma escolha empírica do conjunto de palavras que vão representar os comandos (e.g. *forward* ou *ahead*?, *stop* ou *brake*?) no sentido de melhorar as taxas de acerto e de evitar semelhanças fonéticas.

iv) Aplicativo “Saudando uma Pessoa”

Responsável pela função de cumprimentar as pessoas que se aproximam do robô, esta aplicação usa os seguintes serviços web: Detecção de Faces, Kinect e TTS. Ele também utiliza as bibliotecas do projeto PyFaces [35], um sistema de reconhecimento facial que usa o algoritmo *eigenfaces* [36]. Na Fig. 4.11, esta aplicação está continuamente à procura de rostos em um raio de poucos metros (1-3m), conforme mostrado no passo 1, usando o serviço web de detecção de faces. No passo 2, se há pessoas por perto, ele verifica se eles são "conhecidos", usando o serviço

web de reconhecimento facial, conforme o passo 3. Se o reconhecimento for bem sucedido, ele procura no serviço de memória de curto prazo se essa pessoa já foi "vista" e se o robô já falou com essa pessoa na última hora, conforme o passo 4. Se assim for, ele só registra o evento de identificação da pessoa na memória de curto prazo, caso contrário, ele interage com a pessoa apenas falando um "Oi" seguido do nome da pessoa usando o serviço web TTS, conforme o passo 5, e depois registra o evento de cumprimentar uma pessoa na memória de curto prazo. O aplicativo foi desenvolvido para detectar rostos humanos presentes na imagem da câmera (cena), selecionar uma face mais próxima do robô e tentar reconhecer e falar com essa pessoa.

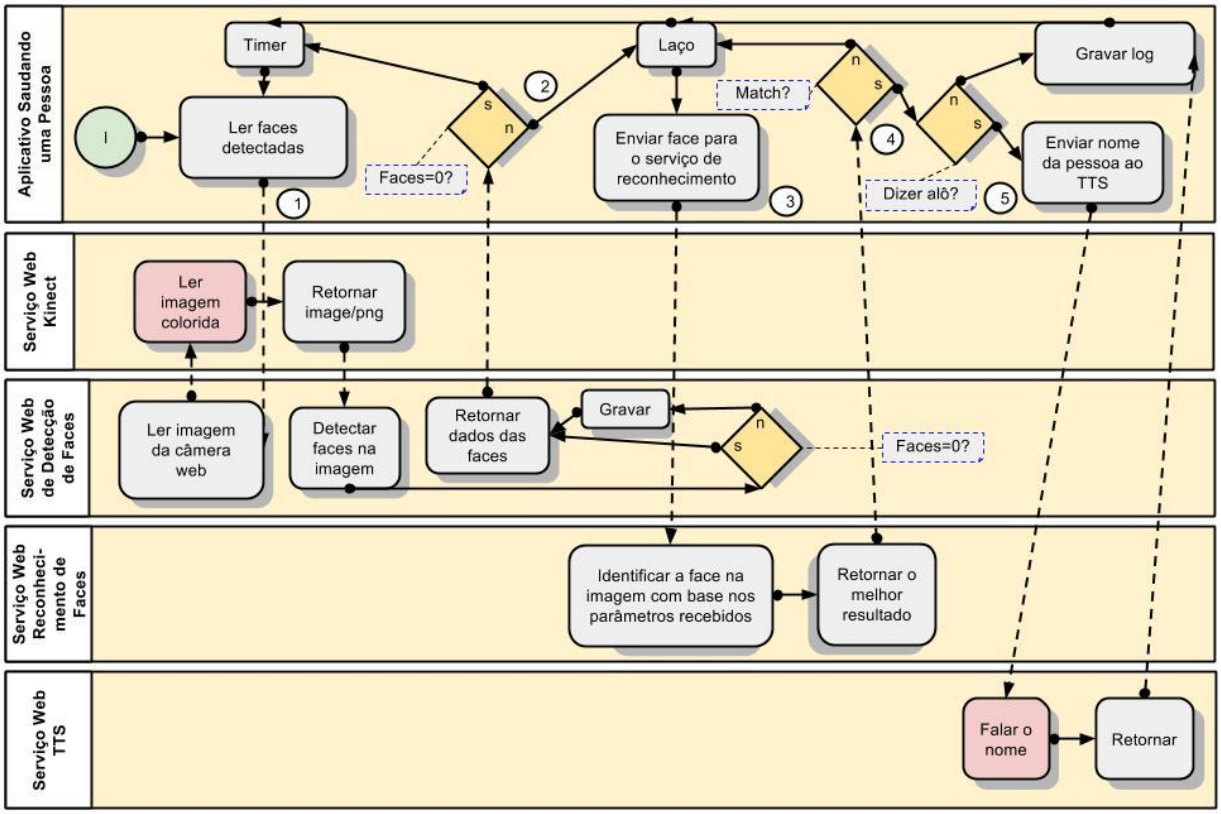

Figura 4.11: Modelagem BPMN do aplicativo “Saudando uma Pessoa”.

v) Aplicativo “Olhe para Mim”

Responsável pela função de “olhar de frente” para uma pessoa que está num certo raio de alcance quando o robô estiver parado por um tempo. Esta aplicação utiliza o serviço web para detecção de faces, o serviço web do veículo, o serviço web STM e o serviço web para o sensor Kinect. No passo 1 da Fig. 4.12, esta aplicação está continuamente à procura de rostos em um raio de poucos metros (1-3m) usando o serviço web de detecção de faces. Quando há pessoas por perto (faces são detectadas), conforme o passo 2, ele verifica se o robô está parado e procura no serviço de memória de curto prazo (STM) se houve comandos de movimentação do veículo

robótico no último minuto (passo 3). No passo 4, se o robô estiver parado por algum tempo mínimo (configurável), o aplicativo gira o veículo na direção do rosto da pessoa mais próxima.

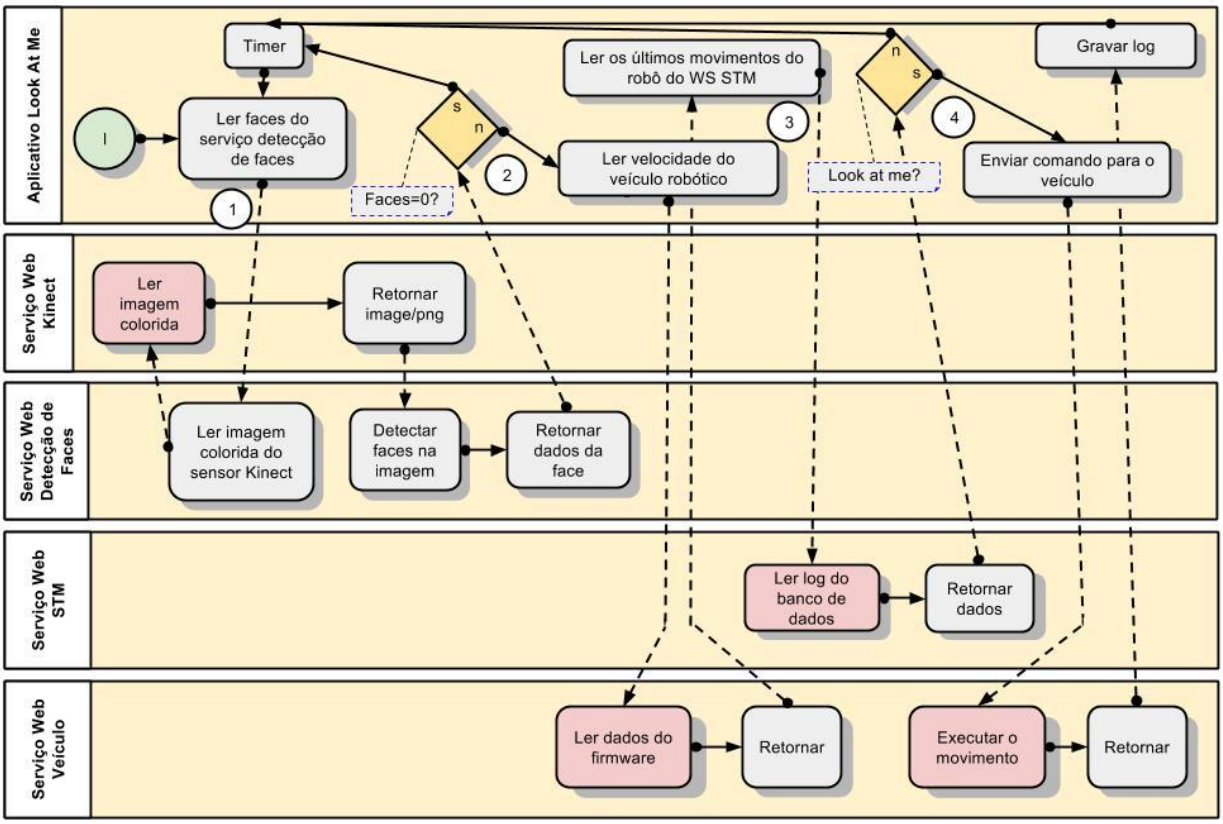

Figura 4.12: Modelagem BPMN do aplicativo “Olhe para Mim”.

Os testes do aplicativo *Look at Me* apresentaram resultados compatíveis com o objetivo proposto de detectar faces e direcionar o robô para a pessoa mais próxima. A Fig. 4.13 apresenta os resultados experimentais com relação as medidas dos tempos necessários para detecção de faces em uma cena (imagem capturada da câmera) sob a forma de *boxplot*. Foram utilizadas 500 medições coletadas durante os testes do aplicativo em ambientes externos (*shopping center*). O tempo mínimo foi de 560ms e o máximo de 1096ms, o primeiro quartil ficou em 673ms, a mediana ficou em 724ms, o terceiro quartil foi de 766ms e a amplitude inter-quartil foi de 93ms.

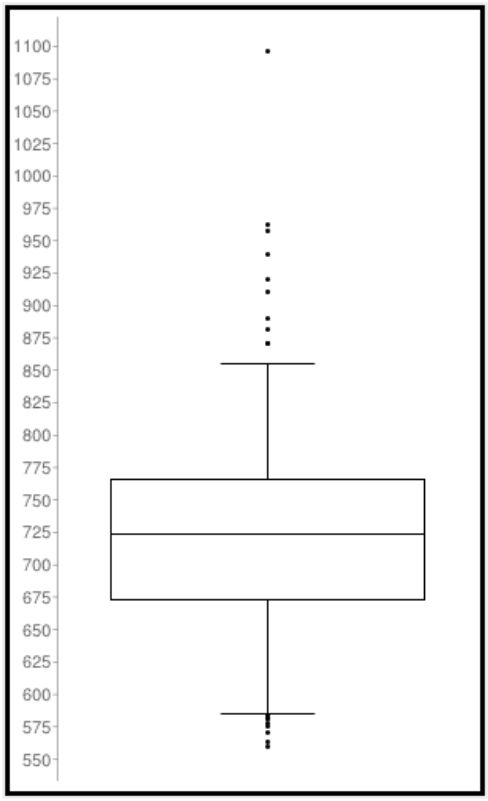

Figura 4.13: Boxplot - tempo de detecção de faces (ms).

vi) Aplicativo de Recarga

Responsável pela função de monitorar a carga da bateria antes dela atingir um nível baixo crítico. Uma vez nesse nível crítico, o aplicativo assume imediatamente o comando do serviço web do veículo, suprimindo qualquer outra requisição de comando enviada dos aplicativos para o veículo robótico. Depois o aplicativo realiza uma busca no ambiente ao redor procurando localizar um marcador específico, predefinido como indicador do local da base de recarga e, finalmente, ele navega em direção à base disponível mais próxima. Depois de recarregar a bateria, a aplicação retorna ao estado de monitoramento, de modo que o robô pode voltar para a execução de sua rotina normal. Esta aplicação utiliza o serviço web monitor de carga da bateria, o serviço web do veículo, o serviço web de detecção de marcadores e o serviço web para o *kinect*.

Apesar da base de recarga não ter sido construída, os testes com a biblioteca ARToolkit demonstraram a viabilidade da solução proposta. Contudo, o ARToolkit apresentou consideráveis taxas de falso positivo (43%), com o reconhecimento de marcadores (*patterns*) onde não havia nada. Isso gera uma grande “indecisão” no momento de definir a direção na qual o robô deve se deslocar, pois além do movimento do próprio robô, existe uma incerteza considerável na posição tridimensional do *marcador* informada pelo ARToolkit, o que leva ao

aumento do percurso e por conseguinte do tempo de deslocamento até o local de interesse. Tudo isso leva à necessidade de uma escolha experimental do conjunto de marcadores que vão representar os locais de interesse (e.g. base de recarga), pois alguns marcadores têm taxas melhores do que outros em função do ambiente físico onde estão inseridos. Nos testes, o objetivo do aplicativo era localizar e aproximar o robô do alvo, sendo utilizado o marcador "Hiro" para simular o local da base de recarga (alvo). Em outro teste, o alvo foi colocado em movimento e o robô assumiu um comportamento de perseguição ao marcador. A cada leitura da posição do marcador, o robô calculava (via lógica Fuzzy [56]) duas variáveis (ângulo e distância) e realizava dois movimentos distintos, primeiro girando na direção desejada e depois avançando nessa direção uma certa distância, que variava em função da distância ao alvo, com velocidade constante.

4.3 Interface Humano-Máquina - IHM

No SOM4R, a interface de interação humano-máquina do robô é um portal web, servindo como painel de controle que permite o monitoramento e a configuração dos diversos recursos do robô (vide Figuras 4.14 e 4.15). Vale ressaltar que a interface humano-máquina implementada através de um portal web é uma inovação importante, uma vez que este tipo de abordagem não está presente nos projetos de *middleware* mencionados anteriormente neste documento. A IHM baseada na Web permite que o portal web do robô seja acessado via *browser* por vários clientes simultaneamente, através de computadores *desktop, ultrabook, smartphone* e *tablet*, utilizando as redes Wi-Fi, 3G ou 4G. É necessário apenas ter um navegador web instalado e saber o endereço IP ou domínio, *login* e senha do portal do robô.

Na implementação da infraestrutura de *software* do portal web, utilizou-se o servidor web de código aberto Apache HTTP Server e o sistema de gestão de conteúdo (CMS - *Content Management System*) Joomla. Um CMS tem a vantagem de estruturar e facilitar a criação, administração, distribuição e publicação da informação de forma dinâmica, em tempo real, através de uma interface de acesso via Internet. Optou-se pelo *software* de código aberto Joomla por ele oferecer uma grande quantidade de funções presentes através de complementos (componentes, módulos e/ou *plugins*) que podem ser agregados ao CMS.

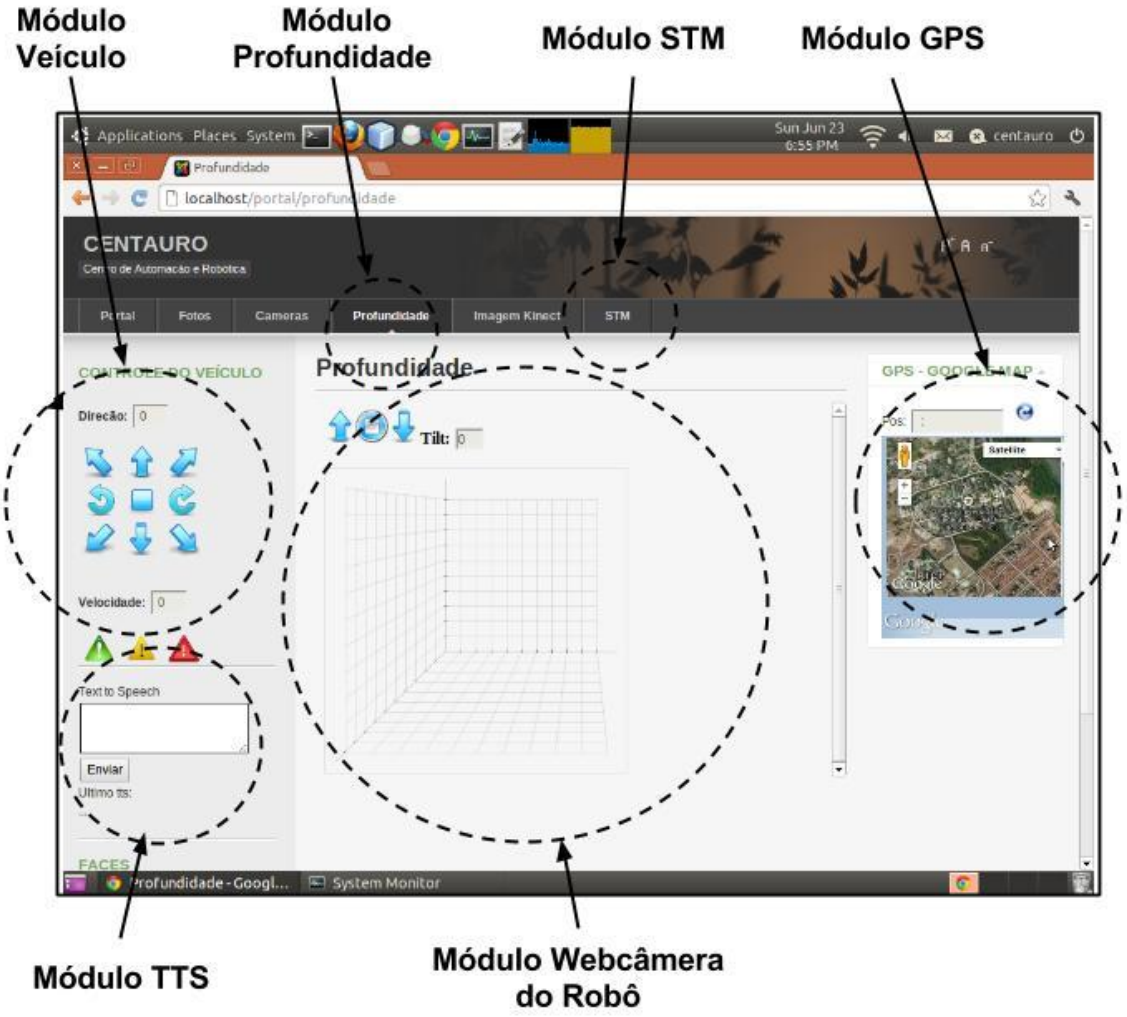

Figura 4.14: Interface Humano-Máquina baseada na Web.

Figura 4.15: Acesso a IHM utilizando um *smartphone* com Android (acima) e iOS (abaixo).

As seguintes extensões para o CMS Joomla foram desenvolvidas neste trabalho:

i) Controle do Veículo

Módulo responsável pelo controle da movimentação do veículo através de setas na IHM, como ilustrado a Fig. 4.16. Permite que o usuário tenha o controle manual do robô que está remoto. O módulo usa o serviço web do veículo para ler o status (velocidade e direção) do dispositivo robótico e para enviar os comandos para o dispositivo.

Figura 4.16: Módulo de Controle Remoto Manual do veículo robótico.

ii) GPS

Módulo responsável por mostrar a posição do robô na IHM usando o Google Maps API [31], conforme ilustrado na Fig. 4.17. Ele usa o serviço web para o GPS. Este serviço depende de acesso a internet para funcionar.

Figura 4.17: Módulo de GPS da IHM do robô.

iii) TTS

Módulo responsável por receber uma palavra ou frase digitada e enviá-la para o serviço web do TTS do robô, conforme ilustrado na Fig. 4.18, permitindo-lhe uma interface de voz do robô com o usuário.

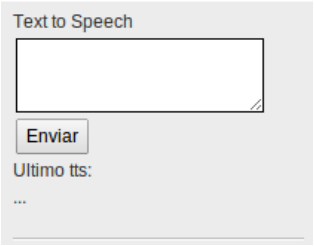

Figura 4.18: Módulo de TTS da IHM do robô.

iv) Imagem de Profundidade

Módulo responsável por mostrar na IHM a imagem de profundidade do sensor kinect, conforme ilustrado na Fig. 4.19. As imagens são apresentadas a uma taxa de 5 fps, permitindo apenas um acompanhamento parcial. Taxas maiores podem ser conseguidas com um notebook mais moderno (e.g. i7). Este módulo usa o serviço web para o sensor kinect. O vídeo do teste pode ser encontrado em http://www.youtube.com/watch?v=CSv3zVnXd0k.

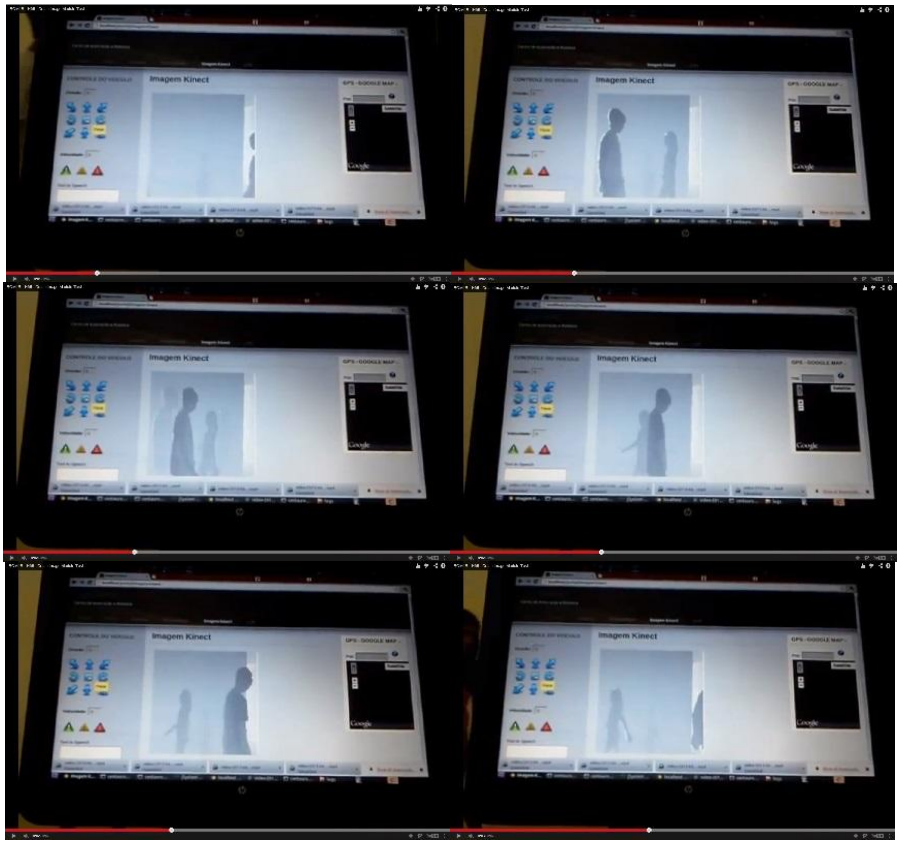

Figura 4.19: Testes do módulo de imagem de profundidade da IHM do robô.

v) Câmera do Robô

Módulo responsável para mostrar imagens em "tempo real" da câmera web interligada ao *notebook* instalado no robô. Esse módulo utiliza o fluxo de vídeo fornecido pelos *softwares* ffmpeg [40] e *Red5 Media Server* [52].

vi) Visualizador do *STM*

Módulo responsável por exibir o log do STM em “tempo real”, permitindo aos usuários monitorar e verificar as atividades dos serviços web e aplicativos do *middleware*, conforme ilustrado na Fig. 4.20. Nesta figura, são ilustradas as duas últimas atividades dos *softwares* em ordem cronológica (e.g. gps 448 dias e 43 dias). O módulo STM facilita a implantação (*deploy*) de novos aplicativos no ambiente distribuído do SOM4R, fornecendo informações sobre os eventos ocorridos e em andamento. Por exemplo, durante os testes do aplicativo Runaway, foi possível monitorar a informação sobre a movimentação e a “força repulsiva” calculada, identificando quando o Runaway suprimia o serviço web do veículo, parando e reposicionando o robô. Em seguida o serviço web do veículo voltava a aceitar os comandos dos outros aplicativos.

Software	.	..
gps	448d : gps velocidade=0 altitude=0 longitude=-38.53 angulo=0 latitude=-3.75	43d : gps velocidade=0 altitude=0 longitude=-38.53 angulo=0 latitude=-3.75
kinect	155d : kinect screen_cols=640 col_closest=482 repulsive_module=121.242601183 angle_deg=1.544567194 depth_closest=923 repulsive_vector= (121.198549014,3.26803588033) screen_rows=480 angle_rad=0.0269577830535	86ms : kinect screen_cols=640 col_closest=386 repulsive_module= angle_deg=1.54627 depth_closest=924 repulsive_vector= (121.157761643,3. screen_rows=480 angle_rad=0.02698

Figura 4.20: Tela do módulo visualizador do STM da IHM do robô.

Comentário Final

Durante os testes experimentais utilizando o *notebook* considerado, o consumo de recursos das CPUs foi medido em torno de 35% quando todos os serviços web estavam em execução. Sabemos que o modelo requisição-resposta da arquitetura SOA tende a minimizar o consumo de recursos das CPUs, pois os serviços web são executados somente sob demanda. Quando os aplicativos de usuário (e.g. *runaway*, comando de voz) também estavam em execução, o nível médio de consumo das CPUs ficou em torno de 55%, pois eles estão executando processos com um determinada frequência. Tais níveis de consumo das CPUs justificam plenamente as inovações propostas pelo SOM4R.

A interface humano-máquina apresentou excelente desempenho, tendo sido acessada simultaneamente por vários computadores e *tablets*, incluindo alguns *smartphones* rodando os sistemas operacionais Android (Google) e iOS (Apple), e mostrou ser uma solução que atende as necessidades de uma interface com o usuário, completamente independente da plataforma de sistema operacional do cliente.

Alguns vídeos dos testes e das apresentações realizadas com o robô considerado estão disponíveis nos endereços abaixo.

- http://www.youtube.com/watch?v=vqF8QrWX6LU
- http://www.youtube.com/watch?v=InUTArXFyMc
- http://www.youtube.com/watch?v=G2iMuNAkWkE
- http://www.youtube.com/watch?v=CSv3zVnXd0k

Todos os códigos e interfaces dos serviços estão devidamente documentados e implementados na forma de projeto *open source*, disponíveis no site **http://som4r.net**. Devido a extensão desses documentos, optamos por não ilustrá-los nos capítulos ou no apêndice.

CAPÍTULO 5

5 - CONCLUSÕES

Propomos um *middleware* concebido através de uma metodologia de projeto para aplicações em robótica baseada na integração de diversas tecnologias emergentes, a saber: protocolos de comunicação, metodologias de segurança, arquiteturas de *softwares* para sistemas distribuídos, estruturas de controle, linguagens de programação, sistemas operacionais e interface humano-máquina. Esta integração buscou combinar as melhores contribuições das tecnologias de suporte aos principais *middlewares* em uso, e concomitantemente desenvolver novas funcionalidades ao projeto de um *middleware* para aplicações em robótica.

O *middleware* desenvolvido neste trabalho permite o acesso aos recursos do robô com aplicações escritas em diferentes linguagens de programação a partir de diferentes computadores através da rede local ou Internet. Os *softwares* (serviços e aplicativos) implementados neste trabalho foram desenvolvidos utilizando as linguagens de programação Python, Php, Java, Javascript, Flash/Flex, e C/C++, tornando flexível e portável a implementação modular de novos aplicativos e serviços web, reduzindo significativamente as restrições para a evolução do *middleware* de forma colaborativa.

A adoção do protocolo HTTP e da linguagem XML, que possuem menores restrições em *firewall*, facilitou a interoperabilidade entre diferentes domínios na rede. O *middleware* foi desenvolvido empregando o sistema operacional Linux, com alguns aplicativos para o sistema operacional Android.

A segurança incorporada ao *middleware* abriu a possibilidade de integrar sistemas robóticos através da internet, pois o uso do método HTTP *Digest Authentication* combinado com o protocolo seguro HTTPS aumentou a segurança de acesso ao *middleware* e reduziu a fraqueza do HTTP Digest quanto a incapacidade do cliente em confirmar a identidade do servidor. Os outros projetos de *middleware* citados não empregam métodos padronizados de autenticação, como por exemplo o HTTP Digest. Esta melhoria na segurança também foi resultante do uso de *tokens* para o controle, praticamente em tempo real, das permissões de acesso entre os serviços web e aplicativos autenticados pelo *middleware* (fluxo de dados).

A estrutura de controle baseada em subsunção é uma inovação em relação ao outros projetos de *middleware* considerados. Tal estrutura possibilitou a implementação de uma arquitetura de controle flexível, adaptando o comportamento dos serviços e aplicativos ao

contexto dinâmico do ambiente do robô. Como método de arbitração, a subsunção, dada a simplicidade da sua formulação, resultou bastante eficiente para a resolução de comandos conflitantes, conforme ilustrado no capítulo 3.

Outra característica importante neste projeto foi a opção pelas arquiteturas ROA e SOA, que possibilitaram, dada a complexidade das aplicações de robótica, a flexibilidade para especificação do grau de granularidade de recursos, ou seja, a geração de um número variável de recursos e cada recurso, por sua vez, podendo oferecer um número variável de atributos. Além disso, práticas de computação distribuída podem ser aplicadas para processamento colaborativo de tarefas computacionalmente pesadas.

O uso de navegadores web (*browsers*) como interface de interação humana (IHM) com o robô foi uma inovação, devido ao fato de que permite o acesso local e remoto (fixo e móvel) de forma concorrente e segura, reduzindo a necessidade de instalação de *software* no cliente. Esta opção aumentou a portabilidade da IHM entre diferentes sistemas operacionais, computadores e dispositivos móveis dos clientes.

A arquitetura modular do *middleware* proposto facilita a adição de outros recursos robóticos simples, e tem-se revelado bastante útil em outras aplicações de robótica em nossos laboratórios.

5.1 Trabalhos Futuros

Nesta tese, a partir do *middleware* proposto e do protótipo implementado, propomos como perspectivas futuras os seguintes trabalhos:

- **Implementar núcleo do SOM4R para o sistema operacional Android.**
  O Android é considerado como o sistema operacional mais popular do mundo inteiro, com uma grande comunidade de desenvolvedores de uma série de aplicativos, possibilitando a evolução do SOM4R para ampliar sua abrangência em diversas aplicações robóticas. A vantagem deste *middleware* executando em Android, além da sua portabilidade e facilidade de comunicação, é a capacidade de interação com os dispositivos sensores (e.g. acelerômetros, giroscópio, gps, bússola, entre outros) presentes nos *smartphones* e *tablets*, permitindo a integração desses sensores ao robô e reduzindo significativamente o desenvolvimento de placas de circuito integrado (PCI) adicionais para incorporar ao robô tais sensores.

- **Desenvolvimento de serviços web para integração com outros *hardwares* robóticos.**
  Desenvolver a integração de outros *hardwares* robóticos, através da especificação e implementação de novos serviços web para encapsular a API de controle de outros robôs disponíveis no mercado (e.g. NAO, Pioneer, entre outros), facilitando a reutilização do

software desenvolvido para tais robôs.

- **Integrar com os Simuladores de Ambiente Stage (2D) e Gazebo (3D).**
  Especificar e implementar novos componentes/serviços para permitir a utilização dos simuladores de ambiente do projeto Player. A vantagem de usar esses simuladores, além da estabilidade e ampla utilização, é a possibilidade de realizar *benchmarks* entre o SOM4R e outros *middleware* (e.g. Player, ROS) através desses ambientes virtuais.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] D.E. Bakken, Middleware. Encyclopedia of Distributed Computing. Kluwer Academic Press, 2001.

[2] M. Montemerlo, N. Roy, and S. Thrun, Perspectives on standardization in mobile robot programming: the Carnegie Mellon Navigation (CARMEN) Toolkit. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 3, pp. 2436-2441, 2003.

[3] I. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I.-H. Shu and D. Apfelbaum. Claraty: Challenges and steps toward reusable robotic software. International Journal of Advanced Robotic Systems, vol. 3, no. 1, pp. 023-030, 2006.

[4] G. Metta, P. Fitzpatrick, and L. Natale. YARP: yet another robot platform. International Journal on Advanced Robotics Systems, vol. 3, No. 1, pp. 43-48, 2006.

[5] R. Elfwing, U. Paulsson, L. Lundberg. Performance of SOAP in Web Service environment compared to CORBA. IEEE Software Engineering Conference (APSEC'02), pp. 84-94, 2002.

[6] X. Feng, J. Shen and Y. Fan. REST: An Alternative to RPC for Web Services Architecture. IEEE First International Conference on Future Information Networks (ICFIN), pp. 7-10, 2009.

[7] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, M. J. Mataric. Most Valuable Player: A Robot Device Server for Distributed Control. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001), vol. 3, pp. 1226-1231, 2001.

[8] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and AY Ng, ROS: an open-source Robot Operating System. Proceedings of the Open-Source Software workshop of the International Conference on Robotics and Automation, ICRA 2009.

[9] J. Kramer and M. Scheutz, Development environments for autonomous mobile robots: A survey. Autonomous Robots, vol. 22, no. 2, pp.101-132, 2007.

[10] R. Volpe, IAD Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The claraty architecture for robotic autonomy. Proceedings of the IEEE Aerospace Conference, vol. 1, pp. 1121-1132, Big Sky, Mont, USA, March 2001.

[11] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin, Claraty and challenges of developing interoperable robotic software. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), vol. 3, pp. 2428–2435, October 2003.

[12] J. Meng, S. Mei and Z. Yan. RESTful Web Services: A Solution for Distributed Data Integration. IEEE International Conference on Computational Intelligence and Software Engineering, CiSE 2009, pp. 1-4, 2009.

[13] N. Mohamed, J. Al-Jaroodi, and I. Jawhar, A Review of Middleware for Networked Robots. International Journal of Computer Science & Network Security, vol. 9 No. 5, pp. 139-148, May 2009.

[14] A. Elkady and T. Sobh, Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography. Journal of Robotics, vol. 2012, Article ID 959013, 15 pages, 2012.

[15] H. Overdick. The Resource-Oriented Architecture. Proceedings of the 2007 IEEE Congress on Services, pp. 340-347, 2007.

[16] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. Proceedings of the International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, pp. 407-416, June 2000.

[17] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, I. Horrocks, The semantic web The roles of XML and RDF. IEEE Internet Computing, vol. 4, no. 5 , pp. 63-74, 2000.

[18] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart. RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication. Technical report, The Internet Engineering Task Force, 1999. Acessado em : 02/03/2012. Disponível em: http://www.faqs.org/rfcs/rfc2617.html

[19] D. Peng; C. Li and H. Huo. An extended UsernameToken-based approach for REST-style Web Service Security Authentication. 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2009, pp. 582-586, 2009.

[20] Ibrahim, N. Orthogonal Classification of Middleware Technologies. Mobile Ubiquitous Computing, Systems, Services and Technologies, 2009. UBICOMM '09. Third International Conference on , vol., no., pp.46,51, 11-16 Oct. 2009.

[21] G. Bradski, A. Kaehler. Learning OpenCV. O'Reilly Media, Inc, 2008.

[22] L. Garber. Wi-fi races into a faster future. Computer, vol. 45, no. 3, pp. 13-16, 2012.

[23] Campbell, A.T.; Coulson, G.; Kounavis, M.E. Managing complexity: middleware explained. IT Professional, vol.1, no.5, pp.22,28, Sep/Oct 1999.

[24] OpenKinect Project. Acessado em: 02/03/2012. Disponível em: http://openkinect.org/wiki/Main_Page

[25] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, vol. 2, Issue 1, pp. 14-23, 1986.

[26] R. A. Brooks. A Robot that Walks - Emergent Behaviors from a Carefully Evolved Network. Neural Computation, vol. 2, pp. 692-696, 1989.

[27] D. Hunziker, M. Gajamohan, M. Waibel, R. D'Andrea. Rapyuta: The RoboEarth Cloud Engine. Proceedings of the International Conference on Robotics and Automation (ICRA) 2013.

[28] C. Crick, et al. Rosbridge: ROS for non-ROS users. Proceedings of the 15th International Symposium on Robotics Research (ISRR). 2011.

[29] CMUSphinx Wiki. Acessado em: 02/03/2012. Disponível em: http://cmusphinx.sourceforge.net/wiki/

[30] eSpeak Text To Speech. Acessado em: 02/03/2012. Disponível em: http://espeak.sourceforge.net

[31] Google Maps API. Acessado em: 02/03/2012. Disponível em: https://developers.google.com/maps/documentation/javascript/

[32] H. Bruyninckx. Open robot control software: the OROCOS project. In Robotics and Automation, 2001. Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA'2001), vol. 3, pp. 2523-2528, 2001.

[33] J. Jackson. Microsoft Robotics Studio: a technical introduction. IEEE Robotics and Automation Magazine, vol. 14, no. 4, pp. 82-87, 2007.

[34] A. Elkady, J. Joy, T. Sobh and K. Valavanis. A Structured Approach for Modular Design in Robotics and Automation Environments. Journal of Intelligent and Robotic Systems, vol. 72, issue 1, pp. 5-19, 2013.

[35] Pyfaces: Face Recognition System. Acessado em: 08/27/2011. Disponível em: https://code.google.com/p/pyfaces/

[36] P.N. Belhumeur, J.P. Hespanha, D. Kriegman. Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.19, no.7, pp. 711-720, 1997.

[37] Apache HTTP Server Project. Acessado em: 08/27/2011. Disponível em: http://httpd.apache.org/

[38] Joomla Open Source CMS. Acessado em: 08/27/2011. Disponível em: http://www.joomla.org/

[39] Business Process Model And Notation (BPMN). Acessado em: 02/03/2012. Disponível em: http://www.omg.org/spec/BPMN/index.htm

[40] FFmpeg Software. Acessado em: 02/03/2012. Disponível em: http://ffmpeg.org/

[41] N. G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. J. Ijspeert, M. C. Carrozza, and D. G. Caldwell. iCub - the design and realization of an open humanoid platform for cognitive and neuroscience research. Journal of Advanced Robotics, vol. 21, no. 10, pp. 1151-1175, 2007.

[42] Roy Thomas Fielding. Architectural Styles and the Design of Network based Software Architectures. Ph.D. thesis, University of California, Irvine, 2000. Acessado em 10/11/2011. Disponível em: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[43] L. T.Berners-Lee, R.Fielding, L. Masinter. RFC 3986 - Uniform resource identifiers (uri): Generic syntax. Technical report, The Internet Engineering Task Force, 2005. Acessado em 10/11/2011. Disponível em: http://www.ietf.org/rfc/rfc3986.txt

[44] A.J Gerber, A.Barnard, A.J Van der Merwe. Towards a semantic web layered architecture. International conference on software engineering, 2007.

[45] Thomas Erl. SOA Principles of Service Design. Prentice Hall, SOA Systems Inc, 2007.

[46] Leonard Richardson, Sam Ruby. RESTful Web Services. O'Reilly Media, Inc, 2007.

[47] Subbu Allamaraju. RESTful Web Services Cookbook. O'Reilly Media, Inc, 2010.

[48] Daniel Austin, Abbie Barbir, Christopher Ferris, Sharad Garg. Web Services Architecture Requirements. W3C technical reports, 2004. Disponível em: http://www.w3.org/TR/wsa-reqs

[49] Extensible Markup Language (XML) 1.0 (Fifth Edition). Acessado em 10/11/2011. Disponível em: http://www.w3.org/TR/xml/

[50] M. Billinghurst, H. Kato. Collaborative Augmented Reality. Communications of the ACM, July 2002, vol. 45, no. 7, pp. 64-70.

[51] E. Woods, P. Mason, M. Billinghurst. MagicMouse: an Inexpensive 6-Degree-of-Freedom Mouse. Proceedings of Graphite 2003, Feb 11th-13th, 2003, Melbourne.

[52] Red5 Media Server. Acessado em: 02/03/2012. Disponível em: http://www.red5.org/

[53] Thomas Erl. SOA Design Patterns. Prentice Hall, SOA Systems Inc, 2009.

[54] Elisa Bertino, Lorenzo D. Martino, Federica Paci, Anna C. Squicciarini. Security for Web Services and Service-Oriented Architectures. Springer, 2010.

[55] Antoniou G. and Von Harmelen F. A Semantic Web Primer. The MIT Press 2004.

[56] Timothy J. Ross. Fuzzy Logic with Engineering Applications, 2oEdition, John Wiley & Sons, 2004.

[57] The OAuth 2.0 Authorization Protocol v2-23. Acessado em: 03/02/2012. Disponível em: http://tools.ietf.org/html/draft-ietf-oauth-v2

[58] ARToolKit Homepage. Acessado em: 02/03/2012. Available in http://www.hitl.washington.edu/artoolkit/

[59] Thomas Erl. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall, 2005.

[60] Arno Puder, Kay Rmer, and Frank Pilhofer. Distributed Systems Architecture: A Middleware Approach. ELSEVIER, 2005.

[61] International Federation of Robotics. Executive Summary of World Robotics 2010. Disponível em: http://www.ifr.org/uploads/media/2010_Executive_Summary_rev_01.pdf

[62] Dudek, Gregory; Jenkin, Michael. Computational Principles of Mobile Robotics. Cambridge University Press, 2000.

[63] Siciliano, Bruno; Sciavicco, Lorenzo; Villani, Luigi; Oriolo, Giuseppe. Robotics: Modelling, Planning and Control. Springer, 2008.

[64] Beardmore, Roy. Timing Belts. Acessado em: 27/05/2011. Disponível em: http://www.roymech.co.uk/Useful_Tables/Drive/Timing_belts.html

[65] Roos, David E. David E. Timing Belt Selection And Troubleshooting. Disponível em: http://www.gates.com/facts/documents/Gf000287.pdf

[66] Meggiolaro, Marco Antonio; Soares, Bruno Favoreto Fernandes; Ristow, Eduardo Carvalhal Lage von Buettner; Maimon, Felipe. RioBotz Combot Tutorial. Disponível em: http://www.riobotz.com.br/riobotz_combot_tutorial.pdf

[67] Econobelt. XL Timing Belts Pitch .200 inch (5,08 mm). Acessado em: 27/05/2011. Disponível em: http://www.econobelt.com/Q460/RFQ/TBelt/04.htm

[68] VXB Bearing. SBPP202 FYH Bearing 15mm Steel plate pillow type: Ball Bearings. Acessado em: 27/05/2011. Disponível em: http://www.vxb.com/page/bearings/PROD/15mm/Kit9355

[69] STMICROELECTRONICS. L298: Dual Full-Bridge Driver. Disponível em: http://www.datasheetcatalog.org/datasheet/SGSThomsonMicroelectronics/mXrqqxz.pdf

[70] Filho, José Tarcísio C.; Veloso, Marcus V.D.; Cordeiro, Rafael V. Sistema Dedicado de Aquisição de Dados para Automação do Processo de Produção de Biodiesel. Anais do III Congresso da Rede Brasileira de Tecnologia de Biodiesel, Brasília, DF, 2009.

[71] Cordeiro, Rafael V.; Filho, José Tarcísio C.; Veloso, Marcus V.D. , Formação de Engenheiro de Automação via Projeto de P&D Demandado pela Indústria de Biodiesel. Anais do XXXVIII Congresso Brasileiro de Educação em Engenharia. Fortaleza, CE,2010.

APÊNDICE A

CARACTERÍSTICAS GERAIS DO PROJETO DO ROBÔ

A-1 Introdução

Robôs para todos os fins (por exemplo, militares, industriais e domésticos) estão sendo fabricados já há bastante tempo, e hoje, já se encontram presentes em diversos países, desempenhando as mais distintas funções. Estima-se que em 2013, de acordo com estudo da Federação Internacional de Robótica (IFR) [61], seja vendido um número recorde de até 160.000 unidades de robôs, incluindo robôs industriais e de serviço. Estes dados tratam do crescimento da robótica em países asiáticos, europeus e norte-americanos, mas também mencionam o notável crescimento nas demandas para os mercados do Brasil e Rússia [61].

No Brasil, entretanto, esta demanda por produtos de alto valor tecnológico agregado, na área de robótica, encontra-se reprimida devido aos altos custos de importação e posterior manutenção dos produtos importados. Não são poucas as empresas e universidades no país que chegaram a adquirir robôs importados e, posteriormente, não tiveram suporte técnico adequado ou acesso a peças de reposição, ocasionando perda do investimento. Neste contexto, é clara a necessidade de desenvolvimento de tecnologia nacional, que possa satisfazer as demandas nacionais e permitir pleno aproveitamento dos benefícios que a robótica pode proporcionar.

Visando adequar a realidade brasileira ao cenário global, foi o Centro de Referência em Automação e Robótica (CENTAURO), estabelecido através de parceria entre a Fundação Núcleo de Tecnologia Industrial do Ceará (NUTEC) e o Departamento de Engenharia de Teleinformática (DETI/UFC), cujo principal objetivo é a realização de projetos industriais em Automação, Robótica e Compatibilidade Eletromagnética, com perspectiva de incorporação de máquinas, componentes e sistemas inteligentes aos processos industriais e de serviços. O CENTAURO conta com o apoio da Secretaria da Ciência, Tecnologia e Educação Superior do Ceará – SECITECE e da empresa Rockwell Automation do Brasil Ltda.

Uma das tarefas vinculadas ao Centro de Referência em Automação e Robótica, é o projeto e construção de um Robô Móvel com Rodas (RMR), denominado Hermes I, compreendendo o *hardware* e *software* da plataforma robótica e de seus diferentes subsistemas de: propulsão, computação embarcada, sensoriamento, comunicação externa e energia. A

importância de produzir o robô para navegação em ambiente estruturado surge da necessidade de entender o sistema robô como um todo e, assim, tentar embutir nele, como é aqui o caso, os requisitos necessários para realizar, da melhor forma possível, as tarefas típicas de deslocamento de um ponto a outro em modo autônomo e de interação com o ambiente estruturado em que estão inseridos, como por exemplo: transporte e monitoramento (indústrias e serviços); ajuda ao cliente (lojas, museus e *shopping centers*); grandes áreas (vigilância e limpeza); entretenimento (brinquedos) e auxílio a deficientes. A sua concepção estará focada para: a) melhorar o desempenho do robô ao executar uma trajetória planejada no ambiente estruturado e ao determinar sua localização (como o robô sabe onde ele está) ao longo do tempo, b) a comunicação externa para permitir ao robô interagir com outros robôs, com os seres humanos (funções de reconhecimento de faces e comunicação oral), e com outros computadores (acesso a serviços de internet, bancos de dados). O Robô SCITOS G5 é de fabricação alemão adquirido pelo CENTAURO e que juntamente com o Hermes I são empregados para as nossas atividades de pesquisa, a saber: implementação de novas metodologias de navegação, sensoriamento e interação homem-robô. Estas plataformas robóticas são importantes como suporte ao desenvolvimento desta tese.

A-2 Robôs Móveis com Rodas - Fundamentos Básicos

Neste trabalho, primeiramente foi realizado o estudo sobre os tipos de locomoção existentes para robótica, de modo a validar a escolha de um sistema de propulsão para um projeto de robô móvel. Com base neste estudo, configuração que mais se adequava aos requerimentos de projeto era a de um sistema de locomoção diferencial com rodas detalhado nas subseções seguintes.

A-2.1 Tipos de rodas

Roda é um mecanismo circular, sólido ou vazado, com um eixo rotativo em seu centro, usado comumente para transporte de cargas e movimentação de máquinas. Uma roda ideal possui movimento de rolamento puro, sem translação na direção do movimento e tampouco na direção ortogonal ao movimento [62]. Em robótica, o número de rodas utilizadas depende da aplicação, mas a quantidade mais comum são duas, três ou quatro rodas. Com três ou mais rodas, obtém-se estabilidade estática, enquanto que com duas rodas, a estabilidade é dinâmica e exige controle preciso das acelerações e velocidades angulares do robô para evitar tombamento [62].

Quatro tipos de rodas são comumente utilizados para locomoção de robôs, que serão descritas a seguir.

a) roda fixa: possui 1 grau de liberdade, com rotação ao redor do ponto de contato. A Fig. A-1 mostra um exemplo de roda fixa [62].

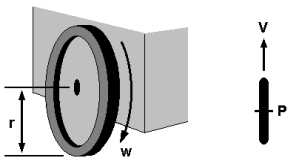

Figura A-1: Roda fixa.

b) roda com orientação: possui dois graus de liberdade, com rotação controlada ao redor do eixo da roda e do ponto de contato, também controlada. A Fig. A-2 mostra um exemplo de roda com orientação [62].

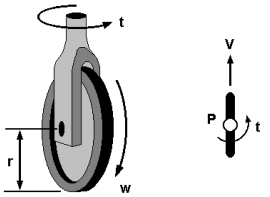

Figura A-2: Roda com orientação.

c) roda de apoio: possui três graus de liberdade, com rotação ao redor do eixo da roda, do ponto de contato e do eixo de apoio. A Fig. A-3 mostra um exemplo de roda de apoio [62].

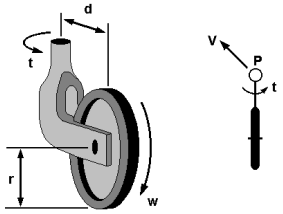

Figura A-3: Roda de apoio.

d) roda sueca: possui três graus de liberdade, com rotação ao redor do eixo da roda (controlado), ao redor dos rolos e ao redor do ponto de contato. A Fig. A-4 mostra uma configuração de roda sueca [62].

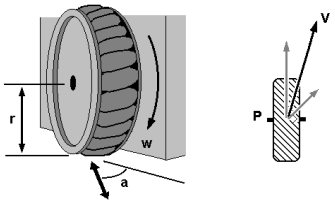

Figura A-4: Roda sueca.

A-2.2 Locomoção de robôs móveis com rodas

A disposição espacial e combinação destes tipos de rodas resultam em várias configurações possíveis para locomoção em robótica, conforme a Fig. A-5, cujas principais são:

i) Triciclo: possui três rodas, duas fixas na parte de trás e uma roda com orientação na frente. A tração pode vir das rodas de trás ou da roda da frente, mas a direção do robô é definida em função do ângulo formado pela roda dianteira [62, 63].

ii) Direção síncrona: podem possuir três ou mais rodas, onde todas são do tipo rodas com orientação. As rodas possuem um mecanismo de sincronização, que faz com que todas se orientem na mesma direção, compondo assim o sistema de direcionamento do robô [62,63].

iii) Omnidirecional: geralmente usam-se três rodas suecas, dispostas em formato de um triângulo equilátero, com uma roda em cada vértice. Sua construção e controle são bastante simples, constituindo uma configuração holonômica. Sua desvantagem está na baixa robustez e durabilidade do conjunto, pois este tipo de roda é formado por vários rodízios em conjunto, embora atualmente já existam versões industriais para estes robôs [62,63].

iv) Direção de carro: também conhecida como *ackerman steering*, possui o mesmo princípio de direção de um automóvel, com duas rodas traseiras fixas e duas rodas dianteiras com orientação. Geralmente é uma configuração escolhida para robôs maiores e que se locomovem em ambientes não estruturados, como terrenos arenosos e acidentados [62,63].

v) Direção diferencial: possui duas rodas fixas independentes e alinhadas num mesmo eixo, podendo possuir uma ou umas rodas de apoio, para fornecer estabilidade estática. É a configuração mais simples do ponto de vista mecânico, com sistema de controle relativamente simples também. Permite mover-se em linha reta, em forma de arco ou girar em torno do próprio eixo [62,63].

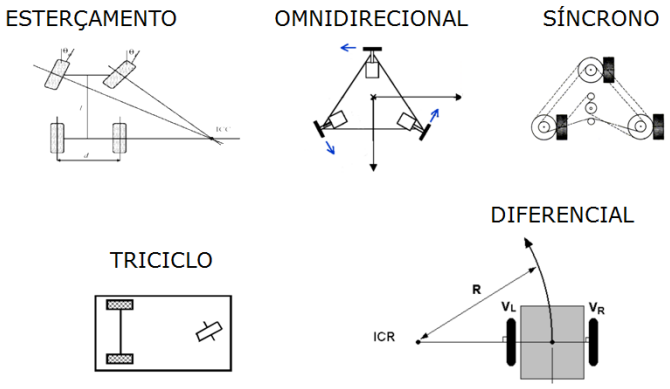

Figura A-5: Possíveis configurações para robôs com rodas.

Considerando os tipos de arranjo de rodas, e a aplicação do robô, escolheu-se o sistema de tração diferencial com uma roda de apoio para dar estabilidade. Acreditou-se que este sistema daria uma boa rigidez mecânica, sendo uma solução de baixo custo, fácil fabricação e com modelagem matemática de controle relativamente simples.

A-3 Projeto Mecânico e de Controle do Protótipo do Robô

A-3.1 Aspectos Gerais

Desenvolvimento de um robô móvel com rodas para operação em ambientes estruturados, ou seja, ambientes que embora não tenham seus elementos plenamente conhecidos internamente ao robô, é formado por padrões que o permitem reconhecer e identificar estes elementos. O robô pode, preliminarmente, locomover-se de modo autônomo, identificar e desviar de obstáculos e pessoas, e realizar monitoramento remoto. Para tanto, foi dotado de um conjunto de sensores e técnicas de visão artificial, que em conjunto, permitem ao robô mover-se sem dificuldades.

Observando outros robôs desenvolvidos anteriormente, verificamos que a ausência de rigidez e durabilidade do conjunto mecânico fazia com que as leituras dos sensores fossem alteradas pela vibração excessiva, e o robô tivesse um comportamento diferente daquele estimado por seu programa de controle. Assim, a preocupação inicial foi a de desenvolver uma estrutura mecânica que fosse rígida e confiável do ponto de vista estrutural.

O projeto foi iniciado em 2009, e quatro versões para o arranjo espacial e geométrico das peças foram idealizadas, até obter-se uma configuração que satisfizesse todas as condições de espaço e funcionalidades requeridas. Outras configurações podem surgir futuramente, mas o sistema que é apresentado neste trabalho foi o que teve melhor adequação às condições impostas, tendo sido fabricado e funcionado satisfatoriamente. As Figuras A-6, A-7, A-8 e A-9 mostram a evolução do projeto mecânico para o sistema de locomoção do robô.

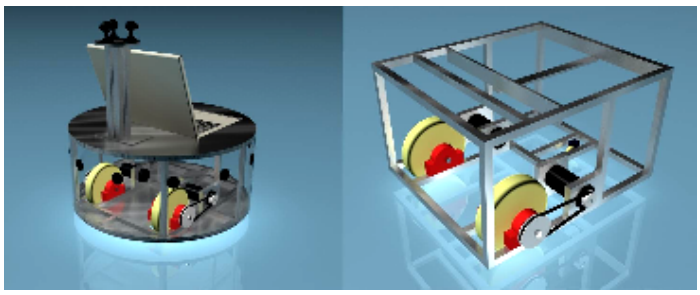

Figura A-6: Primeira versão do robô.

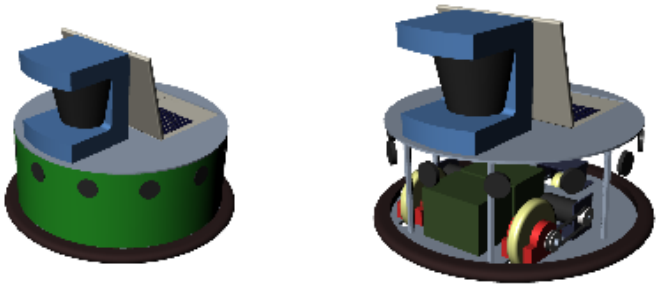

Figura A-7: Segunda versão do robô.

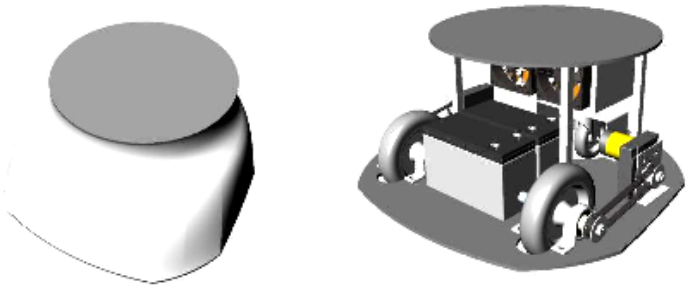

Figura A-8: Terceira versão do robô.

Figura A-9: Quarta e última versão do robô.

A patente do robô foi depositada no Instituto Nacional da Propriedade Industrial (INPI) sob o número PI 1106417-0 e publicada inicialmente na Revista de Propriedade Industrial (RPI) número 2171 de 14/08/2012. Data do depósito: 11/10/2011. Depositantes: Fundação Núcleo de Tecnologia Industrial do Ceará - NUTEC (BR/CE), Universidade Federal do Ceará - UFC (BR/CE). Inventores: Marcus Vinicius Duarte Veloso, Rafael Vasconcelos Cordeiro, José Tarcísio Costa Filho. Título: ROBÔ AUTÔNOMO PARA VIGIL NCIA E PROCESSO DE PRODUÇÃO DE ROBÔ AUTÔNOMO PARA VIGIL NCIA. Devido a este fato, neste trabalho não é permitida a revelação de detalhes mais internos do robô, e estará focalizado em fornecer características gerais do projeto.

A-3.2 Características do Robô

O sistema de locomoção consiste numa base móvel com três rodas, duas fixas e uma roda de apoio. As rodas fixas são acionadas independentemente através de dois motores de corrente contínua, acoplados no eixo de cada roda. O acoplamento entre os eixos dos motores e os eixos das rodas é realizado mediante o uso de correias sincronizadoras. A Fig. A-10 mostra como foi instalado o sistema de transmissão de potência entre o eixos e motores.

Considerando a aplicação do robô, inicialmente foram estimadas algumas características iniciais do projeto. Estas características serviram de restrição e guia para o planejamento das atividades, e algumas sofreram alterações ao longo do desenvolvimento do projeto. Estas estimativas são:

- Peso máximo do robô de 20 kg, incluindo circuitos eletrônicos, baterias, estrutura mecânica, *notebook* de controle embarcado, motores, rodas e revestimento exterior de proteção;
- As dimensões devem permitir ao robô passar por portas. Considerando uma porta de largura de 70 cm, o robô não deve possuir mais de 50 cm de largura, pois sua navegação será também autônoma e se faz necessário ter uma margem de segurança para evitar colisões;
- Autonomia das baterias para tempo de operação mínima de 3 horas. Este é um valor inicial, que deverá ser modificado em outra etapa para mínimo de 6 horas;
- Velocidade máxima de 1 m/s;
- Aceleração suave, de no máximo 0,5 m/s^2;
- Altura do chassi em relação ao solo de 3 cm;
- Diâmetro das rodas mínimo de 10 cm, para possibilitar ao robô subir pequenos batentes e possibilitar uma altura mínima do chassi em relação ao solo. Escolheu-se rodas com diâmetro de 12,7 cm (5 in);

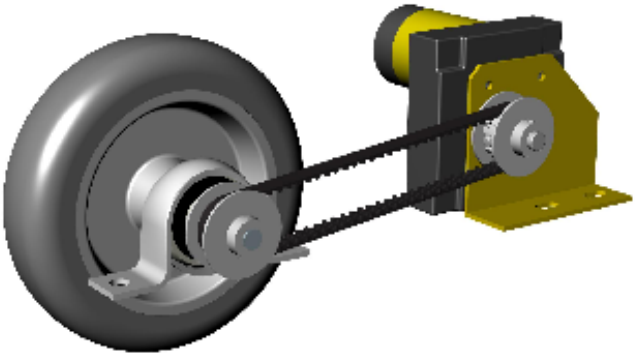

Figura A-10: Sistema de transmissão de potência por correias sincronizadoras.

A-3.3 Motor Elétrico

O projeto foi realizado com base nos motores que estavam à disposição no Laboratório de Robótica (LABOR – CENTAURO / NUTEC). As especificações destes motores não eram desconhecidas, e tivemos de realizar ensaios de torque e velocidade, com objetivo de estimar as características elétricas e mecânicas do motor. A Fig. A-11 mostra duas vistas do motor com dimensões em milímetros.

Os motores são de corrente contínua (CC) e possuem caixa de redução acoplada. O único parâmetro de operação disponível era a tensão nominal de alimentação de 12 V. Para estimação das características elétricas e mecânicas do motor, foram realizadas as seguintes medições:

- Fixou-se o motor elétrico em bancada, e com um alicate de pressão, travou-se o eixo. Aplicou-se uma tensão elétrica de 12 V no motor, e com o auxílio de um multímetro instalado em série com os fios de alimentação, mediu-se a corrente. Nesta condição, a corrente elétrica é máxima e teve valor de aproximadamente $I_{travamento} = 0{,}95\ A$. O valor medido apresentou variação entre *1,05 A* e *0,85 A* devido à imprecisão do multímetro, e o valor adotado de $I_{travamento}$ foi o valor médio.

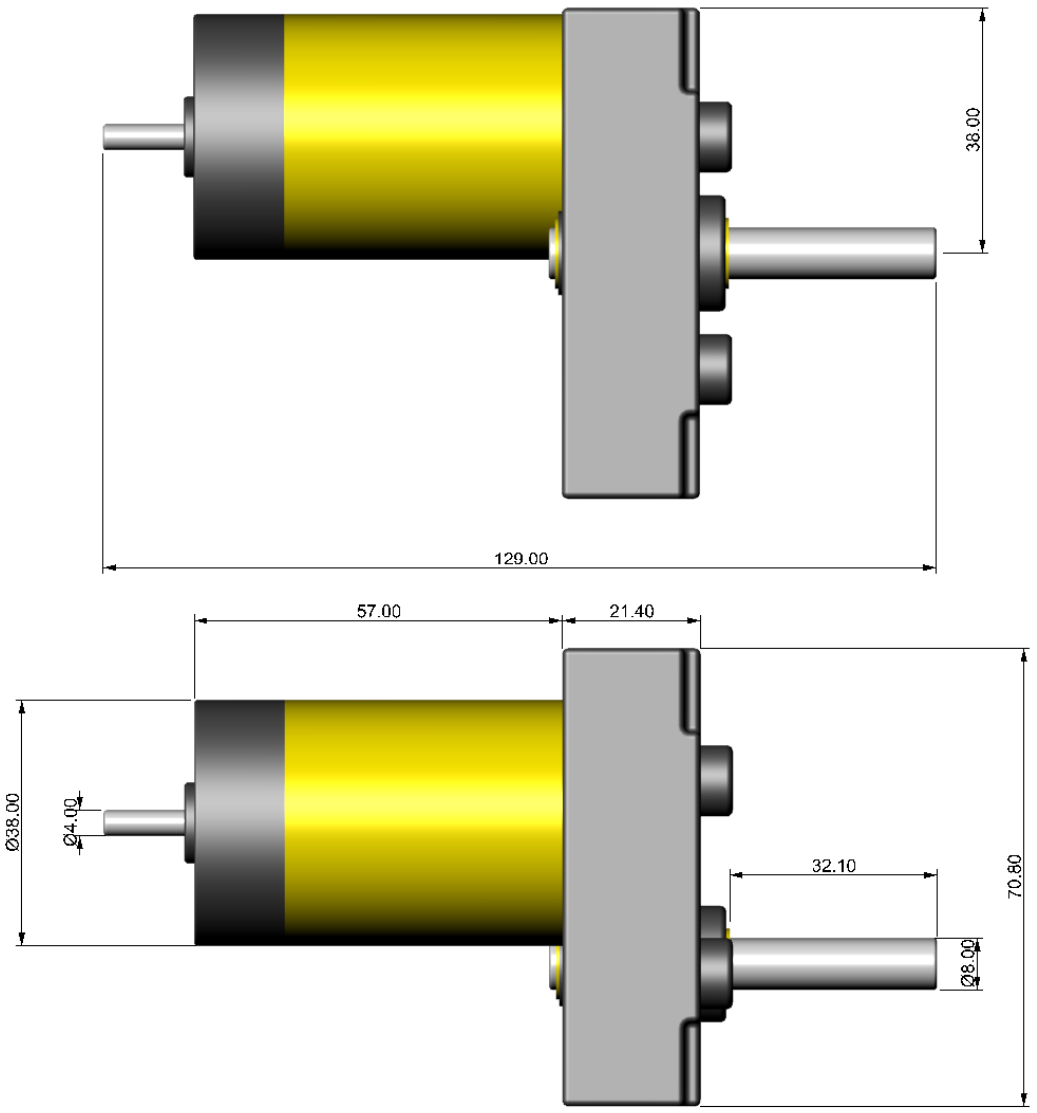

Figura A-11: Vistas do motor de corrente contínua.

- Com o motor elétrico fixado em bancada, prendeu-se o eixo do motor num alicate de pressão. Na outra ponta do alicate, colocou-se um recipiente preso através de um barbante. Segurou-se o alicate na posição horizontal, e ligou-se o motor com uma tensão elétrica de 12 V. Verificando-se que o motor suportava o peso do alicate, desligou-se o motor, e adicionou-se certo volume de água no recipiente. Ligou-se novamente o motor, segurando o alicate na posição vertical, e verificou-se que o mesmo continuava a girar, suportando o torque exercido pela massa de água e alicate. Desligou-se o motor e adicionou-se novamente água ao recipiente e se verificou se o motor conseguia girar, partindo sempre da posição horizontal. Este procedimento de adicionar pequeno volume de água foi sendo repetido até que em uma das tentativas o motor não girou, permanecendo parado na posição horizontal, conforme mostrado na Fig. A-12. Nesta

situação, o motor apresenta torque estático máximo, atingindo seu limite de corrente máxima também. Mediu-se a seguir o volume de água, obtendo-se $V = 0,75\ l$, que corresponde a uma massa $m_{água} = 0,75kg$. O recipiente tem massa de $m_{recipiente} = 0,08kg$, medida em balança de cozinha. A distância entre o ponto de apoio e o eixo do motor é $d_{água} = 20,5\ cm$, resultando num torque $T_{água} = 17,015\ kg.cm$. Mediu-se a massa do alicate, $m_{alicate} = 0,5\ kg$, e a distância do centro de massa do alicate até o eixo do motor, $d_{alicate} = 7\ cm$, resultando num torque adicional $T_{alicate} = 3,5\ kg.cm$. O torque total de travamento registrado para este motor é aproximadamente $T_{travamento} = 20,515\ kg.cm$. Optou-se por utilizar o valor de *20 kg.cm* devido ao fato de este ser um valor comum para este tipo de motor, e devido à incerteza das medições realizadas. Em seguida converteu-se a unidade em *kg.cm* para a unidade padrão do Sistema Internacional, em *N.m*, e o valor do torque de travamento encontrado foi aproximadamente $T_{travamento} = 1,9613\ N\ m$.

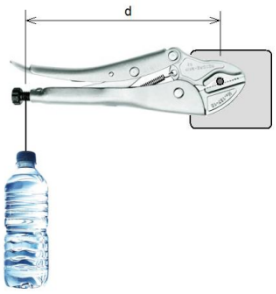

Figura A-12: Esquema representativo do ensaio de torque realizado.

- O eixo do motor elétrico gira lentamente, devido à caixa de redução acoplada. Assim, aplicando-se uma tensão elétrica de 12 V no motor, sem nenhuma carga acoplada a ele, contou-se o número de voltas num intervalo de tempo de dois minutos. Foram contadas 100 rotações, resultando num valor de $\omega_{\text{sem carga}} = 50$ rpm. Este teste foi realizado duas vezes, obtendo-se o mesmo valor.
- Com o motor sem carga acoplada, fixado em uma bancada, aplicou-se uma tensão elétrica. A corrente medida foi aproximadamente $I_{sem\ carga} = 0,06\ A$, observando-se novamente variações nas medições das correntes. O valor adotado para $I_{sem\ carga}$ foi o valor médio entre as medições.
- Com o motor elétrico desligado, acoplou-se os terminais do multímetro com os fios de alimentação do motor. A resistência elétrica medida foi aproximadamente $R_{armadura} = 12,5\ \Omega$.

Deve-se considerar que o resultado destas medições é impreciso, sendo observada variação em grande parte das leituras nos equipamentos de medição. Valores médios foram considerados, após sucessivas leituras, com objetivo de obter características aproximadas dos motores, e serão tomados como referência apenas para início do projeto. Para um cálculo mais refinado destas propriedades mecânicas e elétricas do motor, seria necessário obter os dados de

$T_{travamento}$, $R_{armadura}$, $I_{travamento}$, $I_{sem\ carga}$ e $\omega_{\text{sem carga}}$ diretamente com o fabricante.

De posse dos valores experimentais medidos e utilizando-se as equações 1 a 3, criou-se uma tabela de dados, Tabela A-1, e criou-se a Fig. A-13, que apresenta as curvas características para o motor observado.

Corrente (A)	**Velocidade Angular (rpm)**	**Torque (Nm)**	**Potência Mecânica (W)**	**Eficiência**
0,0600	50,0000	0,0000	0,0000	0,0000
0,1045	47,5000	0,0981	0,4878	0,3890
0,1490	45,0000	0,1961	0,9243	0,5169
0,1935	42,5000	0,2942	1,3094	0,5639
0,2380	40,0000	0,3923	1,6431	0,5753
0,2825	37,5000	0,4903	1,9255	0,5680
0,3270	35,0000	0,5884	2,1566	0,5496
0,3715	32,5000	0,6865	2,3363	0,5241
0,4160	30,0000	0,7845	2,4647	0,4937
0,4605	27,5000	0,8826	2,5417	0,4600
0,5050	25,0000	0,9807	2,5674	0,4237
0,5495	22,5000	1,0787	2,5417	0,3855
0,5940	20,0000	1,1768	2,4647	0,3458
0,6385	17,5000	1,2749	2,3363	0,3049
0,6830	15,0000	1,3729	2,1566	0,2631
0,7275	12,5000	1,4710	1,9255	0,2206
0,7720	10,0000	1,5691	1,6431	0,1774
0,8165	7,5000	1,6671	1,3094	0,1336
0,8610	5,0000	1,7652	0,9243	0,0895
0,9055	2,5000	1,8633	0,4878	0,0449
0,9500	0,0000	1,9613	0,0000	0,0000

Tabela A-1: Dados do motor obtidos através de procedimento experimental.

$$P_{\text{mecânica}} = T \times \omega \tag{1}$$

$$P_{\text{elétrica}} = V_{entrada} \times I_{entrada} \tag{2}$$

$$\eta = \frac{P_{\text{mecânica}}}{P_{\text{elétrica}}} \tag{3}$$

Onde:

- $P_{mecânica}$: potência mecânica liberada no eixo do motor
- T: torque no eixo do motor
- ω: velocidade angular do eixo do motor
- $P_{elétrica}$: potência elétrica absorvida no motor
- $V_{entrada}$: tensão elétrica aplicada ao motor
- $I_{entrada}$: corrente elétrica que circula no motor
- η: eficiência do motor

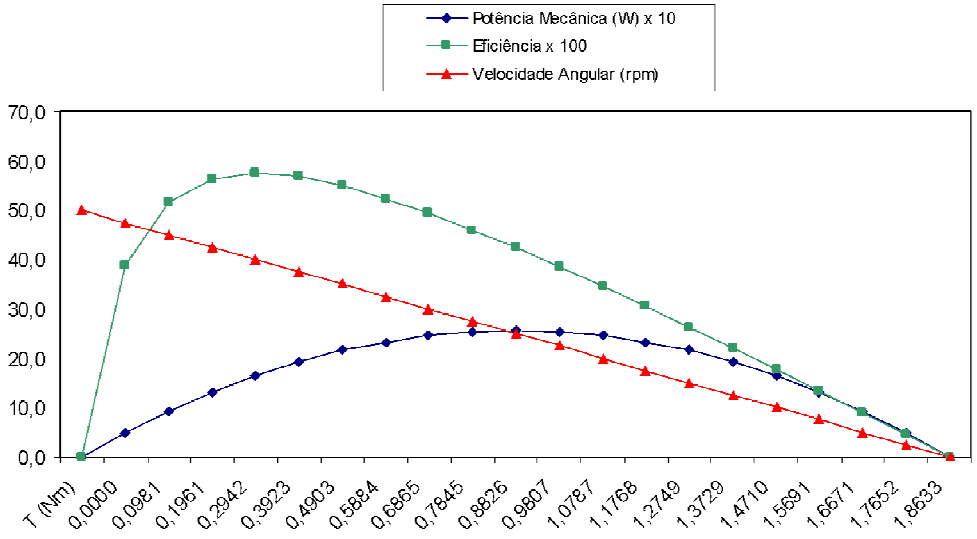

Figura A-13: Curvas características para o motor de corrente contínua utilizado.

As curvas da Fig. A-13 mostram o comportamento que o motor pode assumir para uma tensão constante, de 12 V, onde a presença de forças dissipativas aplicadas ao eixo do motor define o ponto de velocidade, potência, torque e eficiência com a qual o motor irá operar.

Observa-se que a potência é máxima quando o torque é metade de seu valor máximo, e a eficiência máxima ocorre normalmente entre 80% e 90% de $\omega_{\text{sem carga}}$, mais precisamente quando [66]:

$$I_{entrada} = \sqrt{\frac{V_{entrada} \times I_{sem\ carga}}{R_{motor}}} \tag{4}$$

$$I_{entrada} = \sqrt{\frac{12 \times 0{,}06}{12{,}5}}$$

$$I_{entrada} = 0{,}24\ \text{A}$$

Onde:

- $I_{entrada}$: corrente elétrica que circula no motor, em A;
- $I_{sem\ carga}$: corrente elétrica que circula no motor quando não há carga aplicado em seu eixo, em A;
- $V_{entrada}$: tensão elétrica aplicada ao motor, em A;
- R_{motor}: resistência elétrica, em Ω;

Comparando o valor obtido para $I_{entrada}$ com os dados obtidos na Tabela A-1, percebe-se que a eficiência será máxima aproximadamente na velocidade angular de 40 rpm.

A-3.4 Dinâmica das Rodas

Da lei de conservação de energia, se não houvesse perdas por atrito, uma vez o robô operando em velocidade constante, não seria necessário adicionar energia para manter o movimento. Mas devido ao fato de que em aplicações reais as perdas devem ser consideradas, tem-se que a potência mecânica aplicada ao eixo pela ação do motor tem a função de manter a velocidade do robô constante.

A força que move o robô é a força de reação ao atrito das duas rodas com o piso, e é igual a:

$$F_{at} = m_{robô} \times a_{robô} \tag{5}$$

$$T = r_{roda} \times F_{at} \tag{6}$$

$$T = r_{roda} \times m_{robô} \times a_{robô} \tag{7}$$

E a potência mecânica aplicada às rodas é:

$$P_{roda} = T \times \omega \tag{8}$$

$$P_{roda} = r_{roda} \times m_{robô} \times a_{robô} \times \omega \tag{9}$$

$$P_{roda} = m_{robô} \times a_{robô} \times v_{robô} \tag{10}$$

$$P_{roda} = P_{motor} \times f \tag{11}$$

Sendo que:

$$f = \eta_{redução} \cdot \eta_{correia} \cdot \eta_{rolamentos} \cdot \eta_{rodas} \tag{12}$$

Onde:

- F_{at}: força de atrito das rodas com o solo, em N;
- $m_{robô}$: massa do robô, com todas as suas partes, em kg;
- $a_{robô}$: aceleração máxima que se deseja que o robô possa assumir, em m/s²;
- $v_{robô}$: velocidade máxima que se deseja que o robô possa assumir, em m/s;
- r_{roda}: raio da rodas utilizadas no robô, em m;
- T: Torque aplicado nas rodas para movimentar o robô, em N.m;
- P_{motor}: potência liberada nos eixos dos motores elétricos, em W;
- P_{roda}: potência exercida nas rodas para movimentar o robô, em W;
- f: eficiência global do sistema de locomoção do robô
- $\eta_{redução}$: eficiência da caixa de redução
- $\eta_{correia}$: eficiência das correias sincronizadoras
- $\eta_{rolamentos}$: eficiência dos rolamentos de apoio
- η_{rodas}: eficiência das rodas devido ao deslizamento

A-3.5 Correias e Polias Sincronizadoras

Correias sincronizadoras são geralmente utilizadas quando se deseja transmitir potência entre eixos e para garantir que a polia acionada esteja em movimento com razão de velocidade constante em relação à polia motora [64]. A versatilidade destas correias vem substituindo a transmissão por correntes, pois são silenciosas, não necessitam de lubrificação e podem atingir velocidades maiores do que as correntes. Em algumas aplicações, correias estão substituindo também transmissão de potência por engrenagens, pois seus dentes entram e saem das polias por rolamento, com suavidade e com baixo atrito [65].

Um sistema de transmissão por correias, quando bem projetado e instalado, deve ter operação adequada por até entre 8.000 e 12.000 horas, funcionando com eficiência de aproximadamente 98%. No entanto, caso o alinhamento entre as polias não seja exato, sua eficiência e durabilidade são gravemente comprometidas [64].

Os esforços mecânicos são pequenos e não foram considerados relevantes para a escolha do elementos do sistema de transmissão. Então, os aspectos restritivos estavam concentrados nas configurações geométricas e espaciais, pois o espaço é limitado, com distância entre eixos de 15 cm, e diâmetros dos eixos movido e motor de 8mm.

A correia escolhida foi o modelo 140XL037, da empresa Dina, que pode ser vista na Fig. A-14. O primeiro número, 140, diz respeito ao tamanho da correia, que é de 14 in (355,6 mm). As letras XL significam que a correia é especificada para serviços extra-leves. O segundo e último número, 037, fazem referência para a largura da correia, que é de 3/8 in (9,53 mm).

Figura A-14: Correia sincronizadora utilizada.

As polias escolhidas foram do modelo 15XL037. O primeiro número, 15, diz respeito ao número de dentes da polia, que são 15 dentes. As letras XL significam que a polia deve trabalhar com correia do tipo extra-leve. O segundo e último número, 037, fazem referência à largura da polia, que é de 3/8 in (9,53 mm). A Fig. A-15 mostra o desenho técnico para esta polia.

Observando o catálogo do fornecedor, obtém-se os seguintes dados sobre a polia:

- Comprimento L igual a *19,8 mm*;
- Diâmetro primitivo OD igual *24,26 mm*;
- Diâmetro FD igual *30,1 mm*;
- Diâmetro E igual *15,9 mm*
- Furo mínimo de *6 mm* e máximo de *10 mm*;

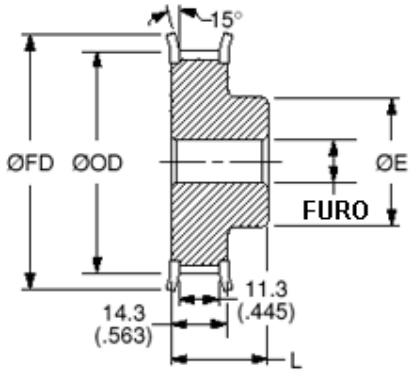

Figura A-15: Desenho técnico da polia sincronizadora utilizada.

Como a razão de velocidades é igual a 1, o fator limitante para a escolha das polias foi a possibilidade de usar um furo de 8 mm (diâmetro dos eixos) e ainda assim permitir uma distância razoável entre o furo e a superfície do cubo. Este espaçamento é importante, pois nele foi realizado furo com rosca interna, para apertar o eixo do motor, fixando a polia no eixo.

O torque de travamento do motor elétrico utilizado foi calculado na seção 4.3, e é de aproximadamente $T_{travamento} = 1{,}9613\ N\ m$. O raio primitivo da polia é de $r_{primitivo} = 12{,}1\ mm$.

Uma tensão máxima aproximada que atua na correia pode ser encontrada com base na equação (13):

$$F = \frac{T_{travamento}}{r_{primitivo}}$$
$$F = \frac{1{,}9613}{12{,}1 \times 10^{-3}} \tag{13}$$
$$F = 162{,}1 \text{ N}$$

A capacidade de carga para este tipo de correia é 60 N/mm [67]. Como a correia tem largura de 9,53 mm, a capacidade de carga da correia é:

$$F_{máx} = 60 \times 9{,}53$$
$$F_{máx} = 571{,}8 \text{ N}$$

A capacidade de carga da correia utilizada é 3,5 vezes a tensão máxima que pode atuar no robô, constituindo uma boa seleção. Nesta análise não foram consideradas as forças de tensionamento das correias, por dificuldade em medi-las.

A-3.6 Mancais de Apoio

Foram utilizados dois mancais de apoio em cada roda, um de cada lado, totalizando quatro rolamentos utilizados para as duas rodas dianteiras, com objetivo de suportar cargas radiais (peso do robô e tração nas correias) e cargas axiais que aparecem quando o robô faz curvas. A Fig. A-16 mostra um desenho técnico com dimensões em milímetros para este mancal.

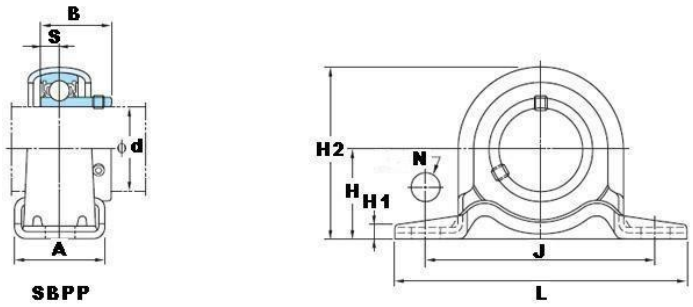

Figura A-16: Mancal de apoio modelo SBPP202.

Onde:

- *d*: 15 mm
- *H*: 22,2 mm
- *L*: 86 mm
- *A*: 25 mm
- *J*: 68 mm
- *N*: 9,5 mm
- *H1*: 3,2 mm

- *H2*: 43,8 mm
- *S*: 6 mm
- *B*: 22 mm

O modelo de mancal de apoio escolhido foi o SBPP202, que foi o menor encontrado no mercado local deste tipo. Ainda assim, o diâmetro interno do rolamento é de 15 cm, forçando o eixo a ter o mesmo diâmetro.

A carga radial em um rolamento é o peso do robô dividido por todos os rolamentos de apoio. Considerando uma massa máxima *m = 20 kg*, especificada na seção 4.2, seu peso será *P = 19,62 N*. A carga radial em cada rolamento será então:

$$F_r = 4{,}91\ \text{N}$$

A carga axial pode ser calculada através da equação da força centrífuga, equação (14). Embora existam quatro rolamentos, a carga axial é suportada por apenas um rolamento em cada eixo, resultando em dois rolamentos com carga axial [23].

$$F_a = \frac{m_{robô} \times v_{robô}^{\ 2}}{2 \times r_c} \tag{14}$$

Onde:

- Fa: carga axial presente no rolamento
- mrobô: massa do robô, com todas as suas partes, em kg;
- vrobô: velocidade máxima que se deseja que o robô possa assumir, em m/s;
- rc: raio de curvatura

A Fig. A-17 mostra um esquema do movimento curvilíneo do robô. Para o robô construído, a distância entre as rodas d = 0,28 m. Considerando rc como a metade da distância entre as rodas, que é onde a força centrífuga será maior, e rc = 0,14 m. A velocidade utilizada será a máxima de 1 m/s, especificada na seção 4.3.

$$F_a = \frac{20 \times 1^2}{2 \times 0{,}14}$$

$$F_a = 71{,}43\ \text{N}$$

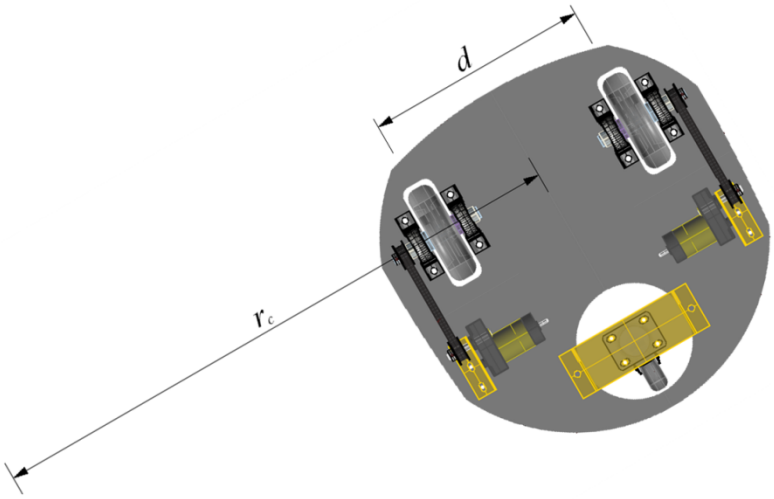

Figura A-17: Esquema representativo de movimentação curvilínea do robô.

O rolamento é o FYH SB202, de esferas, com carga nominal dinâmica $C_r = 9{,}55\ KN$, e carga nominal estática $C_{or} = 4{,}80\ KN$ [68].

Em algumas situações, é possível haver carga axial e radial simultaneamente, e nestes casos, a equação 15 deve ser utilizada para encontrar a carga equivalente [63].

$$P = XVF_r + YF_a \tag{15}$$

Onde:

- P: carga equivalente
- F_r: carga radial constante aplicada
- F_a: carga axial constante aplicada
- V: fator de rotação
- X: fator radial
- Y: fator axial

Os valores para os fatores de rotação, de cargas radial e axial, foram encontrados usando-se a Tabela A-2, obtendo-se $V = 1$, $X = 056$ e $Y = 2{,}30$. O valor de $V = 1$ deve-se ao fato de que o anel interno está rodando em relação à carga. O valor de Y adotado deve-se ao fato de ser o valor limite disponível na tabela.

A carga equivalente é então calculada como:

$$P = 0{,}56 \times 1 \times 4{,}91 + 2{,}30 \times 71{,}43$$

$$P = 167{,}04\ \text{N}$$

E a vida útil dos rolamentos deverá ser no mínimo de:

$$L = \left(\frac{C_r}{P}\right)^3$$

$$L = \left(\frac{9{,}55}{0{,}167}\right)^3$$

$$L = 187.008{,}22 \times 10^{6} \text{ revoluções}$$

$\frac{F_a}{C_{or}}$	X	Y	e
0,014	0,56	2,3	0,19
0,028	0,56	1,99	0,22
0,056	0,56	1,71	0,26
0,084	0,56	1,55	0,28
0,11	0,56	1,45	0,3
0,17	0,56	1,31	0,34
0,28	0,56	1,15	0,38
0,42	0,56	1,04	0,42
0,56	0,56	1	0,44

Tabela A-2: Fatores X e Y para mancais, quando Fa > e V Fr

A-3.7 Circuito de Acionamento dos Motores

Foram utilizados dois circuitos integrados L298, um para cada motor, disponibilizando uma potência máxima de 25 W por motor [69]. Cada circuito integrado possui internamente duas pontes em H que podem ser utilizadas em conjunto para fornecer a potência máxima, permitindo o controle de velocidade, frenagem, e sentido de rotação dos motores. A Fig. A-18 mostra um esquema básico do circuito eletrônico utilizando o circuito integrado L298 para o controle dos motores.

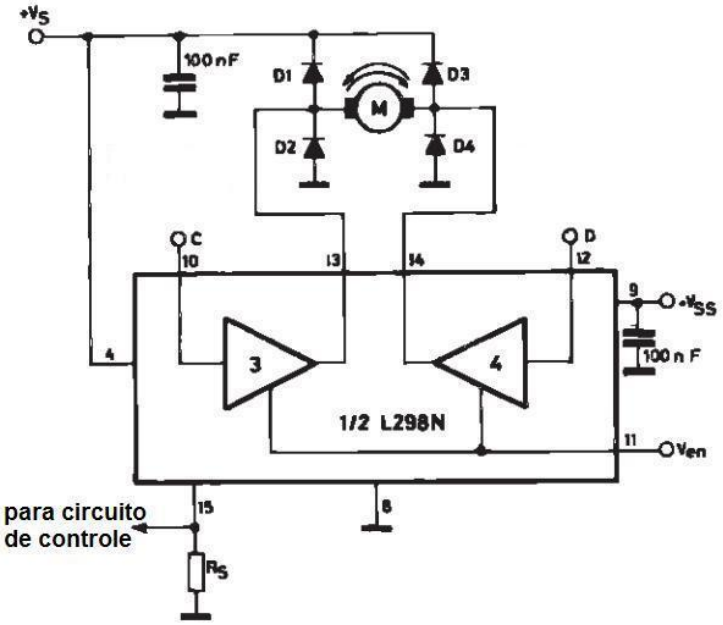

Figura A-18: Esquema do circuito utilizado na placa eletrônica, usando o Circuito Integrado L298.

Para o controle de velocidade em cada roda utilizou-se o conceito de PWM (*Pulse Width Modulation*). Seu funcionamento consiste em alternar a razão entre o tempo em que o pulso possui nível lógico alto e período da oscilação, de acordo com a Equação (17), mantendo a frequência constante. Esta razão é conhecida como Ciclo de Trabalho, sendo normalmente expressa em forma de percentual. A Fig. A-19 mostra três exemplos de tensão elétrica média, obtidas com variação do Ciclo de Trabalho.

$$Ciclo\ de\ Trabalho = \frac{t_{alta}}{t_{total}} \tag{17}$$

Onde:

- t_{alta}: tempo em que o pulso possui nível lógico alto
- t_{total}: período total da oscilação

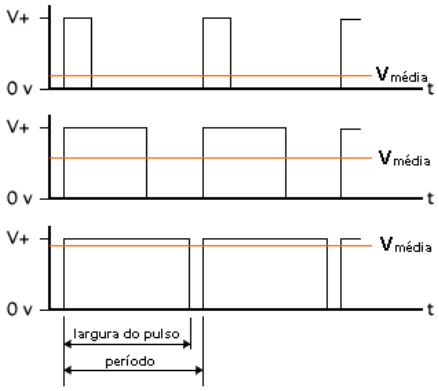

Figura A-19: Modulação por largura do pulso.

O circuito de ponte em H permite a reversão da direção dos motores de corrente contínua, freio eletrônico e controle de velocidade por PWM. Um esquema do funcionamento da ponte em H pode ser visto na Fig. A-20.

Este tipo de circuito permite que o motor possua 4 tipos de operação, a saber:

- Fechando-se as chaves S1 e S4, e mantendo-se as chaves S2 e S3 abertas, o motor deverá girar em um sentido.
- Fechando-se as chaves S2 e S3, e mantendo-se abertas as chaves S1 e S4, o motor deverá girar em sentido contrário ao anterior.
- Deixando-se todas as quatro chaves abertas, o motor fica livre, podendo girar por inércia ou permanecer parado.
- Fechando-se as chaves S2 e S4, ou S1 e S3, os terminais de alimentação do motor estão em curto. Caso o motor esteja em movimento, é induzida uma tensão elétrica reversa que força o motor a parar.

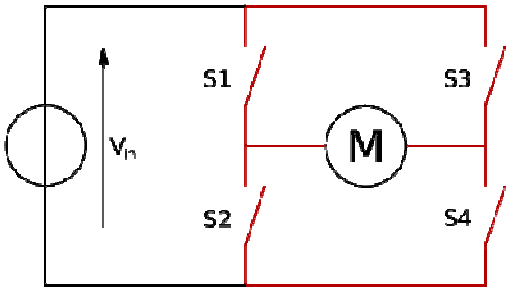

Figura A-20: Esquema de ponte H.

A-3.8 Sistema de Aquisição de Dados e Controle

A comunicação entre a placa controladora dos motores e o *notebook* de controle foi realizada através de interface de aquisição de dados e controle, desenvolvida pela equipe do CENTAURO – Centro de Referência em Automação e Robótica [70,71]. Esta interface realiza comunicação através de protocolo USB 2.0 (*Universal Serial Bus*) segundo o conceito de *plug-and-play* (ligar e usar), permitindo ao computador enviar os dados para controle de velocidade e direção dos motores. A Fig. A-21 mostra o circuito básico para o Sistema de Aquisição de Dados (SAD).

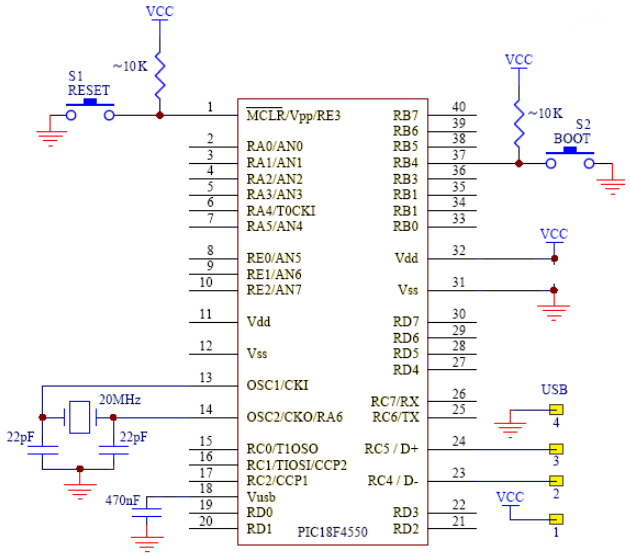

Figura A-21: Circuito eletrônico básico do SAD baseado em microcontrolador PIC 18F4550.

Este circuito tem como principal componente o microcontrolador PIC 18F4550, da empresa Microchip, que possui definido em sua estrutura de *hardware* o protocolo USB 2.0, devendo apenas ser configurado adequadamente. Este microcontrolador pode ser programado em linguagem C, utilizando o compilador gratuito C18, fornecido pela Microchip. O microcontrolador PIC 18F4550 possui 31 portas de I/O, onde todas podem ser configuradas para operar como entrada ou saída digital de dados. Pode-se ainda optar por utilizar até 13 conversores Analógicos Digitais, com 10 bits de resolução cada. Além da comunicação por USB com computador, pode-se realizar comunicação entre outros dispositivos através de SPI, I²C, UART ou USART.

A especificação USB foi criada em 1994 pelas empresas Intel, Compaq, Microsoft, Digital, IBM e Northern Telecom (vide Fig. B-1). Um dos principais motivos que levou à criação da tecnologia USB foi a necessidade de facilitar a conexão de variados dispositivos ao computador. É uma tecnologia de conectividade utilizada em computadores, celulares e outros dispositivos. O padrão USB 2.0 foi lançado em 2000, ele permite o uso da tecnologia pelos fabricantes sem o pagamento de licença, favorecendo a ampliação e barateamento da tecnologia. A versão USB 3.0 foi apresentada em 2008 e caracteriza-se principalmente por um aumento da velocidade de transferência, passando para 4,8 Gbps e full-duplex (transferência de dados bidirecional simultânea).

Como vantagens do padrão USB, podemos citar:

- Velocidade máxima de 480 Mbps (USB 2.0);
- *Plug-and-Play* (PnP) e *Hot Swapping* (dispositivos podem ser conectados e desconectados sem reiniciar o computador);

- Fornece energia para dispositivos de baixo consumo, evitando a necessidade de fontes externas (de 4,75V até 5,25V. 100mA ou 500mA);
- Permite o uso dos dispositivos sem a necessidade de instalação de *drivers* específicos;
- Aceita até 127 dispositivos conectados em uma única porta do computador;
- Utiliza 4 fios blindados, sendo 2 para alimentação (+5V e GND) e outro par para transmissão de dados (D+ e D-);
- Máximo de 5 hubs conectados em série (USB 2.0);
- Comprimento máximo para o cabo é de 5m (USB 2.0).

Para a comunicação USB é necessário um cabo blindado com quatro vias, dos quais dois destes, D+ e D-, formam um par entrelaçado para conduzir o sinal de dados diferencial. Este sinal das duas vias tem como referencia uma terceira, o GND. A quarta via é chamado de VBUS e conduz uma tensão nominal de 5V, que, respeitando o limite de até 500mA, pode ser usada para alimentar o dispositivo USB.

USB é uma arquitetura de comunicação. Os protocolos do barramento serial USB configuram automaticamente os dispositivos na inicialização, ou quando são conectados em tempo de execução, carregando os drivers necessários. O *host*, isto é, o computador ou o equipamento que recebe as conexões, usa um protocolo *master/subordinate* para se comunicar com os dispositivos USB. Todas as transferências de dados são iniciadas pelo *host controller*. Essa abordagem resolve o problema de colisão de pacotes, mas também evita que os dispositivos conectados realizem comunicação direta entre eles. Quando o *host* é iniciado, ele consulta todos os dispositivos conectados ao barramento e atribui um identificador para cada um deles. Esse processo é chamado *enumeration*. Os dispositivos também são enumerados no instante em que são ligados no barramento. No processo de enumeração o host faz a leitura do descritor do dispositivo. Esse descritor contém informações globais sobre o dispositivo, bem como as suas informações de configuração. A configuração define a funcionalidade e o comportamento de *Input/Output* (I/O) do dispositivo.

Um dispositivo USB deve ter uma ou mais configurações, cada uma definida pelo seu próprio descritor. Cada configuração tem uma ou mais interfaces, cada interface pode ser considerada um canal físico de comunicação. Cada interface pode ou não possuir endereços particulares (*endpoint*), cada *endpoint* pode ser um *data provider*, *data consumer*, ou ambos. Interfaces e *endpoints* também são definidos por seus descritores. Além disso um dispositivo USB pode conter *string descriptors* com informações adicionais sobre o fabricante, dispositivo e número serial (*vendor name*, *device name* e *serial number*). As transferências de dados ocorrem entre o *host* e um endereço particular (*endpoint*) no dispositivo USB, o canal de comunicação entre o *host* e o *endpoint* é chamado de *pipe*. Cada *endpoint* possui um canal próprio de comunicação (*pipe*). Um *pipe* pode ser unidirecional ou bidirecional, seu fluxo de dados é independente de todos os outros fluxos de dados (*pipes*) do dispositivo USB.

Há quatro tipos de transferência de dados que podem ser usados para comunicação entre o *host* e o dispositivo USB:

- ***Control*** *transfers*: É utilizado para transmissão de pequenos pacotes de dados usados para controle e configuração dos dispositivos, particularmente quando ele é conectado.
- ***Bulk*** *transfers*: É utilizado por dispositivos que lidam com grandes volumes de dados, como impressoras, scanners e adaptadores SCSI. Conta com recursos de detecção de erro para garantir a integridade das informações transmitidas.
- ***Interrupt*** *transfers*: É utilizado por dispositivos que transferem pouco volume de dados, como mouses e teclados. O *host controller* envia um sinal de interrupção a cada intervalo especificado.
- ***Isochronous*** *transfers*: É utilizado para transmissões contínuas em tempo real (*streams*). Não há recursos de detecção de erros. Dispositivos de áudio e vídeo geralmente usam esse tipo de transferência.

O *host* disponibiliza até 90% da largura de banda do barramento para os dispositivos do tipo *Interrupt* e *Isochronous*. As transferências do tipo *Control* e *Bulk* usam o que restar da largura de banda, ou seja, 10% no mínimo.

Os dispositivos USB são divididos em várias classes. Cada classe define comportamento e protocolo comuns para dispositivos com funções semelhantes (ex: *display*, *communication*, *audio*, *mass storage* e *human interface device*). A classe HID (*Human Interface Device*) consiste primariamente de dispositivos que são usados por humanos para controlar a operação do computador, tipicamente mouse, teclados, *joysticks*, e também dispositivos que talvez não precisem da interação humana mas fornecem dados de modo similar aos dispositivos da classe HID. A definição da classe HID inclui suporte para vários tipos de saídas direcionadas para o usuário final. Diversas plataformas de sistema operacional incluem suporte a HID em nível de *kernel*, dispensando a necessidade de instalação de um *driver* específico.

O SAD é um dispositivo USB do tipo HID (*Human Interface Device*). Esta facilidade, aliada ao seu baixo custo e desempenho robusto e confiável, faz com que sua utilização seja uma solução adequada para aplicações em robótica.

Uma das grandes vantagens em se utilizar o protocolo USB para fazer a comunicação entre os dispositivos periféricos e o computador embarcado é a possibilidade de ter em uma única porta até 128 dispositivos conectados, conforme mostrado na Fig. A-22. Isto simplifica uma eventual mudança ou melhoria no projeto do robô, pois caso se deseje adicionar nova funcionalidade, como um braço manipulador, por exemplo, é necessário apenas inserir nova configuração no *software* para reconhecer e controlar o dispositivo.

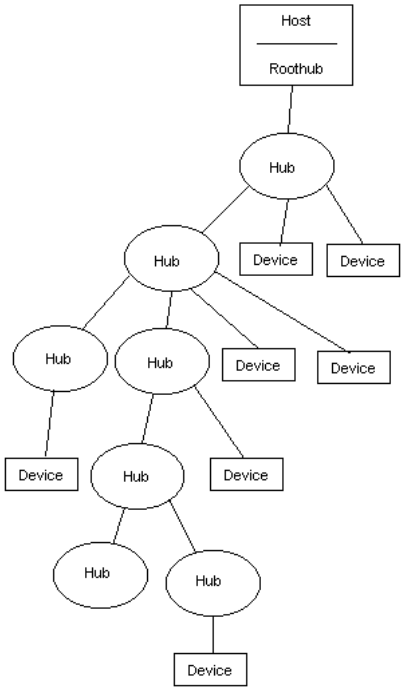

Figura A-22: Possibilidade de conectar diversos dispositivos em uma única porta USB.

A-3.9 Dados Técnicos do Sensor MS-Kinect

Na última versão do robô, a cabeça foi substituída pelo sensor Kinect, acrescentando ao robô funcionalidades como detecção de obstáculos utilizando a câmera de profundidade. As informações técnicas do sensor Kinect são listadas abaixo:

- Câmera de profundidade (depth): 640x480 (as 8 colunas mais a direita são sempre "sem dados", de modo a obter um tamanho de imagem efetivo de 632x480 em um buffer de 640x480).
- Câmera colorida: 640x480 Bayer pattern.
- Formato de saída para o fluxo de IR: 640x488.
- Quando o Kinect não pode ver o reflexo do IR ou não tem dados de profundidade para um pixel, ele retorna 2047 para o valor da profundidade.
- Rastreamento de esqueleto é realizada em um nível mais elevado do que os drivers e a biblioteca libfreenect é basicamente um *driver* de baixo nível dentro do OpenKinect. Os dados brutos são disponibilizadas e uma solução de rastreamento de esqueleto que leva os dados de libfreenect pode ser construída.
- Inductor
- - Color and depth sensing camera
- - Voice microphone array
- - Tilt sensor adjusts

- Visual range
- - Horizontal angle: 57 degrees
- - Vertical angle: 43 degrees
- - Kinect body rotation range: ± 27 degrees
- - Sensing depth range: 1.2 to 3.5 m
- Data Flow
- - Depth sensing camera: 320 x 240,16 bit, 30 fps
- - Color sensor camera: 640 x 480,32 bit, 30 fps
- - Audio: 16bit, 16 khz
- Depth = zero indica um ponto infinitamente longe.
- Depth = 2047 indica erro no pixel.

Com a utilização da câmera de profundidade, é possível considerar que cada ponto vizinho exerce uma "força repulsiva" proporcional ao inverso do quadrado da distância, conforme descrito na Fig. A-23.

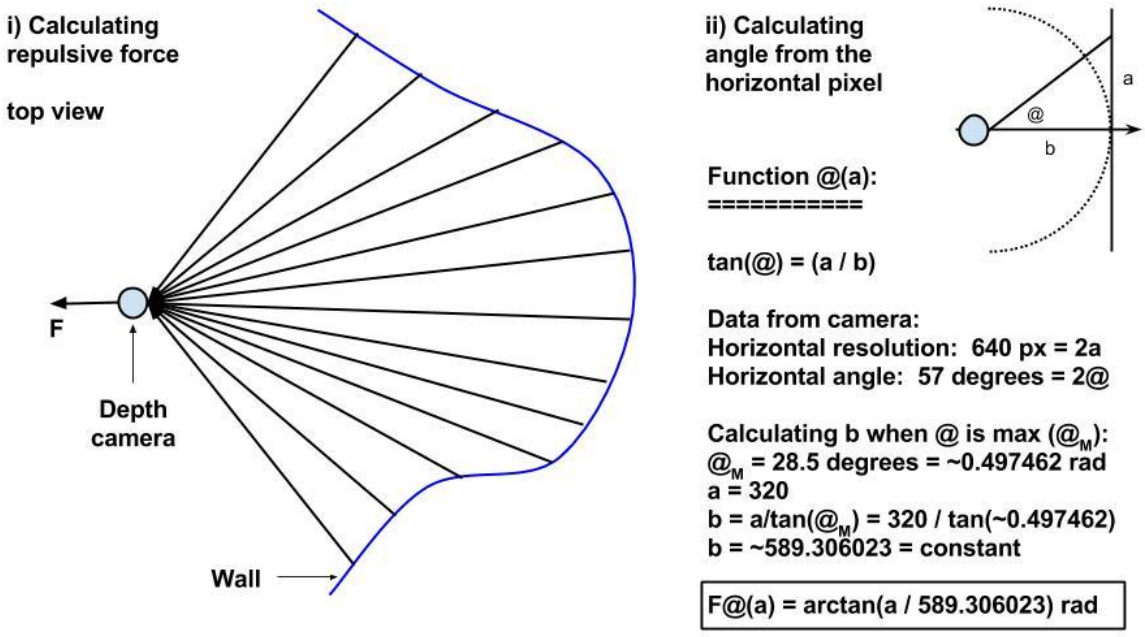

Figura A-23: *i*) Cálculo da "Força Repulsiva" total. *ii*) Cálculo do ângulo em função do pixel horizontal.

Foram realizadas medidas experimentais para verificar a relação entre a unidade de distância informada pela API OpenKinect [24] e a distância real (aproximada) em centímetros. O resultado é apresentado na Tabela A-3.

Depth data from OpenKinect API	Centimetros
945	250
915	200
842	150
824	140
805	130
782	120
756	110
726	100
686	90
636	80
564	70
464	60
386	50
381	45
2047 (erro)	Sem leitura (Não foi possível ver o reflexo ou não tem dados de profundidadepara o pixel)

Tabela A-3: Relação entre a unidade de distância informada pela API OpenKinect e a distância real aproximada em centímetros.

A-4 Resultados e Discussão

O robô descrito neste trabalho foi fabricado e montado, resultando num protótipo rígido e robusto. Seus elementos físicos, com exceção dos motores elétricos, foram superdimensionados, e devem apresentar uma longa vida útil. A Fig. A-24 mostra uma fotografia do robô construído, com todas as suas partes, incluindo o computador de controle e câmeras de monitoramento, estando pronto para entrar em operação.

Figura A-24: Fotografia do robô desenvolvido, com todas as suas partes (penúltima versão).

Para a finalização deste protótipo, falta ainda incorporar o revestimento de proteção externo e os seguintes sensores: sensor de distância baseado em laser, sensor de presença humana por infravermelho, *encoders*, câmera de visão noturna, sensores de temperatura. O projeto mecânico e eletrônico para este robô provou ser robusto, tendo o robô sido desmontado e

transportado em diversas ocasiões, não tendo apresentado problemas em seu funcionamento quando montado novamente. No entanto, algumas observações sobre seu funcionamento devem ser ressaltadas, e serão comentadas a seguir.

A-4.1 Circuitos de acionamento dos motores e SAD

As duas Placas de Circuito Impresso (PCI), de acionamento dos motores e Sistema de Aquisição de Dados, foram fabricadas no Instituto de Telemática (ITTI / IFCE), através de máquina operatriz via CNC. A alta qualidade da fabricação refletiu-se na confiabilidade e facilidade na montagem dos circuitos, permitindo comunicação rápida e direta entre o *notebook* de controle e os motores elétricos. As Figuras A-25 e A-26 mostram, respectivamente, as PCIs para o circuito de acionamento e do SAD.

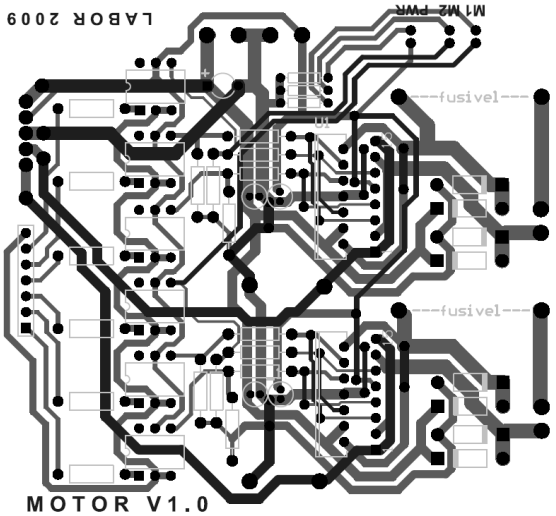

Figura A-25: PCI do circuito eletrônico de acionamento dos motores.

Observou-se, no entanto, que o L298 apresenta queda de tensão de aproximadamente 2 V, quando o robô está em operação. Estando a bateria totalmente carregada, a mesma apresenta tensão elétrica de 13 V. Logo, esta queda de tensão é da ordem de aproximadamente 15% da tensão de entrada, interferindo diretamente no desempenho do motor e em sua potência mecânica disponível.

O SAD demonstrou funcionamento satisfatório, sendo testado em sistemas operacionais Microsoft Windows e diversas distribuições de linux, como Ubuntu e Fedora, realizando comunicação confiável e segura. Foram realizados testes onde o programa embarcado no microcontrolador, *firmware,* enviava continuamente dados sequenciados ao computador, e se verificou se existia perda de dados ou erro nos pacotes de transmissão. O resultado destes testes apresentou que a velocidade de envio de dados do PIC 18F4550 foi aproximadamente 20 vezes superior à velocidade de recebimento de dados na porta USB do computador de controle. Deste modo, foi necessário realizar funções de sincronização de dados entre o *firmware* e o programa

de controle, para garantir integridade da informação transmitida e recebida.

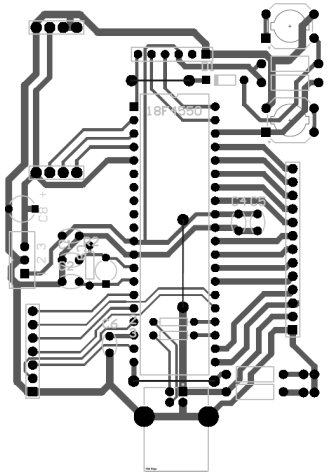

Figura A-26: PCI do Sistema de Aquisição de Dados.

A-4.2 Análise de Energia

Foram realizadas medições para as condições reais de operação do robô móvel em desenvolvimento. Sua velocidade, quando está se locomovendo em linha reta, foi obtida medindo-se o tempo em que percorre uma distância marcada no chão do Laboratório de Robótica – LABOR do CENTAURO. Foram feitas duas marcas no chão com distância de 4 *m* entre elas, e com o auxílio de um cronômetro, mediu-se este tempo. O tempo medido foi de aproximadamente *t = 20 s*, resultando em uma velocidade de aproximadamente *0,2 m/s*. Considerando o raio da roda de 0,127 mm e convertendo esta velocidade para rotações por minuto, tem-se $\omega = 5\ rpm$. Observando a Tabela A-1, a potência mecânica para cada roda correspondente a esta rotação é de aproximadamente $P_{mecânica} = 0{,}92\ W$.

Considerando a queda de tensão provocada pelo CI L298, verificou-se que os motores instalados recebem do circuito eletrônico de controle uma tensão elétrica de aproximadamente *11 V*, que foi medida usando-se um voltímetro embarcado no robô. Mediu-se também as correntes elétricas que circulavam em cada motor, obtendo-se um valor de aproximadamente *I = 0,8 A*. A potência elétrica foi calculada e é aproximadamente:

$$P_{\text{elétrica}} = V_{\text{operação}} \times I_{medida} \tag{18}$$
$$P_{\text{elétrica}} = 11 \times 0{,}8$$
$$P_{\text{elétrica}} = 8{,}8\ W$$

Comparando as potências mecânicas e elétricas, obtém-se a eficiência global do robô para o sistema de locomoção:

$$\eta = \frac{P_{\text{mecânica}}}{P_{\text{elétrica}}} \tag{3}$$

$$\eta = \frac{0,92}{8,8}$$

$$\eta = 0,105$$

Este valor encontrado diz respeito à eficiência para cada motor, mas representa também a eficiência global do sistema de locomoção, pois para a utilização de dois motores tem-se que a potência mecânica e elétrica é dobrada, mantendo-se o mesmo valor de eficiência. Observa-se que esta baixa eficiência de 10,5% deve-se principalmente ao ponto de operação dos motores, que para esta velocidade, conforme pode ser visto na Tabela A-1, é baixa. Nota-se que a eficiência da Tabela A-1 para este ponto é de 9%. Esta diferença se deu devido ao fato de que a potência elétrica considerada nesta seção ter sido medida experimentalmente, sendo ligeiramente menor e aumentando suavemente a eficiência, quando comparada com os dados da Tabela A-1.

Este ponto de operação se deve a fatores como desalinhamento entre as polias das correias, deslizamento das rodas e perdas na caixa de redução, que contribuem para que o robô tenha um alto consumo de energia em relação ao pequeno trabalho realizado. Para otimizar esta relação, seria necessário escolher motores mais potentes, de modo a permitir ao robô operar em uma região de melhor eficiência, e minimizar estas perdas.

A-4.3 Fabricação

A fabricação mecânica do chassi foi realizada através de corte a jato de água por Comando Numérico Computadorizado (CNC), em empresa especializada, MAEMFE. Isto se deu devido ao fato de que o formato escolhido para o chassi possuía muitas curvas e sua fabricação em oficinas mecânicas tradicionais seria dificultada. A Fig. A-27 mostra uma vista de topo do conjunto do chassi com todas as partes detalhadas.

Todas as outras peças foram fabricadas e usinadas em duas oficinas mecânicas dentro da Universidade Federal do Ceará – a oficina do Departamento de Engenharia Mecânica e Produção e a oficina do Departamento de Física – pois achamos que sua geometria não requeria fabricação computadorizada, que resultaria em aumento de custos. Estas peças usinadas são: suportes dos motores, suporte da roda de apoio, eixos das rodas, abertura de rosca nas polias sincronizadoras e furação e abertura de rosca no chassi para fixação das peças.

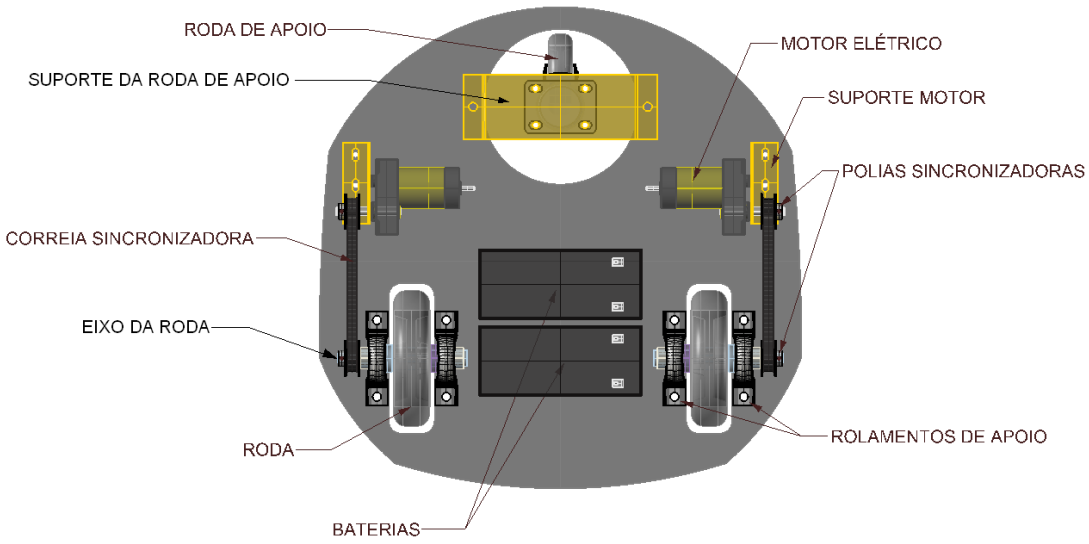

Figura A-27: Vista de topo da base do robô, com detalhe de seus elementos.

Cabe salientar que no momento quando os dois motores recebiam tensão elétrica igual, o robô deslocava-se suavemente para a esquerda, quando deveria ter movimentação puramente retilínea. Isto pode ter ocorrido devido a pequenos desalinhamentos na furação do chassi, para fixação dos rolamentos de apoio das rodas, deixando as rodas com leve inclinação uma em relação à outra. Esta curvatura indesejada pode ter ocorrido também em decorrência da roda esquerda estar girando um pouco mais lentamente do que a roda direita, revelando diferença no alinhamento das correias sincronizadoras, ou diferença entre as velocidades dos motores.

Caso a razão desta curvatura esteja no desalinhamento entre os furos para receber os mancais, uma possível solução estaria em realizar esta furação também por Controle Numérico Computadorizado, para garantir alinhamento preciso e exato das rodas. Este desalinhamento, além alterar a rota prevista para o robô, prejudicando sua odometria e seu sistema de navegação, resulta em desgaste prematuro das rodas, e portanto deve ser evitado.

Caso a curvatura se dê devido às diferenças de alinhamento e tensão nas correias sincronizadoras, uma possível solução seria realizar a furação dos suportes dos motores, que agem também como elemento tensionador das correias, através de CNC. Verificamos que existe diferença entre os alinhamentos destes suportes, conforme mostra a Fig. A-28, e que esta diferença resulta em esforço adicional nas correias, exercendo frenagem nas rodas. Este problema construtivo, além de resultar em curvatura indesejada do robô e ineficiência energética para o sistema de locomoção, acarreta também desgaste e diminuição da vida útil das correias sincronizadoras, portanto devendo ser evitado.

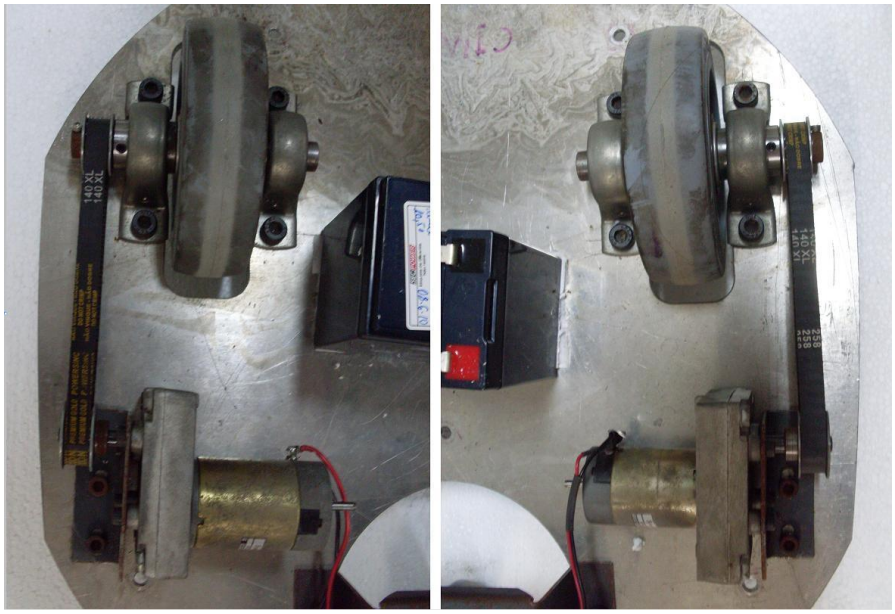

Figura A-28: Fotografia da montagem dos suportes dos motores, revelando desalinhamento entre as correias sincronizadoras.

Caso a razão desta curvatura encontre-se nas diferenças entre as velocidades máximas dos motores, uma possível solução encontra-se na utilização de sensores de velocidade angular ópticos, *encoders*, que permitem a estimação de velocidade, aceleração e posição angular de cada motor individualmente, e deste modo, compensar as diferenças entre os motores através de controle inteligente automático. Desta maneira, além de um funcionamento do sistema de locomoção bastante eficiente, haverá sempre espaço para compensar e equiparar as velocidades entre as rodas.

A-5 Análise de Desempenho do Robô

A rigidez das partes fixas e móveis do robô pôde ser verificada, confirmando que o trabalho desenvolvido possui características de durabilidade e confiabilidade, satisfazendo os objetivos iniciais. A utilização de métodos de fabricação e usinagem através de CNC pode ser bastante vantajoso e eficiente, apesar do custo mais elevado, sendo recomendando para versões e projetos futuros.

A escolha dos motores é um dos pontos críticos, e que neste trabalho, não foi adequada. Em consequência deste fato, o robô se locomoveu lentamente em sua velocidade máxima. Com a continuidade deste projeto, a seleção de motores elétricos com maior potência permitirá ao robô locomover-se com maior velocidade e de modo mais eficiente. Além disso, o Circuito Integrado L298 não teve desempenho adequado para o projeto, pois apresenta queda de tensão elevada, em torno de 15%. Com a continuidade deste projeto, substituiremos este CI por outro mais eficiente, mas considerando que o circuito eletrônico de controle dependerá dos motores escolhidos,

devido à corrente que o motor poderá solicitar. Uma possibilidade discutida pela equipe do LABOR, uma vez definidos os motores, é substituir o CI atualmente empregado pelo circuito integrado LMD18200, que é semelhante ao L298, porém é mais eficiente com relação aos problemas citados. Outra possibilidade complementar é a construção da ponte em H diretamente com o uso de transistores do tipo MOSFET.