



UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE
COMPUTAÇÃO



DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO
E AUTOMAÇÃO INDUSTRIAL

Uma Plataforma para Integração de Redes de Sensores sem Fio a Aplicações de Robótica Móvel

Autor: Ricardo da Silva Souza

Orientador: Prof. Dr. Eleri Cardozo

Co-Orientador(a): Dr(a). Eliane Gomes Guimarães

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Comissão Examinadora:

Prof. Dr. Eleri Cardozo (presidente) — DCA/FEEC/UNICAMP

Prof. Dr. Omar Carvalho Branquinho — PUC-Campinas

Prof. Dr. Romis Ribeiro de Faissol Attux — DCA/FEEC/UNICAMP

Campinas, SP
2011

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

So89p Souza, Ricardo da Silva
Uma Plataforma para integração de redes de sensores sem fio a aplicações de robótica móvel / Ricardo da Silva Souza. –Campinas, SP:[s.n.], 2011.

Orientadores: Eleri Cardozo, Eliane Gomes
Guimarães.

Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de sensores sem fio. 2. Robótica. I. Cardozo, Eleri. II Guimarães, Eliane Gomes. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título


Título em Inglês: A platform for integrating wireless sensor networks to mobile robots applications
Palavras-chave em Inglês: Wireless Sensor Networks, Robotics
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca Examinadora: Omar de Carvalho Branquinho, Romis Ribeiro de Faissol Attux
Data da defesa: 08/07/2011
Programa de Pós Graduação: Engenharia Elétrica


COMISSÃO JULGADORA - TESE DE MESTRADO


Candidato: Ricardo da Silva Souza

Data da Defesa: 8 de julho de 2011

Título da Tese: "Uma Plataforma para Integração de Redes de Sensores sem Fio a Aplicações de Robótica Móvel"

Prof. Dr. Eleri Cardozo (Presidente):  _____

Prof. Dr. Omar Carvalho Branquinho:  _____

Prof. Dr. Romis Ribeiro de Faissol Attux:  _____

Resumo

Com a evolução da robótica móvel, é cada vez mais frequente soluções que buscam realizar o controle de robôs de forma distribuída. Este tipo de controle requer a existência de redes de comunicação sem fio conectando o robô a outros dispositivos no ambiente. Tecnologias como *Wi-Fi* e *Bluetooth* são exemplos de redes utilizadas para este fim. Redes Ad-hoc, como redes de sensores sem fio, podem ser rapidamente instaladas em um ambiente oferecendo cobertura em uma ampla região utilizando um grande número de nós. Além de prover comunicação para os robôs, uma rede de sensores pode também auxiliar na sua navegação, localização e ainda aumentar sua capacidade de sensoriamento. Esta dissertação apresenta uma plataforma que permite a integração de redes de sensores em aplicações de robótica móvel. Com esta integração é possível realizar o controle e a comunicação com robôs móveis por meio de redes de sensores sem fio.

Palavras-chave: Redes de Sensores, Robótica Móvel.

Abstract

A trend in today's mobile robotics is to perform the control of mobile robots in a distributed fashion. This control scheme requires a wireless communication network connecting the robot to the other existing devices in the environment. WiFi and Bluetooth are examples of such networks. Ad-hoc networks such as wireless sensor networks (WSN) can be deployed instantly and offer a wide coverage by employing a large number of communicating nodes. In addition to provide a communication link to the robots, a WSN can aid the robot's navigation, localization, and also enhance its sensory capabilities. This dissertation presents a platform that allows the integration of WSNs into mobile robotics applications. With this integration it is possible to control and communicate mobile robots through WSNs.

Keywords: Wireless Sensor Network, Networked Robotics, Mobile Robots.

Agradecimentos

Aos meus pais, Francisco e Rosana, pelo exemplo de vida, apoio incondicional; e por possibilitarem que eu perseguisse todos os meus objetivos.

Ao meu professor, mestre e orientador Eleri Cardozo, meus agradecimentos pela oportunidade de realizar este projeto; pela amigável e sábia orientação, e pela compreensão e paciência nos momentos difíceis.

À minha co-orientadora Eliane Gomes Guimarães, por todo o incentivo, dedicação, e atenção no decorrer do mestrado.

À Leonora, minha esposa, pelo amor, força e confiança depositados em mim à todo instante, e sem os quais este trabalho não seria possível.

Aos meus padrinhos, madrinhas e amigos de longa data, André, Danilo, Natassia, Carol, Camila, Emídio, Marcela e tantos outros, pela amizade, companheirismo e apoio nos bons e maus momentos.

Aos companheiros de casa, Eduardo e Johannes, que estiveram ao meu lado durante todo o mestrado.

Aos amigos do LCA, Lucio, Diego, Fernando, Fábio, Guilherme e Leonardo, por me ajudarem a realizar este trabalho através de sugestões e críticas; além, é claro, dos momentos de descontração.

À professora Fátima Vieira e aos amigos do LIHM, Ademar e Daniel, pela ajuda e incentivo nos primeiros momentos da vida acadêmica.

À CAPES, pelo apoio financeiro.

Sumário

Lista de Figuras	xiii
Lista de Tabelas	xvii
Glossário	xix
Trabalhos Publicados Pelo Autor	xxiii
1 Introdução	1
2 Redes de Sensores sem Fio	5
2.1 Sistemas Operacionais	6
2.1.1 TinyOS	8
2.1.2 SOS	15
2.1.3 Contiki	17
2.1.4 LiteOS	18
2.1.5 Linux	20
2.2 <i>Hardware</i> para Redes de Sensores	21
2.2.1 Intel Mote2	24
2.3 Aplicações	30
2.3.1 Robótica Móvel	32

3	Plataforma REAL	45
4	Plataforma para Rede de Sensores	55
4.1	Arquitetura da Plataforma	56
4.2	Protocolo de Comunicação	57
4.2.1	TinyAPI	63
4.3	<i>Gateways</i>	63
4.4	Sensoriamento do Ambiente	66
4.5	Comunicação Robô-Servidor	69
5	Simulações e Experimentos	73
5.1	Simulações de Escalabilidade	74
5.2	Experimentos	76
5.2.1	Avaliação de Latência	77
5.2.2	Validação da Comunicação - Campos Potenciais	78
5.2.3	Algoritmo de Campos Potenciais Estendido	79
6	Serviço para Navegação Guiada de Robôs Móveis	85
6.1	Arquitetura de Rede	86
6.2	Definição do Serviço na Plataforma	89
6.3	Avaliação do Serviço	90
6.3.1	Simulações	90
6.3.2	Experimentos	91
6.4	Aplicações do Serviço	93
7	Considerações Finais	97
7.1	Contribuições	97
7.2	Trabalhos Futuros	98

Referências Bibliográficas

99

Lista de Figuras

1.1	Publicações do IEEE sobre redes de sensores no últimos 20 anos [1].	2
2.1	Modelo baseado em <i>threads</i>	7
2.2	Modelo baseado em eventos.	7
2.3	Componente TinyOS que provê uma interface. [2]	9
2.4	(a) Componente TinyOS que usa uma interface. (b) Configuração TinyOS que conecta os componentes. [2]	10
2.5	Componente <i>Timer</i> do TinyOS. [3]	10
2.6	Arquitetura de componentes do TinyOS. [4]	11
2.7	HAA -Arquitetura para suporte a abstração de <i>hardware</i> do TinyOS [5].	12
2.8	Decomposição horizontal [5].	13
2.9	Mensagem CC2420 padrão [6]	14
2.10	Modelo de comunicação do SOS nos dois formatos [7].	16
2.11	Esquema do SO Contiki: programas são divididos em duas classes: serviços principais e programas carregados dinamicamente [8].	18
2.12	Arquitetura do LiteOS [9].	19
2.13	Organização de uma instalação do Linux na memória de um Imote2.	20
2.14	Plataformas <i>Low-End</i> : (a) MicaZ, (b) TelosB, (c) Tmote, (d) EYES.	23
2.15	Imote2 hardware.	24
2.16	Visão detalhada do Imote2. (a) Superior, (b) Inferior.	25

2.17	Arquitetura básica da família Imote.	25
2.18	Módulos adicionais do Imote2: (a) ITS400, (b) IMB2400, (c) IIB2400.	26
2.19	Subsistema de sensoriamento da arquitetura do Imote2.	27
2.20	Pilha de protocolos do <i>Zigbee</i> e do IEEE 802.15.4 [10].	28
2.21	Estruturas topológicas de uma rede IEEE 802.15.4. (a) Estrela, (b) <i>Mesh</i> P2P, (c) <i>Árvore de clusters</i>	30
2.22	Relação entre a corrente e a potência de saída. [2]	31
2.23	Resultados da localização utilizando (a) 4 e (b) 6 nós [11]	34
2.24	Localização em uma rede de sensores: (a) utilizando visão [12] (b) utilizando beacons e RSSI [13]	35
2.25	Cenário onde robô móvel realiza a coleta e entrega de dados entre grupos de sensores e servidores.	36
2.26	Cenário da navegação com vizinhanças demarcadas [14].	37
2.27	Cenário interno: localização do robô feita por triangulação [15].	37
2.28	Traçado de rotas seguras de emergência [16].	38
2.29	Atendimento a chamadas de emergência ou resgate [17].	39
2.30	Localização e Navegação utilizando o efeito <i>Doppler</i> [18].	40
2.31	Dois mapas locais sendo unidos através dos <i>relay-points</i> [19].	41
2.32	(a) Visão geral do <i>testbed</i> (b) Modelo de interoperabilidade através do <i>Player</i> [20]	42
2.33	Cenário da Plataforma AWARE [21].	43
2.34	Plataforma AWARE realizando <i>tracking</i> de robô móvel.	43
3.1	Pacotes principais da plataforma REAL.	45
3.2	Interface <i>Web</i> de teleoperação.	47
3.3	Componentes da plataforma robótica.	49
3.4	Cenário do WebLab.	52
3.5	Arquitetura do REALabs inter-domínios.	52

3.6	Resultado de uma aplicação de mapeamento executada remotamente através da infraestrutura do <i>WebLab</i>	53
4.1	Arquitetura da plataforma para suporte a rede de sensores.	56
4.2	Arquitetura suporte ao <i>Range Service</i> estendido pela comunicação Wi-Fi.	57
4.3	Estrutura de uma mensagem de requisição de operação.	58
4.4	Estrutura de uma mensagem de resposta genérica.	59
4.5	Definição da mensagem de resposta do <i>Info Service</i>	59
4.6	Estrutura de uma mensagem de leitura de sensor.	61
4.7	Estrutura de uma mensagem de controle.	62
4.8	Definição da mensagem para o serviço de ações.	62
4.9	Comunicação interna do robô em diferentes cenários. (a) Comunicação pela rede de sensores. (b) Comunicação pela rede Wi-Fi.	64
4.10	Modelo para transformação entre referências [22].	65
4.11	Cenário de leitura da região ao redor do robô.	68
4.12	Diagrama de sequência da operação de leitura de uma região.	68
4.13	Mensagem de leitura de sensores da rede.	69
4.14	Arquitetura de comunicação da plataforma.	70
5.1	Simulador TOSSIM.	74
5.2	Simulador MobileSim com uma mapa de uma sala.	74
5.3	Cenário de simulações realizadas.	75
5.4	Resultados de simulação para diferentes configurações.	76
5.5	(a) Motes utilizados nos experimentos. (b) Robô com mote conectado.	77
5.6	Resultados comparativos entre Wi-Fi, <i>hop</i> único e <i>multihop</i>	77
5.7	Resultados da execução do algoritmo de campos potenciais utilizando: (a) Wi-Fi e (b) a rede de sensores.	79
5.8	Comparação entre redes de sensores e Wi-Fi.	80

5.9	Campos potenciais utilizando diferentes luminosidades: (a) pouca luz (b) penumbra (c) e completo escuro. S representa a posição do sensor.	81
5.10	Simulação para calibração dos campos potenciais. (a) campos potenciais tradicional (b) campos potenciais com obstáculos reais e ambientais.	82
5.11	Resultados da navegação do robô para duas configurações de luminosidade: (a) penumbra (b) escuro.	83
5.12	(a) Navegação utilizando campos potenciais tradicionais. (b) Navegação utilizando obstáculos físicos e “ambientais”.	84
6.1	Arquitetura de rede para implementação da solução proposta.	87
6.2	Mensagem de definição de rota da camada de navegação.	87
6.3	Diagrama de sequência para a formação de rotas e processo de <i>handover</i>	89
6.4	Resultados de simulações do algoritmo de traçado de rotas.	91
6.5	Cenário simples de traçado de rota. O LED azul indica o nó sensor alvo, o LED vermelho indica que o nó pertence a rota estabelecida e o LED verde indica, nos nós da rede, o nó alvo atual e no nó do robô que a navegação pode ser inicializada.	92
6.6	Diferentes cenários de testes com diferentes topologias.	92
6.7	Teste da solução de recálculo de rotas utilizando um evento temporizado. (a) Formação da rota inicial. (b) Nó que “detectou” o evento realiza um <i>broadcast</i> desta informação. (c) Nova rota calculada.	93
6.8	Teste da solução de recálculo de rotas utilizando sensoricamente real. (a) Formação da rota inicial. (b) Nó coberto detecta o evento e realiza um <i>broadcast</i> desta informação. (c) Nova rota calculada.	94
6.9	Cenários para navegação utilizando a rede de sensores através do algoritmo baseado em visão.	95
6.10	Cenários para navegação utilizando a rede de sensores através do algoritmo baseado em RSSI.	96

Lista de Tabelas

2.1	<i>Mote Hardware.</i>	24
2.2	Características dos sensores do ITS400.	26
2.3	Especificação de Operação do ITS400.	26
2.4	Características da alimentação IBB2400CA.	28
2.5	Configurações de potência de saída do TI/Chipcon CC2420. [23]	30
2.6	<i>Transceivers</i> utilizados em redes de sensores sem fio [24].	31
3.1	Serviços suportados pela plataforma.	51
5.1	Tempo e número médio de saltos nas diferentes configurações.	75
5.2	Tempos médios e desvio padrão de cada cenário de teste.	78
6.1	Operações do serviço de navegação.	90
6.2	Tempo médios de RTT para cada cenário	91

Glossário

API - *Application Programming Interface*

ASK - *Amplitude-shift Keying*

AVC - *Acidente Vascular Cerebral*

AWARE - *Autonomous self-deploying and operation of Wireless sensor-actuator networks cooperating with Aerial Objects*

CGI - *Common Gateway Interface*

CTI - *Centro de Tecnologia da Informação Renato Archer*

CTP - *Collection Tree Protocol*

EEPROM - *Electrically Erasable Programmable Read-Only Memory*

ETX - *Expected Transmissions*

EYES - *Energy-efficient Sensor Networks*

FEEC - *Faculdade de Engenharia Elétrica e da Computação*

FFD - *Full Function Device*

FHSS - *Frequency-hopping Spread Spectrum*

FIFO - *First In First Out*

FSK - *Frequency-shift Keying*

FTDI - *Future Technology Devices International*

GFSK - *Gaussian Frequency-shift Keying*

HAA - *Hardware Abstraction Architecture*

HAL - *Hardware Adaptation layer*

HIL - *Hardware Interface Layer*

HPL - *Hardware Presentation Layer*

HTTP - *Hipertext Transfer Protocol*

HTTPS - *Hipertext Transfer Protocol Secure*

IEEE - *Institute of Eletrical and Electronics Engineers*

JFFS2 - *Journalling Flash File System version 2*

LR-WPAN - *Low-Rate Wireless Private Area Network*

MAC - *Media Control Accesss*

NAT - *Network Address Translation*

O-QPSK - *Offset Quadrature Phase-shift Keying*

OOK - *On-Off Keying*

OSI - *Open Systems Interconnection*

P2P - *Peer-to-Peer*

RAM - *Random Access Memory*

REAL - *Remotely Accessible Laboratory*

REST - *Representational State Transfer*

RF - *Rádio Frequência*

RFD - *Reduced Function Device*

RFID - *Radio-Frequency Identification*

RSSI - *Radio Signal Strength Indicator*

SAML - *Security Assertion Markup Language*

SIP - *Session Initiation Protocol*

SO - *Sistema Operacional*

SOAP - *Simple Object Access Protocol*

TI - *Texas Instruments*

TinyOS - *Tiny Microthreading Operating System*

TOSSIM - *TinyOS Simulator*

TTL - *Time-To-Live*

UART - *Universal Asynchronous Receiver/Transmitter*

UAV - *Unmanned aerial vehicle*

URL - *Uniform Resource locator*

VHF - *Vector Histogram Field*

WPAN - *Wireless Private Access Netowrk*

WSN - *Wireless Sensor Network*

XML - *eXtensible Markup Language*

Trabalhos Publicados Pelo Autor

1. Souza, R. S.; Agostinho, L. R.; Teixeira, F.; Rodrigues, D.; Olivi, L.; Guimarães, E.; Cardozo. E.. “Control of Mobile Robots Through Wireless Sensor Networks”. *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC’11)*, Campo Grande, MS, pg. 805-818, Maio, 2011.
2. Rocha, L. A.; Olivi, L.; Paolieri, F.; Feliciano, G.; Souza, R. S.; Pinho, F.; Teixeira, F.; Rodrigues, D.; Guimarães, E.; Cardozo, E.. “Computer Communications and Networks - Advances in Educational Robotics in Cloud with Qualitative Scheduling in WorkFlows”, Springer, University of Derby, 2011.
3. Cardozo, E.; Guimarães, E.; Agostinho, L. R.; Souza, R. S.; Paolieri, F.; Pinho, F.. “A Platform for Networked Robotics”. *IEEE/RSJ International Conference On Intelligent Robots and Systems (IROS’10)*, Taipei, Setembro, 2010.
4. Agostinho, L. R.; Cardozo, E.; Guimarães, E.; Souza, R. S.; Paolieri, F.. “Uma Infraestrutura para Experimentos Robóticos Bio-inspirados em Grades Colaborativas“. *VIII Workshop em Clouds, Grids e Aplicações - XXVIII SBRC’10*, (WCGA), Gramado, RS, Junho, 2010.

Capítulo 1

Introdução

O rápido avanço da tecnologia na última década tornou a mobilidade uma realidade. A cada dia surgem dispositivos móveis e portáteis com capacidades de armazenamento e processamento cada vez maiores. Paralelamente, tecnologias de comunicação sem fio também são desenvolvidas buscando otimizar a relação dados transmitidos por unidade de energia. Dentro deste cenário, as redes de sensores sem fio foram uma das áreas que mais se beneficiaram.

Redes de sensores sem fio (*Wireless Sensor Network* - WSN) são formadas por dispositivos, tradicionalmente conhecidos como nós sensores (ou *moten*), compactos, autônomos, com capacidades de processamento e armazenamento limitadas, capazes de sensoriar variadas características do ambiente, além de se comunicarem em curtas distâncias via rádio frequência (RF), usualmente de maneira *ad-hoc*.

Aliando a evolução ao barateamento da tecnologia, cada vez mais surgem aplicações, em diversas áreas, que fazem uso de redes de sensores, monitoramento de ambientes, rastreamento de animais, automação industrial e gerenciamento de utilidades urbanas são exemplos destas aplicações [1].

É possível perceber o enorme crescimento do interesse na área não apenas observando o surgimento de novos dispositivos e plataformas no mercado, mas também analisando o crescimento das pesquisas neste campo, por exemplo, no banco de publicações do IEEE. A Figura 1.1 ilustra o aumento de publicações nos últimos 10 anos. É notório o grande crescimento da atenção voltada à

esta tecnologia na última década, assim como a indicação de uma estabilização nos próximos anos.

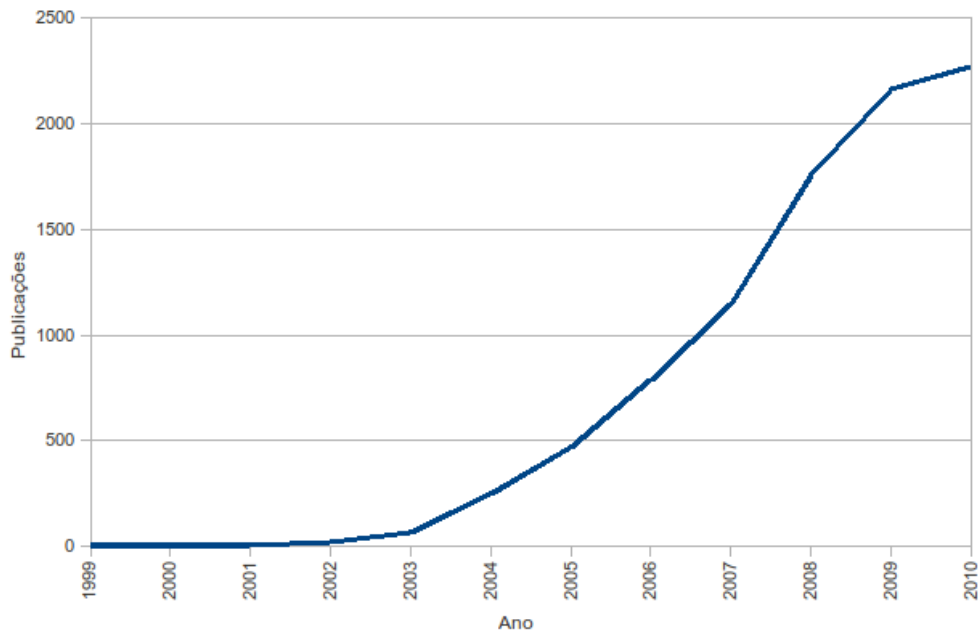


Fig. 1.1: Publicações do IEEE sobre redes de sensores no últimos 20 anos [1].

Dentro da enorme variedade de áreas que fazem uso de redes de sensores sem fio, encontra-se a robótica móvel. O uso em conjunto das duas tecnologias torna-se bastante interessante uma vez que uma pode ser vista como complementar à outra. Robôs móveis podem ser equipados com nós e serem interpretados como nós móveis dentro de uma rede de sensores, ou ainda serem utilizados como um canal de comunicação entre redes distantes. Por outro lado, uma rede de sensores pode ser vista como uma extensão das capacidades de sensoriamento e comunicação de robôs móveis. É possível encontrar vários trabalhos na literatura que exploram ambas as características citadas, porém um ponto em comum entre a grande maioria destes trabalhos é o desenvolvimento de soluções específicas para uma determinada aplicação.

A principal contribuição deste trabalho é a definição de uma plataforma que possibilita o desenvolvimento de aplicações de robótica móvel, que utilizem redes de sensores sem fio, mais rapidamente e através de uma interface padronizada, além de tornar possível o uso da rede de sensores

como uma rede de comunicação para o robô. Para tal, foi tomada como ponto de partida a plataforma REAL (*Remotely Accessible Laboratory*), desenvolvida pelo nosso grupo.

A plataforma REAL possibilita o acesso a robôs móveis através da internet, ou de redes privadas, utilizando protocolos abertos e bem difundidos. O trabalho proposto nesta dissertação consiste de uma extensão desta plataforma de forma a incluir funcionalidades da rede de sensores. Esta extensão foi implementada em duas etapas: a primeira consiste em possibilitar o uso da rede de sensores como uma rede de comunicação entre robôs móveis e o servidor da plataforma; a segunda é a integração das capacidades de sensoriamento da rede à plataforma. Testes foram realizados buscando apresentar o desempenho deste novo modelo de comunicação, utilizando uma rede de sensores, frente a infraestrutura existente, utilizando uma rede Wi-Fi.

Além da definição desta plataforma, também foi implementada uma aplicação para auxiliar a navegação autônoma de robôs móveis em ambientes monitorados por uma rede de sensores. A rede é utilizada para definir rotas navegáveis, de acordo com parâmetros pré-estabelecidos, para o robô atingir uma determinada posição alvo. Para interagir com esta aplicação, um novo serviço também foi proposto para a plataforma.

Esta dissertação está dividida em sete capítulos organizados da seguinte forma: o Capítulo 2 apresenta o estado da arte das redes de sensores sem fio, além de uma revisão de trabalhos correlatos; o Capítulo 3 introduz a plataforma REAL, utilizada como ponto de partida para este trabalho. O Capítulo 4 apresenta a extensão proposta para integrar redes de sensores a plataforma robótica, enquanto o Capítulo 5 contém as validações para as soluções propostas. O Capítulo 6 aborda a implementação do auxílio à navegação de robôs móveis e, finalmente, o Capítulo 7 apresenta as conclusões e propostas de trabalhos futuros.

Nesta dissertação, foi adotada a convenção de manter alguns termos técnicos em inglês buscando manter uma fidelidade semântica com o seu significado dentro da área.

Capítulo 2

Redes de Sensores sem Fio

Redes de sensores sem fio (WSN) são sistemas formados por pequenos dispositivos capazes de formar uma rede ad-hoc [25], ou seja, de se comunicar entre si, em curtas distâncias, sem a necessidade de uma infra-estrutura fixa ou controle central. Estes dispositivos aliam esta capacidade de comunicação à habilidade de monitorar certas características do ambiente e a uma certa capacidade de processamento. A enorme velocidade com que estes dispositivos evoluem, principalmente em capacidade de processamento e memória, incentiva cada vez mais o surgimento de aplicações nos mais diversos contextos.

Com o crescimento da busca pela tecnologia, diferentes plataformas surgem procurando melhor consumo de energia e desempenho. Paralelamente às plataformas, surgem também novos dispositivos, como *transceivers* de rádio oferecendo uma melhor relação energia \times taxa de transmissão. Consequentemente, os *softwares*, como sistemas operacionais, são diretamente influenciados pelas características destas novas plataformas e dispositivos.

Este capítulo apresenta uma visão geral sobre as plataformas, dispositivos e sistemas operacionais existentes, além de aplicações que utilizam redes de sensores, principalmente na área de robótica móvel.

2.1 Sistemas Operacionais

O sistema operacional (SO), tradicionalmente, é uma camada de *software* localizada entre a aplicação e o *hardware*. Entre suas funções estão: controlar e proteger o acesso a recursos, gerenciar alocação destes recursos para diferentes usuários, gerenciar processos, evitar que processos ou serviços bloqueiem recursos, além de prover uma interface simplificada para o acesso aos recursos de *hardware* [26]. No entanto, estas características não são todas necessárias em sistemas embarcados, considerando que as execuções são mais restritas e tendem a ser mais harmoniosas que em sistemas comuns [3].

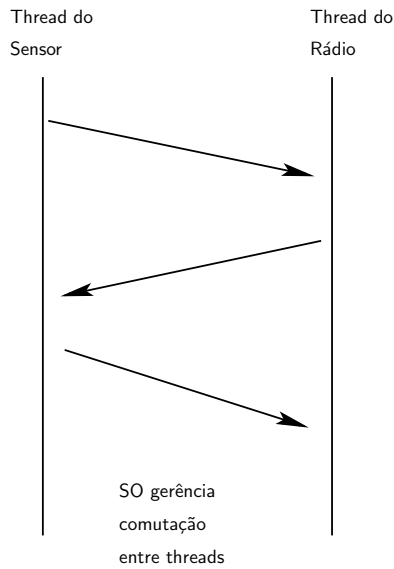
Diante deste cenário, um SO ou um ambiente de execução deve prover as necessidades específicas de sistemas desta natureza. Essas necessidades incluem: gerenciamento de memória, energia e arquivos; comunicação; fornecimento de ferramentas e ambientes de programação; disponibilizar entradas no sistema para acesso aos recursos sensíveis, dentre outras. [2].

SOs são comumente classificados quanto à execução de tarefas em multi ou monotarefa. Um sistema monotarefa é capaz de processar apenas uma tarefa por vez enquanto um multitarefa pode processar várias tarefas simultaneamente. Apesar de um SO multitarefa ser uma melhor opção para a maioria das aplicações de rede de sensores, o maior requerimento de recursos deste tipo de sistema, principalmente memória, o torna inviável para estes dispositivos.

Outro ponto importante quando tratamos de SOs para redes de sensores é a discussão sobre a escolha de um paradigma de programação: baseado em eventos ou em *threads*.

Programas baseados em *threads* utilizam múltiplas *threads* de controle dentro de um único programa. Desta forma, uma *thread* que está bloqueada por uma operação E/S pode ser posta em espera enquanto outras tarefas são executadas em outras *threads*, Figura 2.1. Entretanto, dentro deste paradigma os programas devem proteger estruturas de dados compartilhadas e a execução das *threads* deve ser coordenada. Normalmente, códigos desenvolvidos utilizando *threads* são mais complexos e sujeitos a mais falhas, podendo ainda levar ao bloqueio indefinido de recursos.

Já em SOs baseados em eventos, a figura da *thread* deixa de existir, dando lugar a eventos e

Fig. 2.1: Modelo baseado em *threads*.

handlers de eventos. Os *handlers* se registram ao escalonador do SO para serem comunicados quando da ocorrência de um evento específico. O núcleo, tradicionalmente, utiliza funções em laço que periodicamente verificam a ocorrência de eventos, e, quando estes ocorrem, o *handler* apropriado é chamado. O modelo baseado em eventos é ilustrado pela Figura 2.2.

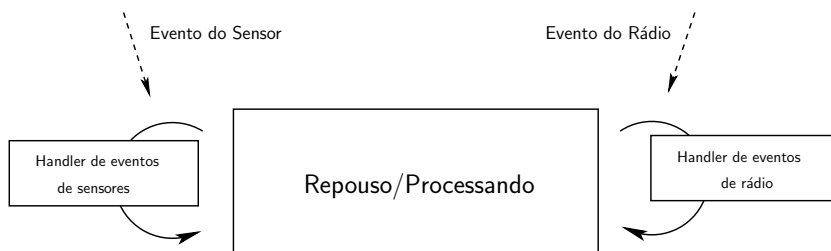


Fig. 2.2: Modelo baseado em eventos.

Na literatura, podem ser encontrados vários SOs e plataformas desenvolvidos para redes de sensores, entre eles estão: TinyOS, LiteOS, Contiki, SunSPOT e SOS. Também é possível encontrar plataformas que suportem a execução de um sistema Linux.

Por este trabalho ser baseado no TinyOS, este sistema será discutido mais detalhadamente. Para os demais, será apresentada uma breve revisão.

2.1.1 TinyOS

O TinyOS [27] é um sistema operacional para plataformas de redes de sensores sem fio. Originalmente desenvolvido pela Universidade de Berkeley na Califórnia, era baseado inteiramente na plataforma MicaZ. A segunda versão do SO foi completamente remodelada, tornando-o mais genérico e suportando diferentes plataformas, mas acarretando incompatibilidade entre as duas versões [28]. Hoje, o TinyOS é um sistema *Open Source* que é mantido por uma vasta comunidade.

A linguagem utilizada para o desenvolvimento do TinyOS foi a nesC [29]. Esta é uma linguagem baseada em C com algumas alterações visando melhor compatibilidade com dispositivos que não dispõem de grande capacidade de processamento e memória. NesC é uma linguagem baseada em componentes e, conseqüentemente, o TinyOS também apresenta esta característica.

O TinyOS destaca-se por ser uma sistema desenvolvido buscando otimização tanto no uso de memória quanto no consumo de energia. Portanto, optou-se pela adoção de um modelo baseado em eventos ao invés de uma abordagem baseada em *threads*¹.

A segunda versão do TinyOS procura implementar um *framework* genérico dando suporte a diferentes *hardwares* e a opção por um modelo baseado em eventos torna mais simples a interação entre o sistema e as diversas plataformas suportadas. A maior parte dos *hardwares* possui um modelo *split-phase*. Neste modelo, para realizar a leitura de um sensor, o *software* modifica o valor de alguns registradores iniciando a leitura. Quando a leitura é concluída, o *hardware* gera uma interrupção e o *software* lê o resultado de um registrador específico. Ao invés de tentar tornar essa interação síncrona, o que seria mais intuitivo ao programador, o TinyOS mantém este modelo, já que esta alteração implicaria num aumento do consumo de memória e energia. É importante notar que a comunicação neste modelo é bidirecional: uma chamada (*downcall*) que inicializa a operação e uma resposta (*upcall*) que sinaliza o fim da mesma. Dentro do TinyOS, *downcalls* são chamados comandos enquanto *upcalls* são eventos.

Como mencionado anteriormente, a linguagem nesC é orientada à componentes, ou seja, um

¹O projeto TOSTHEADS busca implementar um suporte à *threads* para o TinyOS

programa é formado pela união de vários componentes. Neste contexto um componente é similar a um objeto para linguagens orientadas à objeto. A união de dois componentes se dá através de interfaces. Uma interface define comandos e eventos que vão constituir interações entre os componentes. Um componente pode usar ou prover uma interface. Uma vez que um componente usa uma interface, este deve tratar todos seus possíveis eventos, através de *handlers*. Enquanto o componente que provê uma interface deve implementar todos seus comandos.

Na implementação de uma aplicação TinyOS, utilizando a linguagem nesC, definem-se componentes, conhecidos como *modules*. Para implementar a conexão entre os diferentes componentes, um arquivo de configuração, é definido e nele são estabelecidas as conexões entre os componentes, através das devidas interfaces.

A Figura 2.3 ilustra o componente A que provê a interface C, o componente provê o comando D1 e sinaliza o evento D2. Na Figura 2.4(a), o componente B declara interesse em utilizar a interface C, enquanto na Figura 2.4(b) um arquivo de configuração conecta o componente que provê a interface, A, ao componente que a utiliza, B.

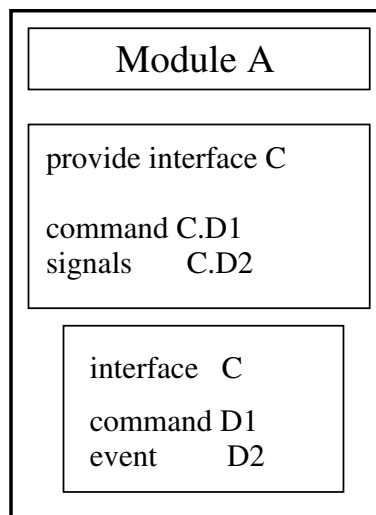


Fig. 2.3: Componente TinyOS que provê uma interface. [2]

Um componente pode ser subdividido em quatro partes: um conjunto de *handlers* para os comandos, outro conjunto de *handlers* para os eventos, um quadro de tamanho fixo e uma ou mais tarefas. O quadro mantém o estado do componente, enquanto comandos, eventos e tarefas atuam

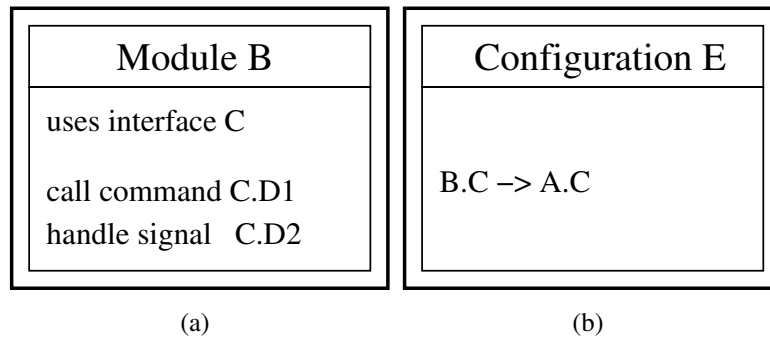


Fig. 2.4: (a) Componente TinyOS que usa uma interface. (b) Configuração TinyOS que conecta os componentes. [2]

sobre este estado. Como o tamanho dos quadros é alocado estaticamente, os requerimentos de memória de um componente são determinados em tempo de compilação e, devido a isso, não existe *overhead* por alocação dinâmica. Essa característica impõe algumas restrições de desenvolvimento, mas também implica em menor consumo de energia. A Figura 2.5 ilustra o componente *Timer* do TinyOS.

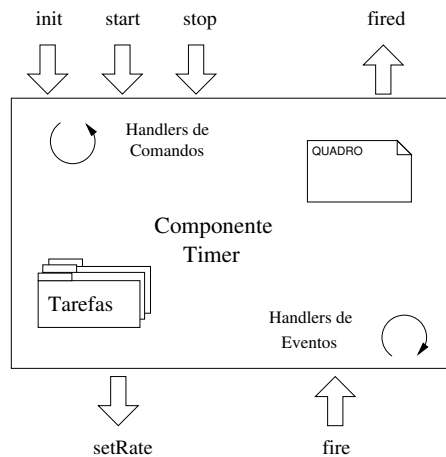


Fig. 2.5: Componente *Timer* do TinyOS. [3]

No modelo de execução do TinyOS, comandos, eventos e tarefas desempenham um papel fundamental. São os responsáveis por permitir que haja comunicação entre partes de um mesmo quadro. Devido ao modelo adotado pelo TinyOS, é importante ressaltar que comandos e eventos devem ambos ser executados até o fim e apenas executarem tarefas simples. Particularmente, comandos devem ser não bloqueantes; devem ser simplesmente requisições onde tarefas de componentes mais baixos na

hierarquia devem agir. *Handlers* de eventos, analogamente, devem apenas deixar informações no quadro do seu componente e preparar a execução de uma tarefa posteriormente. Um evento também pode realizar chamadas a comandos de outros componentes ou sinalizar diretamente um evento para componentes acima.

Tarefas, por sua vez, são processos monolíticos que também têm que ser executadas até o fim. Isto implica que tarefas são atômicas entre si, mas podem ser interrompidas por eventos. É desta forma que o modelo do TinyOS suporta concorrência. A gerência da execução destas tarefas é realizada por um simples escalonador FIFO (*First In First Out*) voltado para o consumo de energia; isso quer dizer que um nó é “desligado” caso não esteja processando ou esperando nenhuma tarefa.

Dentro de uma aplicação do TinyOS, os componentes são organizados em uma estrutura hierárquica. Níveis mais altos realizam chamadas aos níveis mais baixos, enquanto estes sinalizam eventos aos níveis mais altos. O *hardware* está localizado na base desta hierarquia. A Figura 2.6 ilustra este relacionamento entre aplicação, SO e *hardware*.

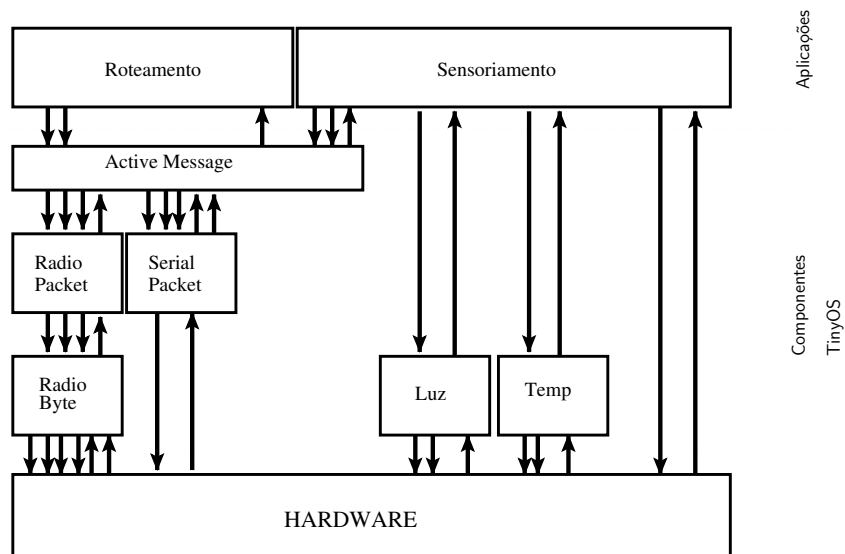


Fig. 2.6: Arquitetura de componentes do TinyOS. [4]

A arquitetura proposta pelo TinyOS para realizar a abstração de *hardware*, a *Hardware Abstraction Architecture* (HAA), é apresentada na Figura 2.7. Neste modelo, a abstração de *hardware* é organizada em três camadas conhecidas como: *Hardware Interface Layer* (HIL), *Hardware*

Adaptation Layer (HAL) e *Hardware Presentation Layer* (HPL). A camada de apresentação de *hardware*, HPL, está posicionada diretamente acima do *hardware* e procura prover uma interface mais simples do mesmo para o resto do sistema. O acesso ao *hardware* é realizado de maneira usual, por meio de memória ou portas de E/S. A HAL, camada de adaptação de *hardware*, é o principal elemento da arquitetura. Esta camada utiliza as interfaces providas pela HPL para definir uma abstração mais usual, escondendo a complexidade intrínseca da manipulação direta de *hardwares*. Finalmente a HIL, camada de interface de *hardware*, converte as abstrações específicas de plataformas providas pelo HAL em interfaces independentes de plataformas para serem utilizadas pelas aplicações.

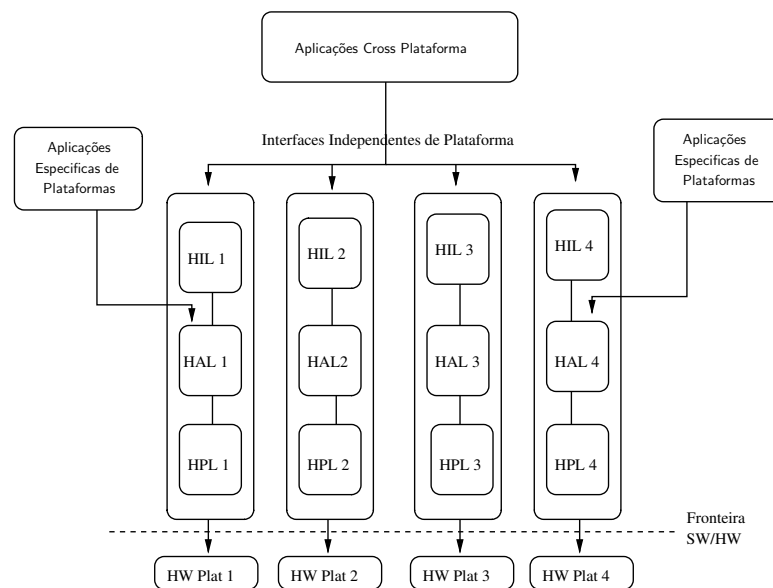


Fig. 2.7: HAA -Arquitetura para suporte a abstração de *hardware* do TinyOS [5].

Na Figura 2.6 esta abstração de três camadas pode ser observada para o *hardware* de comunicação sem fio.

O uso deste modelo também possui outras vantagens. Dois dos três níveis podem ser acessados pelas aplicações, as camadas HIL e HAL, e, como consequência, é possível acessar um *hardware* pelos dois níveis paralelamente. Isto é chamado de decomposição vertical. Uma outra vantagem é a possibilidade de reuso de abstrações que são comuns à plataformas diferentes. No TinyOS cada elemento de hardware, como microcontroladores ou rádios, é definido como um *chip*, uma abstração de *hardware* que também segue o modelo HAA, provendo implementações HAL. Desta forma, a

definição de uma nova plataforma pode ser realizada apenas conectando os *chips* necessários. Isso também é conhecido como decomposição horizontal, Figura 2.8

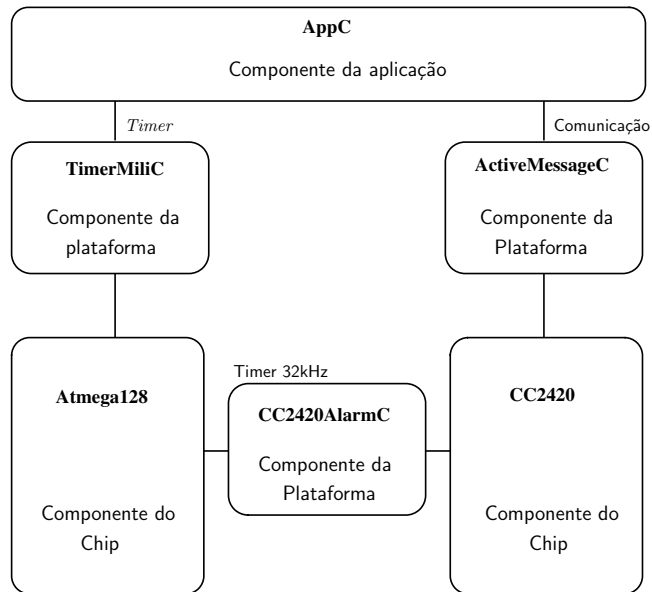


Fig. 2.8: Decomposição horizontal [5].

Uma das principais características do TinyOS é a sua abstração de *hardware*. Para realizar a comunicação isto não é diferente. Na segunda versão do TinyOS, um importante elemento da comunicação é a abstração das mensagens pelo tipo *message_t* [30]. A definição do tipo *message_t* é apresentada no Quadro 2.1.

```

typedef nx_struct message_t {
    nx_uint8_t header[sizeof(message_header_t)];
    nx_uint8_t data[TOSH_DATA_LENGTH];
    nx_uint8_t footer[sizeof(message_footer_t)];
    nx_uint8_t metadata[sizeof(message_metadata_t)];
} message_t;
  
```

Quadro 2.1: Definição do tipo *message_t* na linguagem nesC.

O uso desta estrutura genérica permite que aplicações sejam desenvolvidas independentemente do *hardware* de rádio utilizado, já que estas interagem diretamente apenas com campo *data*. Os campos restantes, *header*, *footer* e *metadata*, são apenas manipulados pelas camadas mais baixas, possuindo implementações específicas para cada *hardware*. O Quadro 2.2 apresenta a definição do

header para o rádio CC2420. Todos os campos desta definição até o *network* vem da especificação do IEEE 802.15.4 e são utilizados pelo dispositivo. O campo *network* deve ser utilizado apenas quando necessário realizar a interoperação com a especificação 6LowPAN. Já o campo *type* é uma definição específica do TinyOS.

Um exemplo da flexibilidade oferecida pela estrutura *message_t* é que no caso do CC2420 não existe a definição do campo *footer*. Em vez disto, o próprio *hardware* acrescenta um CRC de dois bytes ao final de cada pacote enviado.

```
typedef nx_struct cc2420_header_t {
    nxle_uint8_t length;
    nxle_uint8_t fcf;
    nxle_uint8_t dsn;
    nxle_uint16_t destpan;
    nxle_uint16_t dest;
    nxle_uint16_t src;
    nxle_uint8_t network; // optionally included with 6LowPAN layer
    nxle_uint8_t type;
}cc2420_header_t;
```

Quadro 2.2: Definição do *header* para o CC2420 na linguagem nesC.

Através da *macro* TOSH_DATA_LENGTH, é possível determinar o tamanho do *payload* das mensagens em tempo de compilação. Esta variável possui um valor padrão de 28 bytes, mas pode ser alterada para até 128 bytes.

Desta forma, um pacote enviado pelo rádio CC2420 possui o formato apresentado na Figura 2.9, sem utilizar o campo *network* da definição do *header*.

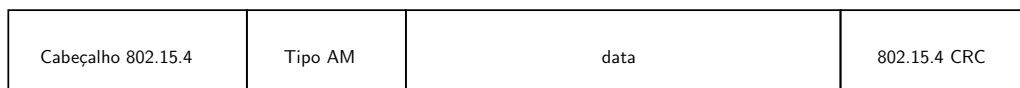


Fig. 2.9: Mensagem CC2420 padrão [6]

TOSSIM

O TinyOS possui um simulador próprio, o TOSSIM. TOSSIM é um simulador a eventos discretos para redes de sensores sem fio que pode ser executado em *desktops*. Seu principal objetivo é prover um ambiente confiável para simulações de aplicações do TinyOS, permitindo que usuários possam testar e analisar códigos em um ambiente controlado e de mais simples manipulação. Em vez de compilar as aplicações para uma plataforma específica, os usuários compilam aplicações explicitamente para o TOSSIM, desta forma, não apenas os algoritmos podem ser analisados, mas também a implementação [31].

Este simulador funciona substituindo os componentes utilizados pela aplicação por implementações de simulação, ou seja, existem implementações análogas a componentes reais, que são carregadas no lugar destas quando uma aplicação é compilada para o TOSSIM. Este simulador, na verdade é uma biblioteca: isto implica que, para executar uma simulação, um outro programa precisa ser escrito para configurá-la e executá-la. Para tal, existem duas interfaces de programação, Python e C++. Em simulações executadas utilizando Python, é possível interagir dinamicamente com a mesma facilitando uma análise mais detalhada do programador. Entretanto, simulações realizadas utilizando C++ apresentam melhor performance.

2.1.2 SOS

O SOS [7] é um sistema operacional embarcado que procura otimizar a relação entre flexibilidade e eficiência na interação com recursos. Uma de suas principais características é a presença de um núcleo onde módulos podem ser carregados e removidos. O funcionamento de um módulo pode ser comparado a um componente do TinyOS, já que este implementa tarefas e funções específicas. Além desta similaridade, uma aplicação do SOS é composta pela interação entre um ou mais módulos. Porém, enquanto um componente do TinyOS possui uma posição estática de memória, módulos do SOS são binários com posição independente, o que possibilita que módulos sejam conectados dinamicamente. O núcleo do SOS disponibiliza interfaces para acesso ao *hardware*, além de um

mecanismo de escalonamento baseado em prioridade e alocação dinâmica de memória.

As interações entre os módulos do SOS são classificados entre comunicação síncrona e assíncrona. A comunicação assíncrona se dá por meio de mensagens, enquanto que a síncrona ocorre por chamadas diretas a funções, Figura 2.10. Após ter sido enviada, uma mensagem passa pelo escalonador que a coloca numa fila de prioridades. Quando o núcleo realiza a chamada do *handler* apropriado, a mensagem é repassada ao módulo destinatário. Interações realizadas por meio de chamadas de funções são mais rápidas do que as realizadas por mensagens. Esta solução requer que os módulos registrem suas funções públicas no núcleo no momento de sua inicialização. Módulos que desejam realizar chamadas destas funções devem registrar esse interesse, em um modelo publicador/consumidor.

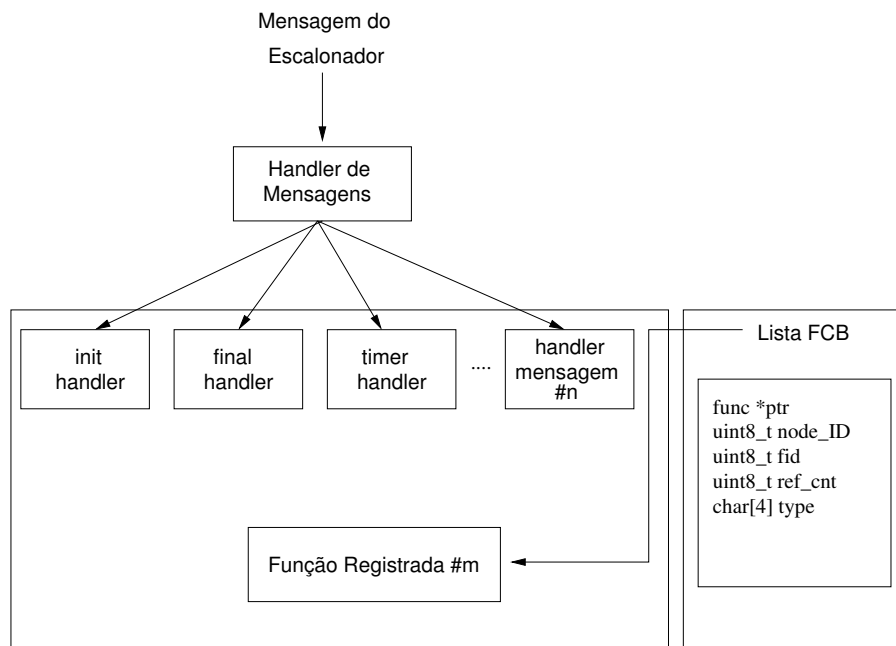


Fig. 2.10: Modelo de comunicação do SOS nos dois formatos [7].

Uma importante característica do SOS é o suporte à reprogramação dinâmica. Como já foi dito anteriormente, uma aplicação SOS é formada pela “união” de um ou mais módulos. Desta forma, uma operação simples de reprogramação nada mais é do que o acréscimo ou substituição de um módulo. Este processo é dividido em três etapas:

1. Quando um módulo está disponível, cabe a um protocolo de distribuição de código publicar informações sobre este para a rede. Estas informações consistem do ID do módulo, da versão e do tamanho de memória requerido. Ao receber essas informações, um protocolo local avalia se o módulo é uma atualização ou um novo módulo, além de avaliar a disponibilidade de espaço físico para o armazenamento do módulo.
2. Uma vez decidido que o módulo será atualizado ou instalado, o protocolo inicia o seu *download*. Neste ponto, uma nova avaliação é realizada para determinar se há espaço suficiente para armazenar o estado local do módulo, e se não houver, o processo é cancelado.
3. Caso todos os requisitos acima sejam atendidos, a inserção e inicialização do módulo é finalizada com sucesso.

2.1.3 Contiki

O Contiki [8] é um sistema que procura apresentar uma solução híbrida para redes de sensores. Enquanto seu núcleo funciona como um sistema orientado a eventos, o suporte a multitarefa é disponibilizado como uma biblioteca. Uma estratégia de ligação dinâmica é utilizada para unir a biblioteca às aplicações que a requerem.

Assim como o SOS, o Contiki também realiza a separação entre o núcleo e o restante dos serviços, que podem ser carregados dinamicamente, mas, neste contexto, estes são chamados de processos. A comunicação entre processos ocorre por meio da postagem de eventos através do núcleo. Na arquitetura do Contiki o núcleo não dá suporte a nenhuma abstração de *hardware*; em vez disso, *drivers* e aplicações podem se comunicar diretamente com o recurso.

O Contiki pode ser dividido em duas partes fundamentais: serviços principais e programas dinamicamente carregados, Figura 2.11. Essa divisão ocorre em tempo de compilação. Os serviços principais são: o núcleo, o carregador de programas, uma pilha de comunicação (com *drivers* dos dispositivos de comunicação) e serviços adicionais. O conjunto desses serviços é compilado em uma imagem binária e instalado nos nós da rede. Este bloco não pode ser alterado dinamicamente.

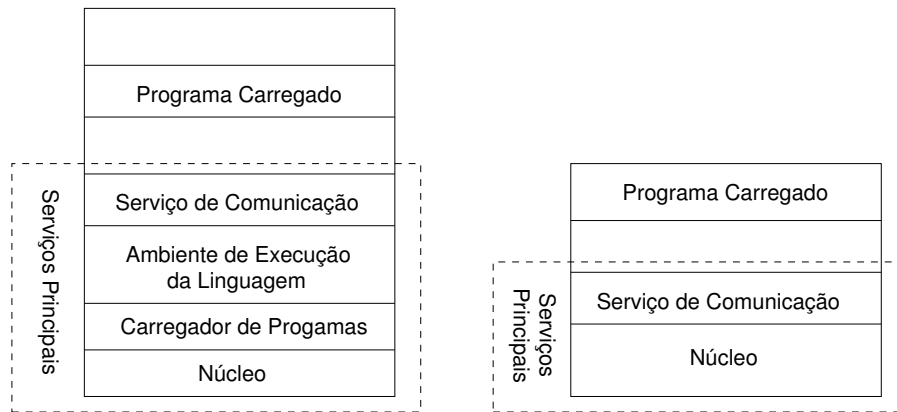


Fig. 2.11: Esquema do SO Contiki: programas são divididos em duas classes: serviços principais e programas carregados dinamicamente [8].

Um serviço consiste de uma interface e sua implementação, que também é chamada de processo. Numa interface, estão definidos o número de versão e a lista de funções com ponteiros para suas respectivas implementações. Uma aplicação realiza chamadas às funções de um serviço através de sua interface, utilizando um *stub*. Uma camada de serviços é implementada para registrar e gerenciar todos os serviços ativos e disponíveis. Serviços ativos registram-se à camada de serviços informando a descrição de sua interface, ID e número de versão. Um serviço do Contiki pode ser visto como análogo a um módulo do TinyOS.

O uso de *stubs* para a interação entre aplicações e serviços implica que programas não precisam conhecer detalhes de implementação ou localização em memória dos serviços. Quando um serviço é chamado, cabe a camada de serviços responder com o ponteiro para uma implementação que esteja ativa. Esta característica dá ao Contiki a possibilidade de realizar alterações e atualizações de serviços sem nenhum impacto nas aplicações.

2.1.4 LiteOS

O LiteOS [32] [9], por sua vez, é um sistema baseado em *thread* e com suporte a múltiplas aplicações. Este SO, diferentemente da maioria dos sistemas desenvolvidos para plataformas desta natureza, não é baseado na conexão entre vários componentes de *software* para a criação de uma

aplicação. O LiteOS busca uma completa separação entre o SO e a aplicação executando sobre ele. Por isso, o sistema disponibiliza várias chamadas de sistema, um *shell* que separa chamadas de diferentes usuários, um sistema de arquivos hierárquico e um protocolo para reprogramação dinâmica.

Dentro do modelo imposto pelo LiteOS, uma rede é vista como um sistema de arquivo distribuído. Um usuário pode acessar e manipular qualquer nó na rede através de um nó base conectado a um computador. Cada nó executa um núcleo, com suporte a multitarefa, que possui três componentes principais: um escalonador, um conjunto de chamadas de sistema e um instalador para binários. Através das chamadas de sistema disponibilizadas pelo núcleo, o usuário é capaz de acessar e gerenciar arquivos e diretórios locais. Estes arquivos são classificados entre dados de sensores, *drivers* de dispositivos e binários de aplicações. Todo o histórico de manipulação do usuário na rede é mantido pelo computador remoto, já que, dentro da hierarquia do sistema, um nó da rede é um componente sem estado. Uma visão geral da arquitetura do LiteOS é ilustrada na Figura 2.12.

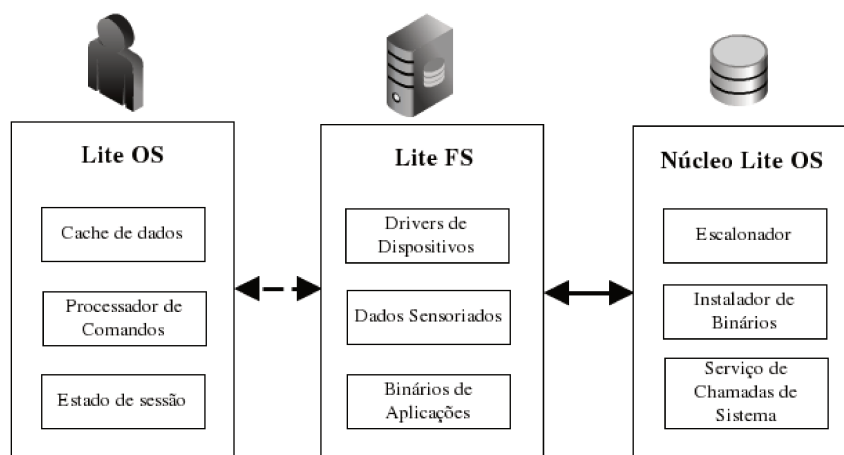


Fig. 2.12: Arquitetura do LiteOS [9].

O LiteOS tenta prover um sistema familiar a usuários de ambiente Linux, tanto o *shell* quanto as chamadas de sistema são baseadas em Linux.

Uma importante característica deste SO é o seu sistema de arquivos, o LiteFS. O LiteFS é um sistema de arquivos distribuídos que permite ao usuário acessar e gerenciar toda a rede de sensores, além de programar e manipular cada nó individualmente. Localmente, o sistema de arquivos é

organizado da seguinte forma: a memória RAM contém a lista de arquivos ativos, ou abertos, e informações sobre a alocação nas memórias EEPROM e Flash.

O LiteOS também oferece suporte à reprogramação dinâmica de aplicações de usuários. Esta operação pode ser realizada de duas formas. Se o código fonte estiver disponível para o SO, este será recompilado com uma nova configuração na memória e todos os ponteiros e referências da versão anterior são alterados para a nova versão. Caso o código fonte não esteja disponível, o SO faz uso de um mecanismo de *patching* diferencial para substituir a versão anterior.

2.1.5 Linux

Além dos SOs desenvolvidos especificamente para redes de sensores, em dispositivos com maiores capacidades (Seção 2.2) é possível executar uma versão simplificada do núcleo do sistema Linux. Através de projetos como o *OpenEmbedded* [33], é possível compilar imagens do núcleo para as arquiteturas desejadas. A instalação do sistema Linux em um Imote2, por exemplo, consiste da instalação de três elementos: um bootloader, o núcleo propriamente dito e o sistema de arquivos, no formato JFFS2 [34]. A organização destes elementos na memória é apresentada na Figura 2.13.

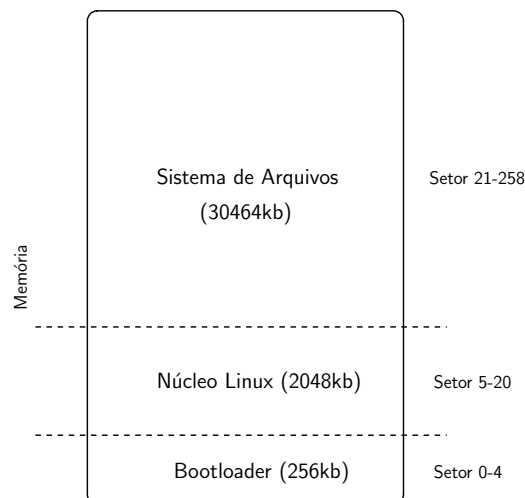


Fig. 2.13: Organização de uma instalação do Linux na memória de um Imote2.

No caso do Imote2, é possível encontrar facilmente *drivers* para seus sensores e dispositivos. Existe também uma implementação inspirada na camada MAC do TinyOS, chamada TOSMAC,

que implementa o modelo de pacotes do IEEE 802.15.4 para realizar a comunicação sem fio. A implementação atual do TOSMAC é compatível apenas com o TinyOS 1.x.

2.2 Hardware para Redes de Sensores

Um dispositivo que compõe uma rede de sensores, tipicamente, é capaz de: sensoriar o ambiente através de diferentes sensores, processar informações localmente e se comunicar com seus vizinhos através de um protocolo sem fio. Nós de redes desta natureza, usualmente, são constituídos por três componentes básicos, que podem tanto ser dispositivos individuais ou embarcados [24]:

- **Módulo principal ou Mote:** componentes principais de uma rede de sensores. Possuem a capacidade de comunicação e a memória onde aplicações são armazenadas. Um *mote* consiste de um microcontrolador, *transceiver*, alimentação de energia, unidades de memória e podem incluir alguns sensores.
- **Módulo de sensores:** dispositivos embarcados com uma variedade de sensores, que podem ser acoplados aos *notes*. Entre os módulos de sensores disponíveis estão MTS300/400 e MDA 100/300 da família Mica. Diferentes sensores estão disponíveis para diferentes plataformas.
- **Placa de programação:** também chamada de *gateway* ou *interface board*, disponibiliza múltiplas interfaces como *Ethernet*, WiFi, USB ou portas seriais para conexão de *notes* à redes industriais ou comerciais ou ainda a um computador. Estas placas são utilizadas tanto para o carregamento de programas quanto para recuperar dados da rede. Exemplos de dispositivos desta natureza incluem o MIB510, MIB520 e IIB2400, fornecidos pela Memsic [35].

As plataformas existentes são usualmente divididas em duas classes, de acordo com suas capacidades e uso: *low-end* e *high-end* [24]. Plataformas classificadas como *low-end* são caracterizadas por limitadas capacidades de processamento, memória e comunicação. São desenvolvidas visando cenários onde grandes quantidades de *notes* são distribuídos em um ambiente para realizar

sensoriamento ou prover uma infraestrutura de comunicação. Tais plataformas têm sido vastamente utilizadas no desenvolvimento e testes de protocolos de comunicação.

Dentre as plataformas *low-end*, podem ser destacadas:

A família Mica: esta família é constituída pelas plataformas Mica, Mica2, MicaZ (Figura 2.14(a)) e IRIS, produzidos pela Memsic. Cada nó é equipado com um microcontrolador Atmel AVR de 8 bits com velocidade de 6-16 MHz e 128 kB de memória *flash*. Enquanto o microcontrolador foi mantido para todas as plataformas, diferentes *transceivers* foram utilizados. O Mica possui um *transceiver* de 868 MHz à 10/40kbps, enquanto o Mica2 vem equipado com um de 433/868/916 MHz à 40kbps. Já o MicaZ e o IRIS possuem *transceivers* compatíveis com o padrão IEEE 802.15.4, que operam a 2.4 GHz à 250kbps. Todas as plataformas possuem memória entre 4-8 kB e memória de armazenamento na ordem de 512 kB. Além disso, são equipadas com conectores de 51 pinos para conexão de sensores, como barômetros ou sensores de umidade e placas de programação.

Telos/Tmote: uma arquitetura similar ao MicaZ foi utilizada tanto no TelosB (Figura 2.14(b)) da Memsic quanto no Tmote Sky (Figura 2.14(c)) da Sentilla. O *transceiver* foi mantido idêntico, porém os Telos/Tmote possuem uma memória maior, já que utilizam um microcontrolador TI MSP430 de 6MHz com 10 kB de memória RAM. Estas plataformas possuem uma variedade de sensores embarcados como de luminosidade, infra-vermelho, umidade e temperatura, além de um conector USB, tornando desnecessário o uso de placas de programação. Conectores de 6 ou 10 pinos possibilitam o uso de sensores adicionais.

EYES: esta plataforma (2.14(d)) é resultado do projeto homônimo de três anos financiado pela União Européia, e possui uma arquitetura similar ao TelosB/Tmote. O microcontrolador MSP430 de 16 bits utilizado possui 60 kB de memória de armazenamento e 2 kB de memória RAM. Os seguintes sensores são embarcados no *mote*: bússola, acelerômetro, de temperatura, de luminosidade e de pressão. A plataforma é equipada com um *transceiver* TR1001, que suporta taxas de transmissão de até 115.2 kbps, além de uma interface RS232 para programação.

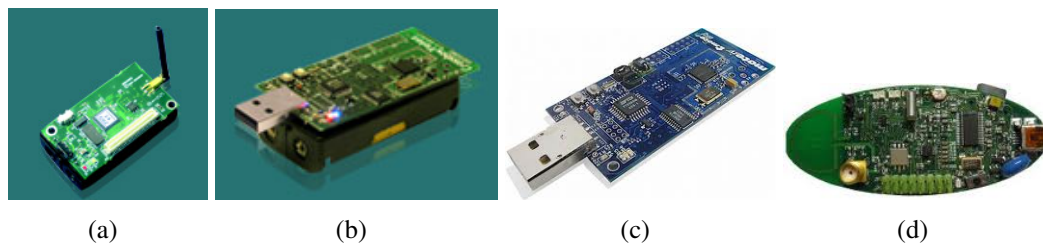


Fig. 2.14: Plataformas *Low-End*: (a) MicaZ, (b) TelosB, (c) Tmote, (d) EYES.

Além de sensoriamento, processamento local e comunicação, redes de sensores também necessitam de algumas funcionalidades que não podem ser eficientemente executadas por plataformas *low-end*. Tarefas como gerenciamento de rede, tipicamente, requerem maior capacidade de processamento e memória do que estas plataformas possuem. Além disso, cenários onde há a integração de WSN com infraestruturas de redes existentes, através de nós *gateways*, ou onde *hubs* de processamento ou armazenamento são integrados a nós de uma rede de sensores, também exigem maiores recursos dos dispositivos. Para cenários desta natureza, foram desenvolvidas as plataformas *high-end*, tal como a Stargate.

Stargate é uma plataforma de alta performance desenvolvida para sensoriamento, processamento de sinais, controle e gerenciamento de redes de sensores. É baseada no processador Intel PXA-255 Xscale 400MHz RISC, processador similar ao encontrado em dispositivos como Compaq IPAQ e Dell Axim. Possui 32MB de memória *flash* 64MB de memória SDRAM, um conector compatível com a família Mica, além de interfaces *Bluetooth* ou IEEE 802.11. Pode ainda ser conectado a *webcams* ou outros dispositivos de captura, porém com alto consumo de energia. O Stargate NetBridge foi desenvolvido para ser o substituto do Stargate. Utiliza, por sua vez, o processador Intel IXP420 Xscale 266MHz. Possui uma interface *Ethernet* e duas USB 2.0.

Como todo o trabalho apresentado nesta dissertação foi desenvolvido utilizando a plataforma iMote2, esta plataforma *high-end* será abordada mais detalhadamente. A Tabela 2.1 apresenta uma comparação entre características das principais plataformas existentes.

Tab. 2.1: *Mote Hardware.*

Tipo <i>Mote</i>	Vel. CPU (MHz)	Prog. Mem. (kB)	RAM (kB)	Freq. Radio (MHz)	Tx. rate (kbps)
<i>Berkeley</i>					
mica	6	128	4	868	10/40
mica2	16	128	4	433/868/916	38.4 kbaud
micaZ	16	128	4	2.4 GHz	250
Cricket	16	128	4	433	38.4 kbaud
EyesIFX	8	60	2	868	115
TelosB/Tmote	16	48	10	2.4 GHz	250
Sun SPOT	16-60	2MB	256	2.4 GHz	250
IRIS	16	128	8	2.4 GHz	250
Imote	12	512	64	2.4 GHz	100
Imote2	13-416	32MB	256	2.4 GHz	250
Stargate	400	32MB	64MB (SD)	2.4 GHz	Varia

2.2.1 Intel Mote2

A Intel desenvolveu duas gerações de plataformas de alta performance para redes de sensores, o iMote e o iMote2. O iMote possui processador ARM7 de 8 bits e 12 MHz, interface *Bluetooth*, 64 kB de memória RAM e 32 kB de memória *flash*, além de várias opções de E/S. A segunda geração de *motes* produzidos pela Intel, o iMote2 (Figura 2.15), foi desenvolvido sobre um processador PXA271 Xscale de 32 bits e 13-416 MHz e um rádio IEEE 802.15.4 TI/Chipcon CC2420. Possui memória RAM e *flash* de 32 MB, suporte a diferentes tipos de rádios e uma variedade de E/S de alta velocidade para conexão de sensores ou câmeras.



Fig. 2.15: Imote2 hardware.

Uma visão mais detalhada do iMote2 é apresentada na Figura 2.16. A arquitetura básica do Imote

é apresentada na Figura 2.17. Esta é dividida em cinco subsistemas: gerenciamento de energia, processamento, sensoriameto, comunicação e interfaceamento.

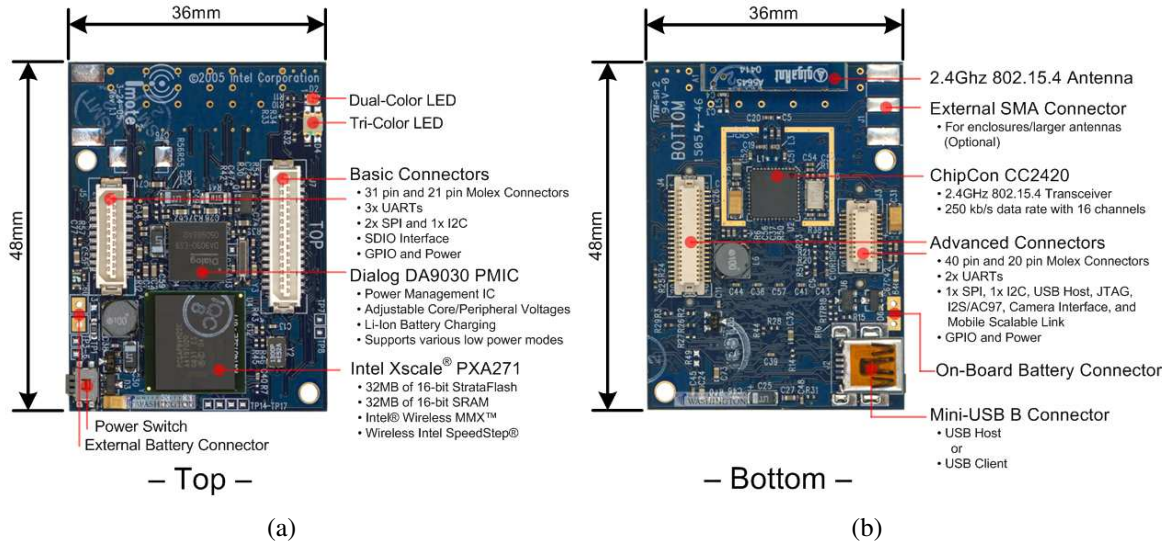


Fig. 2.16: Visão detalhada do Imote2. (a) Superior, (b) Inferior.

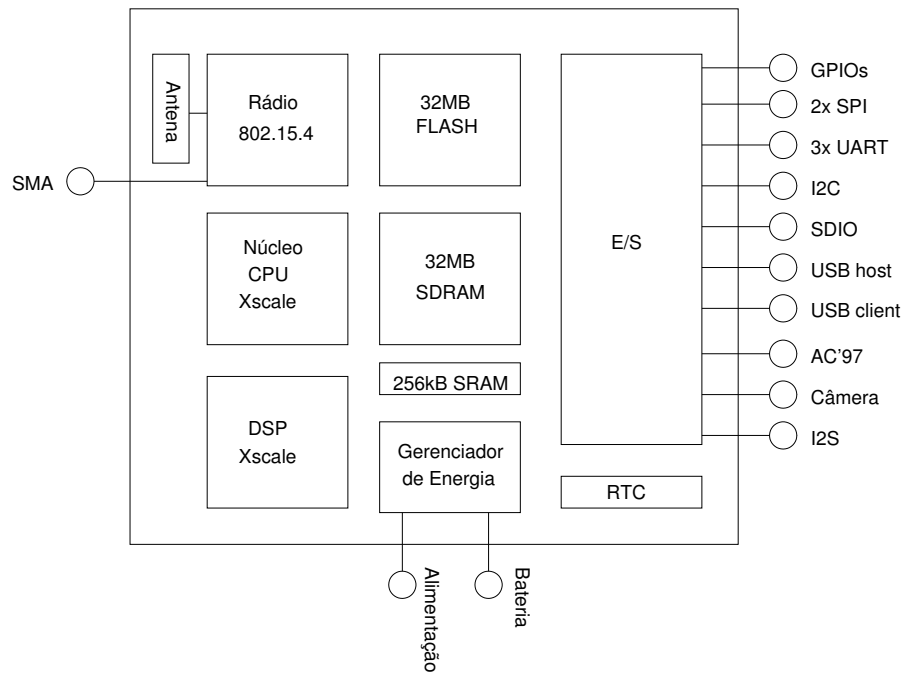


Fig. 2.17: Arquitetura básica da família Imote.

O subsistema de processamento engloba o processador principal, microprocessador, e um processador digital de sinais (DSP - *Digital Signal Processor*). O processador possui a habilidade

Tab. 2.2: Características dos sensores do ITS400.

Sensor	Range	Resolução	Interfaces Com.
Acelerômetro	+/- 2g	12-bit	SPI, I2C
Umidade	10-90% RH (+/-2%) ²	12-bit	GPIO
Temperatura	-40-120°C(+/-2°C) ³	14-bit	GPIO
Temp. Digital	-25-85°C(+/-1.5°C)	-	I ² C
Luz	Luz visível / IR	16-bit	I ² C

Tab. 2.3: Especificação de Operação do ITS400.

Característica	Intervalo de funcionamento
Temperatura de Operação	0 à 70°C
Temperatura de armazenamento	-40 à +150°C
Umidade (sem condensar)	80 %

de operar em modo de baixa tensão (0.85V) e baixa frequência (13MHz), além de vários outros modos de economia de energia.

Para realizar funções de sensoriamento, o iMote2 necessita do seu respectivo módulo de sensoriamento, o ITS400, Figura 2.18(a). O ITS400 possui quatro sensores embarcados: luminosidade, temperatura, umidade e acelerômetro de 3-eixos. As características destes sensores, como resolução e *range*, podem ser visualizadas na Tabela 2.2. A Tabela 2.3 mostra a especificação operacional para o sensor ITS400. Além destes sensores, o módulo multimídia, Figura 2.18(b), também pode ser visto como um módulo de sensoriamento. Este módulo possui uma câmera, alto-falante e um sensor de presença infravermelho.

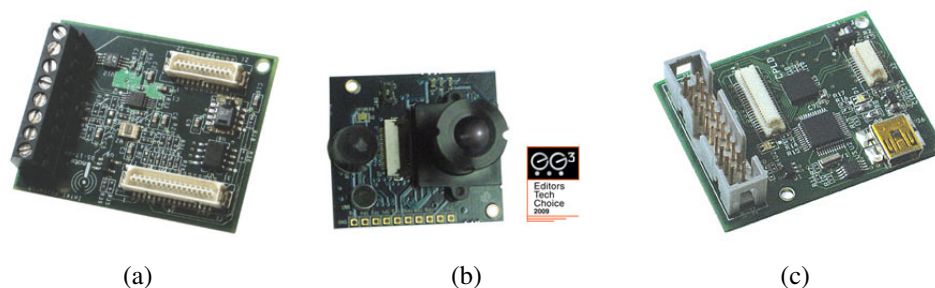


Fig. 2.18: Módulos adicionais do Imote2: (a) ITS400, (b) IMB2400, (c) IIB2400.

O subsistema de sensoriamento, Figura 2.19, provê uma plataforma extensível para conexão de

múltiplos sensores. Os sensores nesta arquitetura se comunicam com o subsistema de processamento através de barramentos SPI e I^2C . O barramento I^2C é utilizado para conectar recurso com baixa taxa de transmissão enquanto o SPI é utilizado para recursos com taxas mais elevadas.

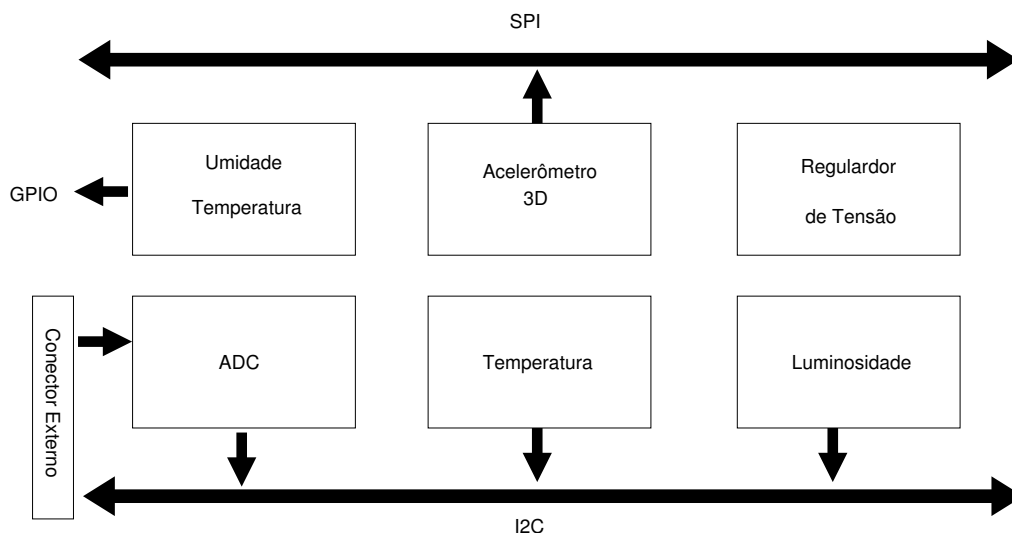


Fig. 2.19: Subsistema de sensoriamento da arquitetura do Imote2.

Outro componente importante é a placa de interface IIB2400, Figura 2.18(c). Através dela, é possível tanto realizar a programação dos *motés* quanto a comunicação de um PC/laptop com a rede. A comunicação, fisicamente, é realizada através de uma porta USB, porém no hardware, essa porta é mapeada como duas UARTs (*Universal Asynchronous Receiver/Transmitter*) através de um *driver* da FTDI [36], uma para troca de mensagens de dados e outra para mensagens de depuração. O módulo também oferece uma conexão JTAG que pode ser utilizada tanto para realizar atualização de *firmware* como para descarregar programas no dispositivo. A conexão pela interface JTAG é realizada utilizando o OpenOCD [37].

A alimentação é responsabilidade de um módulo específico de baterias, IBB2400CA, mas também pode ser realizada pelas portas USB do módulo principal ou da placa de Interface. A Tabela 2.4 apresenta as características deste componente.

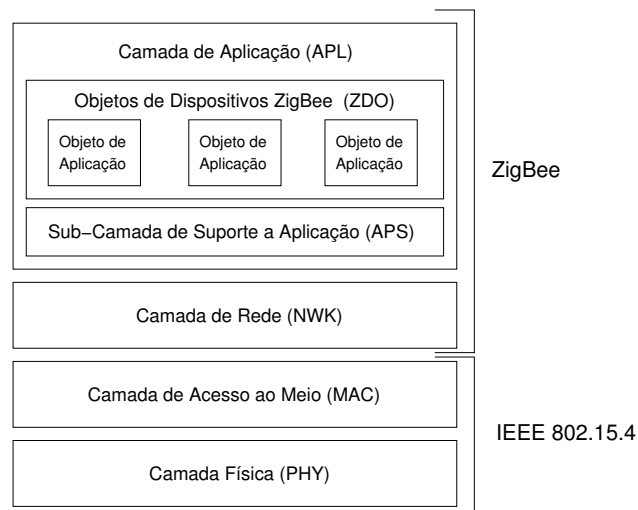
Tab. 2.4: Características da alimentação IBB2400CA.

<i>Battery Board</i>	IBB2400CA
Baterias	3xAAA
Corrente Máxima	500mA Fused
Dimensões	52mm x 43mm x 18mm
Peso com baterias	51g
Peso sem baterias	14g

CC2420 e padrão IEEE 802.15.4

O IMote2 utiliza o padrão IEEE 802.15.4 como sua interface de comunicação sem fio e, para tal, o *transceiver* escolhido foi o CC2420 da TI/Chipcon. O CC2420 é um *transceiver* de 2.4GHz do padrão IEEE 802.15.4 com suporte a até 250kbps desenvolvido especificamente para aplicações sem fio com recursos de energia limitados [23].

Da mesma forma como o IEEE 802.11 ficou comercialmente conhecido como o Wi-Fi, o IEEE 802.15.4 ficou conhecido como o padrão *Zigbee* [38]. Porém o padrão definido pela *Zigbee Alliance* [10] é constituído acima do IEEE 802.15.4, como mostra a Figura 2.20. O padrão 802.15.4 traz apenas a especificação das duas camadas mais baixas do modelo OSI para redes privadas sem fio com baixo fluxo de dados (LR-WPAN), as camadas física e MAC. É possível implementar o restante da pilha de protocolo e comunicar qualquer dispositivo com CC2420 com equipamentos *Zigbee*.

Fig. 2.20: Pilha de protocolos do *Zigbee* e do IEEE 802.15.4 [10].

A especificação do padrão IEEE 802.15.4 WPAN prevê a existência de dois tipos de dispositivos em uma rede: dispositivos de funcionamento pleno (FFD - *Full Function Device*) e de funcionamento reduzido (RFD - *Reduced Function Device*). Os FFDs são dispositivos que possuem a capacidade de executar funções de coordenadores de rede ou formar uma rede própria, enquanto RFDs são dispositivos que implementam apenas um subconjunto das funcionalidades do IEEE 802.15.4. RFDs são recomendados para aplicações simples onde não é necessário o envio de grande volume de dados.

Dentro de uma rede ad-hoc IEEE 802.15.4, dispositivos FFD e RFD podem se relacionar em três diferentes topologias: estrela, *mesh* P2P (*Peer-to-Peer*) ou árvore de *clusters*. Numa topologia estrela, a comunicação é estabelecida com apenas nó (FFD) realizando a tarefa de um coordenador da rede. Nesta topologia, todo nó da rede (FFD ou RFD) se comunica com outros dispositivos através do coordenador. Desta forma, o consumo de energia deste coordenador será significativamente maior que a dos outros nós da rede.

Na topologia P2P também existe a figura do coordenador, porém cada dispositivo pode se comunicar diretamente com qualquer um de seus vizinhos, sem a participação do coordenador. Esta característica aumenta a flexibilidade da rede, apesar de também aumentar sua complexidade. Já a topologia árvore de *clusters* pode ser interpretada como um caso particular de uma topologia P2P em que a maioria dos dispositivos são FFD. Neste cenário, qualquer dispositivo FFD pode desempenhar funções de coordenação e dispositivos RFD podem se conectar aos FFD como folhas no fim de galhos de uma árvore. Esta topologia não é definida pela especificação, há apenas indicações de que é possível construí-la.

O *transceiver* CC2420 possui diferentes níveis de programação de potência de saída, partindo de 0dBm a -24dBm. A Tabela 2.5 mostra a variação de potência e de atenuação do sinal de saída para oito das possíveis programações. A relação entre o consumo de corrente, normalizada em relação a corrente mínima consumida, e a potência de transmissão, em dB, é apresentada na Figura 2.22.

Uma comparação entre as principais características dos *transceivers* encontrados nas principais plataformas utilizadas em redes de sensores é apresentada na Tabela 2.6. O *transceiver* utilizado neste

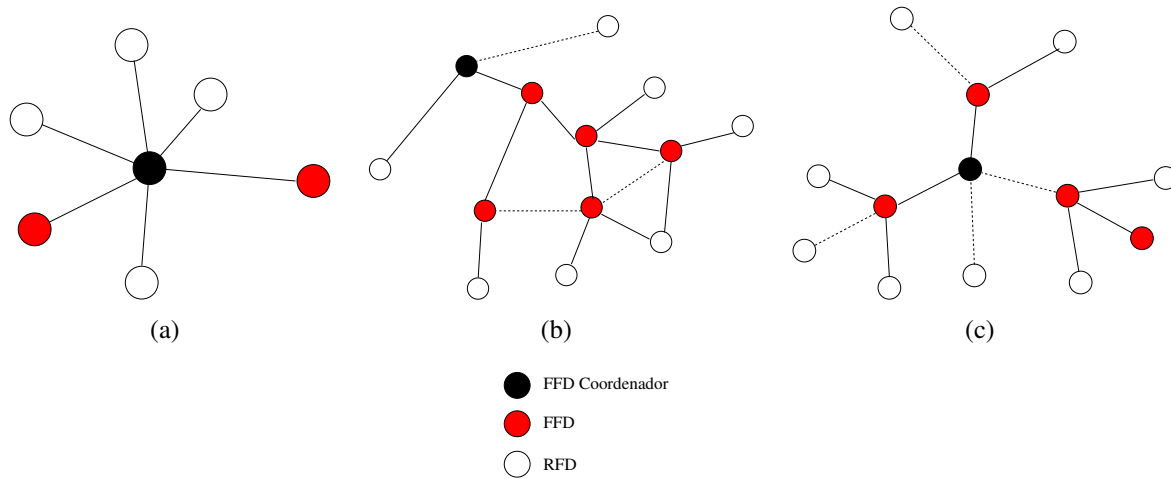


Fig. 2.21: Estruturas topológicas de uma rede IEEE 802.15.4. (a) Estrela, (b) *Mesh* P2P, (c) Árvore de *clusters*.

Tab. 2.5: Configurações de potência de saída do TI/Chipcon CC2420. [23]

Nível de potência	Potência de Sáida dBm	Potência de Sáida mW	Corrente mA	Potência ($V_{dd} = 1.8V$) mW
31	0	1	17.4	31.32
27	-1	0.794328235	16.5	29.7
23	-3	0.501187234	15.2	27.36
19	-5	0.316227766	13.9	25.02
15	-7	0.199526231	12.5	22.5
11	-10	0.1	11.2	20.16
7	-15	0.031622777	9.9	17.82
3	-25	0.003162278	8.5	15.3

trabalho foi o TI CC2420.

2.3 Aplicações

O rápido desenvolvimento das redes de sensores sem fio nos últimos anos trouxe um enorme aumento nas pesquisas realizadas na área. Uma WSN pode fazer uso de diferentes tipos de sensores, entre eles sísmicos, magnéticos, térmicos, visuais, acústicos, que, por sua vez, são capazes de monitorar uma gama de condições ambientais como: temperatura, umidade, pressão, aceleração, direção, luminosidade, som, etc. Esta característica possibilita uma enorme quantidade de cenários

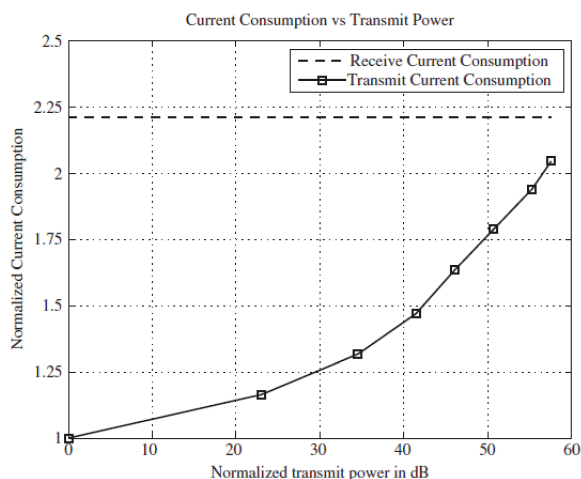


Fig. 2.22: Relação entre a corrente e a potência de saída. [2]

Tab. 2.6: *Transceivers* utilizados em redes de sensores sem fio [24].

	Rádio				
	RFM TR1000	Infineon TDA5250	TI CC1000	TI CC2420	Zeevo ZV4002
Plataformas	WeC, Rene Dot, Mica	eyesIFX	Mica2Dot, Mica2 BTNNode	MicaZ, TelosB SunSPOT, Imote2	Imote BTNNode
Padrão	N/A	N/A	N/A	IEEE 802.15.4	Bluetooth
Trans. (kbps)	2.4-115.2	19.2	38.5	250	723.2
Modulação	OOK/ASK	ASK/FSK	FSK	O-QPSK	FHSS-GFSK
Frequência	916MHz	868MHz	315/433/868/915MHz	2.4GHz	2.4GHz
Alimentação (V)	2.7-3.5	2.1-5.5	2.1-3.6	2.1-3.6	0.85-3.3
TX max (mA/dBm)	12/-1	11.9/9	26.7/10	17.4/0	32/4
TX min (mA/dBm)	N/A	4.9/-22	5.3/-20	8.5/-25	N/A
RX (mA)	1.8-4.5	8.6-9.5	7.4-9.6	18.8	32
<i>Sleep</i>	5 μ A	9 μ A	0.2-1 μ A	0.02 μ A	3.3mA
<i>Startup</i> (ms)	12	0.77-1.43	1.5-5	0.3-0.6	N/A

onde redes de sensores podem ser aplicadas ou integradas. Dentre possíveis cenários se destacam: monitoramento ambiental, doméstico e industrial, além de aplicações militares e na área da saúde.

Talvez a primeira grande área onde redes de sensores foram aplicadas tenha sido a área militar, sendo esta provavelmente a responsável pelo enorme crescimento da tecnologia nos seus primórdios. Neste cenário, existe um forte interesse na coleção de informações, usualmente voltadas para rastreamento (*tracking*) de inimigos, vigilância de campos de batalha ou classificação de

alvos [39]. Por exemplo, uma aplicação que realiza a detecção de intrusos hostis através de uma rede de sensores pode facilmente substituir o uso de minas terrestres [40]. Diferentes aplicações de rastreamento foram desenvolvidas, onde cada um delas busca otimização para tipos de alvos específicos: múltiplos veículos, objetos de metal (veículos, soldados armados) ou ainda cenários onde alvos são explicitamente marcados [41].

Outra área em que o uso de rede de sensores se tornou difundido é no monitoramento de ambientes, tanto internos (domésticos) quanto externos (industriais, ambientais). Uma aplicação usual em cenários internos é realizar o monitoramento e até mapeamento de ambientes considerados não seguros para humanos, por exemplo, escombros de uma edificação que desabou [42]. Outra possibilidade é o uso desta tecnologia para realizar monitoramento de aspectos específicos de uma residência, como consumo de água [43]. Já em ambientes externos, um projeto que chamou a atenção foi o monitoramento de uma espécie de pássaro no seu habitat natural através de uma rede com 32 nós, instalados na *Great Duck Island* [44]. Os nós foram colocados tanto dentro de ninhos, para detectar a presença do pássaro, quanto no ambiente ao seu redor, para adquirir informações sobre temperatura, umidade, pressão, entre outras. Ainda tratando de monitoramento em ambientes externos, um outro ponto de interesse é o monitoramento de áreas agrícolas. Um protótipo para o monitoramento de vinícolas foi desenvolvido e aplicado não somente a plantações, mas como uma forma eficiente de monitorar toda a produção [45].

2.3.1 Robótica Móvel

Durante os últimos anos várias aplicações visam integrar redes de sensores a robótica buscando soluções para problemas clássicos na área: localização [11], mapeamento [46], navegação [19], planejamento de missões [47], entre outros. Além de utilizar rede de sensores para auxiliar na resolução de problemas de robótica, a união entre estas tecnologias também permite o uso de robôs móveis no auxílio de gerencia e manutenção de redes, executando ações como: instalação [48], substituição [49] e aquisição de informações de nós de uma rede [50]. Robôs móveis também podem

ser vistos como uma extensão da capacidade de monitoramento da rede [51] (e vice-versa), além de poderem ser utilizados para mapear nós de uma rede [12] e prover serviços de *gateway* entre diferentes redes.

Um dos problemas clássicos da robótica móvel é a localização, isto é, o robô móvel ser capaz de saber sua exata localização no espaço, baseada numa referência. Uma forma de realizar a localização de robôs móveis é utilizar *landmarks*, pontos com posição conhecida e que é possível inferir uma relação entre a posição do robô e a destes pontos. Uma rede de sensores pode ser utilizada ao invés de *landmarks*. Um protótipo apresentado em [11] utiliza este conceito para realizar a localização de um robô móvel. Nós com posições conhecidas são colocados no ambiente por onde o robô vai navegar. O robô, por sua vez, também possui um nó da rede instalado. Desta forma, para a rede, o robô é um nó móvel, e este pode se comunicar diretamente com os demais nós. Utilizando leituras do RSSI (*Radio Signal Strength Indicator*), a rede pode inferir a distância entre o robô e os nós na sua vizinhança. Realizando uma triangulação, tomando como referência a posição conhecida dos nós fixos da rede, é possível calcular a posição do robô. A Figura 2.23 apresenta dois resultados da técnica proposta para realizar a localização: a Figura 2.23(a) apresenta o resultado utilizando quatro nós, enquanto a Figura 2.23(b) apresenta o resultado do mesmo experimento utilizando seis nós. Pela trajetória resultante do rastreamento do robô, é fácil perceber que, quanto mais nós presentes no ambiente, melhor será o resultado do processo de localização.

Uma outra abordagem para localização em sistemas que integram robôs móveis e redes de sensores é realizar a localização dos nós da rede de sensores utilizando a posição do robô. Existem diferentes implementações para este problema. Uma solução, baseada em visão [12], utiliza uma marca conhecida colocada sobre o robô e câmeras instaladas em nós da rede para inferir a posição dos nós. Os nós são dispostos de forma a não conseguirem se comunicar entre si; desta forma, é necessária a presença de um agente móvel que se mova entre os nós da rede para localizá-los, Figura 2.24(a). Um nó é capaz de detectar a marca sobre o robô, utilizando sua câmera, e inferir uma relação entre as suas posições e, em conjunto com a odometria do robô, é possível definir a posição do nó.

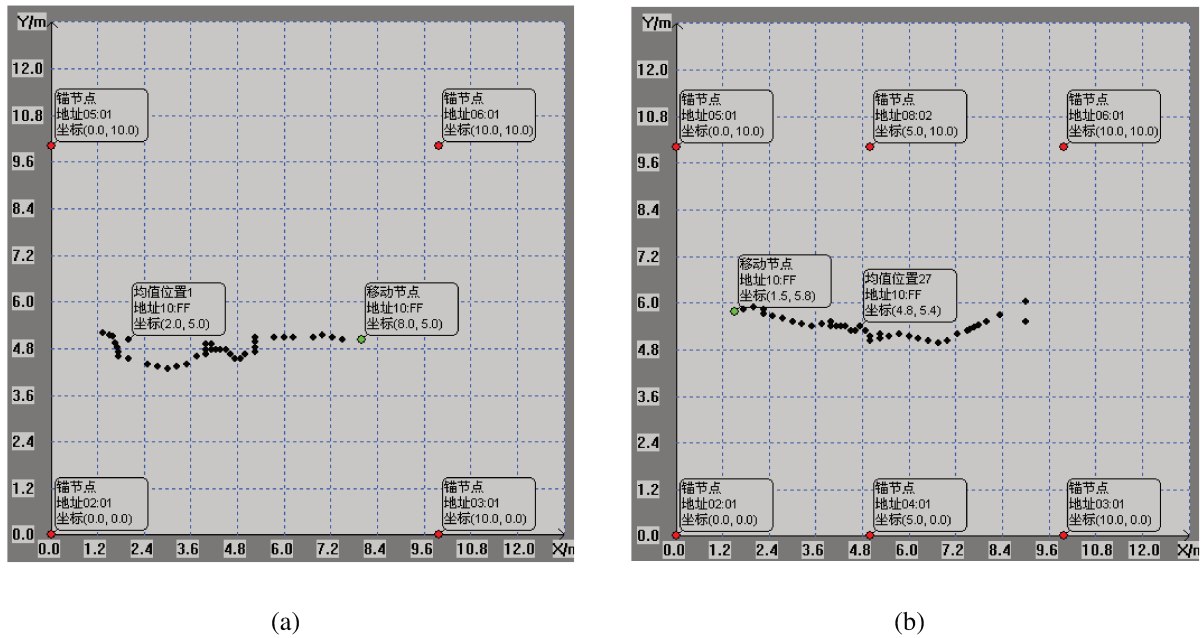


Fig. 2.23: Resultados da localização utilizando (a) 4 e (b) 6 nós [11]

Uma outra solução utiliza um robô móvel para navegar através da rede emitindo *beacons* que são utilizados pelos nós para inferir suas próprias posições [13]. Um robô navega através de uma rede enviando *beacons* periódicos contendo as coordenadas do robô, Figura 2.24(b). Qualquer nó que receba um *beacon* é capaz de retirar uma medida de RSSI, referente ao pacote recebido, e estimar sua posição com uma certa probabilidade. Quanto mais *beacons* um nó recebe melhor será a estimativa da sua posição. Porém para atingir resultados mais precisos, a trajetória tomada pelo robô deve ser tal que cada nó receba pelo menos três *beacons* não colineares. As estimativas são calculadas utilizando inferência Bayesiana.

Redes de sensores também são utilizadas para o auxílio do mapeamento de ambientes. Utilizando as leituras dos sensores de uma rede, um mapa do ambiente pode ser construído baseado em uma característica deste [46]. Mapas no formato de grades são gerados e podem ser utilizados por algum algoritmo de navegação para guiar o robô pelo ambiente. Desta forma, informações como temperatura e umidade podem ser utilizadas como obstáculos para o robô.

Aplicações de rastreamento também podem ser implementadas com o auxílio de redes de

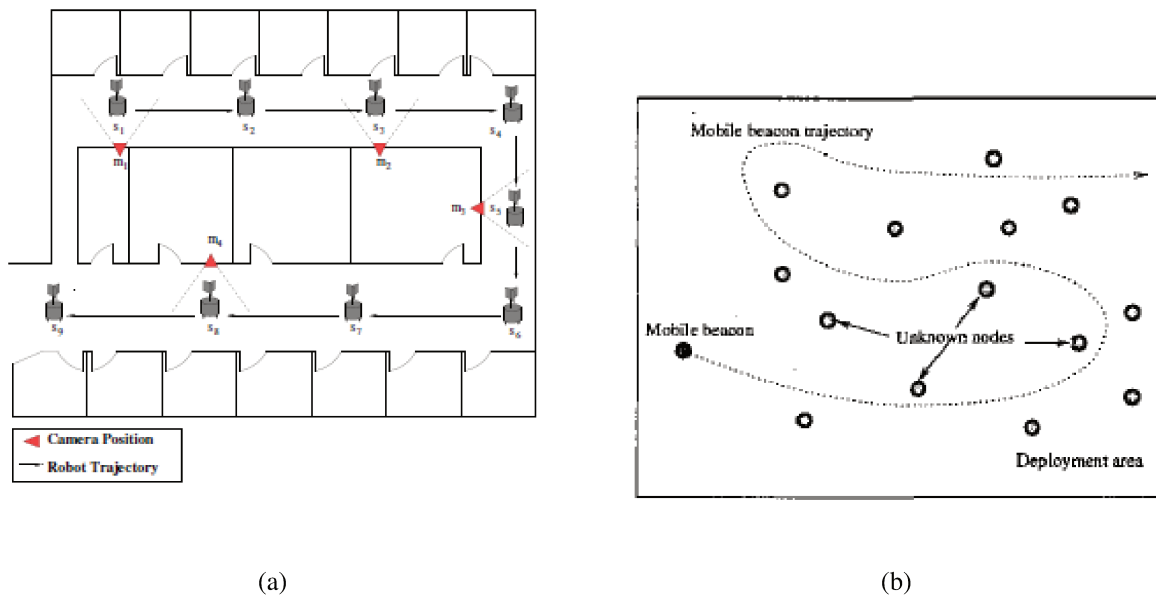


Fig. 2.24: Localização em uma rede de sensores: (a) utilizando visão [12] (b) utilizando beacons e RSSI [13]

sensores. Uma abordagem ao problema, apresentada em [52], utiliza um algoritmo simples para realizar o rastreamento, sem a necessidade de um grande volume de informações. Os nós da rede são equipados com um sensor binário, de alcance limitado e não muito confiável, que é capaz de detectar se o alvo encontra-se ou não ao seu redor. Quando a presença do alvo é detectada, o nó que realizou a detecção constrói uma mensagem contendo suas coordenadas, uma área aproximada de onde o alvo deve estar localizado, centrada em sua posição, e ainda uma informação sobre o momento em que a informação foi adquirida. Esta mensagem é transmitida por *broadcast* até o rastreador. As mensagens acompanham um TTL (*Time-To-Live*) para evitar uma inundação descontrolada na rede. Com a sua posição e uma estimativa da posição do alvo, que é atualizada a cada mensagem recebida da rede, o rastreador pode navegar até o alvo. Estes resultados também são apresentados em [53].

Talvez uma das aplicações mais intuitivas na união entre redes de sensores e robôs móveis seja a coleta e entrega de dados entre grupos de sensores e servidores [50]. Aplicações desta natureza usualmente são elaboradas em situações onde o posicionamento dos sensores ou grupos de sensores é muito distante dos servidores onde as informações serão processadas, ou disponibilizadas, e o custo

de utilizar um robô como *link* de comunicação é mais vantajoso do que conectar toda a rede através da instalação de mais nós, principalmente levando em consideração que a quantidade de energia gasta por transmissão é bem mais elevada do que em operações de sensoriamento ou processamento [54]. Este cenário pode ser visualizado na Figura 2.25.

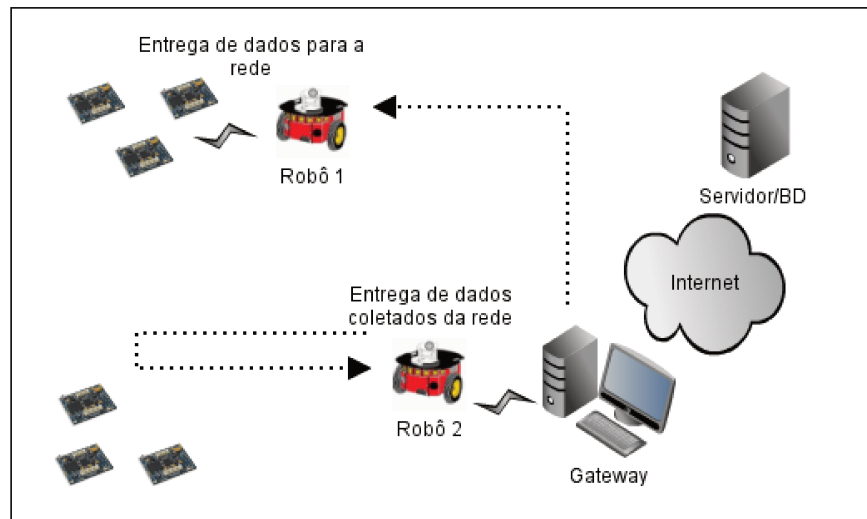


Fig. 2.25: Cenário onde robô móvel realiza a coleta e entrega de dados entre grupos de sensores e servidores.

Um outro grande grupo de aplicações está na área de navegação de robôs móveis. Diferentes propostas de navegação podem ser encontradas na literatura. Normalmente, estas propostas diferem umas das outras pela estrutura da rede, quantidade de informação transmitida e ainda quantidade de informações conhecidas, por exemplo, localização de robôs e sensores.

Uma rede de sensores estruturada, com posição conhecida, pode ser utilizada para guiar um robô móvel através de um ambiente [14]. Quando um objetivo é determinado (robô requer à rede ou rede requer presença do robô), o nó mais próximo daquela localização inicializa a computação do campo de navegação. Nesta fase cada nó determina a melhor direção a ser tomada pelo robô quando este está em sua vizinhança. A união de todas estas direções forma o campo de navegação, que fornece ao robô o melhor caminho possível para atingir o objetivo. A navegação, propriamente dita, é realizada de maneira discreta, ou seja, o caminho total é separado em alvos intermediários, neste caso os nós da

rede, Figura 2.26. O robô considera que atingiu um alvo intermediário e requer o próximo ao entrar na vizinhança deste nó. Isso é determinado utilizando leituras de RSSI. A navegação entre dois nós é realizada utilizando o algoritmo VHF (*Vector Histogram Field*) [55]. Esta solução, especificamente, não impõe restrições sobre cenários internos ou externos, porém, como parte da solução utiliza valores de RSSI para determinar a presença do robô, é de se esperar que o sistema seja mais estável em ambientes externos, já que o RSSI é muito influenciado pela reflexão das ondas, tornando-o ainda menos confiável em ambientes internos [56].

Uma solução para navegação voltada para ambientes internos é apresentada em [15]. Os nós, instalados em um ambiente como o ilustrado pela Figura 2.27, possuem posições conhecidas e são equipados com um emissor de rádio capaz de transmitir ondas eletro-magnéticas em frequências específicas. O robô, por sua vez, é equipado com uma antena direcional capaz de detectar os sinais enviados pelos nós da rede. Nesta implementação o robô também navega de nó em nó da rede, mas desta vez trocando informações sobre posição e distâncias. Além de navegar, o robô é capaz de se localizar realizando uma triangulação com os nós que estão ao seu redor.

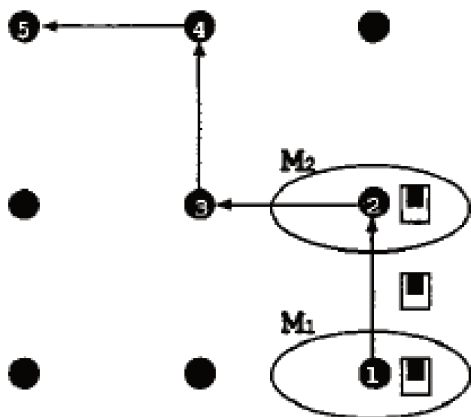


Fig. 2.26: Cenário da navegação com vizinhanças demarcadas [14].

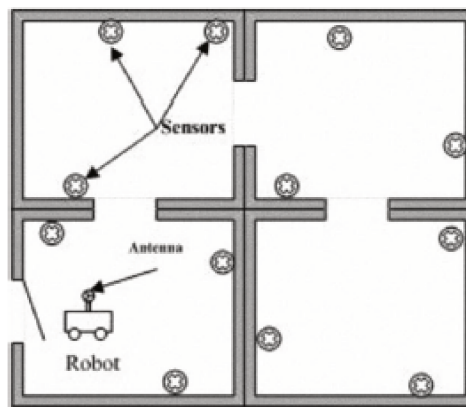


Fig. 2.27: Cenário interno: localização do robô feita por triangulação [15].

Uma outra abordagem para auxiliar a navegação de robôs móveis pode ser utilizar informações do ambiente, captadas pelos nós da rede, para influenciar os caminhos tomados pelo robô. Uma rede faz leituras de temperatura, umidade e, ao fundir essas informações com a altura dos nós, estabelece

a posição dos obstáculos do robô no ambiente [57]. Com esta informação um nó é capaz de informar ao robô para onde este deve se locomover. Neste caso, os nós também têm suas posições conhecidas. Existem duas formas para se determinar os caminhos: o mais seguro e o mais curto. Para navegar no caminho mais seguro o robô deve dar preferência aos nós com menor valor resultante da fusão de seus sensores. Já para navegar pelo caminho mais curto os nós preferenciais devem ser aqueles com menor distância para uma reta imaginária que liga a posição inicial do robô ao alvo.

Redes de sensores também podem ser utilizadas para sinalizar rotas de saídas de emergência tanto para humanos quanto para robôs [16]. Nesta implementação uma rede de sensores é instalada em um ambiente. Nós posicionados próximos a saídas de emergência recebem altitudes menores, enquanto à nós mais afastados são atribuídas altitudes maiores. A altitude de um nó nada mais é do que a quantidade de *hops* entre o dado nó e a saída mais próxima. Quando uma emergência é detectada, uma mensagem é disseminada pela rede informando-a da mesma. Nós próximos ao evento de emergência formam uma zona de perigo e as altitudes dos nós da rede são recalculadas levando em consideração também sua distância à emergência. Com o novo valor de altitude, os nós definem qual o próximo *hop* para a saída segura mais próxima, Figura 2.28.

Number	Simulation	Experiment	Packet count/ simulation experiment	Number	Simulation	Experiment	Packet count/ simulation experiment
1			25/26	4			39/47
2			36/40	5			38/44
3			40/44	6			36/40

Fig. 2.28: Traçado de rotas seguras de emergência [16].

Um cenário semelhante ao de buscas de rotas de emergência é o de atendimento a um chamado de emergência [17]. Dado a ocorrência de um evento, o robô pode ser requisitado a comparecer a

um certo local. Um evento pode ser definido como um pedido de ajuda enviado por um outro robô ou algum elemento da rede. Quando da ocorrência de um evento, o nó que o detectou envia uma mensagem requerendo a presença do robô. Esta mensagem é transmitida através de um “inundação organizada” de forma a diminuir a quantidade de informações desnecessárias que transitam na rede. A rede, ao transmitir esse pacote, define uma árvore de navegação por onde o robô deverá navegar, Figura 2.29. A navegação é realizada utilizando uma antena direcional para guiar o robô na direção do nó desejado.

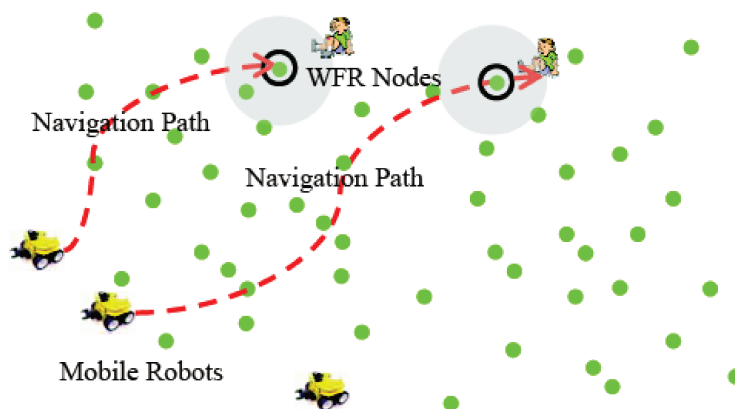


Fig. 2.29: Atendimento a chamadas de emergência ou resgate [17].

Sistemas de redes de sensores equipados com sensores acústicos também podem ser utilizados para auxiliar na navegação de robôs móveis, além da localização, como apresentado em [18]. Através dos princípios do efeito *Doppler* o sistema é capaz de localizar, estimar e corrigir as trajetórias tomadas pelo robô. O ambiente é composto por três elementos: nós receptores estáticos e de posição conhecida; um emissor secundário, também de posição conhecida; e o robô móvel equipado com o emissor principal (Figura 2.30). Os emissores transmitem sinais senoidais puros com frequências bem próximas, provocando uma interferência no sinal captado pelos receptores. Neste ponto, um filtro de Kalman estendido [58] é utilizado para realizar a localização e estimativa da trajetória atual do robô.

Redes em que sensores são equipados com sensores mais poderosos, como lasers ou câmeras,

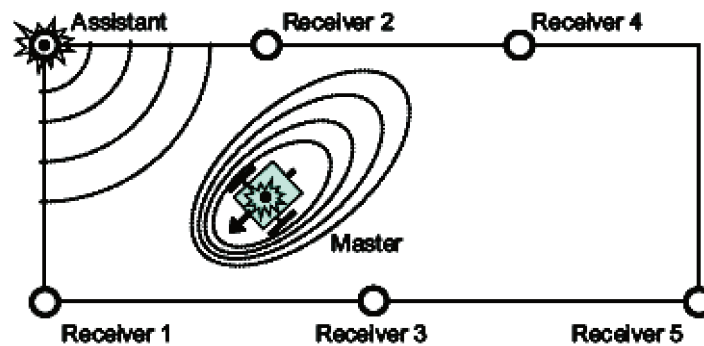


Fig. 2.30: Localização e Navegação utilizando o efeito *Doppler* [18].

podem ser utilizadas para realizar um mapeamento mais minucioso de um ambiente e determinar rotas para a navegação do robô. Neste cenário, a aplicação apresentada em [19] é capaz de mapear e guiar um robô através de um ambiente monitorado e demarcado com marcas que os sensores são capazes de detectar. Um nó é capaz de construir um mapa local da sua redondeza, incluindo todas as marcas presentes. Estes mapas locais, mais tarde, são fundidos em um único mapa global do ambiente. A união destes mapas é realizada utilizando as demarcações do ambiente detectadas pelos nós. As marcas presentes em duas áreas são utilizadas para uní-las e se tornam conhecidas como *relay-points*. A Figura 2.31 ilustra a união de dois nós vizinhos em um ambiente: os pontos brancos são as marcas na área do nó j , os pontos pretos são as marcas na área do nó i , enquanto os pontos cinzas são os *relay-points*.

Frameworks e Plataformas

Até este momento, foram apresentadas variadas aplicações que utilizam redes de sensores e robótica móvel. Um ponto em comum entre estas aplicações é que todas são desenvolvidas com o propósito de cumprir uma função específica, e muitas vezes necessitam de *hardwares* específicos. Uma outra abordagem, é o desenvolvimento de *frameworks* e plataformas na tentativa de prover um ambiente onde aplicações que utilizam robótica móvel e redes de sensores possam ser desenvolvidas

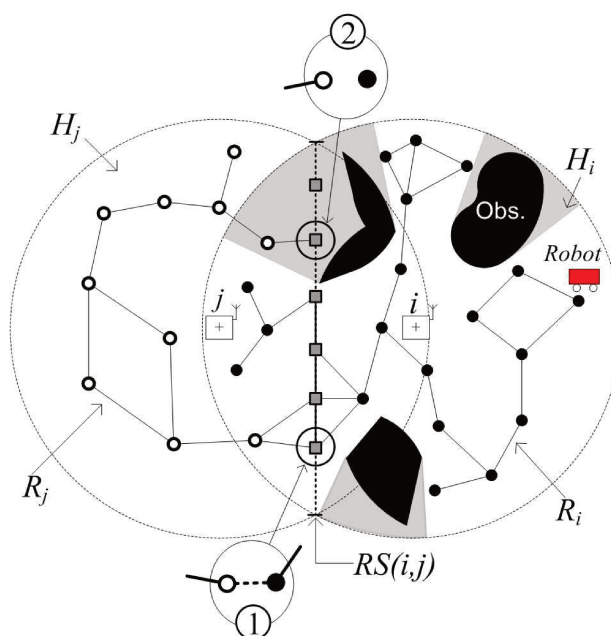


Fig. 2.31: Dois mapas locais sendo unidos através dos *relay-points* [19].

com um esforço reduzido, como é o caso do trabalho exposto no capítulo 4 desta dissertação.

Uma idéia interessante é a de utilizar um *framework* robótico, como o *Player* [59], para prover acesso não só a robôs em um ambiente, mas também aos recursos de uma rede de sensores, como apresentado em [20]. Para realizar esta interação, uma interface entre a rede de sensores e o *Player* foi desenvolvida para tornar transparente aos objetos da rede, e ao usuário, aspectos do funcionamento interno da rede de sensores como: formato de mensagens, protocolos, linguagem de programação, etc (Figura 2.32(b)). Além desta interface, um novo *driver* foi desenvolvido para o *Player*, específico para dispositivos da família *Xbow*. Com a proposta de unificar todo o acesso a objetos cooperativos (sensores de uma WSN, robô, etc.) por meio de chamadas do *Player*, um *testbed* foi construído contendo quarenta nós estáticos e seis robôs móveis, sendo cinco Pioneer 3-AT e um robô externo, Figura 2.32(a). O *testbed* foi validado através de execuções de diferentes tipos de experimentos.

Outra tentativa de desenvolver uma camada de *software* que dê suporte à integração entre redes de sensores e robôs móveis foi o objetivo do projeto europeu AWARE [60]. Mais especificamente, o objetivo do projeto foi o desenvolvimento de uma plataforma que permitisse a cooperação de UAVs com redes de sensores e atuadores terrestres compostas por nós estáticos e móveis [61]. O projeto

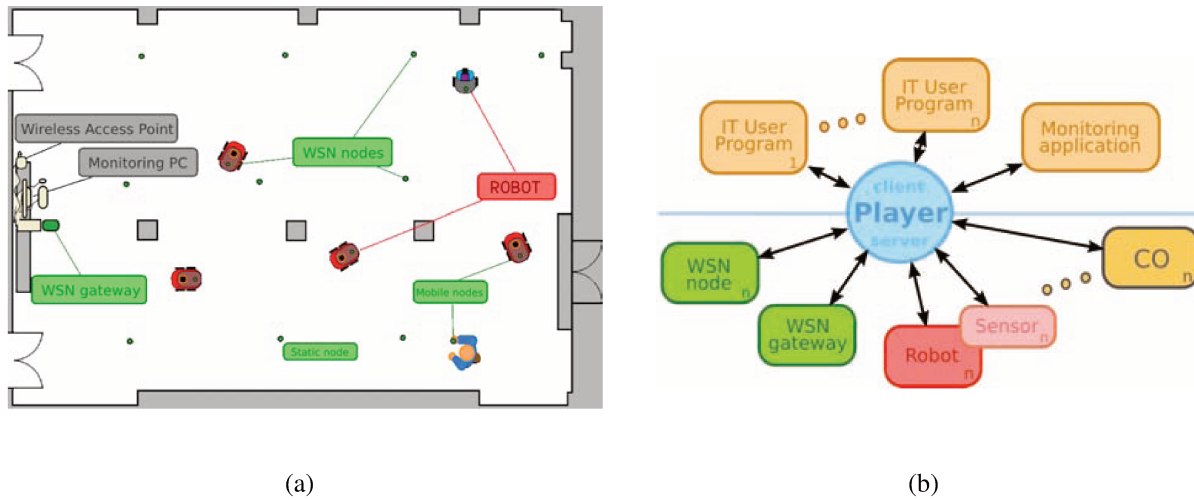


Fig. 2.32: (a) Visão geral do *testbed* (b) Modelo de interoperabilidade através do *Player* [20]

desenvolveu um *middleware* para integrar as informações obtidas por diferentes tipos de sensores provenientes de redes de sensores e robôs móveis [21]. A plataforma AWARE, como um todo, é formada por dois tipos de redes, uma rede de maior banda formada por redes Ethernet e WiFi (IEEE 802.3 e IEEE 802.11) e uma rede de menor banda formada pelos nós de uma rede de sensores. A comunicação entre as redes é realizada por meio de *gateways*, que podem ou não ser os robôs, Figura 2.33. A rede se organiza dinamicamente em grupos baseado na similaridade de suas leituras. Desta forma, a rede pode definir abstrações de “objetos” do ambiente, como um foco de incêndio. Essa abordagem também procura tornar a agregação de dados mais eficiente, já que cada grupo tem um líder que agrega os dados locais e o servidor precisa apenas recolher dados destes líderes. O sistema foi desenvolvido utilizando o TinyOS para as plataformas Tmote e Mica2.

A plataforma foi testada através de um experimento de *tracking*. Enquanto o robô navegava através do ambiente monitorado a rede de sensores deveria informar continuamente onde o robô deveria estar. Este rastreamento foi realizado montando grupos baseados em leituras de RSSI do robô, Figura 2.34.

Este capítulo realizou uma breve introdução dos principais conceitos envolvendo redes de sensores sem fio, além de uma discussão sobre aplicações e propostas para integrar esta tecnologia a robótica

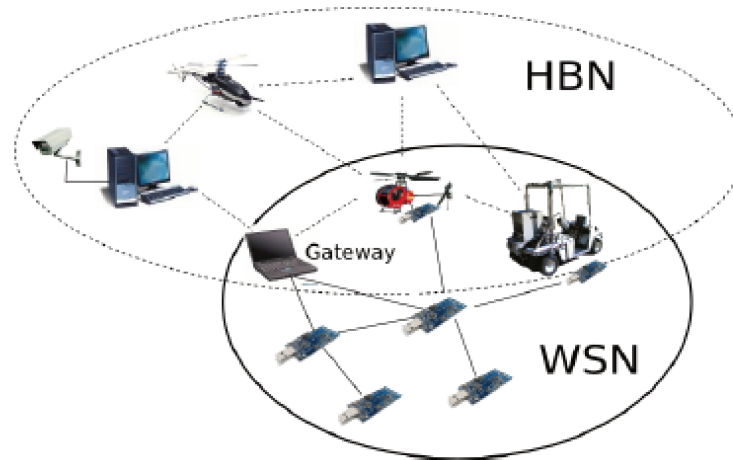


Fig. 2.33: Cenário da Plataforma AWARE [21].

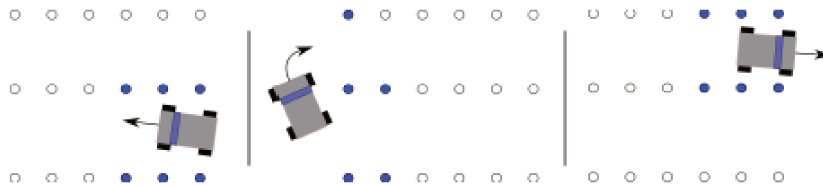


Fig. 2.34: Plataforma AWARE realizando *tracking* de robô móvel.

móvel.

Apesar da grande evolução na última década, constatou-se que ainda há muito espaço para evoluir na interação entre redes de sensores e robótica móvel, principalmente pelo fato que a grande maioria das pesquisas e implementações são muito específicas para determinadas aplicações e cenários. Tendo em mente estes aspectos, propõe-se uma plataforma para facilitar a interoperação entre as duas tecnologias como primeiro passo para uma integração não tão voltada a problemas específicos.

Capítulo 3

Plataforma REAL

Este capítulo introduz a plataforma REAL, que serviu como base para este trabalho, apresentada em [62]. A plataforma provê acesso a robôs móveis através da Internet, ou redes privadas, utilizando protocolos abertos, serviços e soluções de segurança vastamente utilizados por aplicações distribuídas atuais.

A plataforma é estruturada em quatro pacotes principais, como apresentado na Figura 3.1.

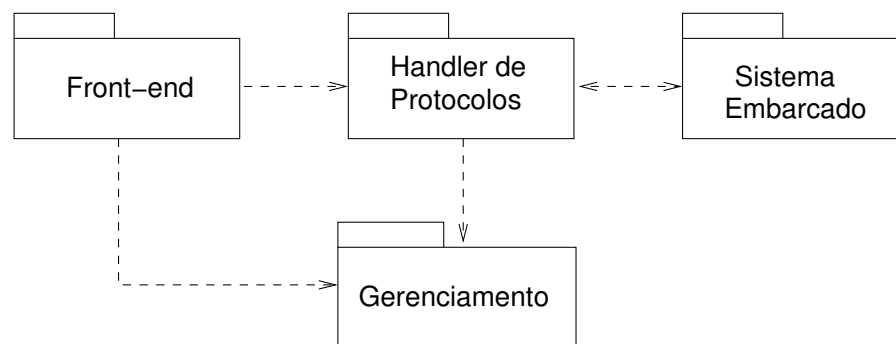


Fig. 3.1: Pacotes principais da plataforma REAL.

Dentro da arquitetura proposta para a plataforma, o pacote *Handler* de Protocolos desempenha funções como checagens de segurança, *proxying* e NAT. Este componente intercepta todos requerimentos direcionados aos recursos da plataforma (robôs, câmeras, sensores, etc), verifica se essas mensagens possuem as devidas credenciais de segurança e encaminha esses requerimentos ao recurso desejado, localizado numa rede privada. O serviço de *proxying* pode executar ainda uma “tradução”

de protocolos: por exemplo, um requerimento HTTPS é normalmente transformado num HTTP antes de ser encaminhado ao recurso. Em suma, este componente realiza as funções que requerem uma maior capacidade computacional como verificações de segurança e manipulação de protocolos.

Este pacote, porém, depende de roteadores ou servidores presentes entre a aplicação cliente e os recursos, já que operações de NAT e *firewall* são executadas por módulos das camadas mais baixas do *kernel*, como o *iptables* em roteadores que rodam Linux. A funcionalidade de *proxying* é executada por servidores HTTP. Além disso, estes servidores precisam suportar verificações de autorização e autenticação. Servidores Apache (*httpd*) permitem que este processamento extra seja realizado nas requisições antes destas serem encaminhadas ao recurso.

O pacote Gerenciamento implementa funcionalidades de gerência no nível de federações, domínios e recursos. A utilização de uma infraestrutura federada permite a definição de entidades de confiança para os domínios. Um gerenciamento típico no nível de federação executado por este pacote é a gerência e verificação de certificados. O gerenciamento de domínios permite a criação de relações confiáveis entre domínios dentro da federação. Tarefas de gerenciamento de domínios incluem: gerência de usuários, credenciais e certificados. Já o gerenciamento de recursos permite um acompanhamento e configuração de todos os recursos mantidos pelo domínio. Tarefas deste tipo de gerência englobam controle de acesso e reservas para uso de recursos.

As funcionalidades descritas acima para o pacote de gerenciamento usualmente são implementadas como serviços *Web* com acesso seguro, via HTTPS, e estão sujeitas a autenticação e autorização. Esses tipos de serviços são tipicamente suportados por *Web containers* como o Apache Tomcat ou o Oracle Glassfish. Mais especificamente, os serviços de autenticação podem ser baseados em sistemas como o Shibboleth ou o OpenSSO. Já o serviço de autenticação implementado na plataforma é baseado em SAML (*Security Assertion Markup Language*) [63], através do OpenSAML. Para cumprir os requisitos de autorização, o pacote oferece um serviço de reserva de recurso, por meio do qual o acesso ao mesmo é controlado, em conjunto com um serviço de controle de sessão. Todos os requerimentos provenientes de usuários devem conter um identificador de sessão, e se uma sessão válida não estiver presente na requisição, esta é bloqueada.

O pacote *Front-End* contém componentes que possibilitam a interação entre aplicações e recursos através da Internet ou redes privadas. Estes componentes oferecem interfaces de alto nível, APIs para a manipulação de robôs móveis em distintas linguagens de programação (C/C++, Java, J2ME, Python, Android, C# e MatLab). Essas APIs visam esconder das aplicações todas as interações entre protocolos existentes na execução de uma operação. As funcionalidades deste pacote são similares às providas por *frameworks* robóticos bem conhecidos como o ARIA [64] e o Player [59], porém os componentes deste pacote podem ser executados em *Web browsers* (como a aplicação de teleoperação apresentada na Figura 3.2), *soft phones* ou qualquer dispositivo que possua acesso a Internet. Outra importante característica é que mensagens geradas por este pacote não são bloqueadas por NAT ou *firewalls*.

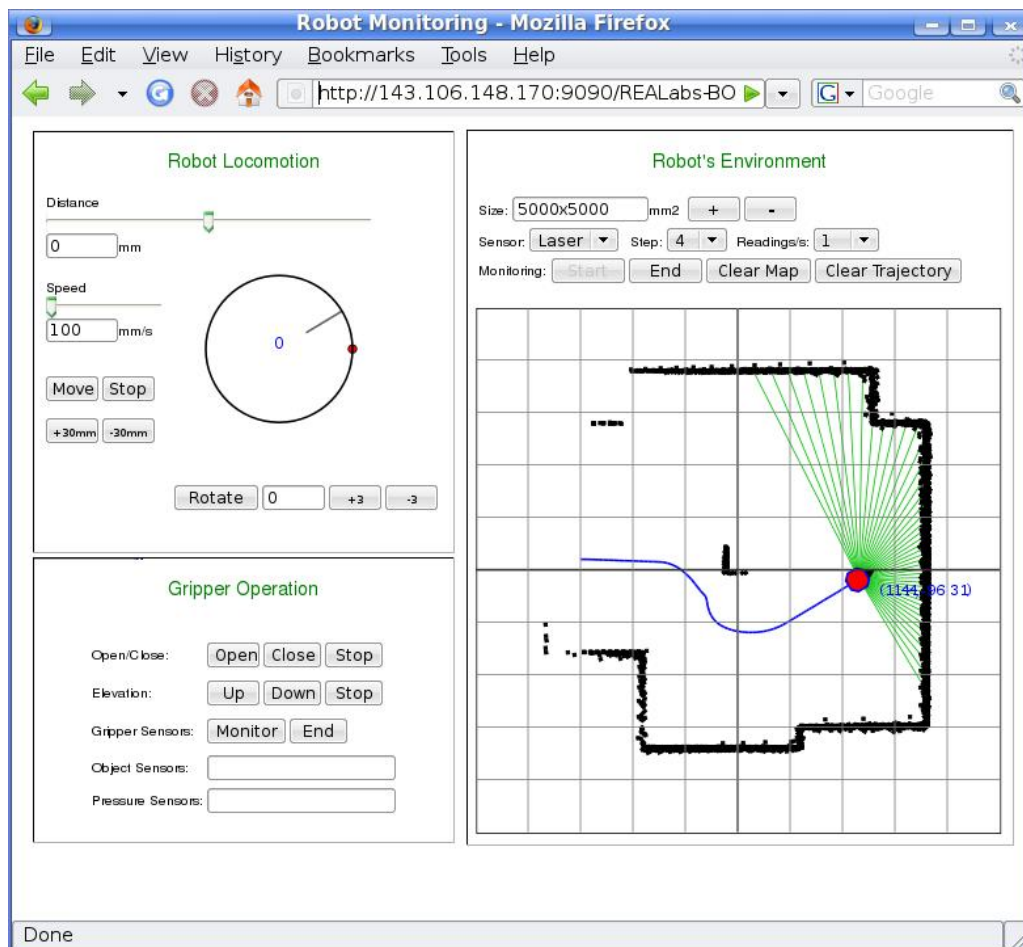


Fig. 3.2: Interface *Web* de teleoperação.

Por último, temos o Sistema Embarcado. Este pacote agrega todos os componentes que são executados embarcados nos recursos. É basicamente composto de microsistemas que executam nos processadores internos dos robôs. Apesar de robôs de grande porte possuírem maiores capacidades de processamento e memória, suficientes para executar servidores mais complexos, isso não é o caso quando tratamos robôs de pequeno e médio porte. Esta característica impõe restrições que precisam ser levadas em consideração durante o desenvolvimento dos microsistemas. Exemplos destas restrições são: requisitos de tempo real, segurança e interações com o *hardware* dos robôs.

As interações entre aplicações clientes e os microsistemas são realizadas utilizando protocolos bem conhecidos e difundidos, como HTTP e SIP. Características como: simplicidade de protocolos baseados em texto, maior facilidade de processamento utilizando *proxies* no lado do servidor e a existência de aplicações clientes bem difundidas (navegadores *Web* por exemplo) são fatores que contribuem para a escolha destes protocolos.

O modelo de comunicação cliente-servidor implementado na plataforma é baseado no modelo de interação REST (*Representational State Transfer*) [65]. Esse modelo faz uso de mensagens para consultar e alterar o estado de objetos mantidos em um servidor *Web*. Um objeto é uma estrutura de dados que representa uma entidade lógica, como uma entrada de banco de dados, ou física, como um robô. As mensagens utilizadas no modelo REST podem ser baseadas em qualquer protocolo, sendo HTTP e SOAP (*Simple Object Access Protocol*) os mais utilizados. SOAP é um protocolo complexo baseado em mensagens XML (*eXtensible Markup Language*) transmitidas utilizando protocolos da Internet, incluindo HTTP. No caso de REST sobre HTTP, sem SOAP, as mensagens já definidas para o protocolo HTTP são utilizadas na interação, principalmente GET, POST, PUT and DELETE. Na plataforma optou-se pelo uso de REST sobre HTTP ao invés de SOAP de forma a simplificar o lado do microservidor.

Além da comunicação cliente-servidor, também é necessário definir como se dá a interação entre o microservidor e o hardware do robô. A solução mais adotada é o uso de um *framework* robótico, como o ARIA ou Player como um intermediador. Nesses casos, o *framework* torna-se responsável por todas as interações com o hardware do robô, simplificando o processo de tradução das mensagens, contendo

operações desejadas pelos clientes, em comandos que possam ser processados pelo microcontrolador do robô. Normalmente as chamadas para esses *frameworks* são feitas através de C/C++.

Uma visão expandida da arquitetura da plataforma é apresentada na Figura 3.3.

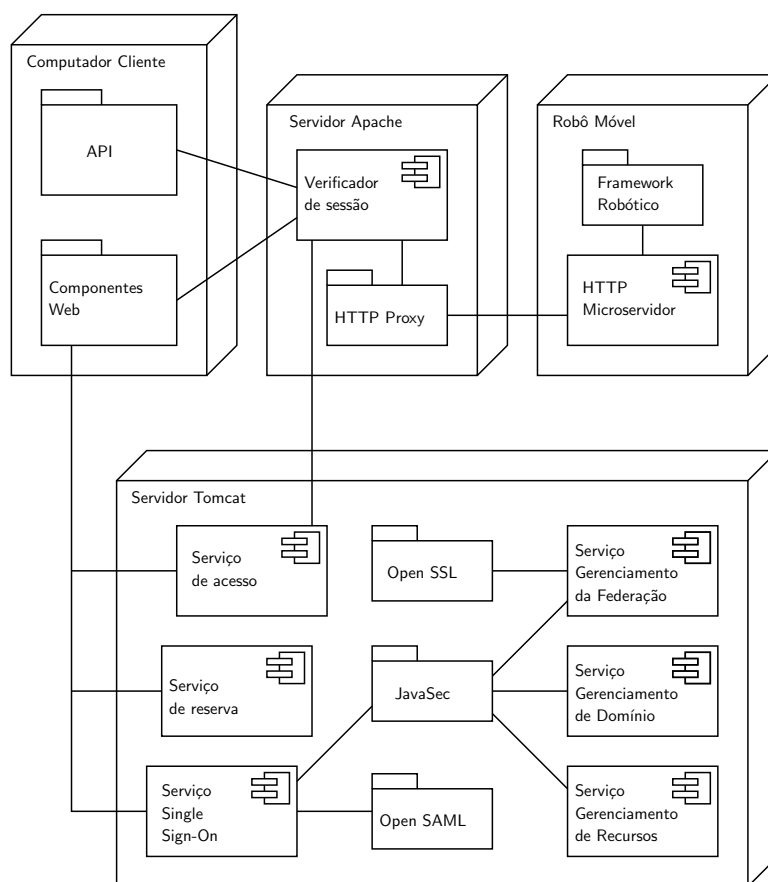


Fig. 3.3: Componentes da plataforma robótica.

Quando uma aplicação cliente envia uma mensagem para executar uma operação, por exemplo uma requisição de informações do robô (Quadro 3.1), essa mensagem chega primeiro ao *Proxy Server* onde os requerimentos de segurança são verificados. Após as checagens de segurança, o *Proxy* envia a mensagem para o robô desejado. Quando o microservidor HTTP recebe a requisição, é realizada uma análise na URL em que é determinada a operação a ser executada e seus parâmetros, se existirem. Uma vez adquiridas as informações necessárias, o microservidor faz uma chamada C/C++ da função referente a operação desejada; neste caso informações sobre o robô, com os parâmetros enviados, se houverem. Para cada chamada de uma função do *framework* robótico, uma resposta é enviada ao

microservidor. Essa resposta é encapsulada em um documento XML e enviada de volta à aplicação cliente. Neste caso a resposta contém informações sobre o robô, Quadro 3.2.

```
http://143.106.148.168:9090/Robot1/info.cgi?op=getAllInfo
```

Quadro 3.1: Requisição de informações sobre o robô.

```
<result >
  <sizes >
    <width >425</width >
    <length >511</length >
  </sizes >
  <sonar >
    <number >16</number >
  </sonar >
  <bumper >
    <front >true </front >
    <rear >true </rear >
  </bumper >
  <gripper >
    <hasgripper >true </hasgripper >
  </gripper >
  <laser >
    <readings >181</readings >
    <minangle >-90</minangle >
    <maxangle >91</maxangle >
    <stepangle >1</stepangle >
  </laser >
  <battery >
    <voltage >13</voltage >
  </battery >
</result >
```

Quadro 3.2: Resposta XML do microservidor com informações do robô.

As operações que podem ser executadas, através do microservidor, estão divididas em seis diferentes serviços. A Tabela 3.1 apresenta os serviços, e algumas de suas operações, suportados pela plataforma.

As APIs disponíveis no pacote *Front-End* suportam todas as operações disponibilizadas pelo

Tab. 3.1: Serviços suportados pela plataforma.

Serviço	Definição
Move Service	Operações relacionadas com o movimento do robô. (e.g., setVel, setHeading, getPosition)
Range Service	Operações relacionadas com o sensoriamento. (e.g., getSonarReadings, getLaserReadings)
Video Service	Operações de captura de vídeo e imagem (e.g., getPicture, getVideo)
Camera Service	Operações de configuração de cameras (e.g., setPan, setTilt)
Info Service	Serviço de informações (e.g., getRobotSizes, hasGripper)
Action Service	Comportamentos programáveis no robô (e.g., constantVelocity, avoidFront)

microservidor. O uso das APIs torna a construção das requisições HTTP, Quadro 3.1, e o processamento do documento XML, Quadro 3.2, transparentes para os usuários.

O modelo desta plataforma foi empregado para a construção de um laboratório de acesso remoto (*Weblab*) que foi então utilizado para a validação do modelo. Este *Weblab* visa realizar a integração entre diferentes domínios de forma transparente ao usuário. As Figuras 3.4 e 3.5 ilustram o ambiente utilizado no *Weblab*. Neste caso, foi realizada a integração entre dois domínios, a FEEC e o CTI, através de uma rede de alto desempenho, a Kyatera [66]. O *proxy* responsável por redirecionar as requisições para os recursos específicos está localizado num servidor Apache na FEEC, único ponto da infraestrutura conectado diretamente à internet. Esta infraestrutura foi utilizada na realização de experimentos por alunos de um curso de robótica móvel oferecido pelo programa de pós-graduação da FEEC da Unicamp.

A Figura 3.6 apresenta o resultado de uma aplicação de mapeamento de uma sala realizada como parte do curso de robótica móvel. Este resultado foi obtido realizando o acesso remoto ao robô através da plataforma REAL e suas APIs.

Este capítulo introduziu a plataforma REAL, na qual foi baseado o desenvolvimento apresentado no próximo capítulo. Junto à plataforma, sua arquitetura e serviços, também foram apresentados cenários onde a plataforma foi utilizada, bem como resultados de experimentos.

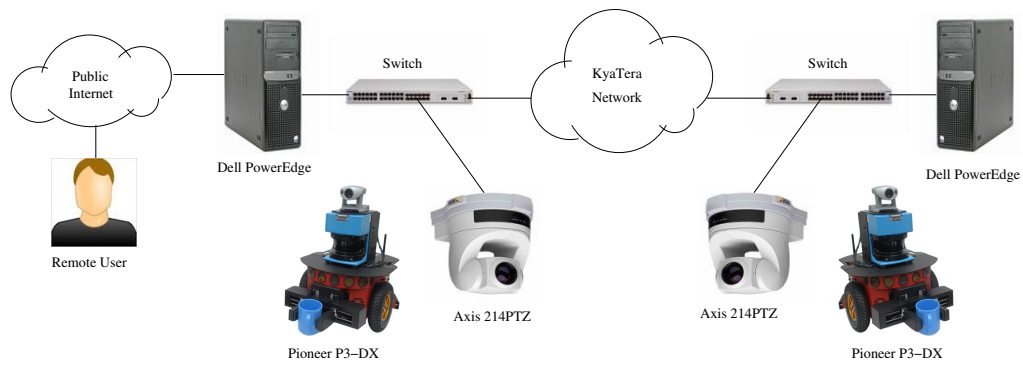


Fig. 3.4: Cenário do WebLab.

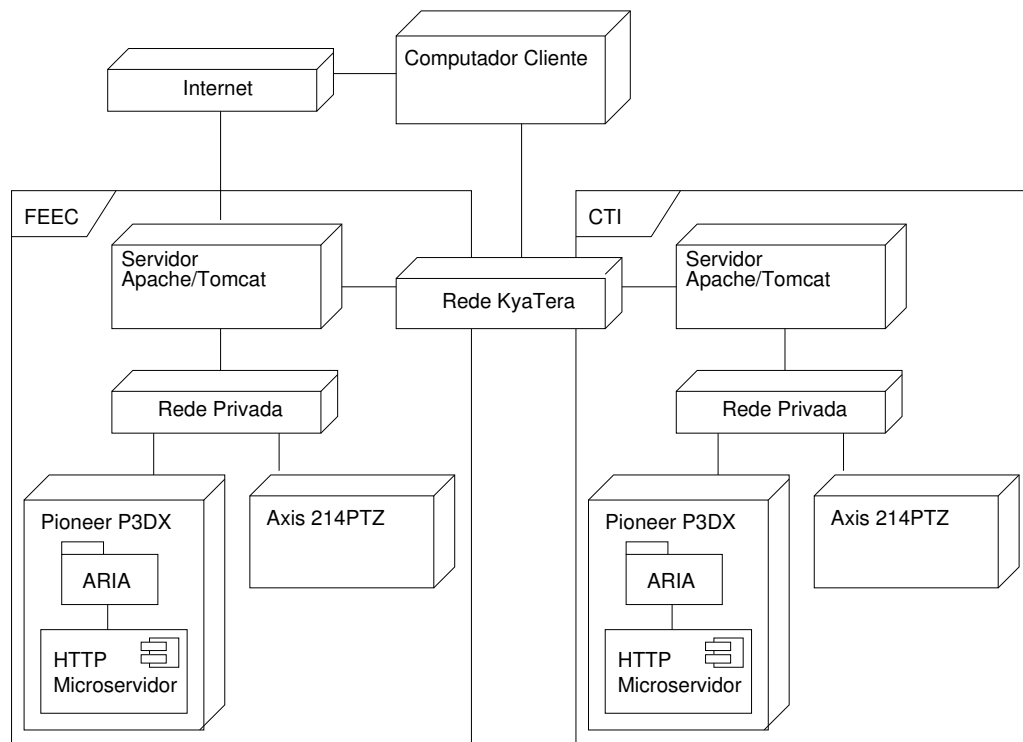


Fig. 3.5: Arquitetura do REALabs inter-domínios.

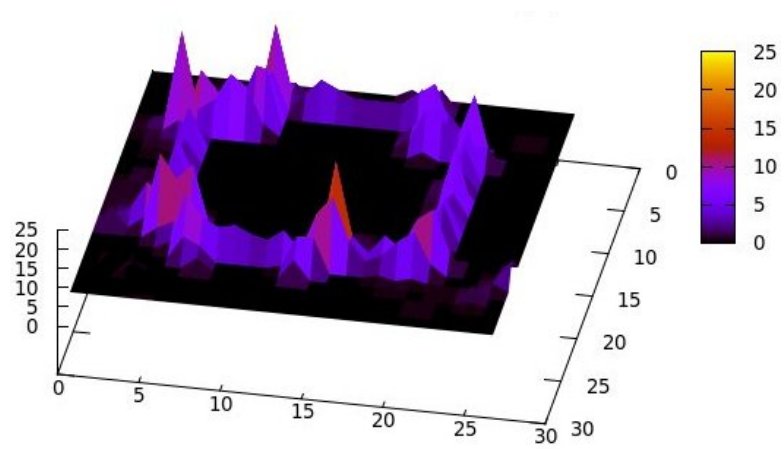


Fig. 3.6: Resultado de uma aplicação de mapeamento executada remotamente através da infraestrutura do *WebLab*.

Capítulo 4

Plataforma para Rede de Sensores

Este capítulo apresenta uma extensão da plataforma apresentada no Capítulo 3. Esta extensão tem como objetivo integrar redes de sensores e robótica móvel, mais especificamente, integrar as funcionalidades e a capacidade de sensoriamento diferenciada de dispositivos como rede de sensores à aplicações de robótica.

Dentre os requisitos desta nova arquitetura, destacam-se:

- Suportar a comunicação entre servidor e robô, ao menos em suas funcionalidades básicas, mantendo as alterações transparentes à plataforma e usuários. São tidas como funcionalidades básicas da plataforma as operações definidas pelos serviços de movimentação (*Move Service*), sensoriamento (*Range Service*) e informação (*Info Service*), Tabela 3.1;
- Estender o *Range Service* da plataforma com as capacidades de sensoriamento da rede, atendendo o formato já definido pelo serviço.

Desta forma, o sistema deve prover as funções de sensoriamento a todo momento e prover ainda um canal de comunicação, limitado, entre o robô e o servidor em momentos em que nenhum outro canal está disponível ou ainda quando aplicações demandam uma maior largura de banda. Por exemplo, em aplicações que utilizam alto tráfego de vídeo, que é uma informação que demanda muita banda, um canal de comunicação mais simples, como o de uma rede de sensores, pode ser

utilizado para transmitir os comandos para o robô paralelamente a uma rede Wi-Fi dedicada ao tráfego mais custoso. Além disso, todo o modelo de comunicação apresentado, XML sobre HTTP, deve permanecer inalterado (alterações permanecem transparentes tanto à plataforma quanto ao usuário).

4.1 Arquitetura da Plataforma

Para ser possível prover as funcionalidades exigidas pelos requisitos do sistema, um modelo arquitetural foi proposto para a plataforma, Figura 4.1.

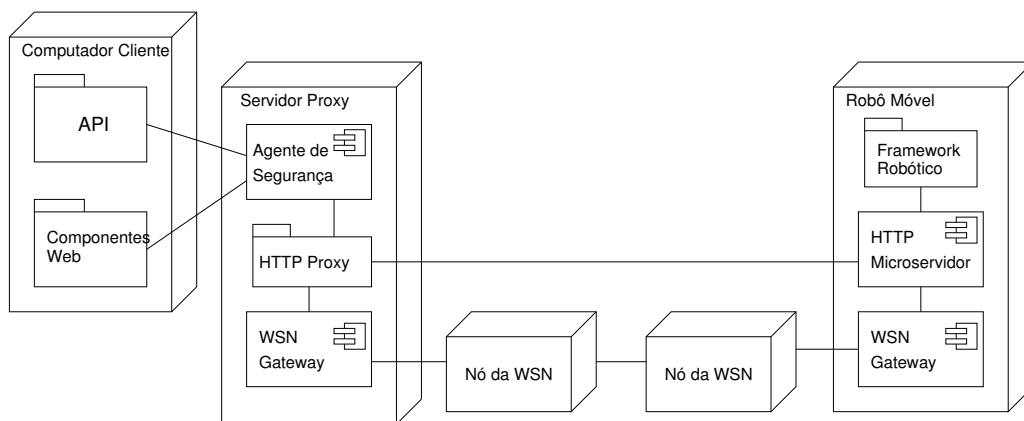


Fig. 4.1: Arquitetura da plataforma para suporte a rede de sensores.

No modelo proposto, existem dois componentes que desempenham as funções fundamentais: os *WSN Gateways* e os nós *WSN*. Os *WSN Gateways* são *softwares* que realizam a comunicação diretamente com a rede de sensores. Todos os pacotes trocados entre servidores e robôs devem passar primeiro por um dos *gateways*. O segundo componente corresponde aos nós conectados no servidor e no robô. O *software* presente nestes dois nós complementa as funcionalidades dos *gateways* dentro da rede propriamente dita. Estes componentes serão abordados na Seção 4.3.

Como dito anteriormente, um dos requisitos desta extensão da plataforma é a integração das capacidades de sensoriamento da rede de sensores ao *Range Service*. Isto implica que, quando o robô está em um ambiente monitorado por uma rede de sensores, mas a comunicação é realizada por uma rede Wi-Fi, o sistema ainda deve ser capaz de prover leituras dos nós da rede. Para prover essa

funcionalidade, a arquitetura do sistema tem que ser vista como apresentado na Figura 4.2.

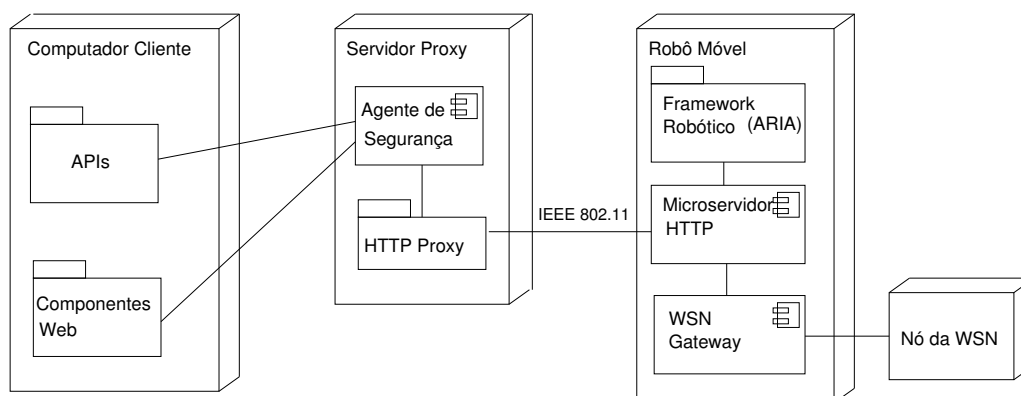


Fig. 4.2: Arquitetura suporte ao *Range Service* estendida pela comunicação Wi-Fi.

Neste cenário, o microservidor HTTP realiza um serviço de *proxy*. O microservidor precisa diferenciar e direcionar requisições a recursos distintos, dentro do mesmo serviço. Para tal, é necessário que o microservidor seja capaz de se comunicar com ambos, *framework* robótico e WSN *gateway*. A comunicação entre o servidor e o *framework* é realizada através de chamadas C/C++ diretas. Já para realizar a comunicação entre microservidor e *gateway* foram utilizadas chamadas HTTP. O motivo desta escolha foi a padronização do modelo de comunicação entre estes dois elementos, já que, visando manter essas alterações transparentes para a plataforma, quando toda a comunicação é realizada através da WSN, o *gateway* interage com o microservidor através de chamadas HTTP.

4.2 Protocolo de Comunicação

Um dos requisitos propostos para a extensão da plataforma é a possibilidade de utilização da rede de sensores como uma canal de comunicação entre robô e servidor ao invés da rede Wi-Fi já existente. Entre as principais limitações encontradas na troca de tecnologias de comunicação do IEEE 802.11 para IEEE 802.15.4, está a grande disparidade entre o volume de dados transmitidos por pacote nas duas. O tamanho padrão do *payload* em pacotes Wi-Fi, utilizando TCP, é de 1460 bytes, enquanto na implementação do IEEE 802.15.4 do TinyOS, o tamanho padrão do *payload* é de 28 bytes.

Devido a esta limitação, um protocolo de comunicação foi desenvolvido de forma a transmitir as informações necessárias pela rede de sensores, para manter a alteração de tecnologia transparente para recursos e clientes. Para a grande maioria das mensagens trocadas usualmente entre aplicações e o robô, a definição de mensagens simples é suficiente para realizar a comunicação. Por exemplo, para realizar uma operação de consulta a velocidade do robô, uma única mensagem é enviada, contendo a operação desejada, e uma mensagem é retornada contendo a velocidade do robô.

Para enviar as requisições contendo as operações desejadas, uma mensagem genérica foi definida, a qual contém, obrigatoriamente, um tipo, que define o serviço ao qual a operação pertence, e a operação. A mensagem possui também três campos para a inserção de parâmetros adicionais, caso seja necessário. A definição da mensagem de requerimento de operação é apresentada na Figura 4.3.

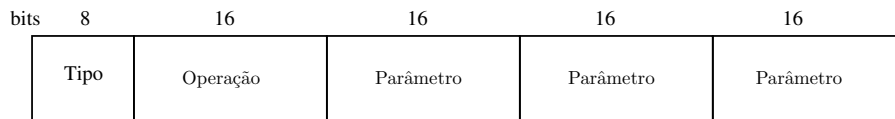


Fig. 4.3: Estrutura de uma mensagem de requisição de operação.

O Quadro 4.1 apresenta o resultado do encapsulamento de uma requisição de leitura de sonar para uma mensagem de requerimento de operação da rede de sensores.

```
http://RobotWSN/range.cgi?op=getSonarReadings&range=0:15:1
```

```
Message <OperationRequestMsg>
```

```
[msgType=0x2]
```

```
[operation=0x21]
```

```
[par1=0x0]
```

```
[par2=0x10]
```

```
[par3=0x1]
```

Quadro 4.1: Uma requisição de leitura de sonar e a respectiva mensagem para o protocolo da rede de sensores.

Para as respostas da maioria dos serviços, uma única mensagem, genérica, é suficiente para encapsular as respostas, mesmo que estas não sejam todas do mesmo formato. A Figura 4.4 mostra a definição desta mensagem. O Quadro 4.2 mostra a resposta de uma leitura da velocidade do robô no

formato XML da plataforma e da mensagem encapsulada no formato da rede de sensores.

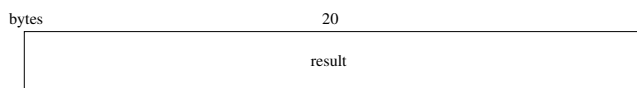


Fig. 4.4: Estrutura de uma mensagem de resposta genérica.

```
<result >200</result >
-----
Message <ResultMsg>
 [ result=0xC8]
```

Quadro 4.2: Resposta XML a uma requisição de velocidade do robô e a mensagem enviada pela rede de sensores

O serviço de informação, *Info Service*, também pôde ser implementado com apenas uma mensagem de resposta, porém uma mensagem específica para o serviço foi definida. A resposta para uma requisição das informações do robô foi apresentada no Quadro 3.2. Uma mensagem análoga ao XML de resposta a este serviço foi definida, Figura 4.5.

bits	16	16	16	16	16	16	16	16	16	16
	<i>Width</i>	<i>Length</i>	<i>NumSonar</i>	<i>FrontBumper</i>	<i>RearBumper</i>	<i>Gripper</i>	<i>LaserReadings</i>	<i>LaserMinAngle</i>	<i>LaserMaxAngle</i>	<i>LaserStepAngle</i>

Fig. 4.5: Definição da mensagem de resposta do *Info Service*.

Um dos requisitos para o emprego de uma rede de sensores como uma rede de comunicação em aplicações de robótica é o suporte aos serviços básicos da plataforma. Um destes serviços básicos é o *Range Service*. Este serviço provê funcionalidades de leituras de sensores, como sonar e laser, que normalmente envolvem o envio de grandes volumes de dados. O quadro 4.3 apresenta o resultado para a requisição de leitura de um único sonar. Uma leitura de laser pode conter até 180 destas leituras; isto resulta em um arquivo de 16Kbytes.

Apesar de as requisições para operações deste serviço serem similares às de outros serviços e poderem ser facilmente encapsuladas em uma única mensagem, numa rápida observação no Quadro 4.3, é fácil perceber que as respostas a estas requisições são grandes demais para serem encapsuladas em mensagens únicas. Desta forma, para ser capaz de enviar os dados da resposta XML, as informações precisam ser particionadas. Também é importante salientar que todas requisições de leituras de sensores, sonar ou laser, retornam acompanhadas da posição do robô no momento da leitura. Isto também pode ser observado no Quadro 4.3.

```
<result >
  <robotPosition >
    <x>0</x>
    <y>0</y>
    <th>0</th>
  </robotPosition >
  <readings >1</readings >
  <sensors >
    <sensor >
      <number>0</number>
      <sensorPosition >
        <x>69</x>
        <y>136</y>
        <th>90</th>
      </sensorPosition >
      <localPosition >
        <x>69</x>
        <y>5136</y>
        <th>0</th>
      </localPosition >
      <globalPosition >
        <x>69</x>
        <y>5136</y>
        <th>0</th>
      </globalPosition >
      <reading >5000</reading >
    </sensor >
  </sensors >
</result >
```

Quadro 4.3: Resposta XML do microservidor com leitura de um único sonar.

Assim, é possível interpretar que a principal tarefa do protocolo de comunicação é prover este serviço. Para tal, o protocolo foi definido da seguinte forma: uma resposta a qualquer requisição de leitura de sensor é composta por dois tipos de mensagem: uma mensagem de controle e mensagens de leitura de sensores. Por exemplo, a resposta para uma requisição de leitura de todos os sonares ocorre da seguinte forma. Primeiramente, uma mensagem de controle é enviada contendo informações sobre a resposta mais a posição do robô e em seguida, uma mensagem é enviada para cada leitura de sonar realizada, neste caso 16. Ou seja, cada documento XML de resposta de leituras de sensores é particionado em uma mensagem para cada sensor mais uma mensagem de controle.

Após a definição deste modelo, também se faz necessária a definição de quais informações sobre as leituras serão transmitidas. Cada leitura de sensor contém cinco informações: o número/identificador do sensor; a posição do sensor em relação ao robô; a posição da leitura em duas referências, a global e a local; além da distância lida propriamente dita. A posição do sensor em relação ao robô é fixa, para cada modelo, logo, esta informação pode ser armazenada localmente, ou requerida ao robô num estágio de inicialização. As posições da leitura tanto na referência global quanto local podem ser calculadas. Logo, dentre todas estas informações, as únicas que são indispensáveis são a distância lida e o identificador do sensor. Consequentemente, estas são as informações enviadas numa mensagem de leitura de sensor junto a um identificador de mensagem, como é apresentado na Figura 4.6.

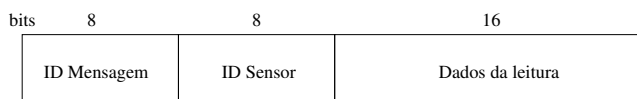


Fig. 4.6: Estrutura de uma mensagem de leitura de sensor.

Já a mensagem de controle contém, além da posição do robô no momento da leitura, o número total de leituras realizadas nesta requisição, para que o *gateway* possa controlar o recebimento das leituras. A estrutura da mensagem de controle é apresentada na Figura 4.7.

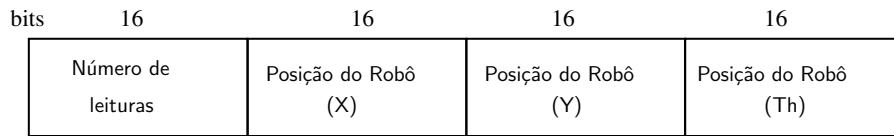


Fig. 4.7: Estrutura de uma mensagem de controle.

Utilizando este simples protocolo, a plataforma, agora, é capaz de prover suas funcionalidades básicas também através de uma rede de sensores.

Para a definição deste protocolo e de suas mensagens, foi assumido um cenário de execução simplificado, no qual existiria apenas um robô e uma aplicação cliente. Desta forma, a sincronia das requisições HTTP garantem a relação entre as requisições e suas respostas, não sendo necessária a definição explícita desta relação. Porém, em cenários onde exista mais de um robô, ou múltiplas aplicações acessando informações de um robô, é necessário garantir a relação entre requisição e resposta. Isto pode ser realizado utilizando um identificador único para cada requisição, definido pelo *gateway* do servidor. Este identificador seria carregado pela requisição e suas respostas, permitindo aos *gateways* encaminhá-las aos destinos corretos.

Além das funcionalidades básicas da rede, um serviço adicional também foi implementado, o *Action Service*. Este serviço define certos comportamentos do robô que podem ser ativados através do mesmo. Diferentemente dos anteriores, para este serviço, foi necessária a definição de uma mensagem específica, devido ao seu formato particular, Figura 4.8.

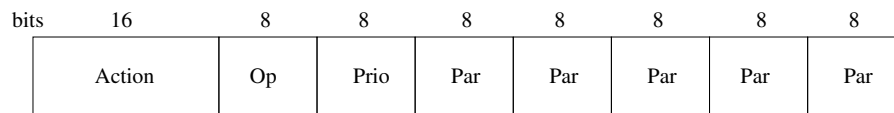


Fig. 4.8: Definição da mensagem para o serviço de ações.

As respostas deste serviço são normalmente encapsuladas na mensagem resposta padrão, Figura 4.4.

4.2.1 TinyAPI

Uma vez definidas as mensagens necessárias para ser possível suportar os serviços básicos da plataforma pela rede de sensores, surge a possibilidade de disponibilizar uma API simplificada para que aplicações que executam embarcadas na rede possam também atuar diretamente no robô. As mesmas mensagens definidas pelo protocolo podem ser utilizadas, o que garante a transparência para os robôs sobre quem está executando a operação.

Desta forma, foi desenvolvida uma API simples para aplicações TinyOS, a TinyAPI. Esta versão extremamente simplificada da API, por enquanto, é capaz de prover apenas dois serviços da plataforma, o de movimentação e o de ações. A escolha destes dois serviços se deu pela possibilidade da implementação de aplicações que realizem a navegação do robô realmente controlada pela rede, nas quais comportamentos de segurança, como evitar obstáculos frontais, podem ser instanciados no robô através do serviço de ações, enquanto comandos de navegação são enviados por nós da rede.

4.3 Gateways

Dentro da arquitetura proposta, os *gateways* desempenham um papel fundamental. Eles são os responsáveis pela maior parte das funcionalidades da plataforma quando executando através de uma rede de sensores.

No lado do servidor, a figura do *gateway* é desempenhada por um servidor HTTP, implementado em Java, que possui a mesma interface CGI (*Common Gateway Interface*) que o microservidor executando embarcado no robô. Em cenários onde a rede de comunicação é provida pela rede de sensores, quando da chegada de uma requisição no servidor, o agente *proxy* encaminha a requisição ao *gateway* em vez de diretamente para o robô. Cabe, então, ao *gateway* enviar a requisição e processar seu retorno.

No robô, outro programa Java implementa as funcionalidades do *gateway*. Diferentemente do *gateway* presente no lado do servidor, este *software* implementa funcionalidades, tanto em cenários

onde a rede de sensores é o canal de comunicação quando a rede Wi-Fi é utilizada.

No primeiro cenário, onde a rede de sensores é utilizada como canal de comunicação, o *gateway* pode ser visto como um cliente HTTP embarcado no robô, Figura 4.9(a). Quando uma requisição chega ao *gateway*, e é referente a uma operação a ser executada no robô, como alterar velocidade ou ler sonares, uma requisição HTTP parte do *gateway* para o microservidor, a resposta XML é processada pelo *gateway* e enviada pela rede de sensores. Se a requisição for para leitura de sensores da rede, o *gateway* retorna uma requisição de leitura de sensores da rede ao nó do robô que executa a operação.

Já no segundo cenário, onde a comunicação é realizada por uma rede Wi-Fi, o *gateway* pode ser visto como um servidor HTTP, Figura 4.9(b). Quando requisições chegam ao robô pedindo leituras de sensores da rede, o microservidor realiza uma requisição HTTP para o *gateway*, que repassa mensagens para a rede através do nó conectado ao robô e aguarda as mensagens de retorno. Quando a resposta está completa, um XML é retornado ao microservidor contendo as informações requeridas.

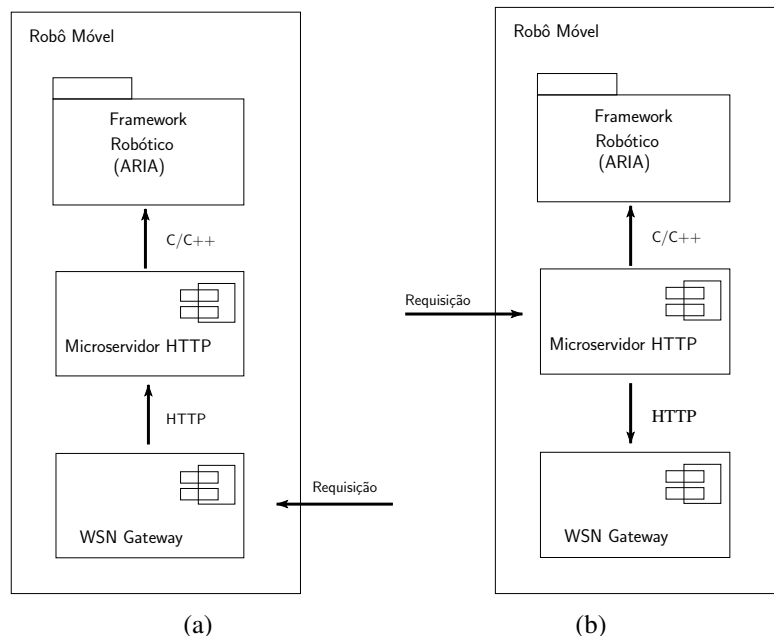


Fig. 4.9: Comunicação interna do robô em diferentes cenários. (a) Comunicação pela rede de sensores. (b) Comunicação pela rede Wi-Fi.

Como foi exposto na Seção 4.2, cabe aos *gateways*, mais especificamente ao *gateway* do servidor,

adquirir, calcular e inserir as informações restantes ao XML de resposta à requisições de leitura de sensores, sonar e laser. As informações sobre as posições dos sensores no robô foram armazenadas localmente junto ao *gateway* do servidor. Ao desencapsular uma mensagem de leitura de sensor e recuperar o identificador referente aquela leitura, o gateway utiliza a leitura do sensor, aliada à posição do sensor no robô, para determinar a posição do obstáculo lido pelo sensor, na referência local e global. A Figura 4.10 apresenta o modelo utilizado para realizar estas conversões de referências.

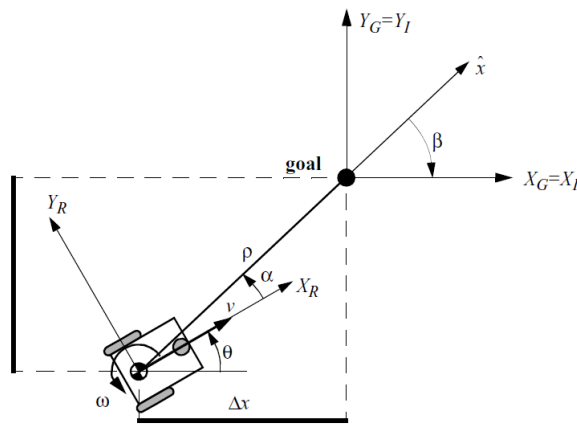


Fig. 4.10: Modelo para transformação entre referências [22].

A partir deste modelo, é possível determinar as equações utilizadas para o cálculo da leitura na referência local, Equação (4.1), e global, Equação (4.2). Como a posição de interesse é uma leitura do sensor, esta não necessita de orientação.

$$Pos_{local} = \begin{cases} X_{local} = pos_sensor(x) + leitura \cdot \cos(pos_sensor(\theta) \frac{\pi}{180}) \\ Y_{local} = pos_sensor(y) + leitura \cdot \sin(pos_sensor(\theta) \frac{\pi}{180}) \end{cases} \quad (4.1)$$

E:

$$Pos_{global} = \begin{cases} X_{global} = pos_robo(x) + cos_r \cdot X_{local} + sen_r \cdot Y_{local} \\ Y_{global} = pos_robo(y) + cos_r \cdot Y_{local} - sen_r \cdot X_{local} \end{cases} \quad (4.2)$$

Onde:

$$\cos_r = \cos\left(-pos_robo(\theta)\frac{\pi}{180}\right) \quad (4.3)$$

$$\sin_r = \sin\left(-pos_robo(\theta)\frac{\pi}{180}\right) \quad (4.4)$$

4.4 Sensoriamento do Ambiente

Outro importante requisito para o sistema foi o da integração das capacidades de sensoriamento dos dispositivos da rede de sensores com o modelo de leitura de sensores empregado na plataforma. Isto implica que a requisição deve seguir o mesmo formato CGI de uma requisição de leitura de sonar ou laser. O Quadro 4.4 apresenta uma requisição típica de leitura de sonares.

```
http://143.106.148.168:9090/Robot1/range.cgi?op=getSonarReadings&range=0:15:1
```

Quadro 4.4: Requisição de leitura de todos os sonares.

Na URL apresentada, existem três informações essenciais para a execução da operação:

- `range.cgi` → informa a qual serviço a operação pertence;
- `op=getSonarReadings` → contém o nome da operação desejada;
- `range=0:15:1` → contém o intervalo desejado para as leituras; neste caso, as leituras do sensor 0 ao 15, de um em um.

Para realizar leituras de sensores da rede, as requisições devem seguir este modelo e, por se tratar de leitura de sensores do ambiente, as operações definidas devem pertencer ao *Range Service*. Foram definidas operações referentes a cada um dos sensores disponíveis na rede: *getLightReadings*, *getHumidityReadings*, *getTempReadings*, *getAccelReadings* para realizar leituras de luminosidade, umidade, temperatura e do acelerômetro, respectivamente. O campo *range*, definido para a determinação do intervalo de leitura, desempenha função similar para estes tipos de leitura. É através

deste campo que é possível definir em qual dispositivo a leitura deve ser realizada, ou se em todos ao alcance do nó do robô. O Quadro 4.5 apresenta uma requisição para a leitura da intensidade luminosa em todos os nós em uma região. Para realizar a leitura de uma região, foi definido o endereço 255, como uma analogia ao *broadcast* em redes IP.

```
http://143.106.148.168:9090/Robot1/range.cgi?op=getLightReadings&range=255:0:0
```

Quadro 4.5: Requisição a leitura de luminosidade de todos os sensores em uma região.

Para direcionar a leitura a um nó específico, basta alterar o valor 255 para o identificador do nó desejado, no parâmetro *range*.

As maneiras como uma requisição individual e uma requisição de leitura de uma região são realizadas são distintas. A requisição de leitura de uma região, por definição, será executada no robô, já que supõe-se a região ao redor do robô como área de interesse, Figura 4.11. A requisição chega ao *gateway* do robô e é processada. Quando é determinado que a requisição é de leitura de sensor da rede, uma requisição específica é enviada ao nó do robô contendo: o tipo de sensor que deseja ser lido e o identificador deste. No caso de uma leitura de região (*range=255:0:0*), o nó faz um *broadcast* da requisição a todos os seus vizinhos e aqueles que receberem a mensagem realizam a leitura e enviam a resposta ao nó do robô, que a repassa ao *gateway*. Cabe, então, ao *gateway* verificar se a requisição foi realizada pela rede Wi-Fi ou WSN, e encaminhar a resposta pela rede apropriada. No caso de uma leitura individual, o identificador é propagado pela rede até atingir o nó desejado, que realiza a leitura e a envia ao nó do robô.

Uma das características de uma rede de sensores é a possibilidade de falhas de nós. Dado que o processo de manter os estados de vizinhos é uma tarefa com custo elevado, o processo de leitura de uma região é realizado utilizando um relógio. Após o *broadcast* da requisição, o relógio é inicializado, e as respostas recebidas antes do expiração deste relógio são repassadas ao servidor. Um diagrama de sequência detalhando toda a comunicação durante o processo de leitura de uma região é apresentado na Figura 4.12.

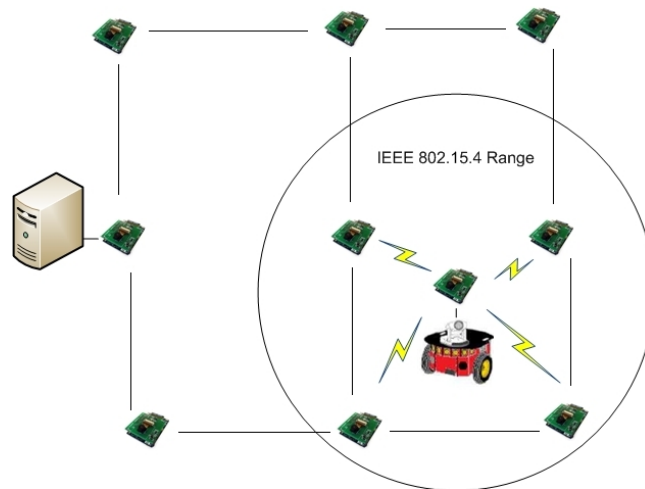


Fig. 4.11: Cenário de leitura da região ao redor do robô.

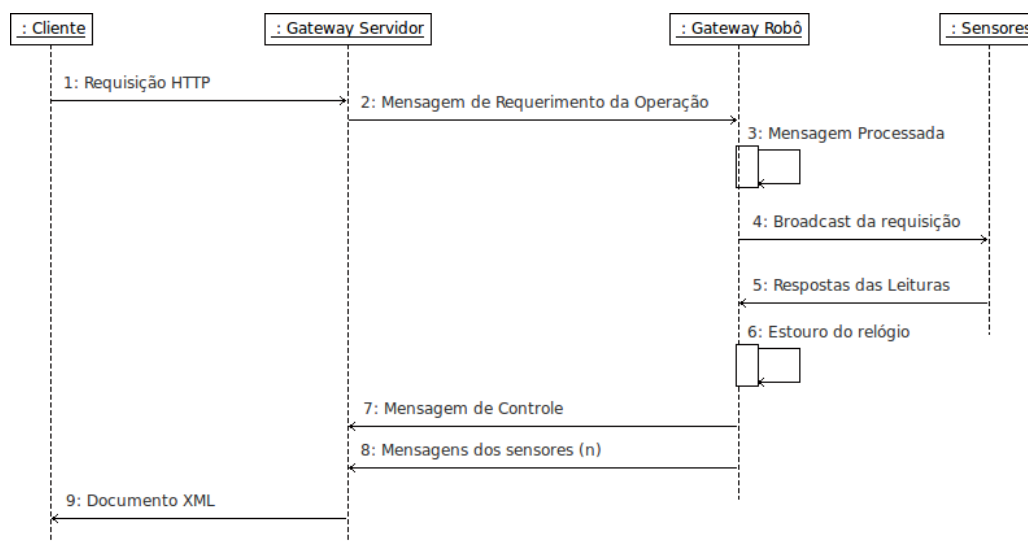


Fig. 4.12: Diagrama de sequência da operação de leitura de uma região.

Assim como a requisição, a resposta deste serviço, ou seja, o documento XML resultante, também precisa seguir o modelo definido pela plataforma (Capítulo 3). Isso significa que uma resposta a uma leitura dos sensores deve conter, pelo menos, a posição do robô no momento da leitura, o identificador do nó, a posição do nó (se houver) e a leitura do sensor especificado. Existem duas diferenças fundamentais relativas às características destes sensores, em relação aos sensores do robô, que impactam na definição das mensagens. Diferentemente de um sonar, por exemplo, a leitura de um sensor da rede não possui nenhuma relação com a posição do dispositivo no ambiente. Também

não existe nenhuma informação que possa ser relacionada à posição de um sensor que possa ser armazenada localmente no servidor, como a posição de sensores de sonar no robô.

Devido a essas diferenças, a definição de uma mensagem de leitura de sensor da rede deve conter campos para a informação da posição deste sensor, para ser utilizada em aplicações onde essa informação está disponível. Esta definição implica que, em aplicações utilizando esta plataforma, a informação de posição de um elemento é mantida pelo próprio elemento, quando esta existir.

A mensagem definida para as respostas de leituras de sensores da rede é apresentada na Figura 4.13.



Fig. 4.13: Mensagem de leitura de sensores da rede.

4.5 Comunicação Robô-Servidor

Após a definição do protocolo para suportar os serviços da plataforma e da especificação de todas as funcionalidades dos *gateways*, ainda é necessário definir como será realizada a comunicação entre robô e servidor. Como, em uma rede de sensores, realizar a manutenção constante de rotas é uma tarefa com custo bastante elevado, foi tomada a decisão que apenas os nós de interesse teriam suas rotas mantidas: o nó do servidor e o nó do robô. Esta decisão também elimina o *overhead* devido ao traçado sob demanda de rotas. Para realizar a comunicação, foi utilizado um protocolo de roteamento em árvore, nativo do TinyoS, chamado CTP (*Collection Tree Protocol*) [67].

Neste protocolo, todos os nós de uma árvore determinam uma rota à raiz desta árvore. Desta forma, uma árvore foi construída para cada um dos nós de interesse da plataforma. Os nós, periodicamente ou quando ocorre alguma alteração em suas tabelas, trocam mensagens com seus vizinhos buscando realizar a manutenção destas rotas. É fácil concluir que este não é um algoritmo que busca suportar mobilidade, porém, no cenário proposto, além de apenas existir um nó móvel,

este se move em velocidades relativamente baixas. Desta forma, alterar o nó do robô para enviar mensagens de manutenção de rota com maior frequência do que o restante da rede é suficiente para que garantir a funcionalidade da solução.

A Figura 4.14 apresenta a arquitetura de comunicação básica da plataforma.

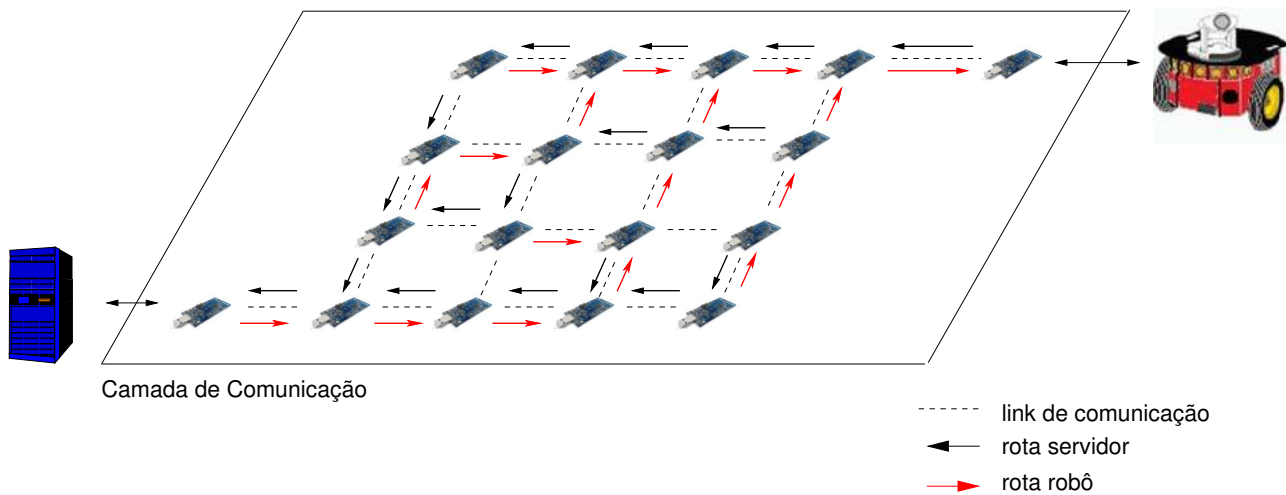


Fig. 4.14: Arquitetura de comunicação da plataforma.

O protocolo CTP usa uma medida de transmissões esperadas, ETX (*Expected Transmissions*), como um gradiente para a construção das árvores de roteamento. Um nó calcula seu ETX em uma árvore, somando o valor do ETX do seu nó pai ao valor do ETX do *link* entre o mesmo e o nó pai. Com o cálculo dos ETX, um nó escolhe como rota ao nó raiz aquele caminho com menor ETX. O valor do ETX de um nó raiz é nulo.

Outro ponto importante quando tratamos de comunicação e roteamento em redes de sensores sem fio é a perda de pacotes. Este problema foi solucionado também nos *gateways*, utilizando relógios para detecção de *timeouts*. Esta solução foi adotada por dois motivos: primeiramente pela maior simplicidade de implementação e segundo, e mais importante, como várias das informações possuem uma íntima relação com o momento em que esta foi adquirida, é mais vantajoso repetir a requisição e adquirir informações mais relevantes do que tentar recuperar as informações que foram perdidas pela rede.

O problema de perda de pacote torna-se um pouco mais complexo quando tratamos o serviço de

sensoriamento, o *Range Service*. Neste caso, existem três possíveis cenários:

1. Pacote da requisição é perdido: quando a mensagem contendo a requisição não é entregue ao robô, a requisição é repetida com a expiração do relógio de *timeout*;
2. Mensagem de controle é perdida: uma mensagem de controle perdida se caracteriza quando mensagens referentes a sensores são recebidas pelo servidor quando nenhuma mensagem de controle foi recebida. Neste caso, todas as mensagens são descartadas e uma nova requisição é realizada com a expiração do relógio;
3. Mensagem de leitura de sensor é perdida. Neste cenário existem duas possibilidades: a última mensagem é perdida ou alguma mensagem intermediária é perdida. Quando uma mensagem é recebida com o último ID esperado é recebida, e o número total de mensagens recebidas é menor do que o esperado, uma nova requisição é enviada imediatamente. No caso de a última mensagem ser perdida, o reenvio da requisição ocorre com a expiração do relógio.

Este capítulo apresentou uma proposta para integrar redes de sensores sem fio a robótica móvel através de uma extensão da plataforma REAL. Foram apresentados: uma nova arquitetura para a plataforma, um protocolo para tornar possível o envio das informações necessárias pela rede de sensores e uma arquitetura de rede que possibilitar a comunicação entre robô e servidor. Dentro de cada um dos pontos introduzidos foram levantados os pontos positivos, que justificaram as escolhas realizadas, e os pontos negativos, que servem como inspiração para trabalhos futuros, das soluções propostas.

Capítulo 5

Simulações e Experimentos

Este capítulo apresenta resultados de diferentes simulações e experimentos realizados para validar as soluções de comunicação e inclusão das capacidades de sensoriamento de uma rede de sensores sem fio à plataforma REAL, apresentadas no Capítulo 4.

Todas as simulações de redes de sensores apresentadas foram realizadas utilizando o simulador do TinyOS, o TOSSIM. A Figura 5.1 ilustra a execução de uma simulação. Em todas as simulações apresentados nesta dissertação, foi utilizado um modelo de propagação gerado a partir de leituras realizadas na biblioteca da Universidade de Stanford, utilizando o algoritmo CPM (*Closest-Fit Pattern Matching*) [68]. Este algoritmo recebe as leituras de ruído como entrada e gera modelos estatísticos, para propagação.

Para as simulações de ambientes robóticos, foi utilizado o simulador MobileSim 0.5.0 [69]. Mapas de ambientes criados pelo usuário podem ser carregados e utilizados em simulações. A Figura 5.2 apresenta o MobileSim aberto e carregado com uma mapa retratando uma sala com obstáculos nos quatro cantos.

```

Applications Plask System
ricardo@alpha-60: opt/tinyos-2.1.0/apps/RadioCountToLedsc
file Edit View Terminal Help
ricardo@alpha-60: opt/tinyos-2.1.0/apps/RadioCountToLedsc$ python test.py
0 2 -68.0
2 0 -07.0
1 2 -54.0
2 1 -55.0
1 3 -68.0
3 1 -68.0
2 1 -64.0
3 2 -64.0
Creating noise model for 1
Creating noise model for 2
Creating noise model for 3
DEBUG (1): Application booted again.
DEBUG (1): Application booted a third time.
DEBUG (2): Application booted again.
DEBUG (2): Application booted a third time.
DEBUG (3): Application booted again.
DEBUG (3): Application booted a third time.
DEBUG (1): RadioCountToLedsc: timer fired, counter is 1.
DEBUG (1): RadioCountToLedsc: packet sent.
DEBUG (2): RadioCountToLedsc: timer fired, counter is 1.
DEBUG (2): RadioCountToLedsc: packet sent.
DEBUG (3): RadioCountToLedsc: timer fired, counter is 1.
DEBUG (3): RadioCountToLedsc: packet sent.
DEBUG (1): Received packet of length 2.
DEBUG (1): Time: 0:0:0.258324115
DEBUG (1): Received packet of length 2.
DEBUG (3): Received packet of length 2.
DEBUG (3): Time: 0:0:0.251184895
DEBUG (2): Received packet of length 2.
DEBUG (2): Time: 0:0:0.253872882
DEBUG (1): Received packet of length 2.
DEBUG (1): Time: 0:0:0.253872882
DEBUG (1): RadioCountToLedsc: timer fired, counter is 2.
DEBUG (1): RadioCountToLedsc: packet sent.
DEBUG (2): RadioCountToLedsc: timer fired, counter is 2.
DEBUG (2): RadioCountToLedsc: packet sent.
DEBUG (3): RadioCountToLedsc: timer fired, counter is 2.
DEBUG (3): RadioCountToLedsc: packet sent.

```

Fig. 5.1: Simulador TOSSIM.

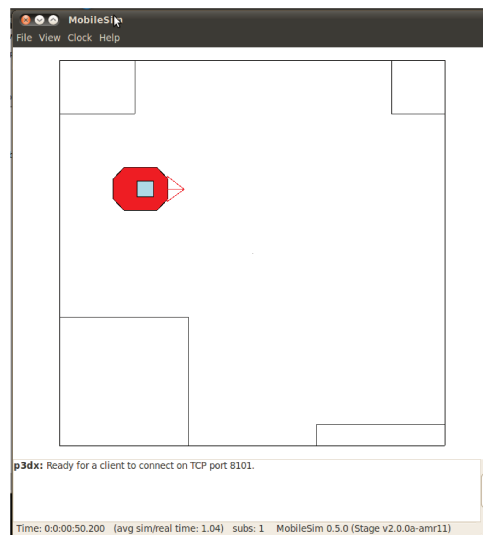


Fig. 5.2: Simulador MobileSim com uma mapa de uma sala.

5.1 Simulações de Escalabilidade

Um dos objetivos traçados para este trabalho foi a implementação da comunicação entre servidor e robô; isso significa que a comunicação deve ser mantida independentemente da posição do robô no interior da rede. Logo, é importante definir se o sistema permanece funcional conforme a rede aumenta, além de determinar o impacto deste aumento.

Devido ao número limitado de dispositivos disponíveis para a realização de experimentos reais,

não seria possível realizar testes que avaliassem a escalabilidade da rede. Desta forma, simulações foram utilizadas para avaliar esta característica. Cenários com diferentes números de sensores foram simulados, em todos eles os sensores foram dispostos numa topologia de grade, Figura 5.3.

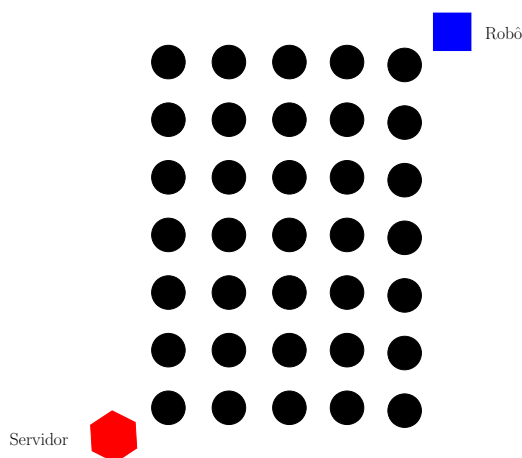


Fig. 5.3: Cenário de simulações realizadas.

Neste cenário, as mensagens precisam cruzar a rede inteira, desde o servidor, hexágono vermelho, ao robô, quadrado azul. A simulação consiste na medida do tempo necessário para uma requisição atravessar a rede e a resposta à esta requisição ser recebida e processada pelo nó do servidor. A operação realizada foi uma leitura da velocidade do robô (*getVel*).

As simulações foram realizadas com 9, 50, 100, 150 nós na rede, além dos nós do robô e do servidor, e repetidas 60 vezes. A Figura 5.4 apresenta os resultados destas simulações. É possível observar que apesar de uma oscilação nas primeiras interações o algoritmo tende a estabilizar, e o tempo médio de RTT (*Round Trip Time*) apresenta um aumento com o crescimento da rede, como esperado. Os tempos médios em cada uma das configurações são apresentados na Tabela 5.1.

Tab. 5.1: Tempo e número médio de saltos nas diferentes configurações.

Tamanho da rede	Número médio de saltos	RTT médios (ms)
9 nós (3x3)	4	79
50 nós (5x10)	14	220
100 nós (5x20)	22	389
150 nós (5x30)	33	517

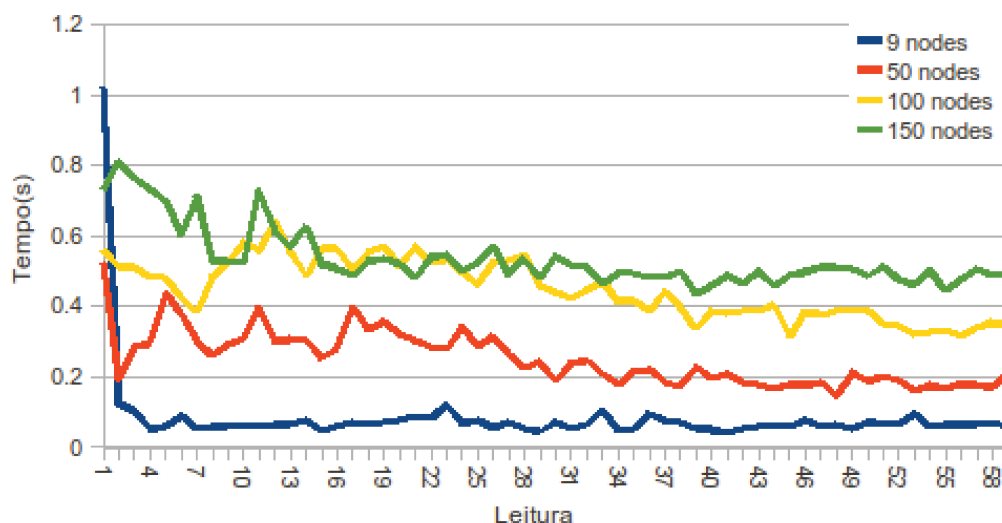


Fig. 5.4: Resultados de simulação para diferentes configurações.

5.2 Experimentos

Para avaliar a extensão proposta, experimentos foram conduzidos com robôs e uma rede de sensores reais. Esta seção apresenta testes para a avaliação da latência. O algoritmo de campos potenciais [70] foi utilizado para a avaliação da comunicação implementada e uma versão alterada deste algoritmo foi empregado para avaliar a extensão do *Range Service* para prover leituras de sensores da rede.

Todos os experimentos apresentados nesta seção foram realizados utilizando um robô móvel Pioneer P3-DX da MobileRobots ao qual foi conectado um IntelMote2 da Memsic por uma porta USB, Figura 5.5(b). Este mote foi equipado com um módulo de sensoriamento e um módulo de programação para realizar a comunicação com o robô. O robô possui um processador Pentium III e 512MB de memória RAM, rodando um sistema operacional Xubuntu 8.04 LTS. Ao servidor é conectado um iMote2 com um módulo de interface. A Figura 5.5(a) apresenta o mote conectado ao servidor além de um outro mote utilizado nos experimentos.

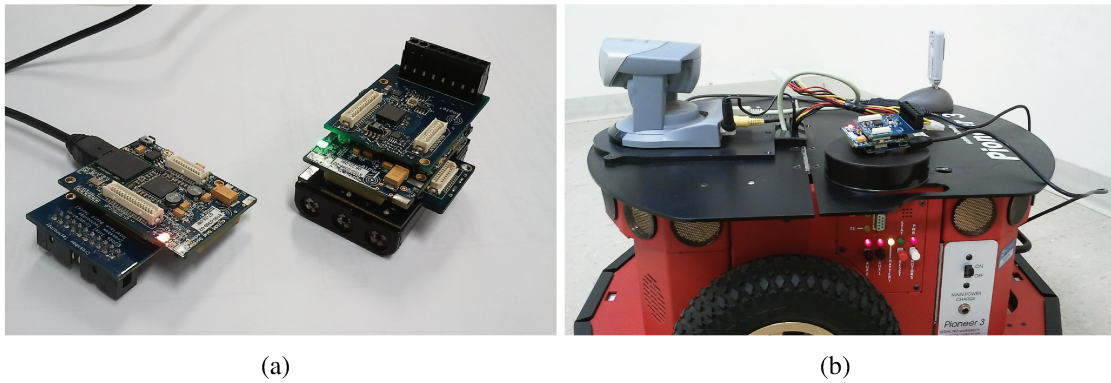


Fig. 5.5: (a) Motes utilizados nos experimentos. (b) Robô com mote conectado.

5.2.1 Avaliação de Latência

O primeiro experimento realizado foi com o objetivo de avaliar a latência da rede de sensores em dois cenários: com *hop* único e *multihop*. Os resultados destes testes foram comparados com o desempenho de uma rede Wi-Fi típica. A Figura 5.6 apresenta os resultados destes testes. Os testes *multihop* foram realizados com três *hops*.

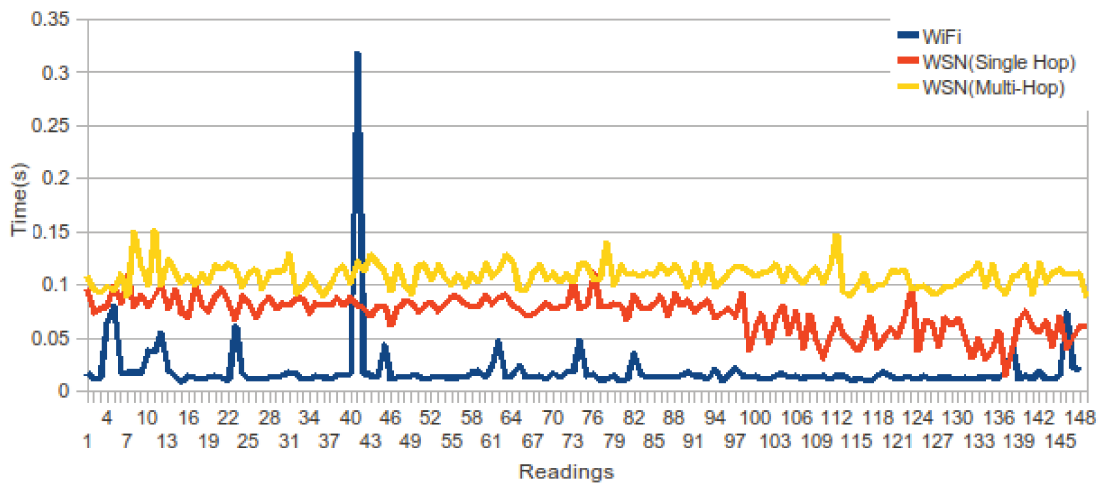


Fig. 5.6: Resultados comparativos entre Wi-Fi, *hop* único e *multihop*.

Como esperado, com um tempo médio de 19 ms, o desempenho da rede Wi-Fi foi consideravelmente superior ao desempenho da rede de sensores. Porém, os resultados da rede de sensores, média de 75 ms com um único *hop* e 108 ms com *multihops*, mostram que é viável utilizar a tecnologia

como uma rede de comunicação. Os resultados das latências fim-a-fim das redes são apresentados na Tabela 5.2.

Tab. 5.2: Tempos médios e desvio padrão de cada cenário de teste.

Scenario	Average RTT (ms)
WiFi	19 ± 27
Single Hop	75 ± 31.5
Multi Hop	108 ± 11.4

5.2.2 Validação da Comunicação - Campos Potenciais

Após uma avaliação comparativa das latências da rede, ainda se faz necessária uma avaliação mais qualitativa do desempenho da rede de sensores. Para realizar esses testes, foi utilizado o algoritmo clássico de navegação, campos potenciais. Este é um algoritmo de navegação autônoma que, dada uma posição alvo, é capaz de navegar um robô de maneira segura por um ambiente até a posição desejada [71]. O algoritmo utiliza uma combinação de campos repulsivos gerados por obstáculos detectados, neste caso utilizando sonares, e um campo atrativo na direção do alvo. Este campo resultante é utilizado para atualizar a velocidade e direção do robô.

No cenário de testes, o robô foi colocado de um lado da sala com uma caixa como obstáculo localizada no centro da sala. O robô deveria atravessar a sala e retornar à posição de origem, desviando da caixa. Na Figura 5.7(a), é possível visualizar o traçado descrito pelo robô na execução do algoritmo utilizando uma rede Wi-Fi típica, enquanto, na Figura 5.7(b), é possível ver o resultado de uma execução utilizando a rede de sensores. Em ambas execuções a velocidade máxima do robô foi configurada para 200 mm/s.

Em ambas infraestruturas de rede, o algoritmo foi capaz de navegar o robô até seu objetivo e de desviar do obstáculo. Porém, devido à maior latência da rede de sensores, uma operação de leitura de sonares leva mais tempo para ser completada utilizando esta tecnologia do que utilizando uma rede Wi-Fi. Logo, o algoritmo toma mais tempo para detectar que o robô está próximo a um obstáculo e iniciar o desvio. Analogamente, o algoritmo também demanda mais tempo para perceber que o

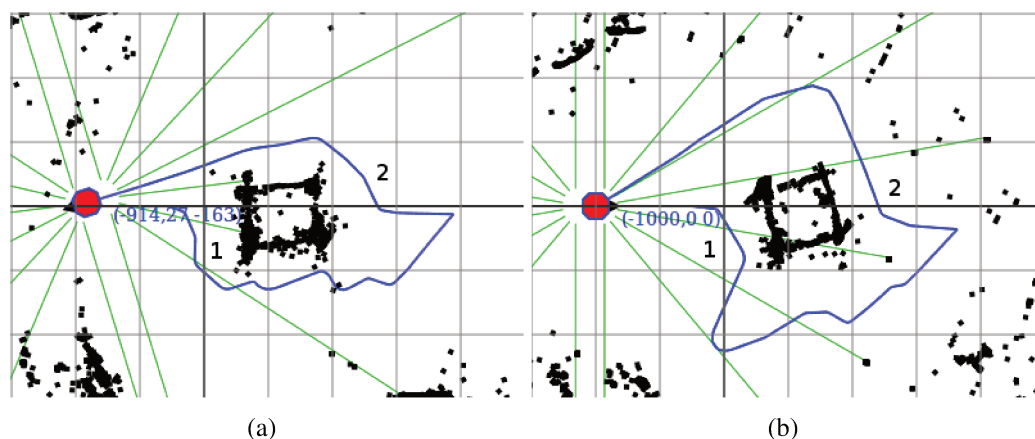


Fig. 5.7: Resultados da execução do algoritmo de campos potenciais utilizando: (a) Wi-Fi e (b) a rede de sensores.

obstáculo não está mais presente e retomar a navegação para o alvo. Estes fenômenos podem ser observados nas regiões demarcadas com os números 1 e 2 na Figura 5.7. É possível observar também que na região demarcada com 1 na Figura 5.7(b) a trajetória tomada pelo robô se afasta muito mais da caixa do que na região demarcada com 2, isso ocorreu devido a perda de um pacote de leitura de sonar, e o tempo necessário para repetir a leitura. Os pontos pretos nas figuras representam as leituras dos sonares.

A Figura 5.8 apresenta a comparação entre uma outra execução do algoritmo de campos potenciais, utilizando a rede de sensores, e o desempenho da rede Wi-Fi. Nesta execução também é possível observar as mesmas características devido à maior latência da rede, apresentadas anteriormente.

5.2.3 Algoritmo de Campos Potenciais Estendido

Para realizar a validação da integração das capacidades de sensoriamento da rede à plataforma, uma versão alterada do algoritmo de campos potenciais foi utilizada. Nesta versão, campos com potenciais repulsivos também são gerados a partir de características do ambiente, além dos obstáculos físicos. Os campos gerados pelas leituras de sensores da rede são funções de duas variáveis: a distância do robô ao sensor (similar ao algoritmo original) e a leitura realizada pelo sensor. Nestes

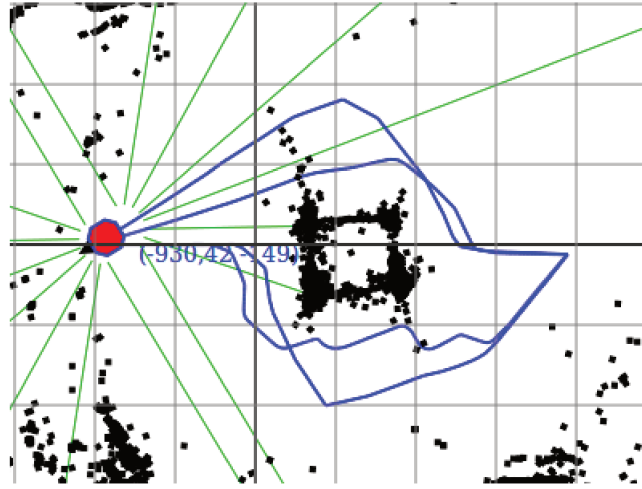


Fig. 5.8: Comparação entre redes de sensores e Wi-Fi.

testes o sensor utilizado foi o de luminosidade devido à maior facilidade na manipulação da incidência de luz sob o sensor do que, digamos, a temperatura.

Para a geração dos campos potenciais tradicionais, as equações utilizadas foram: Equações (5.1) (5.2) [22], onde: $F_{att}(q)$ é o campo atrativo, $F_{rep}(q)$ é o campo repulsivo, q é a posição atual do robô, q_{goal} é a posição do objetivo e $\rho(q)$ é distância euclidiana entre o robô e o objetivo.

$$F_{att}(q) = -k_{att}(q - q_{goal}) \quad (5.1)$$

$$F_{rep}(q) = \begin{cases} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(q)} \frac{q - q_{obstacle}}{\rho(q)} & \text{if } \rho(q) \leq \rho_0 \\ 0, otherwise \end{cases} \quad (5.2)$$

Enquanto para o cálculo dos campos com potencial repulsivo, baseados nas leituras dos sensores ($F_{repSensor}$, foi utilizada a Equação (5.3).

$$F_{repSensor}(q, r) = \begin{cases} \frac{K_{repSensor}}{\rho(q)r} & \text{if } \rho(q) \leq \rho_S \\ 0, otherwise \end{cases} \quad (5.3)$$

Como mencionado anteriormente, o campo repulsivo gerado pela leitura de um sensor é função de duas variáveis: a distância entre o robô e o sensor, $\rho(q)$, e a leitura do sensor, r .

Com esta alteração, a cada iteração do algoritmo são realizadas: uma leitura completa dos sonares do robô, ou de um subconjunto de interesse, onde, para cada leitura abaixo de um limiar, um campo repulsivo é calculado utilizando a Equação (5.2); e uma leitura dos valores de luminosidade na região do robô, onde um campo repulsivo também é calculado para cada leitura abaixo de um limiar utilizando a Equação (5.3). E, finalmente estes campos são somados para gerar o campo repulsivo resultante.

Para realizar a configuração e calibração do algoritmo, testes foram realizados em um ambiente robótico simulado. O primeiro teste foi realizado com um cenário onde apenas obstáculos gerados por leituras da luminosidade do ambiente estavam presentes. A referência utilizada para a geração dos campos repulsivos foi que, quanto menor a luminosidade, maior o campo repulsivo. Para alimentar a simulação com valores de sensoriamento do ambiente, um nó sensor real foi utilizado. O nó sensor possuía a informação de sua posição no mapa a ser enviada junto às respostas de leituras do sensor, como definido pelo modelo da plataforma. Diferentes configurações de luminosidades foram utilizadas nas simulações. A Figura 5.9 apresenta os resultados.

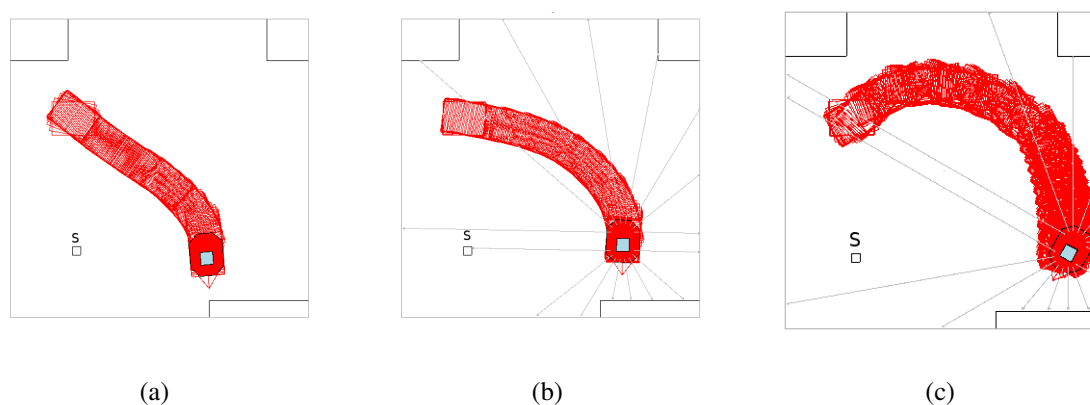


Fig. 5.9: Campos potenciais utilizando diferentes luminosidades: (a) pouca luz (b) penumbra (c) e completo escuro. S representa a posição do sensor.

É notória a influência que alterações nos valores de luminosidade possuem na trajetória do robô, chegando ao ponto, como na Figura 5.9(c), onde o robô não consegue mais atingir seu alvo.

Após uma avaliação sobre o campo gerado pelas leituras da rede, foram realizados testes em

que diferentes tipos de obstáculos estavam presentes no ambiente para que fosse possível calibrar o algoritmo para experimentos reais. Para realizar estes testes foi utilizado um mapa de um ambiente com diferentes obstáculos em cada canto de uma sala, similar ao apresentado na Figura 5.2, e um sensor localizado perto de a um dos cantos da sala.

Um teste preliminar foi realizado executando apenas o algoritmo tradicional, Figura 5.10(a). Em seguida, a alteração proposta para realizar a validação das alterações ao *Range Service* foi executada, Figura 5.10(b). Estes testes foram utilizados para o ajustes das constantes K_{att} , K_{rep} e K_{repesn} .

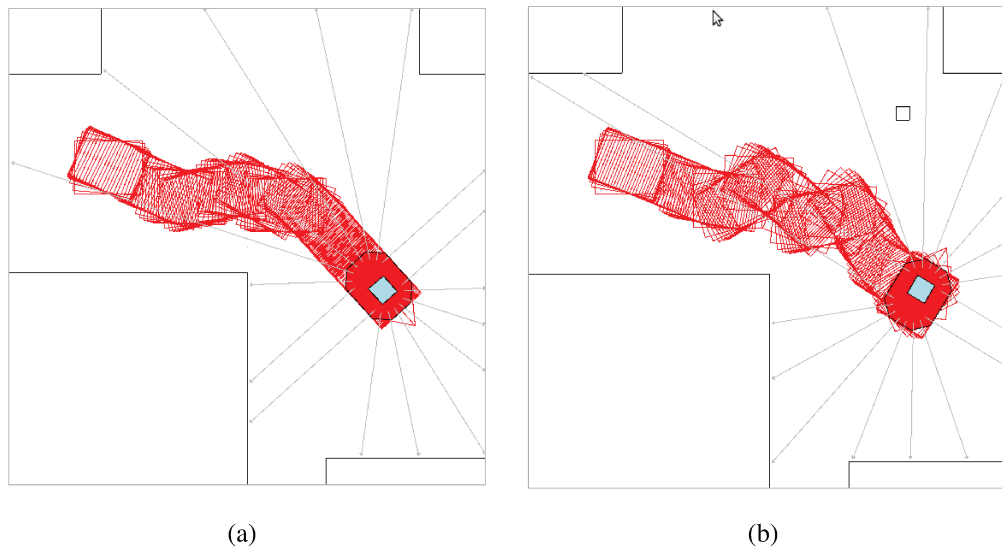


Fig. 5.10: Simulação para calibração dos campos potenciais. (a) campos potenciais tradicional (b) campos potenciais com obstáculos reais e ambientais.

Os melhores resultados foram obtidos com $K_{att} = 0.9$, $K_{rep} = 0.3$ e $K_{repesn} = 0.2$.

As codificações realizadas fazendo uso das APIs da plataforma podem ser utilizadas tanto para controlar robôs em ambientes simulados quanto robôs reais. Na codificação, a única alteração é o endereço do robô. Os experimentos consistiram de navegar o robô de um ponto a outro da sala desviando de obstáculos, físicos ou “ambientais”. O cenário dos testes com o robô real seguiram os cenários das simulações já apresentadas. Primeiro, foi realizada uma avaliação do campo gerado somente pelas leituras de luminosidade. Diferentes configurações foram utilizadas. A Figura 5.11(a) apresenta o traçado do robô com um campo gerado por um sensor recebendo pouca luz, enquanto a

Figura 5.11(b) apresenta o traçado do robô quando o sensor estava quase no total escuro.

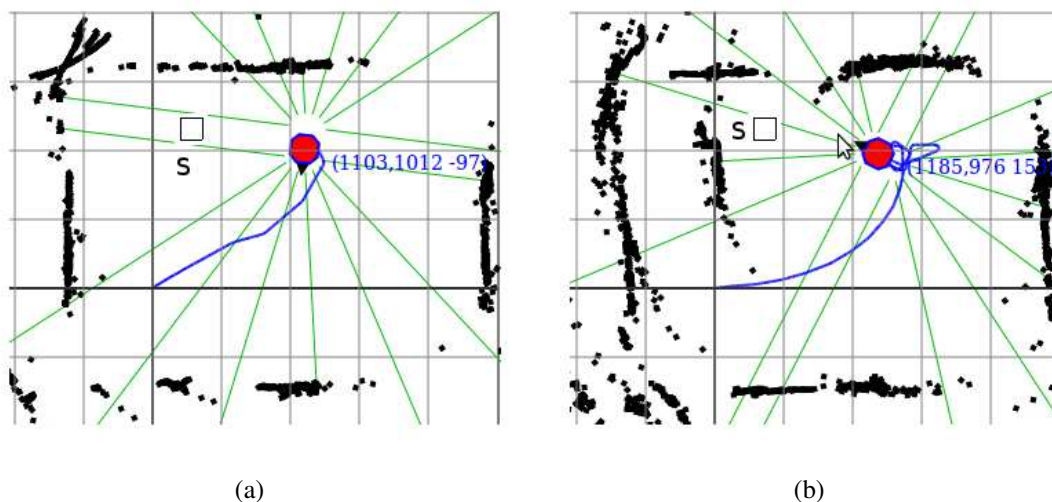


Fig. 5.11: Resultados da navegação do robô para duas configurações de luminosidade: (a) penumbra (b) escuro.

Assim como nas simulações, ainda se faz necessário testes com o algoritmo completo, ou seja, desviando de obstáculos físicos e daqueles gerados pelas leituras de sensores no ambiente. A Figura 5.12(a) mostra o cenário utilizado no experimento sem o sensor; neste caso a trajetória do robô é apenas influenciada pelos obstáculos físicos, detectados pelo sonar. Na Figura 5.12(b) é apresentado o mesmo cenário de navegação com a inserção de um sensor que gera um campo repulsivo ao robô. É fácil verificar a alteração nas rotas entre as imagens da Figura 5.12.

Em todos os experimentos, a rede de sensores foi utilizada como rede de comunicação. Isto significa que, além de todas as leituras de sensores, as atuações no robô também foram realizadas pela rede de sensores. Também é importante ressaltar que todos os resultados obtidos experimentalmente foram condizentes com os resultados das simulações.

Este capítulo apresentou as simulações e experimentos utilizados para validar as soluções propostas no Capítulo 4 e os resultados obtidos com estes testes. Através de simulação, foi possível avaliar a escalabilidade do sistema, enquanto experimentos reais foram utilizados para avaliar características como latência, realizar uma avaliação qualitativa do uso de uma rede de sensores como canal de comunicação entre robô e servidor, além de validar a integração das capacidades

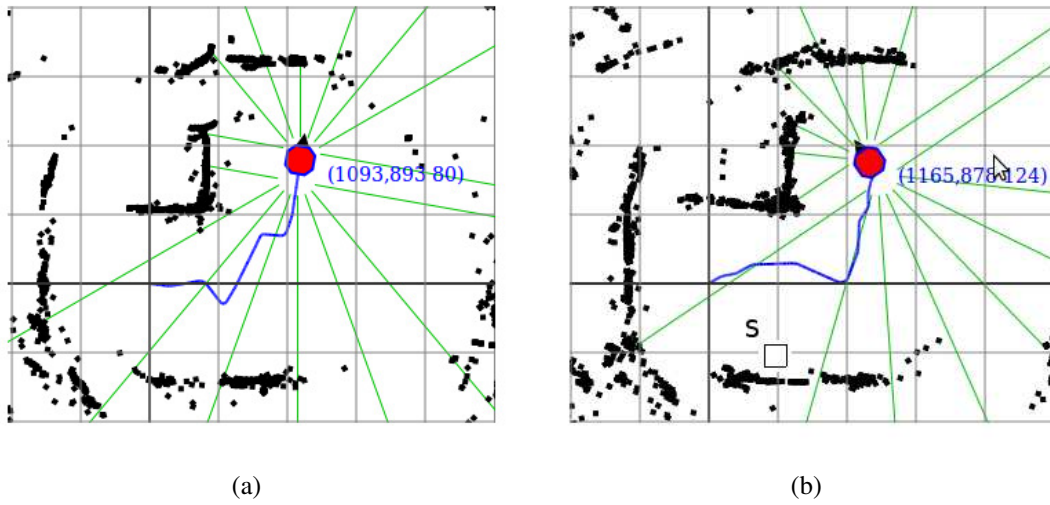


Fig. 5.12: (a) Navegação utilizando campos potenciais tradicionais. (b) Navegação utilizando obstáculos físicos e “ambientais“.

de sensoriamento da rede à plataforma REAL.

Capítulo 6

Serviço para Navegação Guiada de Robôs

Móveis

Esta dissertação apresentou até este momento uma plataforma que integra as funcionalidades de redes de sensores sem fio à robótica móvel, mais especificamente às capacidades de sensoriamento e de formar uma rede de comunicação. Como uma aplicação que utiliza a plataforma proposta, será apresentado um serviço de definição dinâmica de rotas de navegação para robôs móveis em ambientes monitorados por uma rede de sensores.

Este serviço busca uma solução para guiar robôs móveis, inseridos em ambientes monitorados por uma rede de sensores, até pontos de interesse sem a necessidade de utilizar a odometria do robô, uma medida que está muito sujeita a erros, mapas dos ambientes ou informações deterministas sobre as posições de nós sensores (coordenadas (x, y)) dentro destes ambientes. Para realizar uma navegação independente destes pontos, alguns requisitos são impostos tanto ao posicionamento dos nós sensores no ambiente quanto ao robô. Para os nós da rede de sensores, torna-se necessário que estes sejam dispostos em posições “estratégicas”, ou seja, em posições por onde um robô possa navegar. Por exemplo, no caso de um corredor, os nós sensores devem ser instalados ao longo deste corredor, preferencialmente centralizados.

Além de definir rotas navegáveis para o robô quando este requer uma, a rede também realiza

atualizações dinâmicas destas rotas baseadas em alguma característica do ambiente, que pode ser definida pelo usuário, por exemplo, evitar rotas que possuam luminosidade abaixo de um determinado limite. Devido a esta característica de alterações de rota durante a navegação do robô, este não conhece em nenhum instante a rota global até seu alvo, evitando atrasos para atualizar esta rota global todas as vezes que a rota fosse alterada.

Para possibilitar a fácil interação com esta habilidade de guiar a navegação do robô, foi definido um novo serviço na plataforma. Este novo serviço possui operações que permitem que uma aplicação cliente inicialize e configure como a rede vai formar estas rotas para o robô.

6.1 Arquitetura de Rede

Para realizar o traçado de rotas descrito acima, é preciso definir uma arquitetura de rede que possibilite esta implementação. A solução encontrada foi adotar um modelo multi-camadas. Uma camada adicional foi definida acima da camada de comunicação, a camada de navegação, Figura 6.1. Este modelo permite que as tabelas de roteamento das duas camadas possam ser alteradas independentemente uma da outra. Desta forma, as tabelas da camada de navegação podem ser alteradas devido a leituras de sensores da rede, enquanto as rotas da camada de comunicação permanecem alheias a estes eventos. Quando um sensor realiza uma leitura que não é considerada segura, este nó informa seus vizinhos que uma rota de navegação não pode ser criada através dele. A camada de navegação, então, modifica as entradas das suas tabelas de roteamento evitando adotar rotas que passem por este determinado nó.

Este serviço de navegação possui dois modos de funcionamento. No primeiro, a rede monitora um ambiente procurando detectar algum evento pré-definido, como um incêndio, que seria representado pelo aumento da temperatura. O segundo modo de operação é o traçado de rotas para um alvo determinado pelo usuário. Estas rotas são formadas utilizando o mesmo protocolo de roteamento descrito na Seção 4.5.

Quando um nó alvo é determinado, este dissemina esta informação pela rede e se torna nó raiz da

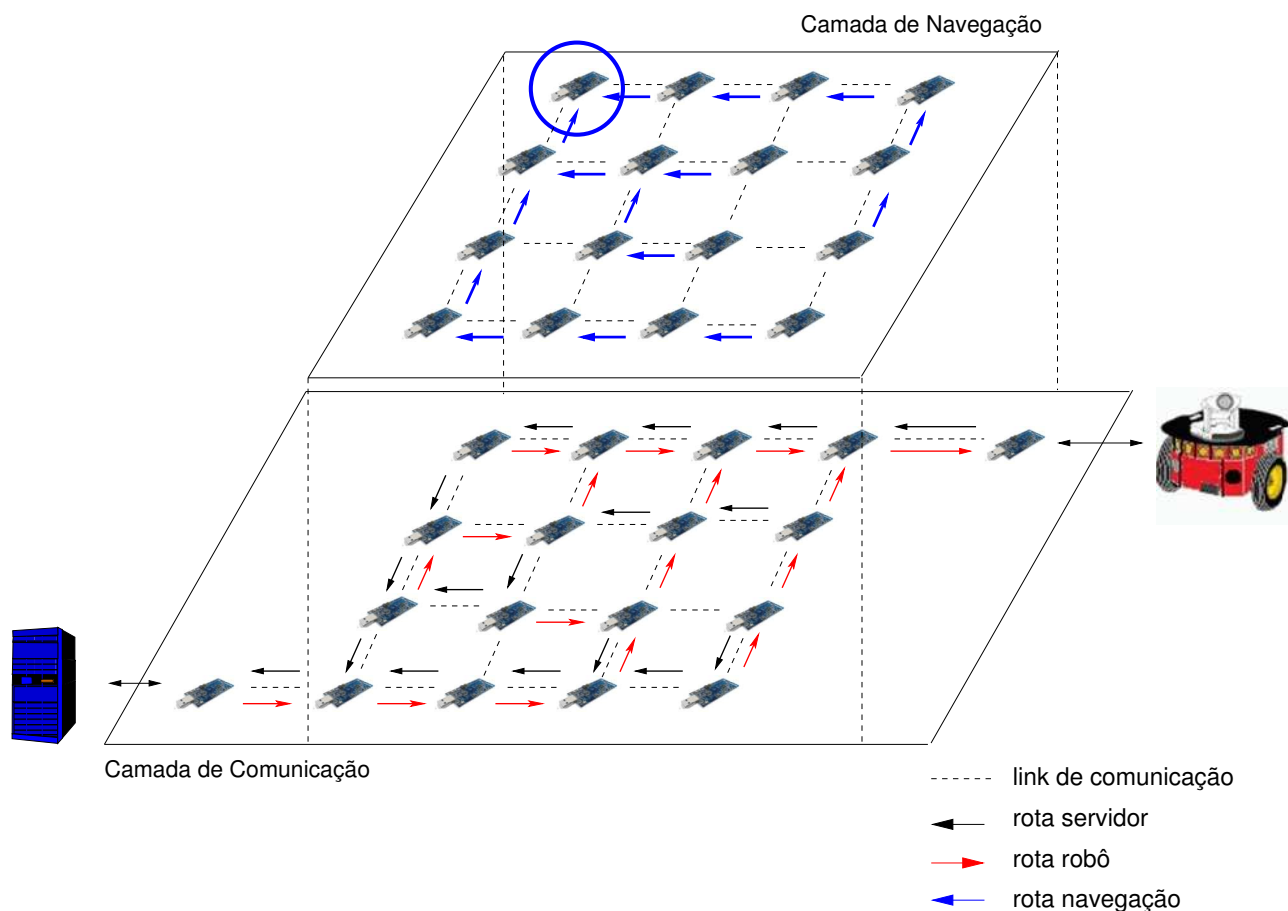


Fig. 6.1: Arquitetura de rede para implementação da solução proposta.

camada de navegação. Uma vez que o nó do robô recebe esta informação, ele realiza um *broadcast* de um requerimento de rota para os seus vizinhos. Ao receber este requerimento, o nó gera uma requisição de rota específica da camada de navegação contendo seu ID, como primeiro *hop* da possível rota, um *hop count* e um número de sequência recebido do nó do robô. A Figura 6.2 apresenta a estrutura desta mensagem.

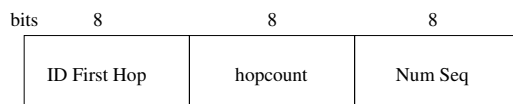


Fig. 6.2: Mensagem de definição de rota da camada de navegação.

Este requerimento é enviado até o nó alvo, mas os nós que propagam esta mensagem do primeiro *hop* até o nó alvo, além de se incluírem no *hop count*, realizam uma verificação nas mensagens.

As requisições que possuem um número de sequência menor ou igual do que o da última requisição transmitida, o que caracterizaria uma duplicata, não são repassadas, buscando assim reduzir o número de mensagens transmitidas na rede. É assumido que a primeira requisição que atingiu o nó alvo é a rota mais curta entre a posição atual do robô e o seu nó alvo, logo esta será a única requisição respondida. A resposta de uma mensagem de requerimento de rota é enviada ao nó do robô contendo o ID do vizinho que é o primeiro nó da rota. Ao receber essa mensagem, o nó do robô envia ao vizinho que é o *hop* da rota um pedido de um *beacon* de navegação. Ao receber um *ack* do requerimento, o nó do robô indica ao robô que a navegação pode iniciar.

Ao atingir um alvo, o nó do robô verifica se é o alvo final. Se o nó não for o alvo final, um processo de *handover* é iniciado, o nó do robô comunica a rede, mais especificamente ao nó atingido, que este pode desativar o *beacon*. Quando o nó alcançado desativa o *beacon*, e não é o nó alvo, uma mensagem é enviada pela rota atual indicando ao próximo *hop* que este pode ativar o seu *beacon*. O novo nó alvo então envia uma mensagem de *ack* para o robô, indicando que ele pode agora prosseguir com a navegação. A Figura 6.3 apresenta o diagrama de sequência de todo o processo de definição da rota de navegação.

Desde o momento em que o serviço é inicializado, os nós da rede equipados com sensores realizam leituras periódicas. Se o valor de uma leitura estiver abaixo de um determinado limiar, o nó entra em um estado bloqueado, o que significa que uma rota de navegação não deve passar por ele. Quando um nó entra neste estado, este envia uma mensagem para todos os seus vizinhos (*broadcast*), informando-os desta condição. Os nós vizinhos atualizam suas tabelas de roteamento com esta informação e recalculam suas rotas. Uma vez que um nó bloqueado realiza uma leitura considerada segura, acima do limiar, este retorna ao estado inicial e comunica seus vizinhos desta alteração.

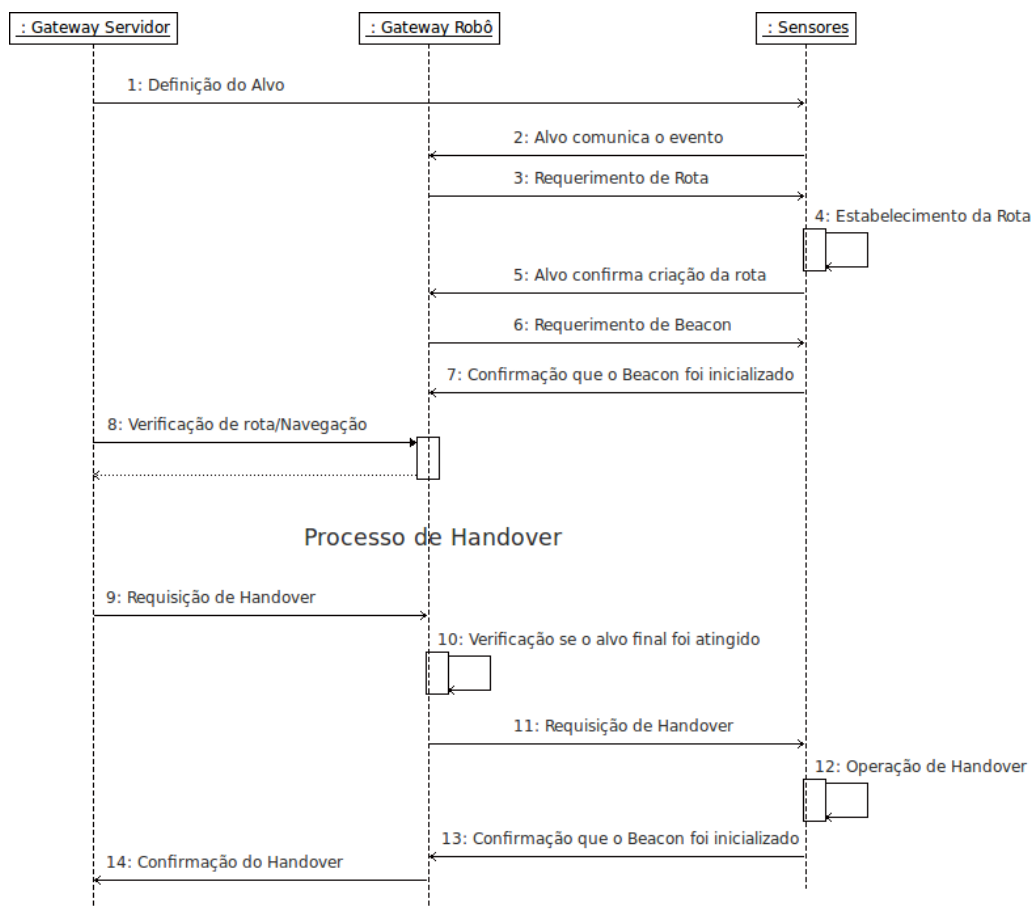


Fig. 6.3: Diagrama de sequência para a formação de rotas e processo de *handover*.

6.2 Definição do Serviço na Plataforma

Buscando manter o modelo adotado pela plataforma, onde toda a comunicação entre uma aplicação cliente e a plataforma é realizada utilizando requisições HTTP, um novo serviço foi definido para que uma aplicação possa interagir com a infraestrutura para navegação proposta. O serviço foi definido como *WsnNavigation Service*; o quadro 6.1 apresenta as operações definidas pelo serviço.

A seguir será feita uma breve descrição de cada operação:

- *Start*: esta operação informa a rede que o usuário deseja iniciar a busca por um evento para enviar o robô. (Neste protótipo os eventos utilizados foram de variação da luminosidade);
- *Stop*: encerra a monitoração por algum evento;

Tab. 6.1: Operações do serviço de navegação.

Operação	Definição	Parâmetros
start	inicializa a detecção de eventos	N/A
stop	encerra a detecção de eventos	N/A
setTarget	indica nó para onde o robô deseja navegar	ID do nó
handover	informa a rede que um nó foi alcançado	N/A
isPathDone	pergunta a rede se existe uma rota para o alvo	N/A

- *setTarget*: informa a rede explicitamente qual nó deve ser considerado alvo;
- *handover*: esta operação deve ser chamada quando o robô atingir o alvo atual, a rede então irá verificar se é o alvo final ou realizar o processo de *handover*. Dependendo da resposta dada pelo servidor a aplicação pode saber se atingiu o alvo final ou não;
- *isPathDone*: operação que responde a aplicação se já existe uma rota para o alvo, sua resposta indica o início da navegação;

6.3 Avaliação do Serviço

6.3.1 Simulações

Assim como na avaliação da plataforma, a primeira avaliação do serviço a ser realizada é do desempenho do sistema frente ao aumento da rede. Simulações do algoritmo de formação de rotas foram executadas repetidas vezes para diferentes tamanhos de rede. O modelo da rede ainda é o mesmo apresentado anteriormente. A Figura 6.4 apresenta os resultados das simulações realizadas.

As simulações apresentam um acréscimo relativamente alto no tempo tomado para a definição de uma rota. Uma análise mais detalhada mostra também que o maior responsável por esse acréscimo é o protocolo de disseminação, utilizado para propagar a mensagem que requer a presença do robô do nó alvo até o robô. A Tabela 6.2 apresenta as medidas detalhadas para cada cenário de simulação. A segunda coluna (Evento) apresenta o tempo necessário para o robô receber a informação de que sua

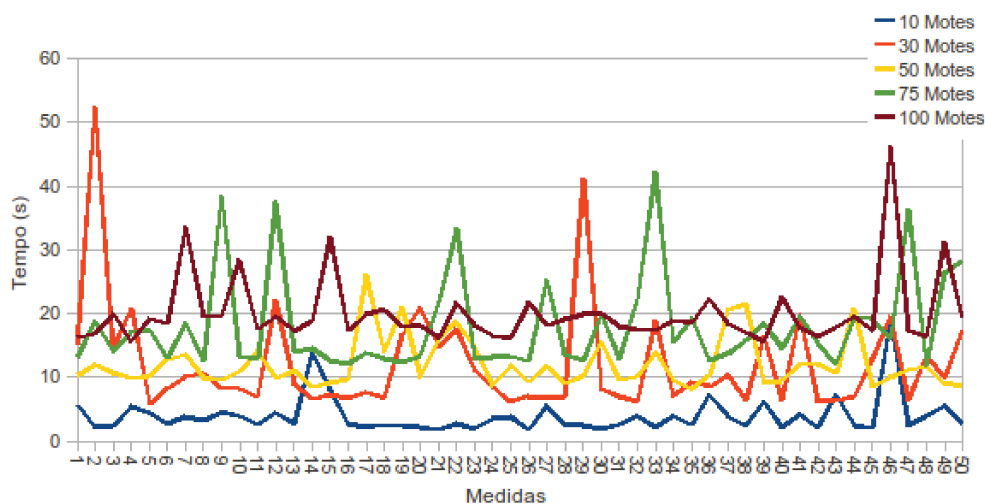


Fig. 6.4: Resultados de simulações do algoritmo de traçado de rotas.

Tab. 6.2: Tempo médios de RTT para cada cenário

N° Motes	Evento(s)	Rede(ms)	RTT(s)
10	3.90	66	3.975
30	12.028	140	12.169
50	11.921	203	12.124
75	18.913	280	19.194
100	19.645	356	20

presença é requerida. A segunda coluna (Rede) é o tempo para a requisição de rota e sua resposta atravessarem a rede. A última coluna (RTT) apresenta o tempo total da operação.

6.3.2 Experimentos

Os primeiros experimentos foram realizados utilizando poucos nós sensores, visando validar o processo de definição das rotas. O primeiro cenário experimental foi constituído por 4 nós, onde existia apenas uma rota possível entre o nó do robô e o nó alvo. Este cenário é apresentado na Figura 6.5. O nó mais a direita representa o nó do robô, enquanto os outros três formam a rota de navegação.

Após este teste inicial, diferentes cenários foram testados buscando aumentar, e alterar, gradativamente a topologia da rede. A Figura 6.6(a) ilustra o resultado de um cenário onde existem duas

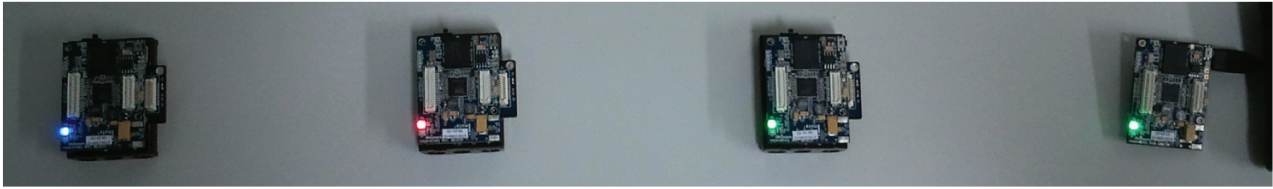
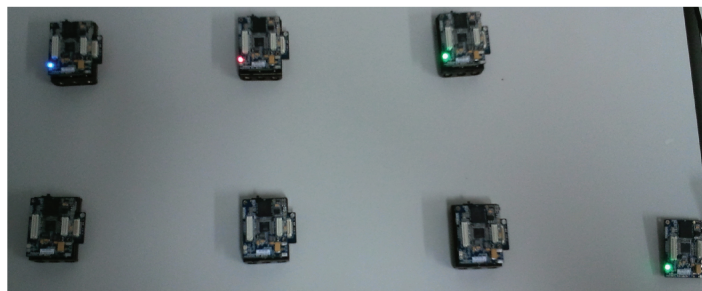


Fig. 6.5: Cenário simples de traçado de rota. O LED azul indica o nó sensor alvo, o LED vermelho indica que o nó pertence a rota estabelecida e o LED verde indica, nos nós da rede, o nó alvo atual e no nó do robô que a navegação pode ser inicializada.

possíveis rotas entre o robô e o seu alvo, enquanto na Figura 6.6(b) existem múltiplas possibilidades.



(a)



(b)

Fig. 6.6: Diferentes cenários de testes com diferentes topologias.

Após os testes do traçado de rota em múltiplos cenários, foram realizados testes para avaliar a alteração dinâmica de rotas baseadas em características do ambiente. Nos experimentos apresentados, assim como na extensão do algoritmo de campos potenciais apresentado na Seção 5.2.3, foi utilizado o sensor de luminosidade.

Para realizar esta avaliação, foi utilizado um cenário simples, análogo ao apresentado na Figura 6.6(a). Em todas as execuções havia duas rotas possíveis do nó do robô para o nó alvo. Após o estabelecimento de uma rota, um evento foi gerado, forçando a camada de navegação a recalculer suas rotas.

Num primeiro momento, um evento temporizado foi utilizado; trinta segundos após inicializada a rede, a ocorrência de um evento é simulada. O resultado deste experimento é apresentado na Figura 6.7.

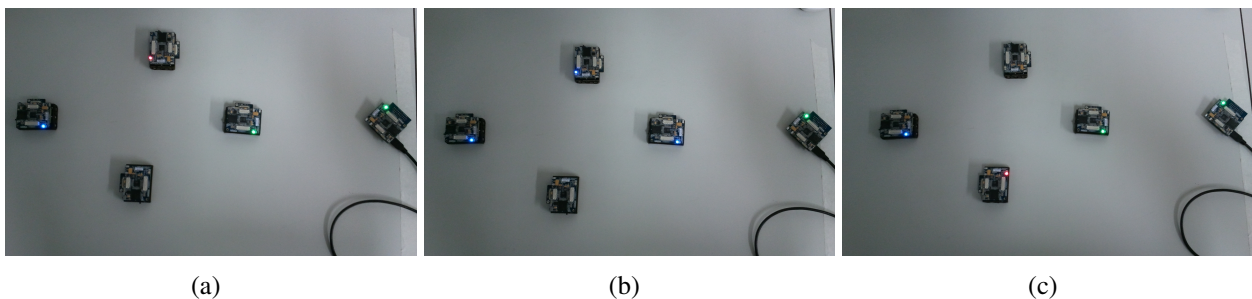


Fig. 6.7: Teste da solução de recálculo de rotas utilizando um evento temporizado. (a) Formação da rota inicial. (b) Nó que “detectou” o evento realiza um *broadcast* desta informação. (c) Nova rota calculada.

Em seguida foram realizados testes onde os motes realmente sensoriaram o ambiente. Dois nós foram equipados com sensores e temporariamente realizavam uma leitura do sensor de luminosidade. Quando a leitura estava abaixo de um determinado limiar o sensor entrava no estado bloqueado. O resultado de uma execução completa é apresentado na Figura 6.8.

6.4 Aplicações do Serviço

Dois protótipos de um sistema de navegação foram desenvolvidos para utilizar a solução de auxílio a navegação proposta: um baseado em visão e outro baseado em leituras de RSSI. Como está fora do escopo desta dissertação a análise e avaliação de algoritmos de navegação para robôs móveis, será apenas comentado cada um dos protótipos e os impactos de cada um na implementação da solução de rede.

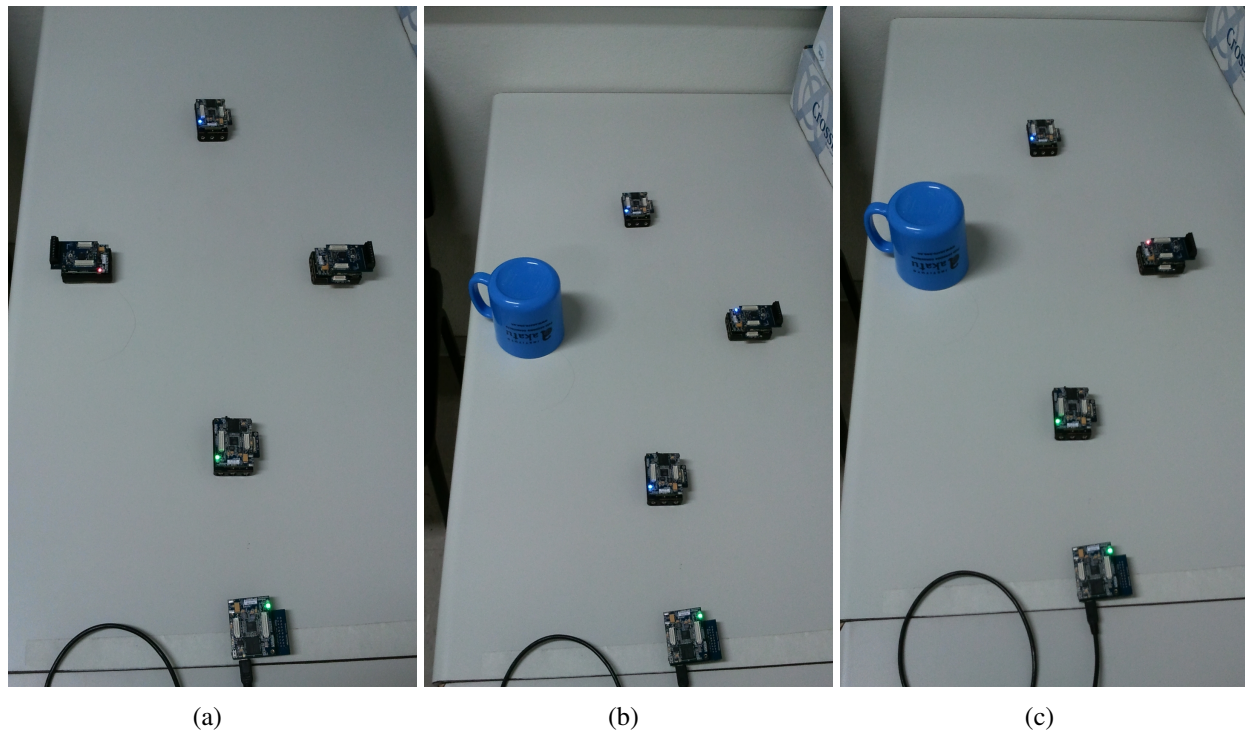


Fig. 6.8: Teste da solução de recálculo de rotas utilizando sensorialmente real. (a) Formação da rota inicial. (b) Nó coberto detecta o evento e realiza um *broadcast* desta informação. (c) Nova rota calculada.

A principal diferença, pelo pontos de vista da rede de sensores, é o tipo de *beacon* utilizado em cada cenário. No cenário baseado em visão, é utilizado um *beacon visual*, um LED verde, enquanto no baseado em RSSI, são utilizadas mensagens enviadas periodicamente pelos nós sensores para atrair o robô.

Uma das características de aplicações que utilizam imagens é o grande volume de dados transmitidos pela rede. Para ser possível realizar a navegação baseado em visão, um cenário híbrido é necessário. O robô se conecta ao servidor através das duas tecnologias de rede disponíveis, a rede de sensores e uma rede Wi-Fi. A rede de sensores é utilizada tanto para prover rotas de navegações pelo ambiente quanto para o envio de comandos para o robô, enquanto a rede Wi-Fi permanece dedicada apenas ao envio das imagens da câmera do robô. Este cenário é ilustrado na Figura 6.9.

A navegação propriamente dita é realizada por uma combinação de um algoritmo de controle nebuloso para realizar o *tracking* do *beacon* nas imagens, combinado a uma rede neural para realizar

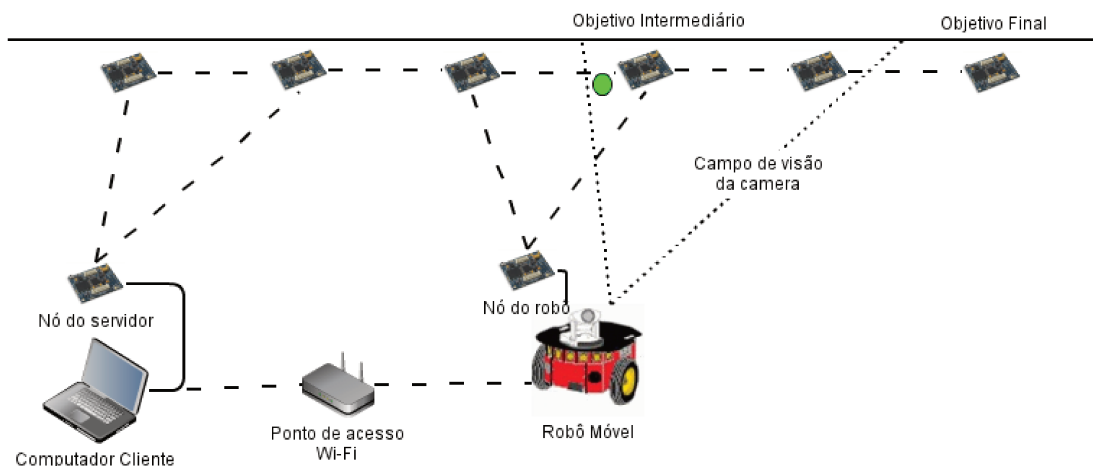


Fig. 6.9: Cenários para navegação utilizando a rede de sensores através do algoritmo baseado em visão.

o desvio de obstáculos.

Para realizar a navegação baseada em RSSI, quando o robô navega pela rota os nós que são alvos intermediários enviam mensagens periodicamente, através das quais o robô captura o valor de RSSI e utiliza este valor para determinar se está se aproximando ou não do alvo. Um algoritmo bio-inspirado baseado em regras é utilizado para determinar os movimentos do robô de acordo com a variação dos valores de RSSI e também para garantir o desvio de obstáculos.

Como a quantidade de informações transmitidas no cenário baseado em RSSI é bem reduzida frente ao baseado em visão, apenas a rede de sensores é utilizada. O cenário baseado em RSSI é apresentado na Figura 6.10.

Ambas as soluções foram capazes de navegar o robô até os alvos desejados. Apesar do maior tráfego de informações necessário para realizar a navegação no primeiro cenário, esta solução se mostrou mais eficiente, permitindo ao robô atingir seus alvos mais rapidamente e realizando traçados mais lineares. Este resultado se deve principalmente pelo tipo de informação utilizada, já que as imagens da câmera do robô são mais confiáveis do que variação de leituras de RSSI. Um outro ponto onde a solução utilizando visão levou vantagem sobre a baseada em RSSI é no número de mensagens trocadas entre os nós da rede de sensores, o que influencia diretamente no consumo de energia. Pelo

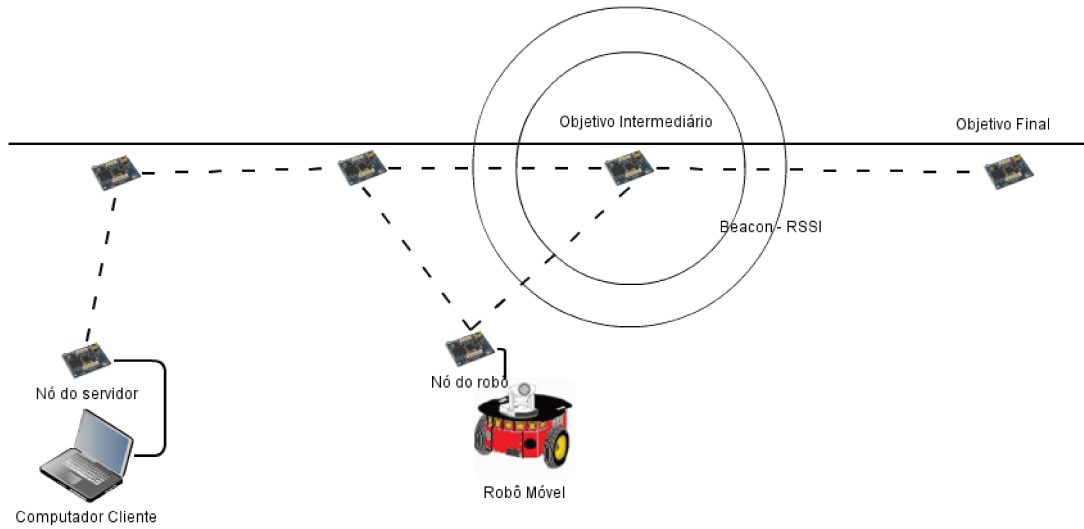


Fig. 6.10: Cenários para navegação utilizando a rede de sensores através do algoritmo baseado em RSSI.

fato de o *beacon* ser visual, não é necessário o envio periódico de mensagens pelos motes que formam a rota. Uma vez ativado o *beacon*, o nó só volta a interagir com o robô quando este considera o mesmo atingido.

Este capítulo definiu um novo serviço, a ser provido pela plataforma, que utiliza a rede de sensores para auxiliar a navegação de robô móveis. Este serviço utiliza uma arquitetura em camadas para definir rotas para a navegação do robô, baseadas em características do ambiente, enquanto provendo as funcionalidades descritas no Capítulo 4. Esta modelo em camadas permite que as tabelas de roteamento sejam independentes, de tal forma que parâmetros distintos podem ser empregados para definir as rotas nas diferentes camadas. Simulações foram utilizadas para avaliar a desempenho e escalabilidade, enquanto experimentos reais, em diferentes cenários, foram empregados para validar a solução proposta.

Capítulo 7

Considerações Finais

Este capítulo final está dividido em duas seções. Na seção 7.1, é apresentada uma retrospectiva dos principais pontos do trabalho e suas contribuições, e, na seção 7.2, além de proposições de trabalhos futuros, também são apresentados desdobramentos atuais das proposições apresentadas nesta dissertação.

7.1 Contribuições

Esta dissertação apresentou uma extensão à plataforma robótica REAL para inclusão de redes de sensores. Esta extensão foi realizada através de dois aspectos principais: o uso da rede de sensores como uma rede de comunicação e a inclusão das capacidades de sensoriamento da rede ao serviço *Range Service* da plataforma.

Foi proposta uma arquitetura para esta nova plataforma, em que surgiu a figura dos *gateways*, responsáveis por manter toda essa nova infraestrutura escondida de aplicações e recursos. Um protocolo de comunicação junto à definição de mensagens específicas foi proposto para possibilitar a transferência de todos os dados necessários para prover os serviços da plataforma por uma rede de sensores. Utilizando este protocolo, uma simples API, a TinyAPI, foi definida, possibilitando que o controle da movimentação fosse realizado diretamente por um nó da rede.

No serviço de sensoriamento da plataforma, o *Range Service*, foram definidas quatro novas operações, referentes aos sensores disponíveis na rede de sensores: *getHumReadings*, *getLightReadings*, *getTempReadings* e *getAccelReadings*. Estas novas operações também seguem o modelo imposto pela plataforma REAL.

Utilizando simulações e experimentos reais, foi possível realizar avaliações sobre o desempenho da solução proposta. Através de simulação, foi possível mostrar que a solução de comunicação é escalável e pode ser aplicada em redes de sensores monitorando grandes áreas.

Uma avaliação mais qualitativa do uso da rede de sensores como uma rede de comunicação foi realizada utilizando o algoritmo de navegação por campos potenciais, e foi demonstrado que é possível fazer uso desta rede para esta função. Uma versão alterada deste mesmo algoritmo foi utilizada para validar a integração dos sensores da rede ao *Range Service*.

Também foi proposta uma estrutura de rede para realizar o auxílio de navegação de robôs móveis em ambientes monitorados por uma rede de sensores. O uso desta estrutura permite que um robô navegue até um ponto de interesse sem o uso da sua odometria.

7.2 Trabalhos Futuros

O trabalho proposto nesta dissertação, mais especificamente a arquitetura de navegação apresentada no capítulo 6, é um dos pontos de partida para o desenvolvimento de uma infraestrutura inteligente na área de robótica assistiva na qual o se encontra o projeto DesTINe. Este projeto tem como objetivo o desenvolvimento de uma infraestrutura assistiva para pessoas que possuam severas sequelas de AVC (Acidente Vascular Cerebral).

Existem várias possibilidades para evoluir o trabalho aqui apresentado, entre estas destacam-se:

- Definição de um algoritmo de roteamento que seja mais eficiente em ambientes dinâmicos;
- Portar o sistema descrito, implementado utilizando o TinyOS, para um sistema embarcado Linux;

- Inclusão de outros sensores como câmeras e leitores de etiquetas RFID, melhorando as informações que a rede pode fornecer ao robô sobre o ambiente.

Referências Bibliográficas

- [1] Tatiana Giorgenon Bonifácio. Implementação de um protocolo *mesh multi-hop* baseado em algoritmo de roteamento geográfico para redes de sensores sem fio. Master's thesis, Escola de Engenharia de São Carlos, Universidade de São Paulo, 2010.
- [2] Walteneus Dargie and Christian Poellabauer. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. Wiley, 2010.
- [3] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. Wiley, 2005.
- [4] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, New York, NY, USA, 2000.
- [5] TEP 2. Tep 2 - hardware abstraction architecture. <http://www.tinyos.net/tinyos-2.x/doc/html/tep2.html>, 2010.
- [6] TEP 125. Tep 125 - tinyos 802.15.4 frames. <http://www.tinyos.net/tinyos-2.x/doc/html/tep125.html>, 2010.
- [7] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *Proceedings of the International Conference on Mobile Systems, Applications and Services*, pages 163–176, New York, NY, USA, June 2005.

- [8] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki: A lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the IEEE International Conference on Local Computer Networks*, pages 455–462, Washington, DC, USA, 2004.
- [9] Qing Cao, Tarek Abdelzaher, John Stankovic, and Tian He. The liteos operating system: Towards unixlike abstractions for wireless sensor networks. In *Proceedings of the International Conference on Information Processing in Sensor Networks*, pages 233–244, Washington, DC, USA, 2008.
- [10] ZigBee Alliance. The zigbee alliance home page. <http://www.zigbee.org>, 2011.
- [11] Chao Peng, Max Q.H. Meng, and Huawei Liang. An experimental system of mobile robot’s self-localization based on wsn. In *Proceedings of the IEEE International Conference on Automation and Logistics*, pages 1029–1034, Shenyang, China, August 2009.
- [12] David Meger, Dimitri Marinakis, Ioannis Rekleitis, and Gregory Dudek. Inferring a probability distribution function for the pose of a sensor network using a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 756–762, Kobe, Japan, May 2009.
- [13] Mihail Sichitiu and Vaidyanathan Ramadurai. Localization of wireless sensor networks with a mobile beacon. In *Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 174–183, Fort Lauderdale, Florida, USA, October 2004.
- [14] Maxim Batalin and Gaurac Sukhatme. Mobile robot navigation using a sensor network. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 636–641, New Orleans, Los Angeles, USA, April 2004.
- [15] Si-Yao Fu, Guo-Seng Yang, and Zeng-Guang Hou. An indoor navigation system for autonomous mobile robot using wireless sensor network. In *Proceedings of the IEEE International Conference on Networking, Sensing and Control*, pages 227–232, Okayama, Japan, March 2009.

- [16] Yu-Chee Tseng, Meng-Shiuan Pan, and Yuen-Yung Tsai. Wireless sensor networks for emergency navigation. *IEEE journal on Computer*, 39(7):55–62, July 2006.
- [17] Jehn-Ruey Jiang, Yung-Liang Lai, and Fu-Cheng Deng. Mobile robot coordination and navigation with directional antennas in positionless wireless sensor networks. In *Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, Mobility '08, pages 80:1–80:7, Yilan, Taiwan, September 2008.
- [18] Isaac Amundson, Xenofon Koutsoukos, and Janos Sallai. Mobile sensor localization and navigation using rf doppler shifts. In *Proceedings of the ACM International Workshop on Mobile Entity Localization and tracking in GPS-less environments*, pages 97–102, San Francisco, CA, USA, September 2008.
- [19] Zhenwang Yao and K. Gupta. Distributed roadmaps for robot navigation on sensor networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3078–3083, Anchorage, Alaska, USA, May 2010.
- [20] Adrián Jiménez-González, José Ramiro Martínez de Dios, and Aníbal Ollero. An integrated testbed for heterogeneous mobile robots and other cooperating objects. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3327–3332, Taipei, Taiwan, October 2010.
- [21] Pablo Gil, Iván Maza, Aníbal Ollero, and Pedro José y Marrón. Data centric middleware for the integration of wireless sensor networks and mobile robots. In *Proceedings of the Conference on Mobile Robots and Competitions*, Albufeira, Portugal, April 2007.
- [22] R. Siegwart. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.
- [23] CC2420. Texas instruments low power rf ics - 2.4ghz - cc2420. <http://focus.ti.com/docs/prod/folders/print/cc2420.html>, 2011.
- [24] Ian F. Akyildiz and Mehmet Can Vuran. *Wireless Sensor Networks*. Wiley, 2010.

- [25] Ian Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communication Magazine*, 40(8):102–114, August 2002.
- [26] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2001.
- [27] TinyOS. Tynyos project web site. <http://www.tinyos.net/>, 2010.
- [28] Rainer Matischek. *A TinyOS-Based Ad Hoc Wireless Sensor Network*. Verlag Dr. Muller, 2008.
- [29] D. Gay, P. Lewis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1–11, San Diego, California, USA, 2003.
- [30] TEP 111. Tep 111 - message_t. <http://www.tinyos.net/tinyos-2.x/doc/html/tep111.html>, 2010.
- [31] Philip Levis and Nelson Lee. Tossim: A simulator for tynyos networks. Technical report, Computer Science Division, University of California at Berkeley, 2003.
- [32] Qing Cao and Tarek Abdelzaher. Liteos: A lightweight operating system for c++ software development in sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems*, pages 361–362, New York, NY, USA, 2006.
- [33] OpenEmbedded. The openembedded project web page. <http://www.openembedded.com>, 2011.
- [34] Jörg Kasteleiner. Principles of applying embedded linux on imote2. Master’s thesis, Faculty of Computer Science and Engineering, University of Applied Sciences Frankfurt am Main, 2010.
- [35] Memsic. Memsic - powerfull sensing solutions for a better life. <http://www.memsic.com>, 2011.

- [36] FTDI. Ftdi chip home page. <http://www.ftdichip.com/>, 2011.
- [37] OpenOCD. Open on-chip debugger. <http://openocd.berlios.de/web/>, 2011.
- [38] Martin Auersvald. Wireless sensor network for monitoring patients with parkinson's disease. Master's thesis, Department of Control Engineering, Faculty of Electrical Engineering, Czech Thechnical University in Prague, 2008.
- [39] D. Li, K. D. Wong, Y. H. Hu, and A. M. Sayeed. Detection, classification, and tracking of targets. *IEEE Signal Processing Magazine*, 19:17–29, March 2002.
- [40] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. An energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the Internatial Conference on Mobile Systems, Applications, and Services*, Boston, MA, 2004.
- [41] Th. Arampatzis, J. Lygeros, and S. Manesis. A survey of applications of wireless sensors and wireless sensors and wireless sensor networks. In *Proceedings of the Mediterranean Conference on Control and Automation*, pages 719–724, Limassol, Cyprus, June 2005.
- [42] Maxim Batalin and Gaurav Sukhatme. Using a sensor network for distributed multi-robot task address. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 158–164, New Orleans, Los Angeles, USA, April 2004.
- [43] Taehong Kim, Young-Guk Ha, Jihoon Kang, Daeyoung Lim, Chong Poh Kit, and Joo-Chan Sohn. Experiments on building ubiquitous robotic space for mobile robot using wireless sensor networks. In *Proceedings of the International Conference on Advanced Information Networking and Applications, Workshops*, pages 662–667, Okinawa, Japan, March 2008.
- [44] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications, WSNA'02*, pages 88–97, Atlanta, Georgia, USA, 2002.

- [45] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: sensor networks in agricultural production. *IEEE journal on Pervasive Computing*, 3(1):38–45, January 2004.
- [46] Huawei Liang, Shuai Li, Wanming Chen, Tao Mei, and Max Meng. A wsns-based map building method for mobile robots. In *Proceedings of the International Conference on Information Acquisition*, pages 415–419, Jeju City, Korea, July 2007.
- [47] Prasanna Mohan Ballal, Abhishek Chaitanya Trivedi, Vincenzo Giordano, José Mireles, and F. L. Lewis. Deadlock-free dynamic resource assignment in multi-robot systems with multiple missions: Application in wireless sensor networks. *Journal of Control Theory and Applications*, 8(1):12–19, 2010.
- [48] Maxim Batalin and Gaurav Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. *Journal of Telecommunication Systems*, 26(2-4):181–196, 2004.
- [49] Jang-Ping Sheu, Jun-Yung Hsieh, and Po-Wen Cheng. Design and implementation of mobile robot for nodes replacement in wireless sensor networks. *Journal of Information Science and Engineering*, 24(2):393–410, 2008.
- [50] Onur Tekdas, Volkan Isler, Jong Hyun Lim, and Andreas Terzis. Using mobile robots to harvest data from sensor fields. *IEEE journal of Wireless Communications*, 16(1):22–28, February 2009.
- [51] Aleksandar Rodic, Dusko Katic, and Gyula Mester. Ambient intelligent robot-sensor networks for environment surveillance and remote sensing. In *Proceedings of the International Symposium on Intelligent Systems and Informatics*, pages 39–44, Subotica, Serbia, 2009.
- [52] Jason O’Kane and Wenyan Xu. Energy-efficient target tracking with a sensorless robot and a network of unreliable one-bit proximity sensors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 750–755, Kobe, Japan, May 2009.

- [53] Jason M. O’Kane and Wenyuan Xu. Network-assisted target tracking via smart local routing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4818–4823, Taipei, Taiwan, October 2010.
- [54] W. Ye, J. Heidemann, , and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of IEEE joint conference of computer and communication societies (INFOCOM)*, pages 1567–1576, New York, USA, June 2002.
- [55] I. Ulrich and J. Borenstein. Vhf*: local obstacle avoidance with look-ahead verification. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2505–2511, San Francisco, USA, April 2000.
- [56] Emanuele Menegatti, Andrea Zanella, Stefano Zilli, Francesco Zorzi, and Enrico Pagello. Range-only slam with a mobile robot and a wireless sensor network. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 8–14, Kobe, Japan, May 2009.
- [57] Wanming Chen, Tao Mei, Huawei Liang, Zhuhong You, Shuai Li, and Max Q-H Meng. Environment-map-free robot navigation based on wireless sensor networks. In *Proceedings of the International Conference on Information Acquisition*, pages 569–573, Seogwipo, South Korea, August 2007.
- [58] G. Welch and G. Bishop. An introduction to the kalman filter. Technical Report 95-041, Department of Computer Science, University of North Carolina at Chapel Hill, 2004.
- [59] Player. The player project. <http://playerstage.sourceforge.net/>, 2010.
- [60] AWARE. The aware project team aware project webpage. <http://www.aware-project.net>, 2010.
- [61] Aysegul Tuysuz Erman, Lodewijk van Hoesel, and Paul Havinga. Aware: Platform for autonomous self-deploying and operation of wireless sensor- actuator networks cooperating

- with aerial objects. In *Proceedings of the Centre for Telematics and Information Technology Symposium*, Twente, Netherland, May 2007.
- [62] Eleri Cardozo, Eliane Guimarães, Lucio Agostinho, Ricardo Souza, Fernando Paolieri, and Fernando Pinho. A platform for networked robotics. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 1000–1005, Taipei, Taiwan, October 2010.
- [63] S. Cantor, J. Kemp, R. Philpott, and E. Maler. Assertions and protocols for the oasis security assertion markup language (saml). <http://www.oasis-open.org>, March 2005.
- [64] ARIA. Mobilerobot’s advanced robot interface for applications. <http://robots.mobilerobots.com/wiki/ARIA>, 2010.
- [65] L. Richardson and S. Ruby. *RESTful Web Services*. O’Reilly, 2007.
- [66] Kyatera. Kyatera - fiber-to-the-lab. <http://www.kyatera.fapesp.br/>, 2011.
- [67] TEP 123. Tep 123 - the collection tree protocol. <http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html>, 2010.
- [68] HyungJune Lee, Alberto Cerpa, and Philip Levis. Improving wireless simulation through noise modeling. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN ’07, pages 21–30, New York, NY, USA, 2007. ACM.
- [69] MobileSim. Mobilesim - mobilerobot’s research and academic customer support. <http://robots.mobilerobots.com/wiki/MobileSim>, 2011.
- [70] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1398–1404, Sacramento, California, USA, April 1991.

- [71] J. Barraquand, B. Langlois, and J. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems Man, and Cybernetics*, 22(2):224–241, March 1992.