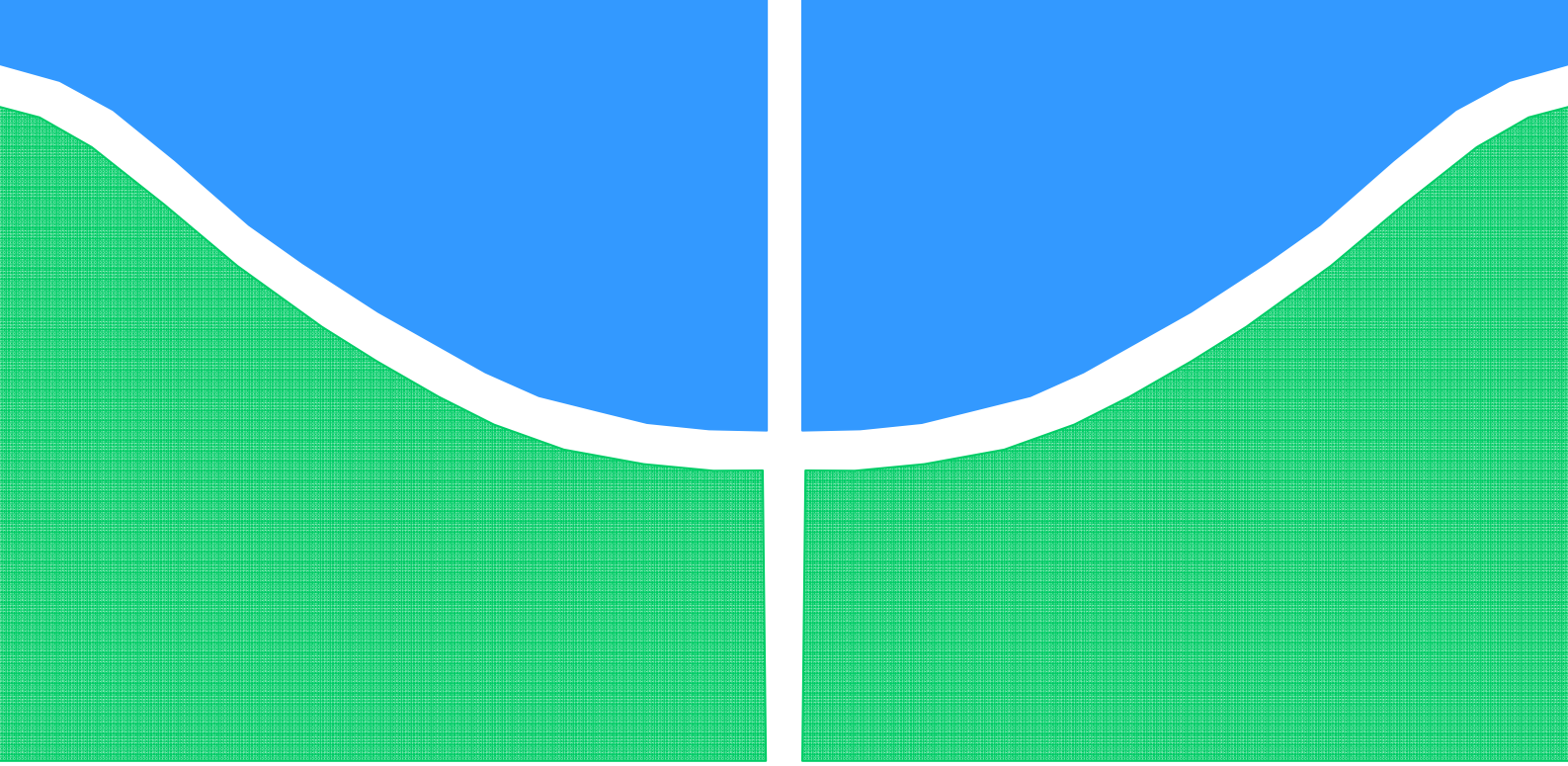


PROJETO DE GRADUAÇÃO

Medição Móvel de Conforto Térmico para Rede de Automação Predial W*ireless*

Fillipe Lopes do Couto
Luis Felipe da Cruz Figueredo

Brasília, Julho de 2008

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

PROJETO DE GRADUAÇÃO

Medição Móvel de Conforto Térmico para Rede de Automação Predial *Wireless*

Fillipe Lopes do Couto
Luis Felipe da Cruz Figueredo

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro de Controle e Automação.

Banca Examinadora

Prof. Adolfo Bauchspiess, UnB/ENE (Orientador) _____

Prof. Geovany Araújo Borges, UnB/ENE _____

Prof. Jacir Bordim, UnB/CIC _____

Brasília, Julho de 2008

FICHA CATALOGRÁFICA

FILLIPE, COUTO
LUIS, FIGUEREDO

Medição Móvel de Conforto Térmico para Rede de Automação Predial *Wireless*, [Distrito Federal], 2008.

xii, 85p., (FT/UnB, Engenheiro, Controle e Automação, 2008). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. ZigBee
2. Técnicas de localização
3. RSSI
4. Propagação de ondas de rádio
5. Redes *Wireless*
6. Redes Neurais

I. Mecatrônica/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

COUTO, F. L., FIGUEREDO, L. F. C., (2008). Medição Móvel de Conforto Térmico para Rede de Automação Predial *Wireless*. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-nº 011/2008, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 85p.

CESSÃO DE DIREITOS

AUTORES: Fillipe Lopes do Couto, Luis Felipe da Cruz Figueredo.

TÍTULO DO TRABALHO DE GRADUAÇÃO: Medição Móvel de Conforto Térmico para Rede de Automação Predial *Wireless*.

GRAU: Engenheiro ANO: 2008

_____

Fillipe Lopes do Couto
Q.I. 25, Bloco 'M', ap 503 – Guará II
71060-250 Brasília – DF – Brasil

_____

Luis Felipe da Cruz Figueredo
SQN 309, Bloco 'D', ap 301 – Asa Norte
70755-040 Brasília – DF – Brasil

RESUMO

O avanço das tecnologias de dispositivos de processamento móveis e a proliferação de redes locais *wireless* têm favorecido o crescente desenvolvimento de tecnologias e serviços na área de localização. Novos conceitos em automação predial e novas tecnologias de sensores promoveram a utilização de redes de sensores em serviços e sistemas de localização.

Inserido neste contexto, a proposta do presente trabalho é desenvolver um sistema de localização que, utilizando uma rede de sensores *wireless*, possa ser incorporado em um ambiente inteligente. Adicionalmente, através da utilização do parâmetro *Received Strength Signal Indicator* – RSSI, investigar e comparar a eficiência do uso da teoria de propagação de ondas de rádio com relação ao uso de métodos inteligentes, como as redes neurais.

Com este intuito, construiu-se uma rede de sensores *wireless* e um módulo sensor portátil capazes de se comunicarem através do protocolo ZigBee IEEE 802.15.4. Os resultados são apresentados com foco na acurácia e na precisão dos métodos utilizados. O sistema de localização utilizando redes neurais artificiais apresentou, com relação a estes aspectos, resultados mais efetivos do que o sistema fundamentado, apenas, na teoria analítica de propagação de rádio.

Palavras Chave: ZigBee, técnicas de localização, RSSI, teoria de propagação de ondas de rádio, redes wireless, redes neurais.

ABSTRACT

The advances of mobile computing devices and the proliferation of local wireless networks have fostered the increasing development of location-aware technologies and services. New concepts in building automation and new sensor technologies promoted the utilization of networks of sensors in services and localization systems.

Within this context, the purpose of this work is to develop a localization system that, by using wireless sensor networks, may be incorporated in an intelligent ambient. In addition, through the Received Strength Signal Indicator – RSSI parameter, investigate and compare the effectiveness of the radio waves propagation theory, in contrast to intelligent adaptive methods, like neural networks.

With this intention, a wireless sensor network and a mobile sensor capable to communicate through ZigBee IEEE 802.15.4 protocol has been built. The results are shown with focus on the accuracy and precision of the used methods. The localization system through neural networks has shown, based on these aspects, better results than those acquired through radio waves analytical theory.

Keywords: ZigBee, localization techniques, RSSI, radio waves propagation theory, wireless networks, neural networks.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AES – *Advanced Encryption Standart*

AoA – *Angle-of-Arrival*

AmI – *Ambient Intelligence*

API – *Application Programming Interface*

CLP – Controlador Lógico Programável

CMOS – Complementary Metal-Oxide Semiconductor

DSS – *Distribution System Service*

GCC – *GNU C Compiler*

IEEE – *Institute of Electrical and Electronics Engineers*

ISM – *Industrial, Scientific and Medical*

LAVSI – Laboratório de Automação, Visão e Sistemas Inteligentes

MAC – *Medium Acess Control*

OSI – *Open System Interconection*

PAN – *Personal Area Network*

PET – *Physiological Equivalent Temperature*

PMV – *Predicted Mean Vote*

RISC – *Reduced Instruction Set Computer*

RSSI – *Received Strength Signal Indicator*

ToA – *Time-of-Arrival*

Wi-fi – *Wireless Fidelity*

1 Introdução

Ambient Intelligence (AmI) é um paradigma emergente relativo ao desenvolvimento de ambientes inteligentes com foco centrado no usuário. Dentro deste contexto, a automação predial prevê a consolidação de uma rede de sensores e atuadores com diversos fins, entre eles, oferecer conforto térmico, economia de energia e segurança. Existem também paradigmas semelhantes ao *Ambient Intelligence*, como o *Ubiquitous Computing*, *Pervasive Computing*, e o mais recente deles, o *Everyware*, que também descrevem tais sistemas.

A rápida evolução das tecnologias eletroeletrônica e digital e a drástica redução de seus custos têm promovido o desenvolvimento da automação predial. Além disso, a proliferação de redes locais sem fio e o avanço nas pesquisas sobre novos protocolos de redes sem fio – *wireless* – promoveram o desenvolvimento de padrões direcionados às aplicações em automação predial. Uma das grandes vantagens da utilização de redes *wireless* para a automação predial é o *retrofitting* de prédios já construídos, ou seja, a instalação de uma rede de sensoriamento sem a necessidade de reformas e com redução nos custos de instalação e manutenção.

Neste contexto, o Laboratório de Automação, Visão e Sistemas Inteligentes – LAVSI – está conduzindo um amplo projeto de automação predial. Este projeto conta com vários colaboradores e visa à instalação de uma planta piloto para o desenvolvimento de ferramentas de promoção do conforto térmico com a racionalização eficiente de energia.

O estímulo para a elaboração deste projeto surgiu após o estudo do trabalho exposto em Indria (2006), no qual se estuda o desenvolvimento de um módulo pessoal de medição, utilizando-se o índice *Predicted Mean Vote* (PMV) para se estimar o conforto térmico de um indivíduo. Este índice possui sete valores, variando de -3 (que corresponde a frio) a +3 (que corresponde a quente), e é estimado com base na ISO 7730. Para seu cálculo, se faz necessário o sensoriamento das seguintes variáveis de conforto térmico: a temperatura do ar, temperatura radiante média, velocidade do ar, umidade relativa do ar (medido indiretamente pela pressão do vapor de água no ar), tipo de vestuário do indivíduo e o nível de atividade do indivíduo.

Tendo o PMV como base, o trabalho de Indria (2006) expôs a construção de um módulo móvel, ao qual há sensores que monitoram algumas das variáveis pessoais de conforto térmico. O sistema sensorial deste módulo móvel é constituído

por um termopar, que estima a temperatura do indivíduo portador do módulo, e um circuito que recebe os dados de um medidor de freqüência cardíaca comum, utilizado por esportistas. Em Indria (2006), ainda, pode-se encontrar estudos relativos a sistemas de localização do indivíduo dentro de um recinto, para que outras variáveis componentes do PMV pudessem ser estimadas. Porém, este sistema de localização não demonstrou resultados significativos, em comparação com os resultados obtidos pelo sistema de monitoração das variáveis de conforto térmico do módulo.

Por esse motivo, o foco do presente projeto foi voltado para a construção de um sistema de localização utilizando-se o módulo pessoal e a rede de sensores do ambiente, completando a funcionalidade do módulo proposto no trabalho de Indria (2006). Dentre os projetos pesquisados que abordam estudos de localização com redes de sensores, dois se destacam. Um deles é a proposta encontrada em Dankwa (2004), onde várias técnicas de localização são citadas, tendo como base a medida da intensidade do nível de sinal (potência) entre os nós da rede e o módulo móvel: assim, o trabalho expõe um intenso estudo entre energia de transmissões e sua relação com a distância, buscando um Modelo de Propagação de Rádio que estime a distância entre dois nós da rede pela medida de nível de sinal. O outro é o trabalho encontrado em Mao *et al*. (2007), onde inúmeras técnicas de localização em redes de sensores sem fio são descritas e analisadas. Além destes trabalhos, Terwilliger *et al.* (2004) e Nunes (2006) também apresentam estudos importantes sobre métodos de localização.

Portanto, o presente trabalho propõe a criação de sensores portáteis que permitam a promoção do conforto térmico direcionada ao usuário. Assim, em recintos coletivos, as variáveis correlatas para cada pessoa seriam tratadas e o sistema de controle adaptado para o conforto individualizado e não mais se utilizando uma região definida por um conjunto de sensores fixos. Entretanto, estes sensores portáteis são pouco úteis sem o conhecimento de sua posição dentro do ambiente de automação predial. Assim, desenvolvem-se, no trabalho, técnicas de localização de baixo custo e com a utilização de recursos da própria rede de sensores *wireless* do ambiente.

O Capítulo 2 descreve o embasamento teórico das diversas ferramentas utilizadas ao longo da confecção deste trabalho, como os protocolos wireless existentes, a teoria de propagação de ondas eletromagnéticas, as técnicas de localização mais comuns e uma breve introdução sobre redes neurais. O Capítulo 3

expõe os dispositivos empregados para o estudo das técnicas de localização elaboradas, mostrando as principais características das ferramentas de *hardware* e *software*. O Capítulo 4 explica como os conceitos teóricos e o aparato experimental foram utilizados em conjunto para obter os resultados presentes neste trabalho. O Capítulo 5 expõe os resultados alcançados e a comparação dos mesmos com os resultados que eram esperados. Por fim, o Capítulo 6 resume as conclusões obtidas da análise dos resultados e os confronta com os objetivos propostos deste trabalho, além de traçar perspectivas futuras de aplicações com as informações fornecidas pelo presente projeto.

2 Fundamentação Teórica

2.1 O Padrão Zigbee

O padrão ZigBee foi desenvolvido por um consórcio de empresas (a *ZigBee Alliance*) em conjunto com o IEEE, visando o desenvolvimento de um protocolo padrão para redes de sensores *wireless*. Devido às necessidades deste tipo de rede, a *ZigBee Alliance* desenvolveu um padrão aberto, global, de baixo consumo de energia, robusto, confiável e de baixo custo utilizando o protocolo IEEE 802.15.4.

A *ZigBee Alliance* surgiu como um consórcio de oito empresas fundadoras: Chipcon, Ember, Freescale, Honeywell, Mitsubishi, Motorola, Philips e Samsung. Atualmente, a organização possui em torno de 175 empresas associadas em mais de 29 países (HEILE, 2005).

2.1.1 Características Gerais

O padrão ZigBee IEEE 802.15.4 foi desenvolvido especialmente para ser utilizado em aplicações de sensoriamento e em aplicações de controle e acionamento de dispositivos. Foi projetado para transmitir com taxa máxima de 250kbps, trabalhando na freqüência ISM (*Industrial, Scientific, and Medical*) de 2,4 GHz (com exceção dos Estados Unidos e da Europa, que trabalham com 915 MHz e 868 MHz, respectivamente), sem necessidade de licença de uso. Um módulo ZigBee consome apenas 50 mA de corrente quando ativo e menos de 50 µA quando em modo *sleep* (ou *standby*).

Outro fator que influenciou na escolha do padrão ZigBee IEEE 802.15.4 foi a versatilidade de configurações de rede deste padrão. Pode-se trabalhar com topologias ponto-a-ponto, ponto-a-multiponto, par-a-par e malha. Segundo Heile (2005), um módulo mestre pode coordenar até 254 módulos escravos e a rede pode trabalhar com até 65536 módulos. Outras vantagens do protocolo ZigBee IEEE 802.15.4 são: possibilidade de configuração automática da rede, endereçamento dinâmico de módulos escravos, controle total por *handshaking* nas transferências de pacotes, garantindo maior integridade dos dados e possibilidade de utilização de criptografia AES de 128 bits.

2.1.2 Arquitetura do Padrão ZigBee

Segundo o artigo apresentado por Baumann (2006), todos os módulos que trabalham com o padrão ZigBee possuem, necessariamente, um rádio transmissor/receptor IEEE 802.15.4, e um microcontrolador com a camada MAC embarcada em seu *firmware*. A definição da camada física e da camada de acesso ao meio é responsabilidade da norma IEEE 802.15.4, enquanto que o ZigBee define as outras camadas do modelo OSI. A arquitetura do padrão é representada pela Figura 2.1.

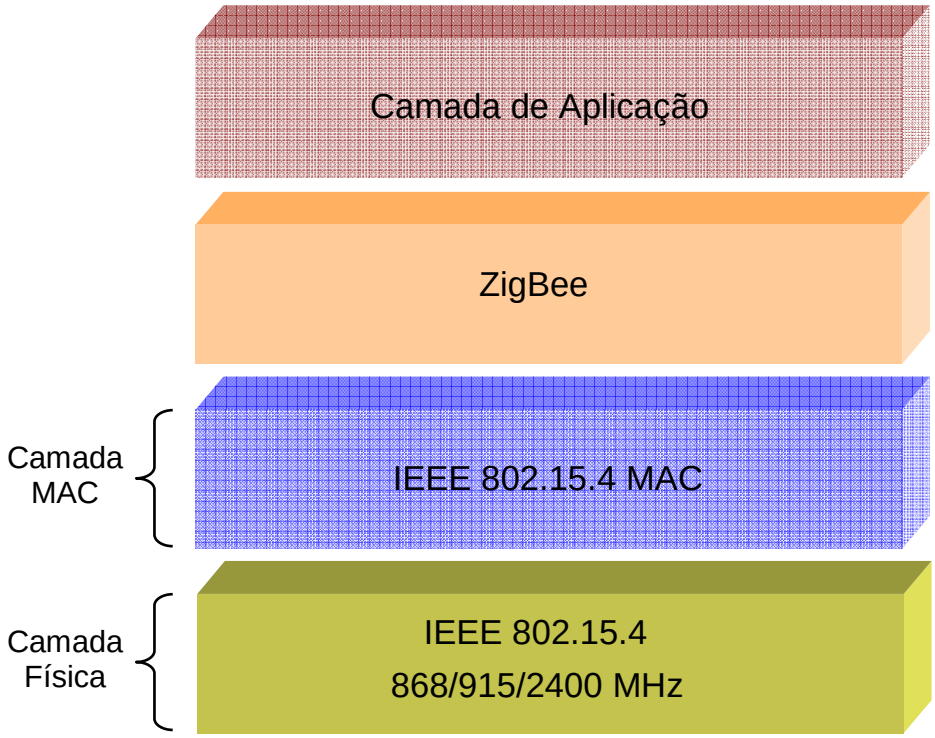


Figura 2.1 – Estrutura do Padrão Zigbee (DVORAK, 2005).

A camada física definida pela norma IEEE 802.15.4 foi projetada para fornecer um alto grau de integração pela utilização da transmissão de Seqüência Direta (DSS) que possibilita a utilização de equipamentos muito mais simples e implementações mais baratas (CARVALHO e PASSARELA, 2006).

A camada MAC foi desenvolvida objetivando soluções de alto desempenho e baixo consumo com baixa complexidade e utilizando pouca memória de programação para a redução dos custos agregados (BAUMANN, 2006). A camada permite a utilização de topologias múltiplas de rede e o controle de um grande número de dispositivos sem a necessidade de operações complexas. A camada

MAC ainda permite a utilização do padrão AES como algoritmo de criptografia (entretanto, seu controle é feito pelas camadas superiores).

Entre as outras camadas, implementadas pela pilha de protocolos do padrão ZigBee, destaca-se a camada de rede/segurança projetada para operar com grandes quantidades de nós com baixa potência e latências relativamente baixas (PINHEIRO, 2004). A pilha de protocolos do padrão permite o balanceamento dos custos em unidades especificas, reduzindo o consumo de energia e fornecendo soluções de melhor custo-desempenho para as aplicações.

2.1.3 Protocolos Concorrentes

A escolha do padrão de comunicação *wireless* a ser implementada na rede de sensores é de vital importância para a aplicação, uma vez que o sistema deve se adequar às necessidades de redes desse tipo, como robustez, segurança na transmissão de dados e módulos com baixo consumo de energia. Além do padrão ZigBee IEEE 802.15.4, outros protocolos e meios de comunicação *wireless* foram estudados e suas características comparadas com as necessidades da rede de sensores.

Na revisão da literatura, foram encontrados trabalhos como o de Raghuvanshi (2006), que cita a comunicação através de radiação infravermelha inadequada para projetos de sensoriamento remoto, devido à necessidade de caminho direto da informação para a transmissão de dados (ou seja, o sinal não atravessa obstáculos, como móveis e paredes).

Com base em Raghuvanshi (2006), Dodd e Windsor (2006) e Monsignore (2007), realizou-se uma comparação entre o ZigBee e outros meios de comunicação *wireless* comumente estudados para aplicação em redes de sensores, como o Bluetooth e o Wi-Fi. As características das principais versões destes protocolos, em comparação com as características do ZigBee, são apresentadas na Tabela 2.1.

Tabela 2.1 – Análise comparativa dos padrões
de comunicação *wireless* (DVORAK, 2005).

Padrão	Taxa de transmissão	Consumo de Energia	Tamanho da pilha	Vantagens	Aplicações
Bluetooth (1.0)	1 Mbps	50 mA TX e 0.2 mA em standby.	100KB	Interoperabili-dade, baixo consumo.	USB wireless
Wi-Fi (802.11g)	54 Mbps	Mais de 400 mA TX e 20 mA em standby.	Mais de 100KB	Alta taxa de transmissão e flexibilidade.	Internet, rede de computadores.
ZigBee	250 Kbps	50 mA TX e menos de 50 µA em standby.	34KB / 14KB	Baixo consumo, baixo custo.	Controle remoto, sensores

Os prós e os contras de cada sistema são detalhados a seguir.

2.1.3.1 Bluetooth

O protocolo de comunicação Bluetooth também trabalha na faixa de freqüência de 2,4 GHz ISM, e é utilizado comumente em produtos que requerem comunicação com pouca distância, como transmissão entre celulares, mouse sem fio e até mesmo monitores de freqüência cardíaca. Esse sistema possui três classificações de nível de sinal, podendo chegar a até 100 metros. Apesar de utilizar um *transceiver* de baixo consumo de energia (característica necessária para a rede de sensores), o Bluetooth possui uma natureza hierárquica que limita suas redes a somente sete dispositivos, o que impossibilita seu uso em redes que necessitem de grande quantidade de nós.

2.1.3.2 Wi-Fi

Wi-fi é a sigla para o termo “*wireless fidelity*”. Este protocolo tem este nome por ser altamente robusto, com significativa imunidade a ruído e capacidade de trabalhar em altas velocidades (comparáveis à Ethernet). Por esse motivo, este é o protocolo de comunicação utilizado em lugares com acesso à Internet *wireless*. Entretanto, seus adaptadores possuem alto custo, e também apresentam alto consumo de energia, características indesejadas ao trabalhar com rede de sensores.

2.2 Comunicação Wireless e Ondas Eletromagnéticas

A comunicação *wireless* baseia-se na transmissão de dados através de ondas de rádio, que são constituídas de perturbações eletromagnéticas periódicas que variam na faixa de freqüência de 30 MHz a 3 GHz. Em Hecht (2002), são encontradas informações que abordam a teoria da propagação de ondas eletromagnéticas, como modelamento físico e matemático e padrões de transporte de energia. Uma vez que o presente trabalho está fortemente ligado à comunicação sem fio, uma análise mais profunda de fenômenos de propagação eletromagnética se faz necessário para melhor compreender as características que envolvem este meio de comunicação.

2.2.1 Revisão das Equações de Maxwell

As contribuições mais importantes de famosos cientistas ao estudo de campos elétricos e magnéticos e suas interações estão resumidas em apenas quatro equações comentadas a seguir, denominadas Equações de Maxwell.

A **Lei da Indução de Faraday** (Equação 1) define que um campo elétrico é criado devido à variação de um fluxo magnético através de uma área "*A*". A **Lei de Gauss – Elétrica** (Equação 2) relaciona o fluxo de campo elétrico através de uma superfície fechada de um volume "*V*" com a carga total em seu interior, onde ***ε*** é a **permissividade elétrica** do meio. A **Lei de Gauss – Magnética** (Equação 3) prevê que não existem (ou, pelo menos, não há como provar a existência de) cargas puramente magnéticas, ou seja, não existem cargas magnéticas pontuais (monopólos magnéticos) que atuam como fontes ou sorvedouros de campos magnéticos. Por fim, a **Lei de Ampère** demonstra que um campo magnético é criado quando há uma corrente elétrica e/ou variação de um campo elétrico, onde ***ε*** é a **permissividade elétrica** do meio e ***μ*** é a **permeabilidade magnética** do meio.

$$\oint_C \vec{E} \cdot d\vec{l} = -\iint_A \frac{\partial \vec{B}}{\partial t} \cdot d\vec{S} \qquad (1)$$

$$\iint_A \vec{E} \cdot d\vec{S} = \frac{1}{\varepsilon} \iiint_V \rho \cdot dV \qquad (2)$$

$$\iint_A \vec{B} \cdot d\vec{S} = 0 \tag{3}$$

$$\oint_C \vec{B} \cdot d\vec{l} = \mu \iint_A \left(\vec{J} + \varepsilon \frac{\partial \vec{E}}{\partial t} \right) \cdot d\vec{S} \tag{4}$$

Uma vez que as ondas eletromagnéticas de dispositivos *wireless* se propagam no ar, as Equações de Maxwell assumem formas mais simples, devido à inexistência de corrente (representada pelo vetor densidade de corrente *J*) e carga elétrica (representada pela integral tripla do lado direito da Equação 2). Assim, as Equações 1, 2, 3 e 4 se tornam, respectivamente, as Equações 5, 6, 7 e 8, onde ε_0 e μ_0 são, respectivamente, a **permissividade elétrica no vácuo** e a **permeabilidade magnética no vácuo**.

$$\oint_C \vec{E} \cdot d\vec{l} = -\iint_A \frac{\partial \vec{B}}{\partial t} \cdot d\vec{S} \tag{5}$$

$$\iint_A \vec{E} \cdot d\vec{S} = 0 \tag{6}$$

$$\iint_A \vec{B} \cdot d\vec{S} = 0 \tag{7}$$

$$\oint_C \vec{B} \cdot d\vec{l} = \mu_0 \varepsilon_0 \iint_A \frac{\partial \vec{E}}{\partial t} \cdot d\vec{S} \tag{8}$$

Particularmente, as Equações 5 e 8 demonstram a existência de oscilações eletromagnéticas, uma vez que a Equação 5 mostra que, ao variar-se um campo magnético, obtém-se um campo elétrico, enquanto a Equação 8 afirma que, ao variar-se um campo elétrico, obtém-se um campo magnético. Assim, estas duas equações descrevem uma propagação, no espaço, de perturbações eletromagnéticas, como, por exemplo, as perturbações harmônicas representadas pela Figura 2.2.

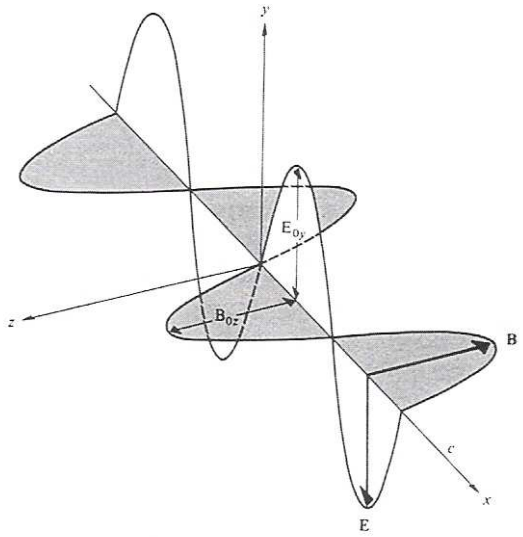


Figura 2.2 – Exemplo de uma onda eletromagnética
se propagando no espaço (HECHT, 2002).

2.2.2 Energia e Irradiância em Ondas Eletromagnéticas

Uma vez que as ondas eletromagnéticas se propagam no espaço, faz-se necessário relacionar a energia transportada em termos da energia radiante por unidade de área. Assim sendo, introduz-se o conceito de **Vetor de Poynting**, representado pela Equação 9.

$$\vec{S} = \frac{1}{\mu_0} \vec{E} \times \vec{B} \tag{9}$$

O módulo do Vetor de Poynting corresponde à potência instantânea por metro quadrado que a onda eletromagnética transmite, enquanto que a direção do produto vetorial *E* **x** *B* define a direção de propagação da energia transportada (e, conseqüentemente, a direção de propagação da onda).

Porém, como o Vetor de Poynting fornece a potência instantânea por área e as ondas de rádio possuem uma freqüência muito alta, torna-se impraticável obter medidas experimentais de energia. Assim sendo, a energia de ondas eletromagnéticas é comumente medida em termos de **Irradiância**, que corresponde ao valor médio do módulo do Vetor de Poynting. Assim, se a propagação é constituída de oscilações harmônicas como as da Figura 2.2, a Irradiância assume a

forma da Equação 10, onde E_0 e B_0 são os valores de amplitude máxima dos campos elétrico e magnético, respectivamente.

$$I = \frac{E_0 B_0}{2\mu_0} \tag{10}$$

2.2.3 Lei do Inverso do Quadrado da Distância

Uma vez que a Irradiância é a medida da potência por unidade de área, então é possível deduzir que a energia que a onda transporta diminui à medida que ela se propaga no espaço, uma vez que sua energia (potência) está distribuída em uma área cada vez maior.

Se imaginarmos ondas eletromagnéticas sendo propagadas por uma fonte pontual, a energia transmitida será distribuída em forma de ondas esféricas, como demonstrado na Figura 2.3. À medida que a onda se propaga, a onda esférica aumenta de tamanho. Como a área de uma superfície esférica é proporcional ao quadrado do raio e a energia se conserva pela superfície da onda, então significa que a potência medida em um ponto diminui com o quadrado da distância, à medida que este ponto se afasta da fonte. Esta afirmação é conhecida como Lei do Inverso do Quadrado da Distância, e é representada pela Equação 11, onde *K* é uma constante e *P* é a potência medida em um ponto distante de *D* unidades de comprimento da fonte.

$$P = \frac{K}{D^2} \tag{11}$$

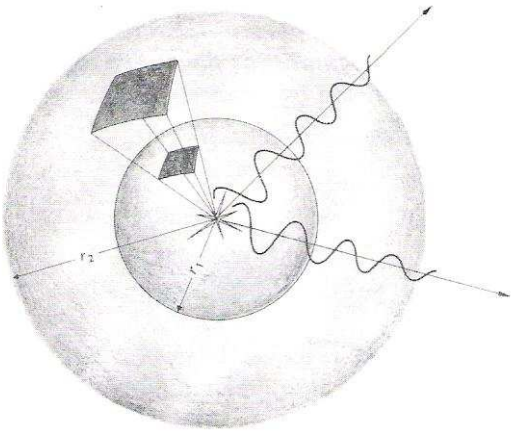


Figura 2.3 – Propagação de ondas eletromagnéticas esféricas por meio de uma fonte pontual (HECHT, 2002).

2.3 Métodos de Localização

Técnicas eficazes sobre localização móvel abarcam sistemas capazes de atender os requisitos da rede e do ambiente de implementação. No caso de ambientes inteligentes, sistemas de localização utilizando redes de sensores sem fio devem atender as restrições de baixo consumo e de baixo custo associadas a estas redes. Nestes sistemas, os métodos de localização são, usualmente, estruturados em duas etapas: a estimação da distância e a combinação das distâncias estimadas (SAVVIDES *et al*., 2002). Entretanto, em Mao *et al*. (2007) e em Nunes (2006) também são encontrados métodos de localização através de técnicas de estimativa da direção do sinal, e métodos de localização através do mapeamento do ambiente.

2.3.1 Métodos Baseados na Estimativa da Direção do Sinal

O método de localização baseado na estimativa da direção de propagação do sinal utiliza o ângulo de chegada do sinal recebido por, ao menos, dois módulos em posições distintas, porém conhecidas. A localização da fonte transmissora é descoberta então, através de geometria básica, onde o encontro das retas definidas pelo ângulo de chegada de cada módulo determina a posição do transmissor (NUNES, 2006). Este processo é representado pela Figura 2.4.

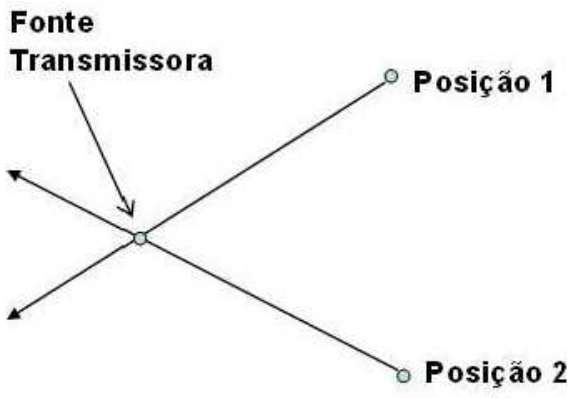


Figura 2.4 – Ilustração do processo de localização a partir do ângulo de chegada do sinal em duas posições distintas (NUNES, 2006).

As técnicas de medição do ângulo de chegada (AoA, do inglês *Angle-of-Arrival*) são baseadas na anisotropia do padrão de radiação das antenas. Um exemplo desta anisotropia é apresentado na Figura 2.5. A idéia, simplificada, é

realizar a rotação da antena receptora com padrão de radiação anisotrópico e analisar o ponto em que se obteve o sinal mais forte, correspondente à direção do módulo transmissor.

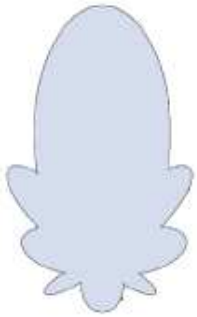


Figura 2.5 – Exemplo do padrão de radiação
de uma antena anisotrópica típica (MAO, 2006).

As grandes desvantagens deste método de localização são a necessidade do uso de antenas com padrão de radiação anisotrópico, o conhecimento prévio deste padrão de radiação, as interferências causadas por multi-caminhos e a necessidade de uma grande quantidade de módulos receptores fixos para se incrementar a acurácia da medição do ângulo de chegada.

2.3.2 Métodos Baseados na Estimativa da Distância entre Módulos

As principais técnicas para a estimativa da distância entre módulos em redes de sensores sem fio são abordadas em Mao *et al*. (2007), onde são descritas as técnicas de obtenção da distância por tempo de chegada e por atenuação do sinal de rádio.

2.3.2.1 Estimativa por Tempo de Chegada

As medições por tempo de chegada (ToA, do inglês *Time-of-Arrival*) relacionam o tempo de propagação do sinal com a distância entre o módulo emissor e receptor. Esta técnica de localização é subdividida em duas outras técnicas: as medições por tempo de propagação direta e por tempo de propagação de retorno.

A técnica de localização por tempo de propagação direta estima a distância através do tempo transcorrido entre a emissão do sinal e a sua recepção em outro módulo. Este método exige um alto grau de sincronia entre o módulo emissor e receptor. Este requisito gera uma demanda adicional de hardware e de

complexidade do sistema, devido à necessidade de relógios altamente acurados e mecanismos eficientes de sincronização (MAO *et al*., 2007).

O tempo de propagação de retorno está relacionado ao tempo transcorrido entre o momento em que um módulo envia um sinal para um segundo módulo até o momento da recepção da resposta advinda deste segundo módulo. Neste caso, como o mesmo módulo, e conseqüentemente, o mesmo relógio é utilizado, eliminam-se os problemas relacionados à sincronia entre os módulos, tornando este método mais atrativo do que a técnica por tempo de propagação direta. Entretanto, mantém-se a necessidade de relógios acurados para o processamento do tempo transcursado.

A grande desvantagem desta técnica de estimativa da distância é que as ondas eletromagnéticas viajam com velocidade próxima a da luz, dificultando as medidas de tempo de chegada e de retorno. Assim sendo, transdutores de ultra-som são mais indicados no uso desta técnica. Entretanto, a adição deste hardware encarece substancialmente o projeto e incrementa o consumo de energia, sendo, portanto, não aconselhado para a utilização em redes de sensoriamento sem fio.

2.3.2.2 Estimativa por Atenuação do Sinal de Rádio

A técnica de obtenção da distância pela atenuação do sinal recebido é baseada em modelos de propagação de ondas de rádio que proporcionam a estimativa da distância entre transmissores de rádio através do nível de energia (potência) do sinal recebido (como apresentado, anteriormente, pela Equação 11). Esta técnica está relacionada à medição do parâmetro RSSI, presente na maioria dos dispositivos *wireless*, que indica a intensidade do sinal recebido.

As desvantagens desta técnica de estimativa da distância estão relacionadas às inúmeras fontes de interferências que atuam sobre as ondas de rádio, reduzindo a precisão da leitura do parâmetro RSSI. Multi-caminhos, reflexão, refração e absorção por objetos físicos são exemplos destas interferências que degradam o sinal de rádio. Outra interferência significativa é a presença de outras redes sem fio com freqüências similares, como o *Wi-Fi*.

Entretanto, apesar destas desvantagens, esta técnica é bastante atrativa para uso em redes de sensores sem fio por não demandar nenhum hardware adicional e por não gerar nenhum impacto significante no consumo de energia, no tamanho dos sensores e no custo do projeto (MAO, 2006).

2.3.2.3 Localização Baseada na Combinação das Distâncias Estimadas

Dankwa (2004) aborda dois métodos de localização baseados na combinação das distâncias estimadas através da medição do nível de energia do sinal de rádio. Na **triangulação hiperbólica** estima-se a distância do módulo móvel para com três outros módulos de posição fixa e conhecida. Para cada módulo são geradas assíntotas hiperbólicas e a partir do cruzamento destas curvas calcula-se a posição do módulo móvel, como ilustrado na Figura 2.6.

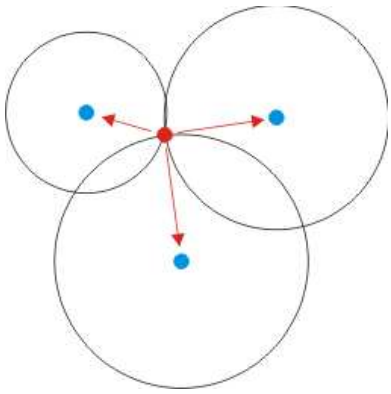


Figura 2.6 – Triangulação Hiperbólica segundo um Modelo de Propagação de Rádio (Dankwa, 2004, com adaptações).

Uma variação da triangulação hiperbólica é o **multilateralismo**, onde, ao invés de somente três, vários outros módulos são utilizados em conjunto para estimar a posição de um módulo móvel, aumentando a acurácia da localização, ao custo de maior quantidade de módulos fixos e cálculo computacional.

2.3.3 Métodos Baseados no Mapeamento do Ambiente

O método de localização baseado no mapeamento do ambiente determina a posição do módulo móvel a partir do uso de determinadas características do sinal de rádio, usualmente o parâmetro RSSI, dependentes da localização do ponto onde tais características são aferidas (NUNES, 2006).

A idéia fundamental desta técnica de localização baseia-se no mapeamento do ambiente com relação ao parâmetro RSSI dos módulos fixos. Este mapeamento é feito através da construção de um banco de dados contendo informações relativas às leituras do parâmetro RSSI de todos os módulos fixos em diferentes posições do ambiente de implementação. A localização do módulo móvel é então estimada

através da comparação dos valores RSSI com os valores armazenados no banco de dados.

A eficácia desta técnica de localização pode ser incrementada pelo uso de algoritmos inteligentes, como as redes neurais artificiais, que estimem as posições não contempladas pelo conjunto de dados do mapeamento do ambiente.

2.4 Redes Neurais

As redes neurais constituem um ramo de sistemas inteligentes que tratam de estruturas compostas por elementos processadores simples, altamente interconectados, utilizando o conexionismo como paradigma (BAUCHSPIESS, 2004). Estes elementos são baseados no sistema nervoso biológico, onde uma função do sistema é largamente determinada pelo número e pela forma de conexão entre os componentes (DEMUTH *et al.*, 2008).

As redes neurais são treinadas para realizar uma determinada função, onde entradas conhecidas gerem saídas esperadas. O treinamento de uma rede neural ocorre através da análise de um conjunto de entradas e saídas conhecidas e de ajustes nos valores das conexões (pesos) entre componentes. Estes ajustes são realizados tomando como base a comparação da saída da rede com o alvo, que corresponde ao resultado esperado. A idéia é que os pesos sejam ajustados até que a saída da rede seja compatível com o resultado esperado. Este conceito é representado pela Figura 2.7.

Os componentes internos da rede são chamados de **neurônios**. O conceito básico de funcionamento destes neurônios é o seguinte: um neurônio recebe uma entrada *p* e a multiplica por um ganho escalar *w* (peso). O produto escalar *wp* é então somado a uma entrada de polarização *b*, também escalar, e o resultado passado como argumento para a função de transferência do neurônio que produz a saída *a*. Este modelo é apresentado na Figura 2.8. Durante o treinamento, uma rede neural reajusta os valores de *w* e de *b* de modo a produzir saídas mais adequadas ao sistema.

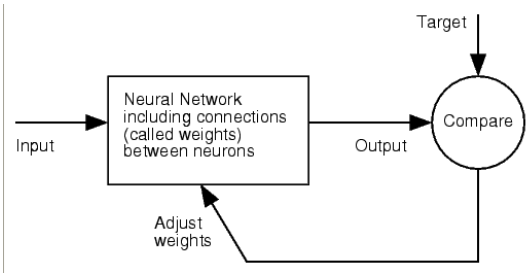


Figura 2.7 – Conceito de ajuste dos pesos entre os componentes da rede (DEMUTH *et al.*, 2008).

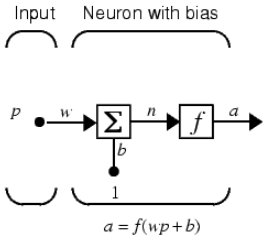
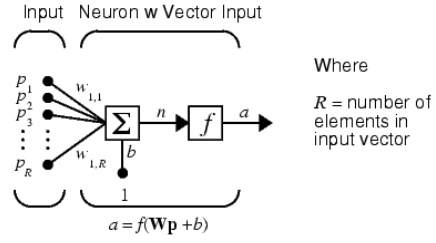


Figura 2.8 – Representação de um neurônio simples (DEMUTH *et al.*, 2008).

Um neurônio pode conter várias entradas (formando um vetor de dados). Neste caso, cada neurônio contém uma matriz **W** que será multiplicada ao vetor de entrada **p** produzindo o vetor **Wp**. O parâmetro passado para a função de transferência do neurônio é a soma dos elementos do vetor **Wp** somado à entrada de polarização *b*. A expressão desta soma pode ser representada como **Wp** + *b*. Tal processo é demonstrado na Figura 2.9.



$$(\mathbf{Wp}+b) = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b$$

Figura 2.9 – Representação de um neurônio com múltiplas entradas (DEMUTH *et al.*, 2008).

Um neurônio é, portanto, constituído de um peso, de um erro de bias e de uma função de transferência. Quando dois ou mais neurônios processam a mesma entrada, este conjunto de neurônios é conhecido como camada de rede. Neste caso, cada neurônio possui uma matriz de peso **W** (que conecta o neurônio a todas as

entradas) e um erro de bias **b**. A saída da camada é o vetor **a** formado pelas saídas de cada neurônio. Este processo é exemplificado pela Figura 2.10, para o caso de R entradas e S neurônios.

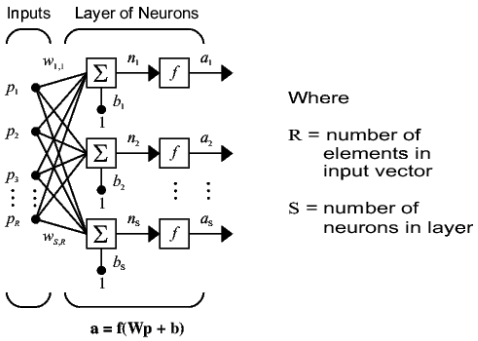


Figura 2.10 – Representação de uma camada com múltiplas entradas e múltiplos neurônios (DEMUTH *et al.*, 2008).

Uma rede neural ainda pode possuir múltiplas camadas em série. Neste caso, cada camada possui seu número de neurônios, e sua matriz de peso e bias associados, não sendo, necessariamente, iguais as outras camadas. Cada camada trabalha como uma camada simples, empregando as saídas de suas camadas precedentes como seu vetor de entrada. Com isso, a representação do sistema tem sua notação alterada. Para melhor visualização dos elementos de cada camada, utiliza-se um índice, sobre o elemento, indicando a que camada este pertence. A camada que produz a saída do sistema neural é chamada de camada de saída, enquanto que as outras camadas são chamadas de **camadas escondidas**.

Um exemplo de uma rede com múltiplas camadas é apresentado na Figura 2.11. Neste exemplo, a rede neural é formada por três camadas em série. A primeira camada possui R_1 entradas e S_1 neurônios. A segunda camada possui S_1 entradas, pois utiliza o vetor de saída da camada 1 como entrada, e S_2 saídas. E a terceira possui S_2 entradas e S_3 saídas. Neste caso a camada 3 é a camada de saída, enquanto que as camadas 1 e 2 são as camadas escondidas.

A classe de rede neural utilizada no projeto foi a *feed-forward backpropagation*. Este método de aprendizagem é utilizado em situações complexas onde mais parâmetros e topologias mais complexas são considerados (ROJAS, 1996).

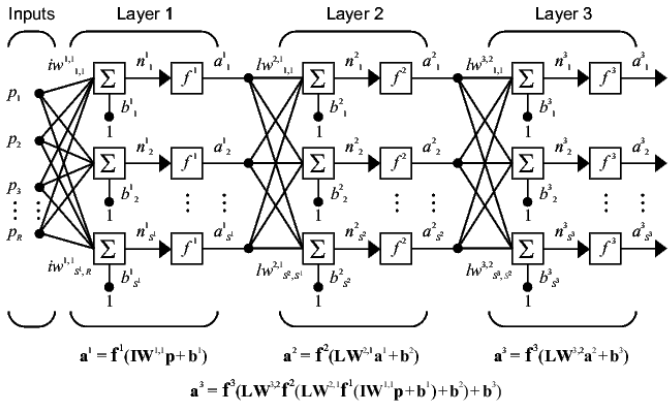


Figura 2.11 – Representação de uma rede com múltiplas camadas (DEMUTH *et al.*, 2008).

Redes *feed-forward* são redes neurais de múltiplas camadas nas quais as saídas dos neurônios se conectam somente com unidades da próxima camada (NEVES, 2006).

O método *backpropagation* (propagação de retorno) está associado diretamente ao gradiente de erro de uma rede. O erro propaga da saída, indo de volta para a entrada (como o próprio nome indica). O algoritmo de *backpropagation* é utilizado para calcular o gradiente de erro da rede e relacioná-lo com os valores de peso modificados (BACKPROPAGATION..., 2008). A combinação de pesos que minimize o gradiente de erro é considerada como a solução para o problema de aprendizagem (ROJAS, 1996). Assim, o algoritmo durante o processamento tende a convergir para pesos que indiquem menores erros (DEMUTH *et al.*, 2008).

3 Aparato Experimental

3.1 O Módulo XBee

3.1.1 Características

Entre as várias empresas membros da ZigBee Alliance que disponibilizam produtos baseados na pilha de protocolos do padrão ZigBee IEEE 802.15.4, trabalharemos com o módulo XBee da empresa MAXSTREAM®, que, atualmente, possui o nome “Digi”. Este dispositivo é um módulo de fácil implementação e baixo custo, além de ser pequeno e atender aos requisitos de baixo consumo de energia. Os módulos XBee podem ser interfaceados com outros dispositivos (como computadores e microcontroladores) através de uma comunicação serial. Os módulos, ainda, podem se comunicar com outros dispositivos de outros fabricantes, desde que estes utilizem o padrão ZigBee IEEE 802.15.4 (FIGUEREDO, 2008). Um módulo XBee típico pode ser visualizado na Figura 3.1.

Figura 3.1 – Exemplo de um módulo XBee (MAXSTREAM, 2006).

A Tabela 3.1 contém as especificações de cada um dos pinos do módulo XBee. Para a utilização do módulo, as conexões mínimas envolvem os pinos 1 (VCC) e 10 (GND), responsáveis pela alimentação do módulo, e os pinos 2 (DOUT) e 3 (DIN), responsáveis pela comunicação serial com outros dispositivos. Os pinos não utilizados devem ser deixados desconectados (MAXSTREAM, 2006). As principais características do módulo são apresentadas nas Tabelas 3.2 e 3.3. Destas tabelas, é possível obter dados interessantes sobre o desempenho e os requisitos de energia do módulo: o alcance é de aproximadamente 30 metros em lugares fechados, e o pico de consumo de 50 mA é verificado no momento de recebimento de dados, caso o XBee seja alimentado com 3,3 V.

Tabela 3.1 – Descrição dos pinos do módulo XBee
(MAXSTREAM, 2006).

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / **CONFIG**	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	**RESET**	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	DTR / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	**CTS** / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / **SLEEP**	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	**RTS** / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

Tabela 3.2 – Características relativas ao desempenho
do módulo XBee (FIGUEREDO, 2008).

Especificações de Perfomance	**XBee**
Alcance Indoor/Urbano	Até 30m
Alcance Outdoor em linha visível	Até 100m
Potência máxima de transmissão	1mW (0 dBm)
Taxa de dados interface serial	Até 115,2 Kbps
Taxa de dados de RF	250 Kbps
Sensibilidade do Receptor	-92 dBm

Tabela 3.3 – Características relativas às especificações elétricas do módulo XBee (FIGUEREDO, 2008).

Especificações Elétricas	XBee
Tensão de Alimentação	2.8 - 3.4 V
Corrente de Transmissão	45mA (@ 3.3 V)
Corrente de Recepção	50mA (@ 3.3 V)
Corrente em modo *Sleep*	< 10 µA

Os módulos XBee possuem três opções de antenas disponíveis para enviar e receber dados: antenas do tipo monopólo (*whip*), antenas do tipo dipólo, e antenas do tipo *chip*. A antena do tipo dipólo possui um ganho de 2,1 dBi, ao passo que as outras antenas possuem ganho de 1,5 dBi. Os padrões de radiação das antenas do tipo monopólo e chip são mostrados nas Figuras 3.2 e 3.3; as antenas do tipo dipólo possuem um padrão de radiação similar à da antena do tipo monopólo.

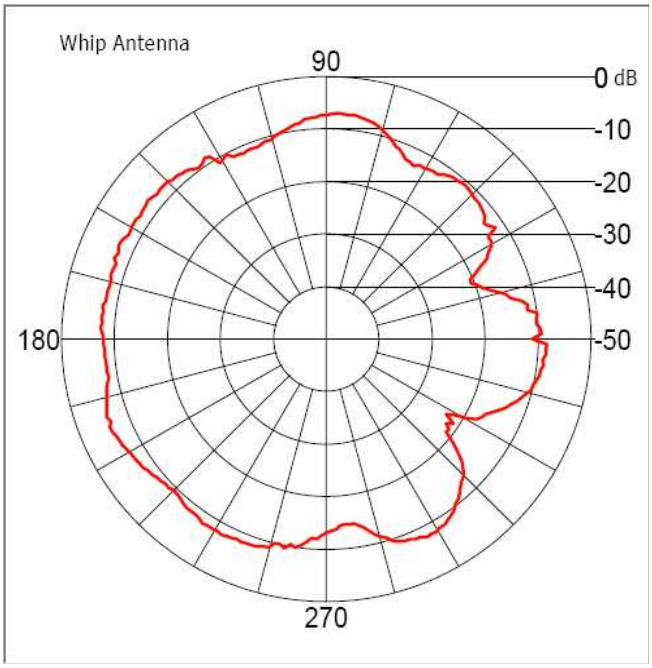


Figura 3.2 – Padrão de radiação das antenas do tipo monopólo (MAXSTREAM, 2005).

Ao analisar a Figura 3.3, verifica-se que a radiação das antenas do tipo *chip* são fortemente influenciadas pela orientação do módulo XBee. Uma vez que o módulo portátil estará sempre mudando sua orientação à medida que se move no recinto, sua utilização para este projeto é desaconselhável. Portanto, para o

presente trabalho, utilizou-se antenas do tipo monopólo (*whip*), pois módulos com antenas do tipo dipólo são mais caros e mais difíceis de se encontrar no mercado.

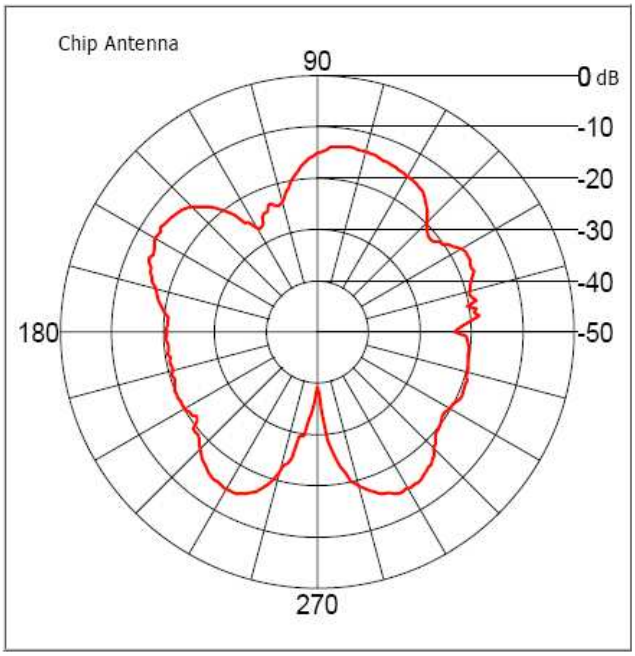


Figura 3.3 – Padrão de radiação das antenas do tipo *chip* (MAXSTREAM, 2005).

3.1.2 Modos de Operação do XBee

3.1.2.1 Modo Transparente

Este é o modo padrão de operação do XBee, onde o módulo trabalha simplesmente como transmissor de dados seriais: todo o dado que é inserido pelo pino de entrada da comunicação serial é transmitido pela antena, e todo dado recebido pela antena é enviado pelo pino de saída da comunicação serial.

3.1.2.2 Modo de Comando

O modo de comando permite acessar variáveis de configuração do módulo. Por meio de uma seqüência específica de caracteres, o XBee entra em um estado no qual parâmetros podem ser configurados e monitorados por meio de seqüências de caracteres, enviadas e recebidas através da comunicação serial do módulo. Entre os parâmetros, pode-se alterar, por exemplo, o canal de transmissão, o endereço do módulo na rede e o endereço de destino.

Para o presente trabalho, alguns comandos são particularmente úteis, pois estão relacionados com a energia de transmissão e recepção. Tais comandos são: o ND, que fornece uma lista com o endereço de fabricação (64bits) e o endereço fonte

(16bits) de todos os módulos XBee que operam no mesmo canal, mesma rede e se encontram ao alcance do módulo que executou o comando, além de fornecer o nível de sinal (em milidecibéis) de cada módulo encontrado; o DB, que retorna o nível de sinal (em milidecibéis) da última mensagem recebida; e o comando PL, que define a potência de transmissão do módulo.

Para mais informações sobre os comandos disponíveis para o dispositivo, consultar MAXSTREAM (2006) e Figueredo (2008).

3.1.2.3 Modo Sleep

Este modo é de grande importância em aplicações wireless devido ao baixo consumo de energia do módulo quando se encontra neste estado. O módulo pode entrar e sair do modo *sleep* via *hardware* (através do pino 9 – SLEEP_RQ) ou via configuração prévia pelo modo de comando: neste último caso, o XBee mantém-se em modo *sleep* em períodos cíclicos. Há, ainda, a possibilidade de utilização de ambos os métodos em conjunto.

3.1.2.4 Modo API

O modo API (*Application Programming Interface*) é uma alternativa para o modo transparente. A diferença é que neste modo os dados seriais são transmitidos através de frames especiais, que possuem dados adicionais para facilitar a configuração e o roteamento de pacotes. Assim, o Modo API provê facilidades como envio de dados para múltiplos destinos sem necessidade de entrar no modo de comando, receber mensagens de sucesso ou falha após cada envio de dados e identificar o endereço do módulo que enviou a mensagem recebida.

3.1.3 Software de Desenvolvimento

A empresa Digi disponibiliza em sua página da Internet (www.digi.com) um software de nome X-CTU, que possibilita a leitura de dados e registradores do XBee através de comunicação serial com um computador. O programa também disponibiliza um terminal, para que comandos possam ser executados e dados possam ser enviados diretamente a outros módulos XBee. Por este *software*, também é possível atualizar o *firmware* do dispositivo.

Para mais informações sobre as características do dispositivo, consultar MAXTREAM (2006).

3.2 Dispositivo de Interfaceamento CON-USBBEE

A empresa ROGERCOM (www.rogercom.com) desenvolve vários produtos para aplicações residenciais e industriais, que utilizam protocolos de comunicação sem fio, como os módulos XBee. Para o presente trabalho, foram adquiridas placas desenvolvidas por essa empresa, de nome CON-USBBEE (Figura 4.4), que constituem circuitos de comunicação entre um módulo XBee e um computador através da interface USB.

O que o dispositivo faz é, simplesmente, utilizar um *driver* (disponível para *download* na página da empresa na Internet), para simular uma porta serial através de uma porta USB. O dispositivo foi de imenso auxílio durante o projeto nas etapas de configuração dos módulos, atualização de *firmwares* e elaboração dos programas de monitoramento e localização.

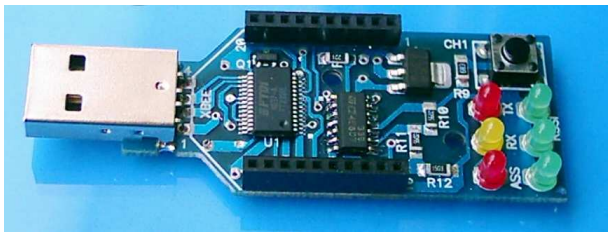


Figura 3.4 – Uma placa CON-USBBEE (www.rogercom.com).

3.3 O Microcontrolador AVR ATmega8

3.3.1 Características

O microcontrolador escolhido para compor o módulo portátil foi o AVR Atmega8, do fabricante ATMEL, devido à familiaridade dos alunos com o dispositivo, advindo de atividades anteriores. Entretanto, este microcontrolador possui outras características interessantes para este projeto, como baixo consumo (3,4 mA trabalhando a 4 MHz), arquitetura de 8-bit de tecnologia CMOS e arquitetura RISC, podendo executar uma instrução por ciclo de relógio. Por fim, o seu conjunto de 130 instruções permite melhor otimização de código de alto nível em linguagem C.

O ATmega8 também já possui vários periféricos integrados dentro de seu encapsulamento, como 6 canais de conversores analógico-digital (quatro com 10 bits de resolução e dois com 8 bits), e oscilador interno de 1 MHz.

A pinagem do encapsulamento P-DIP do ATmega8 com 28 pinos é mostrado na Figura 3.5. Para mais informações sobre o microcontrolador, consultar ATMEL (2004).

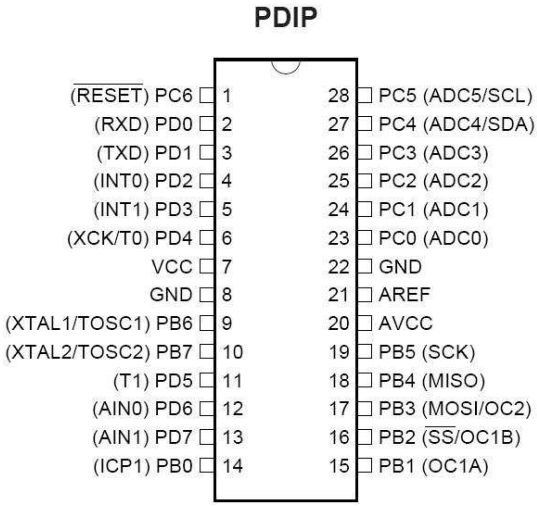


Figura 3.5 – Encapsulamento e pinagem do AVR ATmega8 (ATMEL, 2004).

3.3.2 Software de Desenvolvimento

Em Borges *et al.* (2006), é citado um conjunto de softwares para programação de microcontroladores da família AVR, o WinAVR, disponível em http://sourceforge.net/project/showfiles.php?group_id=68108. O software permite a programação em C ou em *assembler*, mudanças de registradores importantes do AVR (como os *fuses,* que permitem a escolha da fonte de relógio) e possui uma interface gráfica para desenvolvimento de projetos e programação, através do software *Programmer's Notepad*. Para mais informações sobre desenvolvimento de projetos com o AVR, consultar Borges *et al.* (2006).

4 Implementação dos Algoritmos de Localização

4.1 Projeto do *Hardware* de Localização

O *hardware* utilizado no sistema de localização consiste em quatro módulos sensores de localização fixa (previamente definida) e um módulo remoto, cuja localização se deseja obter. O sistema de localização baseia-se na leitura, realizada pelo módulo remoto, do nível de sinal dos módulos sensores. O ambiente no qual as leituras foram realizadas e o sistema de localização implementado foi uma das salas do LAVSI – Laboratório de Automação, Visão e Sistemas Inteligentes. O posicionamento fixo dos módulos sensores dentro deste recinto é representado, na Figura 4.1, pelos X em vermelho, onde foi estabelecido, também, um sistema de coordenadas cartesianas.

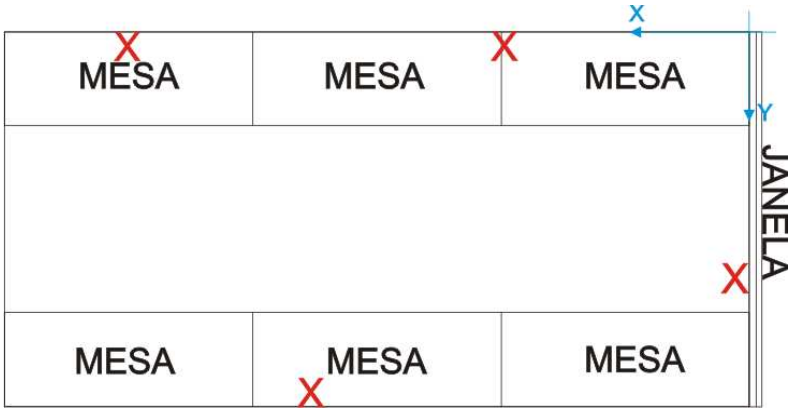


Figura 4.1 – Representação do ambiente onde o experimento de localização foi realizado.

4.1.1 Módulos Sensores

O propósito dos módulos sensores no paradigma *Ambient Intelligence* é realizar a medição da temperatura em determinados pontos do ambiente de pesquisa. Todavia, estes módulos são aproveitados como pontos de referência para o sistema de localização do módulo remoto.

Seu *hardware* foi desenvolvido e simplificado objetivando o menor consumo de energia, visto que há a necessidade de utilização de baterias como fonte de alimentação. O esquemático dos módulos é apresentado na Figura 4.2.

Observa-se do esquemático que o *hardware* é composto basicamente pelo módulo de comunicação XBee e pelo sensor de temperatura (LM35), capaz de medir temperaturas desde -55ºC até 150ºC. Entretanto, esta faixa de temperaturas é muito superior a faixa de aplicação para o sensoriamento térmico de um ambiente, que varia entre 0 ºC e 50 ºC. Como o sensor LM35 fornece uma tensão de saída de 10 mV por ºC, tem-se que a 50 ºC a saída do sensor seria de 500 mV. Esta tensão é, relativamente, muito baixa e qualquer perturbação teria um efeito muito maior sobre a conversão, acarretando um erro sistemático na medição de temperatura. Dessa forma, além da vinculação direta, a saída do LM35 é vinculada aos pinos conversores do módulo XBee através de um controlador proporcional com ganho K_p igual a 6,8, utilizado em conjunto com um saturador de limites entre 0 e 3,3V.

O amplificador operacional utilizado para o projeto do controlador foi o LM324N. Este amplificador foi selecionado por trabalhar com alimentação de fonte simples, fato que elimina a necessidade de fonte simétrica ou de um regulador inversor de tensão deixando o circuito mais simples e compacto.

Os pinos de comunicação serial assíncrona do módulo XBee são conectados à uma barra de pinos de fácil acesso. Esta configuração é mantida para caso haja a necessidade de reconfiguração dos módulos, realizada através de sua USART.

O leiaute da placa de circuito impresso é apresentado nas Figuras 4.3 e 4.4.

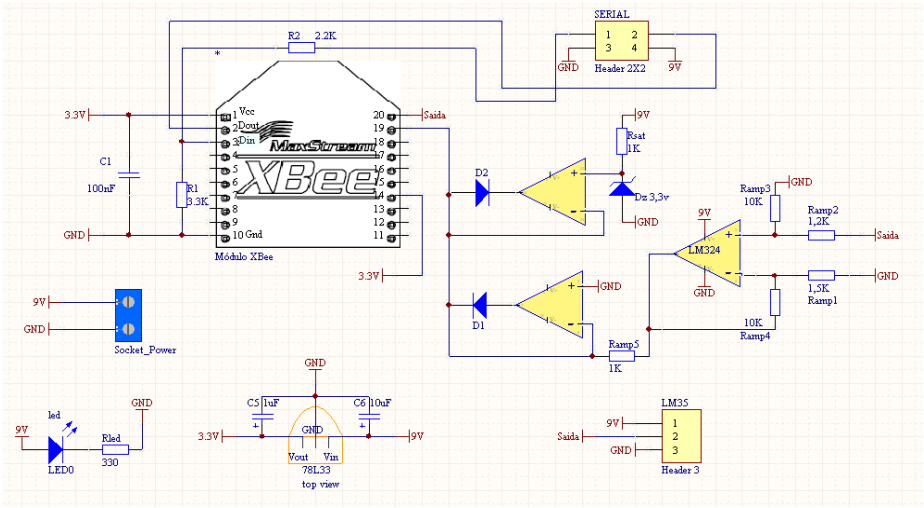


Figura 4.2 – Esquemático do módulo sensor.

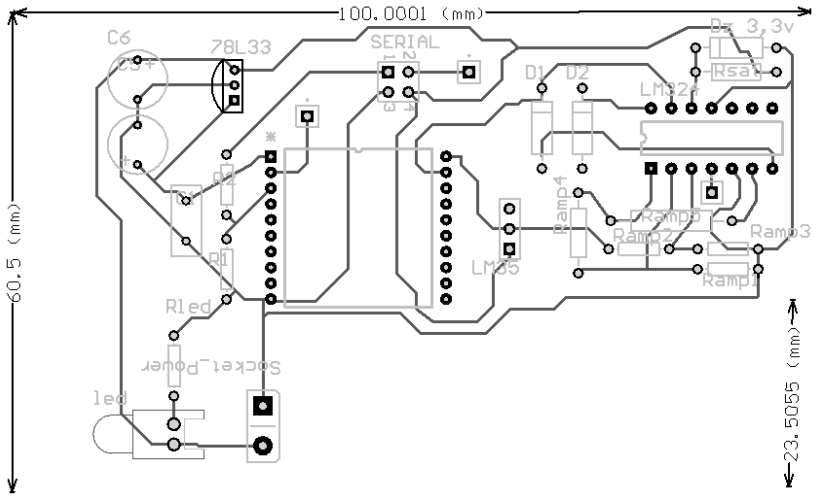


Figura 4.3 – Leiaute do módulo sensor (vista inferior).

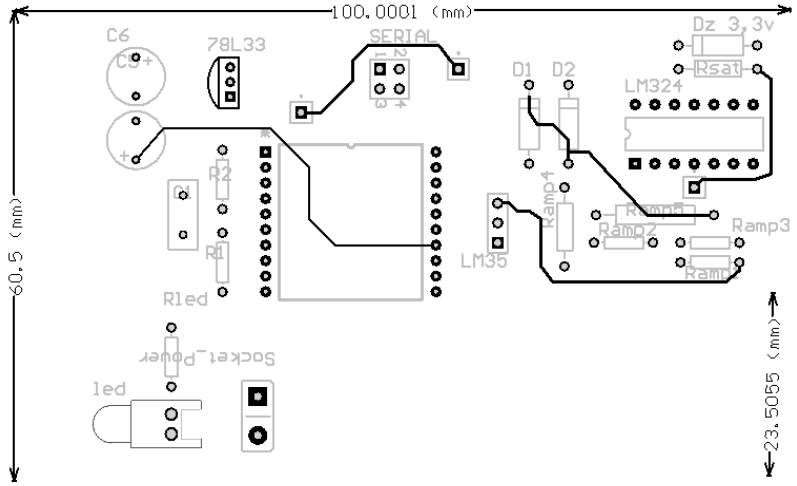


Figura 4.4 – Leiaute do módulo sensor (vista superior).

4.1.2 Módulo Remoto

O *hardware* do módulo remoto é composto por um módulo XBee, por um microcontrolador ATmega8 da Atmel e por um sensor de temperatura LM35 em conjunto com um controlador proporcional, como descrito na seção 4.1.1. O controlador proporcional utilizado no módulo remoto, entretanto, possui ganho K_p igual a 10, visto que a tensão de referência do microcontrolador é de 5 V.

Além destes componentes, o *hardware* do módulo remoto utiliza reguladores de tensão para o ATmega 8 e para o módulo XBee. O esquemático completo do módulo remoto é apresentado na Figura 4.5, enquanto seu referido leiaute de circuito impresso está ilustrado pela Figura 4.6. Por fim, uma foto do circuito construído é mostrada na Figura 4.7.

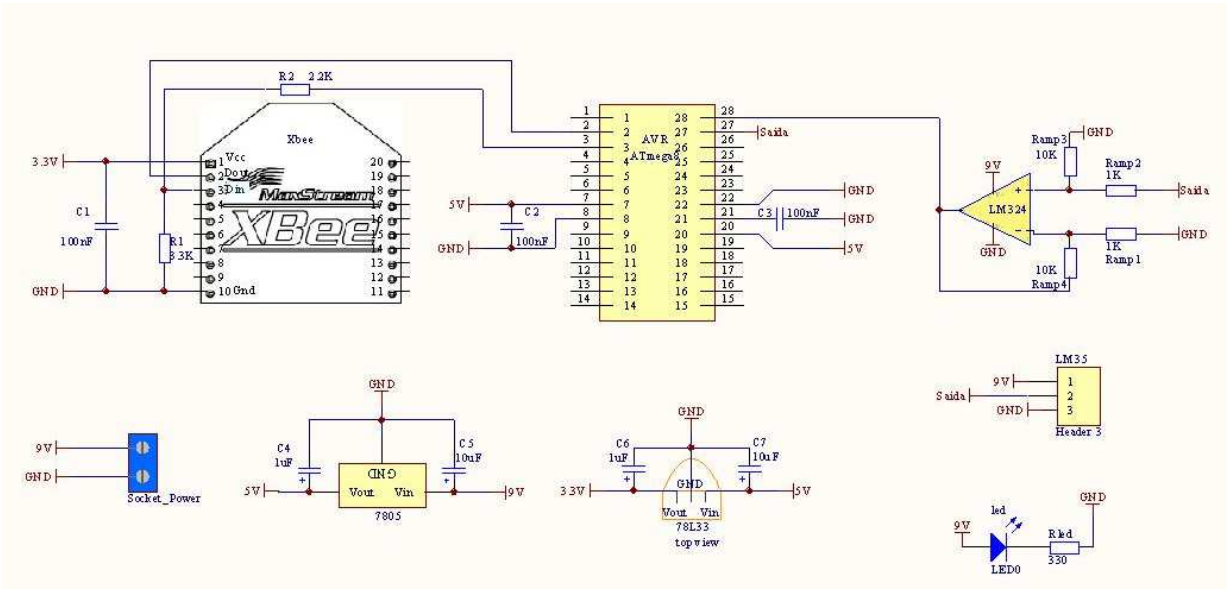


Figura 4.5 – Esquemático do módulo remoto.

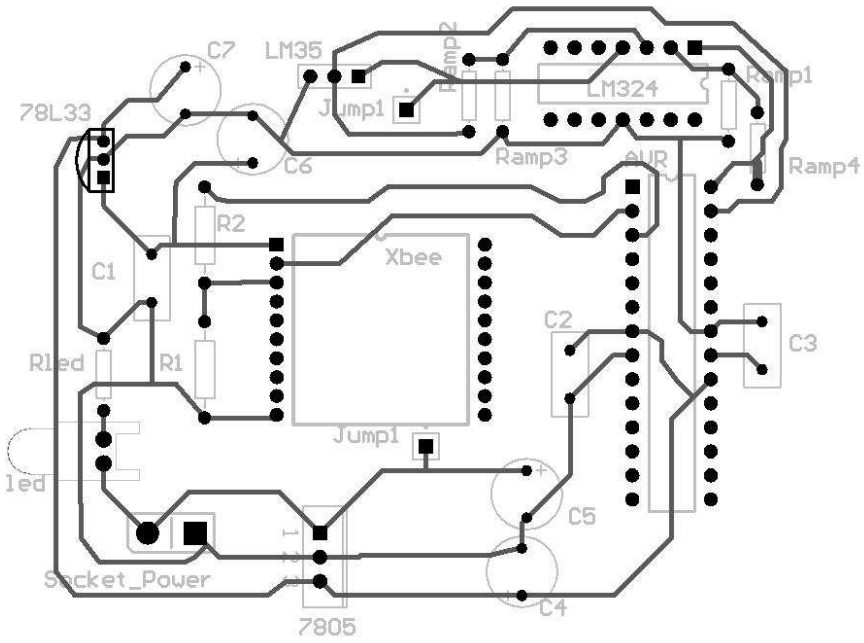


Figura 4.6 – Leiaute do módulo remoto.

O principal objetivo do módulo remoto é realizar a leitura da intensidade do sinal dos módulos sensores e transmitir estes dados para um módulo base, que deverá processar esta informação a fim de obter a localização do módulo. Este módulo base é constituído apenas por um XBee ligado à um computador através de uma CON-USBBEE (dispositivo citado na seção 3.2).

A fim de maximizar a taxa de leitura e transmissão, o XBee do módulo remoto foi configurado para trabalhar em modo API, visto que, em modo transparente, se

faz necessário entrar e sair do modo de comando do XBee, demandando uma quantidade de tempo adicional.

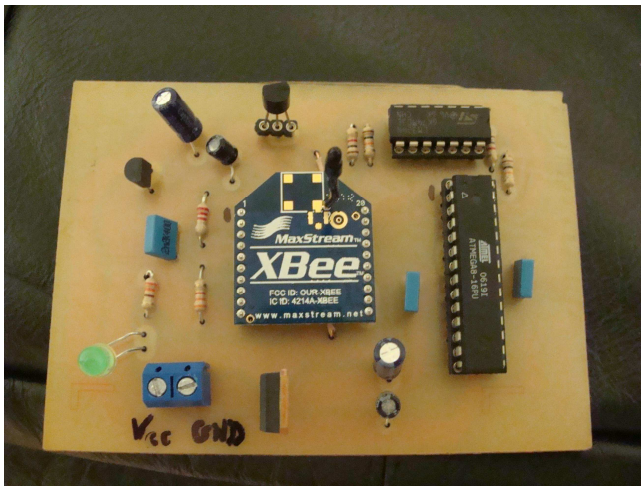


Figura 4.7 – Imagem do circuito do módulo remoto montado.

O programa contido no microcontrolador do módulo remoto deve ser capaz de processar as informações oriundas dos módulos sensores através do comando ND (descrito no sub-item 3.1.2.2), e transmitir apenas as informações úteis para o módulo base, a fim de que este possa implementar os algoritmos de localização. Estas informações são o endereço de cada módulo e seu respectivo valor de intensidade de sinal. Além disso, o programa deve fazer a leitura e o processamento da informação de temperatura oriunda do sensor LM35. Este fluxo de informações é representado pela Figura 4.8.

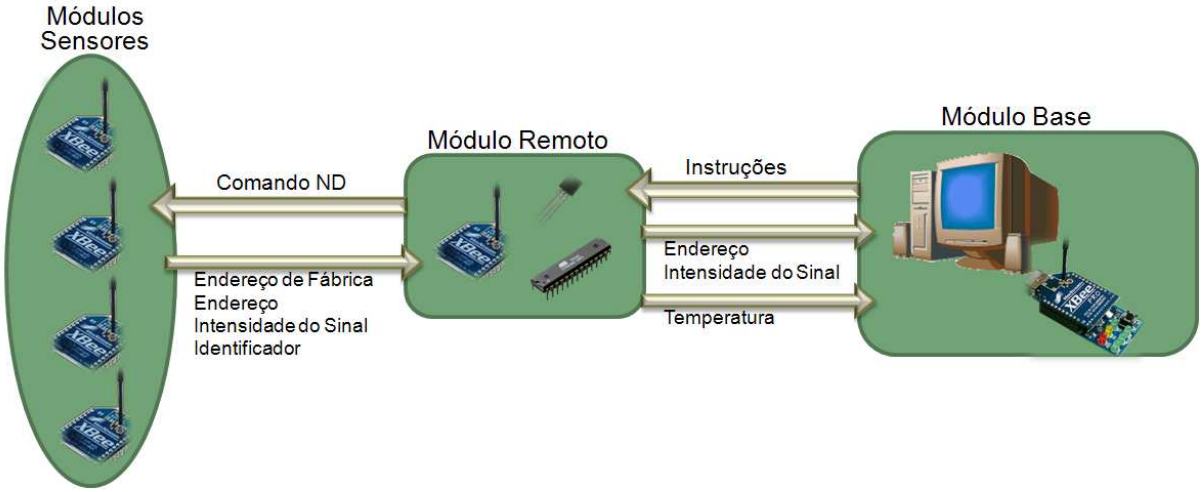


Figura 4.8 – Fluxo de dados entre o modulo remoto,
os módulos sensores e o modulo base.

Todo o programa do AVR foi desenvolvido em C com o auxilio da ferramenta *Programmer's Notepad*, descrita no item 3.3.2. Entretanto, algumas bibliotecas do

compilador GNU-GCC fornecidas pela ferramenta foram substituídas por bibliotecas desenvolvidas para o projeto, devido à instabilidade e ao tamanho destas bibliotecas GCC. Adicionou-se, além destas, uma biblioteca apenas para tratar, ler e transmitir dados no formato API do módulo XBee. Estas bibliotecas se encontram no Anexo V deste documento.

A idéia básica do programa é aguardar a recepção de alguma instrução válida para o processamento. Os comandos válidos, esperados pelo programa do microcontrolador, são: o comando para medir a intensidade do nível de sinal dos módulos sensores, o comando para obter as características gerais do módulo remoto, e o comando para fazer a leitura da temperatura ambiente.

Após o processamento de algum dos comandos, os dados obtidos são, então, transmitidos ao módulo base através de um simples protocolo de transmissão desenvolvido a fim de garantir a confiabilidade dos dados. Caso receba alguma instrução desconhecida ou termine o processamento de alguma instrução válida, o programa é, então, reiniciado. Após a reinicialização, o programa envia um pacote de dados ao módulo base indicando que foi reinicializado e que está pronto para receber nova instrução.

Todos os dados recebidos pela entrada serial do microcontrolador são verificados e caso haja algum dado corrompido, ou se detecte algum *byte* inesperado em um pacote API, o programa é reiniciado e envia-se ao módulo base um pacote de dados informando a reinicialização e o erro que a causou. A descrição detalhada do algoritmo do microcontrolador e do protocolo de comunicação se encontra no Anexo I deste documento.

A leitura da temperatura ambiente é feita através do processamento dos dados advindos da saída do sensor LM35 e da saída do controlador proporcional, como apresentado na Figura 4.5. Para tanto, uma função de medição dinâmica do ganho K_p do amplificador foi adicionada ao programa do microcontrolador. A grande vantagem desta forma de detecção de ganho, em tempo de execução, é a eliminação da medição manual do ganho de tensão do controlador proporcional (esta medição é necessária, pois, mesmo projetado para trabalhar com um ganho K_p igual a 10, perturbações e imprecisões nos valores das resistências levam a um valor distinto de ganho). A idéia é que, a cada leitura feita pelo conversor A/D, calcule-se o ganho do controlador proporcional, através da simples divisão entre o valor lido da entrada A/D, ligada à saída do amplificador, pelo valor lido da entrada A/D ligada diretamente ao LM35. Após uma série de leituras, o programa processa o

conjunto de ganhos e seleciona o ganho mais provável para ser utilizado no cálculo da temperatura. O algoritmo utilizado para este processamento, com seus diagramas de blocos e todos os seus passos lógicos, se encontra no Anexo I deste documento.

4.2 Algoritmos de Localização

4.2.1 Triangulação Hiperbólica

A primeira tentativa de obter um algoritmo de localização foi baseada no método descrito no sub-item 2.3.2.3, que consiste em estimar posição do módulo móvel através da medida de potência do sinal recebido por módulos espalhados ao longo do ambiente. Também foi demonstrado, no item 2.2.3, que a Equação 11 pode ser utilizada como um Modelo de Propagação de Rádio, que é necessário para o cálculo das distâncias no método da triangulação hiperbólica. Assim sendo, fez-se necessário o cálculo empírico da constante *K* presente na Equação 11.

Entretanto, a informação de nível de sinal é fornecida pelos módulos XBee em unidades de dBm (milidecibéis), ou seja, é fornecida uma medida relativa da potência recebida, tendo como referência o valor de potência de 1 mW. Assim, aplicando o logaritmo na base 10 e multiplicando por 10 o lado direito da Equação 11, obtemos a Equação 12, que corresponde a um Modelo de Propagação de Rádio em termos de milidecibéis.

$$dBm = 10\log_{10}\left(\frac{K}{D^2}\right) \tag{12}$$

Por fim, isolando o valor de D para que tenhamos a distância em função do nível de sinal fornecido pelo módulo XBee, chegamos à Equação 13, que corresponde ao Modelo de Propagação de Rádio utilizado no algoritmo de localização.

$$D = \sqrt{\frac{K}{10^{\left(\frac{dBm}{10}\right)}}} \tag{13}$$

4.2.1.1 Obtenção do Valor da Constante K

Para o cálculo do valor de *K*, o aparato experimental descrito na seção 4.1 foi utilizado para medir o nível de sinal entre dois módulos, em distâncias preestabelecidas com o uso de uma trena. Primeiramente, os módulos foram deixados a uma distância de 30 centímetros, e, depois de algum tempo em repouso, mediu-se o nível de sinal através do comando ND (citado no item 3.1.2.2). Em seguida, os módulos foram afastados em passos de 30 em 30 centímetros, e, em cada posição, media-se o nível de sinal correspondente. Os dados assim obtidos estão ilustrados na Figura 4.9.

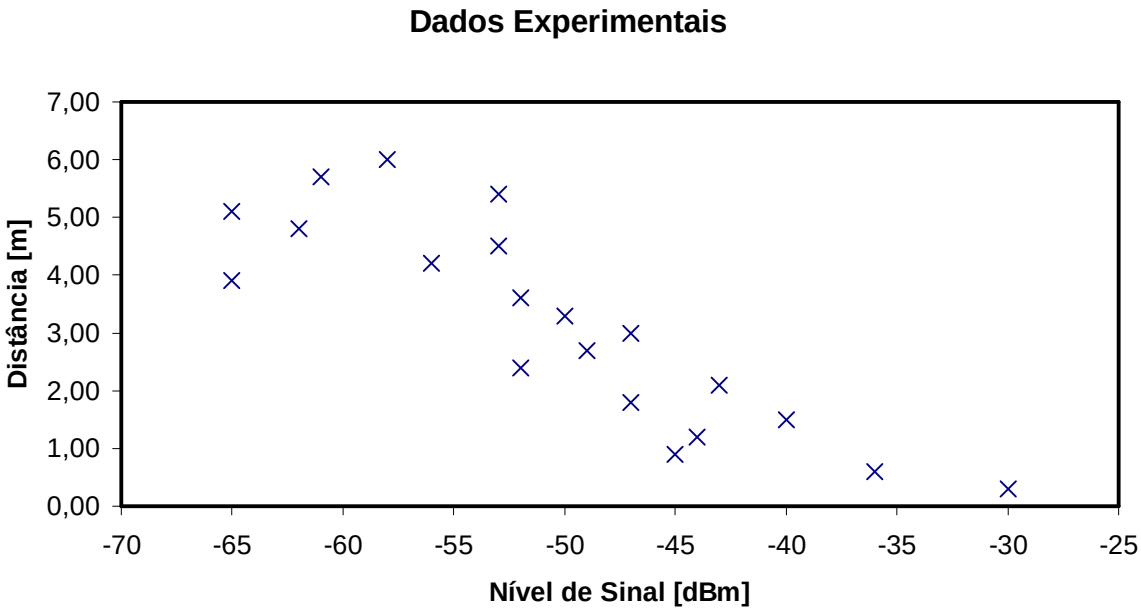


Figura 4.9 – Dados experimentais para o cálculo da constante *K.*

Em seguida, o Método dos Mínimos Quadrados (Edwards e Penney, 2000) foi utilizado para calcular um valor para a constante *K* que minimizasse os erros entre os pontos experimentais e a Equação 13. Assim, foi obtido o valor de 0,000058, que faz com que a Equação 13 seja representada pela curva vermelha da Figura 4.10.

Verifica-se pela Figura 4.10 que a Equação 13 fornece distâncias maiores que 7 metros para valores de nível de sinal menores que -60 dBm. Entretanto, estes dados são incompatíveis com as dimensões do ambiente de medição, uma vez que este consiste em um pequeno recinto de 3 x 7 metros.

Assim, para termos uma noção do erro médio associado ao utilizar a Equação 13 como Modelo de Propagação de Rádio, as diferenças entre os valores de distância medidos e valores de distâncias obtidos pela Equação 13 foram

calculadas, excluindo os quatro pontos experimentas que possuem nível de sinal menor que -60 dBm. Os resultados dos cálculos de erro podem ser vistos na Figura 4.11. Das informações sobre os erros associados para cada distância, foi obtido um erro médio de 0,64 metros.

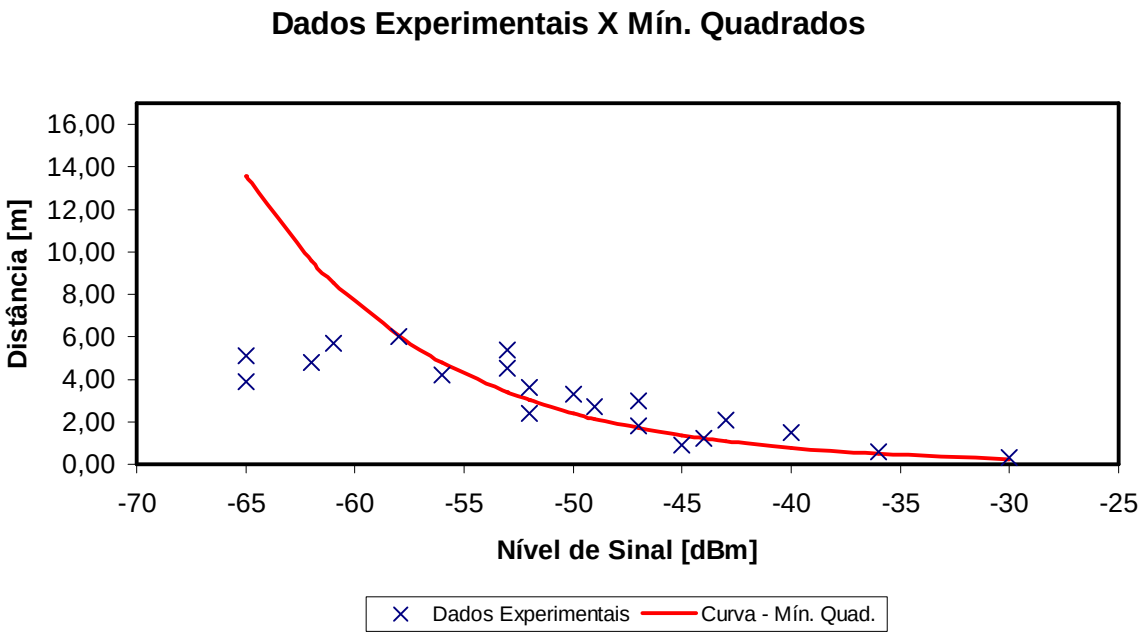


Figura 4.10 – Comparação entre os dados experimentais e a curva obtida para a Equação 13, com *K*=0,000058.

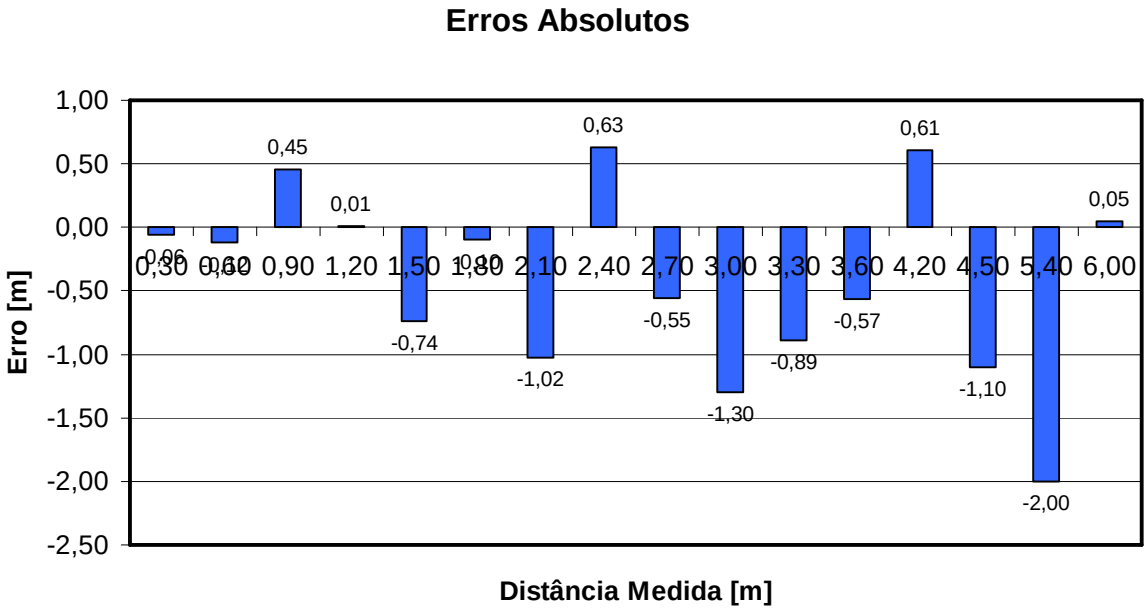


Figura 4.11 – Valores de erro absoluto entre distâncias medidas e distâncias calculadas.

4.2.1.2 Implementação do Algoritmo de Triangulação Hiperbólica

Uma vez que o módulo remoto obtém os valores de nível de sinal em relação aos módulos sensores e os envia para o módulo base, fica a cargo do computador processar os dados e estimar a posição do módulo. Para isso, um programa em linguagem C foi desenvolvido, com o intuito de interagir com o módulo remoto e obter os dados do sistema. Algumas das possíveis funções que o programa pode executar são: obter informações dos módulos, medir o nível de sinal dos módulos sensores em relação ao módulo remoto e executar os algoritmos de localização por triangulação hiperbólica e por redes neurais. A função de localização que utiliza redes neurais é detalhada no item 4.3.2, enquanto que o diagrama de blocos do programa como um todo e todas as suas funções lógicas se encontram no Anexo II deste documento. O Anexo III contém a descrição detalhada do algoritmo de localização por triangulação hiperbólica, explicado de forma simplificada a seguir.

A função de localização por triangulação hiperbólica recebe como entrada os valores da intensidade de sinal enviados pelo módulo remoto, e gera, na saída, informações como a área de provável localização e a estimativa da posição. Uma vez que o algoritmo é baseado em geometria analítica, o ambiente de medição é discretizado, pelo programa, em uma matriz, mapeando o recinto em pequenos quadrados de 10 x 10 cm.

O primeiro passo é estimar a distância radial do módulo remoto em relação aos outros módulos. Neste caso, o raio é fornecido pela Equação 13, que estima a distância do módulo em função do sinal. Como a informação obtida de um módulo sensor não fornece a direção, um raio de proximidade como o da Figura 4.12 é calculado para cada módulo sensor. Através da sobreposição destas circunferências, encontra-se uma área de provável localização do módulo remoto. Esta idéia é apresentada pela Figura 4.13, onde a área em laranja representa a sobreposição das circunferências e corresponde, portanto, a área de provável localização do módulo.

Após obter a área de provável localização, o programa percorre os pontos contidos nesta área e calcula, para cada um deles, um valor de erro que consiste na diferença entre o raio calculado pela Equação 13 e a distância do ponto em questão, com relação à coordenada de cada um dos módulos. A distância é calculada pela Equação 14, que descreve a distância entre dois pontos em um sistema cartesiano, sendo X_{mi} e Y_{mi} as coodenadas de cada um dos módulos sensores, e X_p e Y_p as coordenadas do ponto em questão. Assim, o ponto estimado para a localização do

módulo remoto consiste naquele que apresentar a menor soma de erros calculados a partir da Equação 15. O programa fornece também o valor da área de provável localização e o centro desta área.

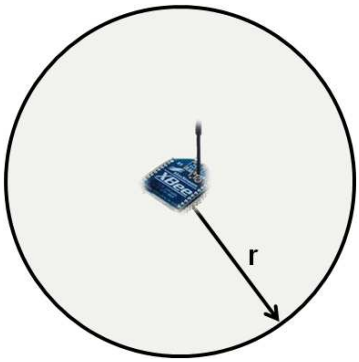


Figura 4.12 – Circunferência estimada pela distância entre o módulo remoto e um módulo sensor, dado seu nível de sinal.

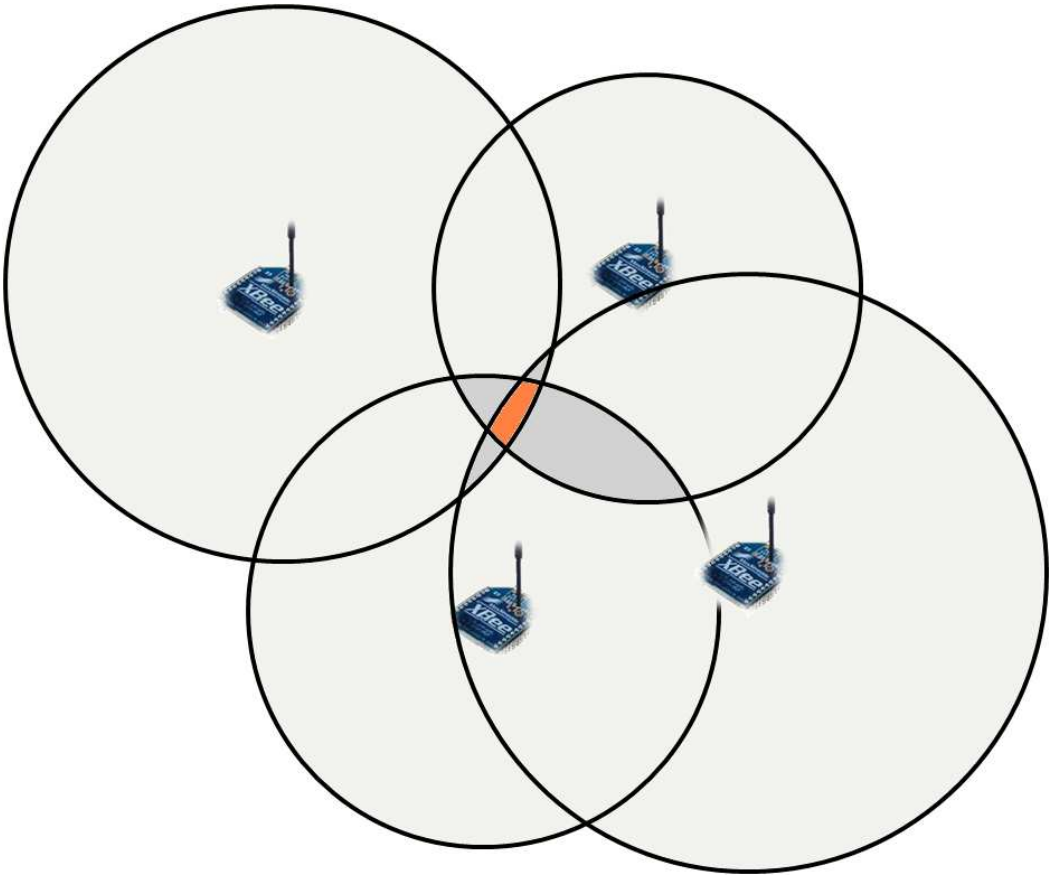


Figura 4.13 – Sobreposição das áreas geradas pelas circunferências de cada módulo.

$$d_p = \sqrt{\left(X_{mi} - X_p\right)^2 + \left(Y_{mi} - Y_p\right)^2} \qquad (14)$$

$$E_p = \sum_{i=1}^{4} R_i - dp_i \qquad (15)$$

Para se ter uma idéia melhor dos raios de cada módulo e da área de localização, o programa fornece seus resultados através de um gráfico que aparece na tela em tempo real. Como o gráfico é baseado no ambiente de localização descrito na Figura 4.1, pode-se ver graficamente a influência de cada dado obtido no comportamento do algoritmo de uma maneira simples. Adicionalmente, o programa também fornece os dados numericamente. As Figuras 4.14 e 4.15 ilustram saídas típicas do programa. Para efeitos de demonstração, nestes casos foram inseridos dados de nível de sinal fictícios.

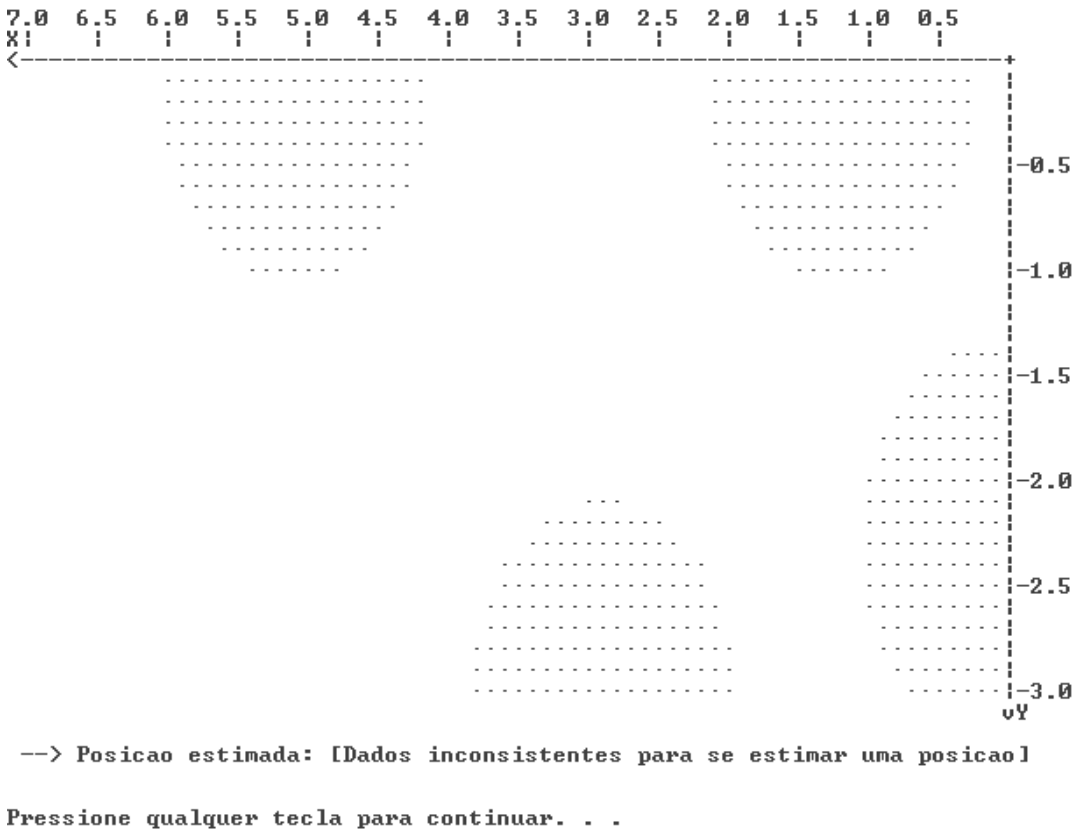


Figura 4.14 – Saída do programa de localização, onde não há área de intersecção entre os raios dos módulos.

Na Figura 4.14, os dados inseridos foram suficientes apenas para que fosse possível visualizar os raios calculados para cada módulo. Nota-se que, se não há intersecção, não há como estimar uma posição. Porém, a Figura 4.15 ilustra quando há intersecção: a área comum entre duas ou mais circunferências é destacada graficamente, e a posição é estimada como sendo o retângulo escuro do gráfico. O

centro da área é informado, graficamente, como o sinal ‘+’. As informações mais importantes obtidas do gráfico são resumidas logo abaixo do mesmo.

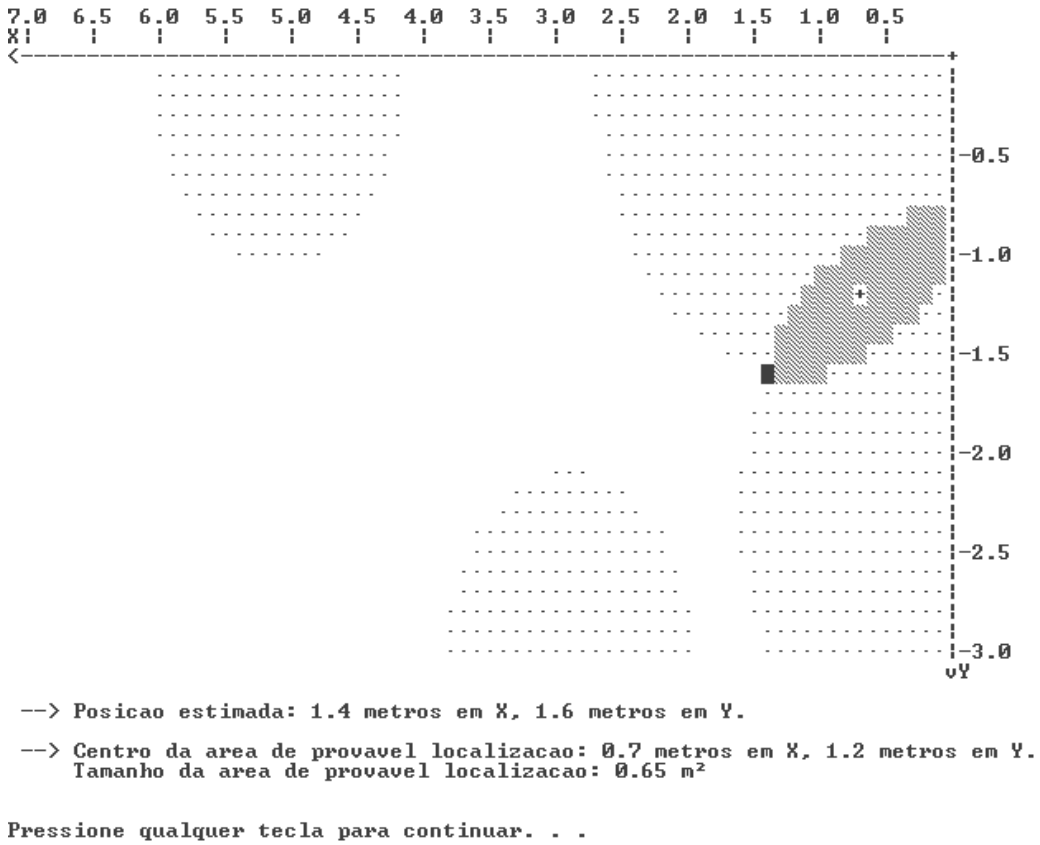


Figura 4.15 – Saída do programa de localização, onde há uma área de intersecção entre os raios de dois módulos.

O algoritmo descrito faz uso da Equação 13 para estimar a distância radial em torno do módulo, sendo empregado o valor de *K* obtido pelo processo descrito no sub-item 4.2.1.1 (ou seja, o valor 0,000058). Entretanto, um segundo valor para a constante *K* é calculada pelo programa em tempo de execução, para considerar os efeitos do sistema sobre o experimento. Este valor alternativo é atribuído a uma variável K_2. Assim, o algoritmo de localização foi executado duas vezes: uma vez utilizando o valor padrão de *K* e outra utilizando K_2. O objetivo deste procedimento é comparar os resultados e verificar se há influência do sistema sobre as medidas, uma vez o valor de *K* foi obtido em um estágio anterior de trabalho, quando foram utilizadas posições bem definidas e leituras de intensidade de sinal de apenas um módulo sensor.

Para o cálculo de K_2, o programa requisita que o módulo remoto seja posto em três posições distintas, fornecidas pelo usuário. O aplicativo, então, calcula a

distância real do módulo remoto para os módulos sensores, dos quais se conhece a localização, e obtém a intensidade de sinal de cada módulo em cada uma das três posições do módulo remoto. Desta forma, são obtidas doze relações de distância por intensidade de sinal (considerando a utilização de quatro módulos sensores). Por meio destas relações a constante K_2 é calcula através do Método dos Mínimos Quadrados (Edwards e Penney, 2000), similarmente ao caso da variável *K* padrão.

4.2.2 Método de Localização Utilizando Redes Neurais

A solução para o problema de localização pode não ser encontrada através de algoritmos analíticos simples. Cobrindo esta possibilidade, desenvolveu-se, também, uma solução baseada em um método inteligente utilizando redes neurais. O método de aprendizagem adaptativo utilizado foi o *Feedforward- Backpropagation*, descrito na seção 2.4.

Para a implementação do sistema inteligente, utilizou-se a *Neural Network Toolbox* do software MATLAB. Esta *Toolbox* do MATLAB contém todas as ferramentas necessárias para se projetar, implementar, visualizar e simular uma rede neural dentro do ambiente MATLAB (DEMUTH *et al.*, 2008).

A rede neural foi configurada da seguinte forma: as entradas da rede são os valores de intensidade de sinal dos módulos sensores, enquanto que as saídas são a posição X e a posição Y do módulo remoto. Desta forma, a rede teria quatro entradas e duas saídas. Entretanto, para simplificar ainda mais a rede e obter melhores resultados, a rede foi dividida em duas outras redes, de quatro entradas e uma saída. A primeira foi utilizada para o processamento da posição X do módulo, enquanto a segunda foi utilizada para o processamento da posição Y.

4.2.2.1 Obtenção do Conjunto de Dados para Treinamento da Rede

O treinamento da rede neural requer uma série de exemplos com entradas e saídas para que a rede possa interpretar adequadamente o comportamento do sistema. Deve-se adquirir, portanto, um conjunto de dados relacionando as entradas, intensidade do sinal dos módulos sensores, com as saídas X e Y relativas à primeira e à segunda redes neurais, respectivamente.

Este conjunto de dados é obtido através do posicionamento do módulo remoto em uma posição com valores conhecidos de X e Y. Nesta posição deve-se, então, medir a intensidade do nível de sinal de cada módulo sensor. A intensidade

do sinal é medida vinte vezes. Calcula-se, então, o valor médio destas vinte leituras, para cada módulo sensor. O resultado desta operação é um conjunto de dados contendo quatro entradas, relativas às intensidades de sinal dos módulos sensores, e duas saídas, relativas às posições X e Y do módulo remoto. A utilização destas vinte leituras tem como objetivo a redução da influência de leituras espúrias no conjunto de dados de treinamento.

Esta operação é repetida para outras cento e dezenove posições dentro do ambiente de implementação, descrito na seção 4.1. O resultado final das leituras é, portanto, um conjunto de cento e vinte elementos, cada um contendo quatro entradas e duas saídas.

4.2.2.2 Treinamento das Redes

Inicialmente, as redes devem ser criadas e seus pesos e bias inicializados. No MATLAB, isto é feito através do comando **newff**, descrito em Demuth *et al.* (2008). As redes, então, são treinadas utilizando o comando **train** (DEMUTH *et al.*, 2008) com, apenas, noventa e seis elementos do conjunto de cento e vinte elementos, previamente obtidos. Os outros vinte e quatro elementos serão utilizados na verificação da eficiência da rede e são chamados de conjunto de teste.

As duas redes, depois de treinadas, são utilizadas na simulação dos elementos do conjunto de teste. As saídas X e Y desta simulação são comparadas com os valores-alvo (valores reais) de X e Y dos mesmos vinte e quatro elementos do conjunto de teste. O somatório da diferença entre as saídas simuladas e os valores alvos nos fornece um erro relacionado às redes. Este erro pode, então, ser utilizado como uma espécie de teste de eficiência das redes treinadas.

4.2.2.3 Seleção das Redes Neurais

Para a seleção das duas redes neurais que serão implementadas no algoritmo de localização, desenvolveu-se um *loop* de duzentas interações, no qual a cada interação duas novas redes são criadas, com valores iniciais aleatórios. A cada vinte e cinco interações, as configurações das redes, como o número de neurônios, o número de camadas e as funções de transferências, são alteradas.

As redes criadas são, então, treinadas e simuladas utilizando-se o processo descrito no sub-item anterior. Os erros associado às redes e ao conjunto de treinamento são, então, comparados e, ao final do processo, selecionam-se as redes com menores valores de erro para serem implementadas no algoritmo.

4.2.2.4 Implementação do Algoritmo de Localização

Após a seleção das duas redes neurais que serão utilizadas no sistema de localização (uma para o eixo **X** e outra para o eixo **Y**), desenvolveu-se o algoritmo que processa, dentro do ambiente MATLAB, os valores de intensidade de sinal para a obtenção da localização do módulo remoto.

O programa de localização requisita a leitura da intensidade do nível de sinal dos módulos sensores com relação ao módulo remoto e, então, grava, dentro do diretório de trabalho do MATLAB, estas informações em conjunto com a localização real do módulo remoto. Este processo é feito através do comando “Localizar por Redes Neurais” do software de localização. Este comando é descrito com maiores detalhes no Anexo IV deste documento.

Dentro do ambiente de trabalho do MATLAB obtém-se os valores da intensidade de sinal dos módulos sensores. Em seguida, a saída do sistema é simulada utilizando as redes neurais previamente criadas. Após a simulação, um gráfico (ilustrado na Figura 4.16) relacionando a posição do módulo remoto simulada pelas redes neurais com a posição real do módulo é gerado.

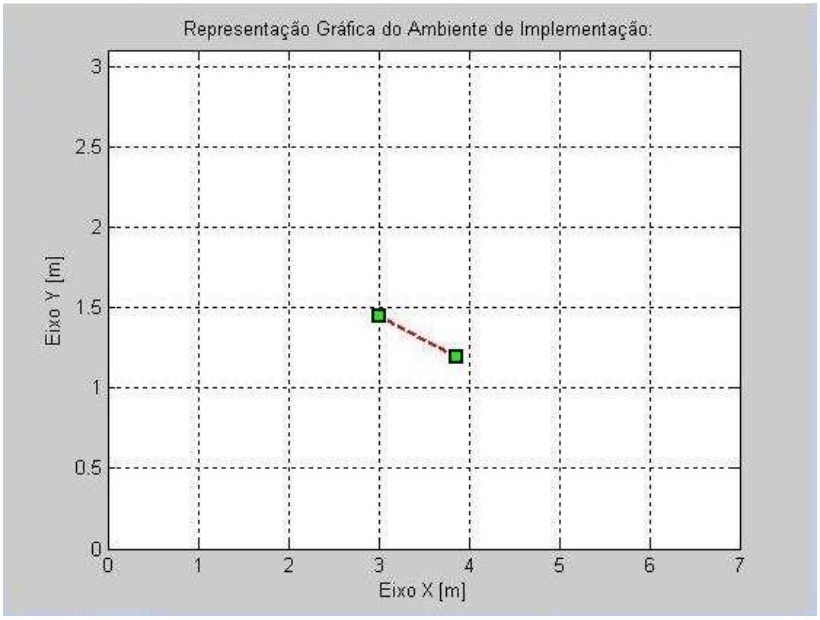


Figura 4.16 – Gráfico que relaciona a posição do módulo remoto gerada pelas redes neurais com a posição real.

5 Resultados Obtidos

5.1 Localização por Triangulação Hiperbólica

A avaliação do método de localização foi feita através da comparação da posição de referência do módulo (medido com uma trena) com os dados fornecidos pelo algoritmo descrito no sub-item 4.2.1.2. Para tanto, posicionou-se o módulo remoto em vinte e quatro posições distintas na sala. As posições dentro do ambiente de localização são apresentadas pela Figura 5.1 e pela Tabela 5.1. Os dados de nível de sinal obtidos de cada uma das posições foram inseridos na função de localização por triangulação hiperbólica, gerando os resultados apresentados na Tabela 5.2.

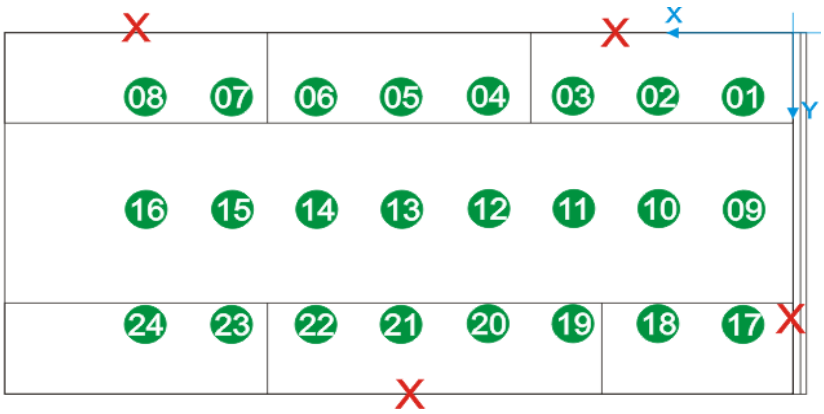


Figura 5.1 – Posições de referência no recinto de localização.

Tabela 5.1 – Valores (em metros) das coordenadas das posições de medição.

Medição	Posição (m)		Medição	Posição (m)	
	Eixo X	Eixo Y		Eixo X	Eixo Y
01	0,6	0,7	**13**	3,0	1,45
02	1,2	0,7	**14**	3,6	1,45
03	1,8	0,7	**15**	4,2	1,45
04	2,4	0,7	**16**	4,8	1,45
05	3,0	0,7	**17**	0,6	2,3
06	3,6	0,7	**18**	1,2	2,3
07	4,2	0,7	**19**	1,8	2,3
08	4,8	0,7	**20**	2,4	2,3
09	0,6	1,45	**21**	3,0	2,3
10	1,2	1,45	**22**	3,6	2,3
11	1,8	1,45	**23**	4,2	2,3
12	2,4	1,45	**24**	4,8	2,3

Tabela 5.2 – Resultados obtidos para a localização.

Posição de Ref. do Módulo (em m)		Posição Estimada do Módulo (em m)		Centro da Área de Localização (em m)		Encontra-se na área de localização ?	Tamanho da área de Localização (em m^2)
X	Y	X	Y	X	Y		
0,6	**0,70**	2,7	0,1	1,3	1,3	SIM	3.15
1,2	**0,70**	0,2	0,4	1,4	0,8	SIM	1.33
1,8	**0,70**	0,1	1,2	1,6	0,9	SIM	3.55
2,4	**0,70**	0,1	1,2	1,6	0,9	SIM	3.55
3,0	**0,70**	0,1	0,2	1,1	1,7	NÃO	4.55
3,6	**0,70**	3,8	2,8	3,2	2,3	NÃO	1.51
4,2	**0,70**	3,7	1,1	3,4	1,2	NÃO	0.19
4,8	**0,70**	2,6	1,3	1,2	1,1	NÃO	3,97
0,6	**1,45**	7,0	3,0	4,2	1,6	SIM	16,63
1,2	**1,45**	0,1	2,9	0,6	2,3	NÃO	1,34
1,8	**1,45**	2,7	1,9	3,2	2,1	NÃO	0,28
2,4	**1,45**	2,8	0,4	3,9	1,5	SIM	3,85
3,0	**1,45**	3,7	1,4	4,0	1,7	NÃO	0.11
3,6	**1,45**	4,3	2,1	3,0	2,1	SIM	5.09
4,2	**1,45**	4,8	3,0	3,5	1,5	SIM	7,61
4,8	**1,45**	5,4	1,4	1,9	1,8	SIM	5,31
0,6	**2,30**	7,0	3,0	3,6	1,6	SIM	20,65
1,2	**2,30**	6,1	3,0	3,1	1,6	SIM	17,55
1,8	**2,30**	7,0	3,0	3,6	1,6	SIM	20,65
2,4	**2,30**	7,0	3,0	3,6	1,6	SIM	20,65
3,0	**2,30**	6,7	3,0	3,2	1,7	SIM	18,21
3,6	**2,30**	7,0	3,0	3,6	1,6	SIM	20,65
4,2	**2,30**	7,0	3,0	3,5	1,6	SIM	19,77
4,8	**2,30**	7,0	3,0	3,6	1,6	SIM	20,65

Da Tabela 5.2, verifica-se que o método empregado não consegue localizar corretamente o módulo remoto; o ponto estimado pelo algoritmo é muito discrepante da posição real em praticamente todos os casos. Além disso, para certos pontos (os últimos oito pontos da Tabela 5.2, por exemplo), a queda do sinal é tão grande que faz com que todo o recinto seja mapeado como área provável, dificultando ainda mais a estimativa de uma posição. Contudo, verificou-se que na maioria dos pontos, o centro da área de provável localização se aproxima mais da posição real do módulo remoto do que o ponto estimado com base no menor erro dado pela Equação 15. Estas informações podem ser mais bem visualizadas nos gráficos da Figura 5.2, que expressam as distâncias, em relação à posição real, da posição estimada e do centro da área de provável localização calculadas pelo algoritmo. Através destes gráficos, verifica-se que o valor médio das discrepâncias entre a

posição calculada do centro da área em relação à posição de referência (representado pela linha tracejada vermelha) é menor do que o valor médio das discrepâncias entre a posição estimada em relação à posição de referência (representado pela linha tracejada azul).

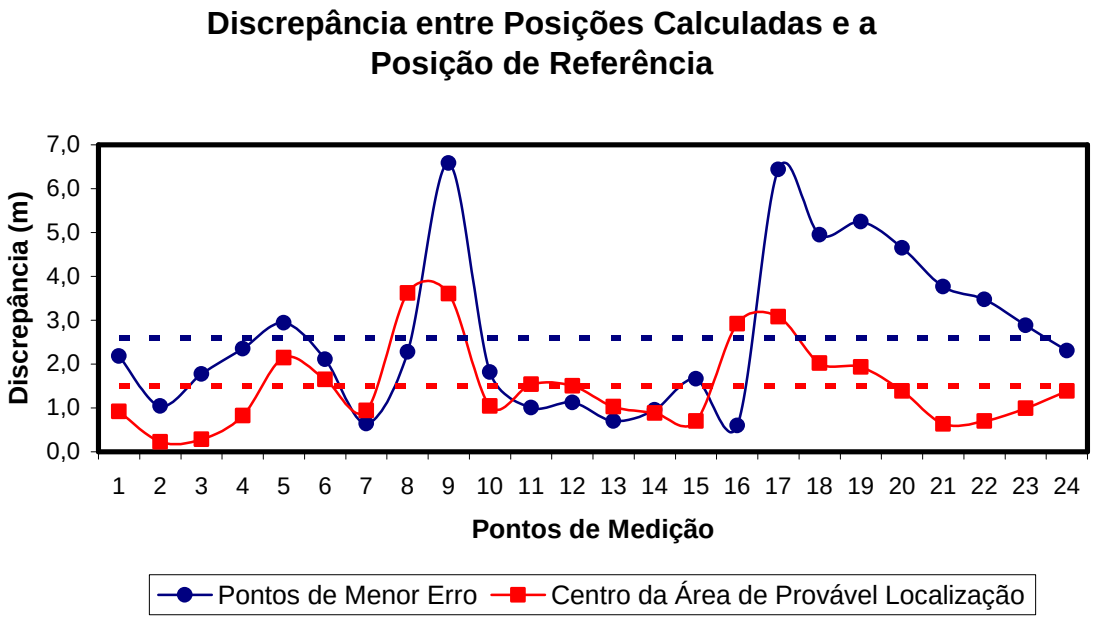


Figura 5.2 – Gráficos das discrepâncias dos pontos calculados em relação à posição de referência.

Utilizando-se os mesmos locais de medição definidos pela Tabela 5.1, o algoritmo foi repetido, porém substituindo o valor da constante *K* da Equação 13 pela variável K_2, de acordo com o procedimento descrito no sub-item 4.2.1.2. Os resultados obtidos estão mostrados na Tabela 5.3. Novamente, faz-se uso de gráficos similares aos da Figura 5.2, para expressar graficamente o desempenho do algoritmo utilizando-se a constante K_2. Estes novos gráficos estão ilustrados na Figura 5.3.

Os resultados obtidos desta vez mostram que, ao utilizar o valor alternativo K_2, os pontos estimados se aproximam um pouco mais das posições reais, mas, ainda sim, o procedimento ainda é impreciso para determinar de forma confiável a posição do módulo remoto no recinto. Além disso, as áreas de provável localização são menores, fazendo com que na maior parte dos casos a posição real esteja fora destas áreas.

Ademais, verifica-se através da análise dos gráficos das Figuras 5.2 e 5.3 que os extremos do ambiente de implementação, representados pelos pontos de medição 8, 9, 16 e 17 apresentam discrepâncias superiores em relação as outras

medidas, para ambos os casos estudados. A observância deste fato sugere que o sistema está sujeito as influencias do ambiente de forma sistemática, sendo necessário outro método de localização que leve em conta estas influencias.

Tabela 5.3 – Resultados obtidos para a localização (utilizando-se K_2).

Posição de Ref. do Módulo (em m)		**Posição Estimada do Módulo (em m)**		**Centro da Área de Localização (em m)**		**Encontra-se na área de localização ?**	**Tamanho da área de Localização (em m^2)**
X	**Y**	**X**	**Y**	**X**	**Y**		
0.6	**0.70**	0,1	1,0	0,7	1.3	NÃO	0,26
1.2	**0.70**	1,6	0,8	3,1	1,8	SIM	4,16
1.8	**0.70**	1,5	1,2	2,3	1,4	NÃO	0,04
2.4	**0.70**	1,5	1,2	2,0	1,2	NÃO	0,04
3.0	**0.70**	0,1	0,9	0,8	1,5	NÃO	1,04
3.6	**0.70**	2,7	2,2	2,2	2,3	NÃO	0,06
4.2	**0.70**	3,9	0,9	2,1	1,6	NÃO	7,96
4.8	**0.70**	1,6	1,3	1,0	1,1	NÃO	0,65
0.6	**1.45**	6,6	2,2	4,7	1,3	SIM	10,0
1.2	**1.45**	0,1	1,7	0,7	1,9	NÃO	0,15
1.8	**1.45**	1,1	2,1	1,0	1,6	NÃO	0,71
2.4	**1.45**	3,8	1,1	3,7	1,5	SIM	10,43
3.0	**1.45**	2,9	1,8	1,8	1,5	SIM	3,13
3.6	**1.45**	3,0	2,0	2,5	2,6	NÃO	1,17
4.2	**1.45**	2,9	0,6	3,1	1,2	NÃO	0,11
4.8	**1.45**	1,3	3,0	1,3	2,5	NÃO	0,70
0.6	**2.30**	5,2	2,4	2,6	1,6	SIM	14,80
1.2	**2.30**	4,1	3,0	2,7	1,5	NÃO	8,00
1.8	**2.30**	7,0	3,0	3,6	1,6	SIM	20,65
2.4	**2.30**	5,8	0,1	2,8	1,6	SIM	15,94
3.0	**2.30**	5,3	2,2	2,9	2,0	SIM	10,30
3.6	**2.30**	7,0	3,0	3,6	1,6	SIM	20,65
4.2	**2.30**	5,2	3,0	2,8	1,8	SIM	12,68
4.8	**2.30**	6,6	3,0	3,3	1,6	SIM	19,02

Uma breve pesquisa para explicar os resultados observados foi realizada em MAXSTREAM (2005), onde são feitas várias considerações sobre nível de sinal e alcance dos módulos, em função do tipo de antena utilizada. O documento salienta, principalmente, a quantidade de fatores que podem influenciar na medida de sinal dos módulos, entre eles, a angulação relativa entre os módulos, a inclinação dos módulos com relação ao solo, obstáculos na trajetória das ondas de rádio e até mesmo os circuitos aos quais os XBee's estão ligados. Estes inúmeros fatores aleatórios aos quais o experimento está sujeito explanam a impossibilidade do uso

exclusivo de métodos analíticos simples para a obtenção de uma solução satisfatória.

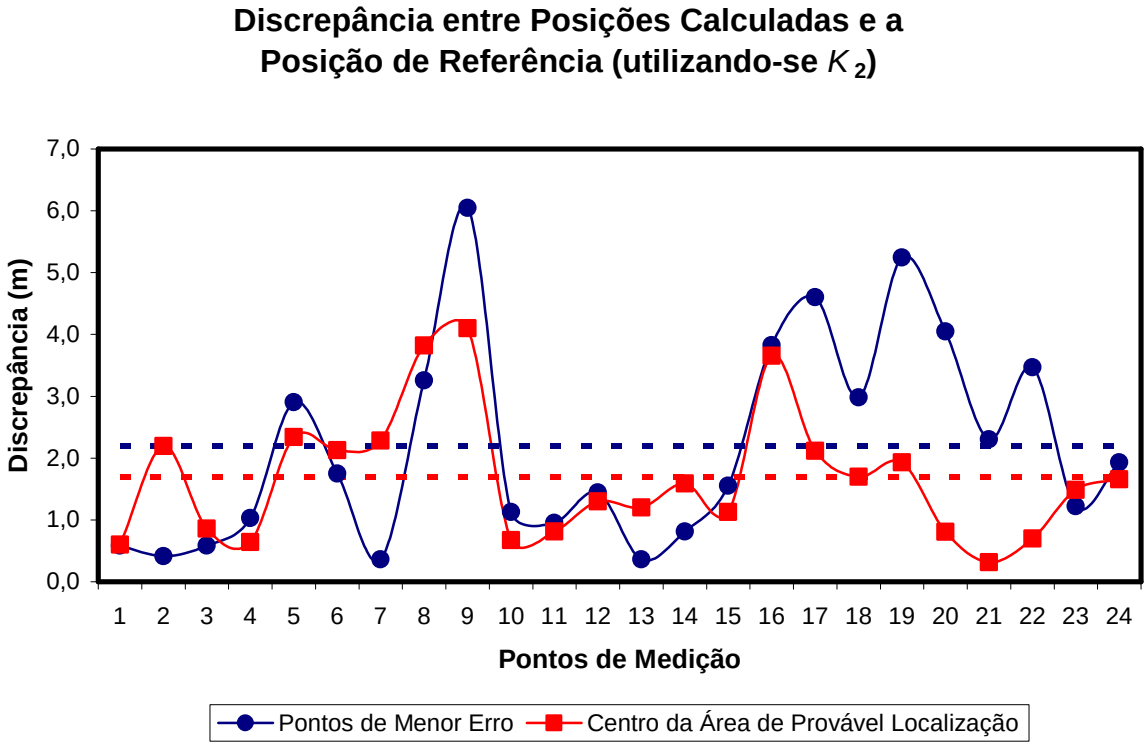


Figura 5.3 – Gráficos das discrepâncias dos pontos calculados em relação à posição de referência (utilizando-se K_2).

5.2 Localização por Redes Neurais

O sistema de localização por redes neurais foi, também, avaliado através da comparação da posição real do módulo remoto com a posição do módulo obtida através do algoritmo de localização. Para tanto, o módulo foi posicionado nas mesmas vinte e quatro posições descritas pela Tabela 5.1 e pela Figura 5.1. Para cada posição simulou-se, através das redes neurais obtidas no item 4.2.2, a posição do módulo remoto. Os resultados dessa simulação são apresentados no gráfico da Figura 5.4, que expressa a distância da posição obtida através do algoritmo de localização com relação à posição real do módulo. Esta distância, tal como nos gráficos das Figuras 5.2 e 5.3, representa a discrepância entre o valor obtido e o valor esperado, sendo, portanto, uma medida de acurácia dos resultados obtidos.

Infere-se da análise do gráfico da Figura 5.4 que o sistema de localização através do uso de redes neurais é capaz de fornecer valores de posição do módulo

remoto com uma acurácia média de 1 m em relação ao valor real da posição do módulo. Verifica-se, portanto, que este sistema de localização é mais eficaz que o sistema de localização por triangulação hiperbólica. A análise comparativa dos gráficos presentes nas Figuras 5.2 e 5.3 corrobora com essa idéia. Não obstante, o incremento da discrepância dos resultados verificado nos extremos do ambiente de implementação, como observado na seção 5.1, não é verificado no sistema de localização por redes neurais.

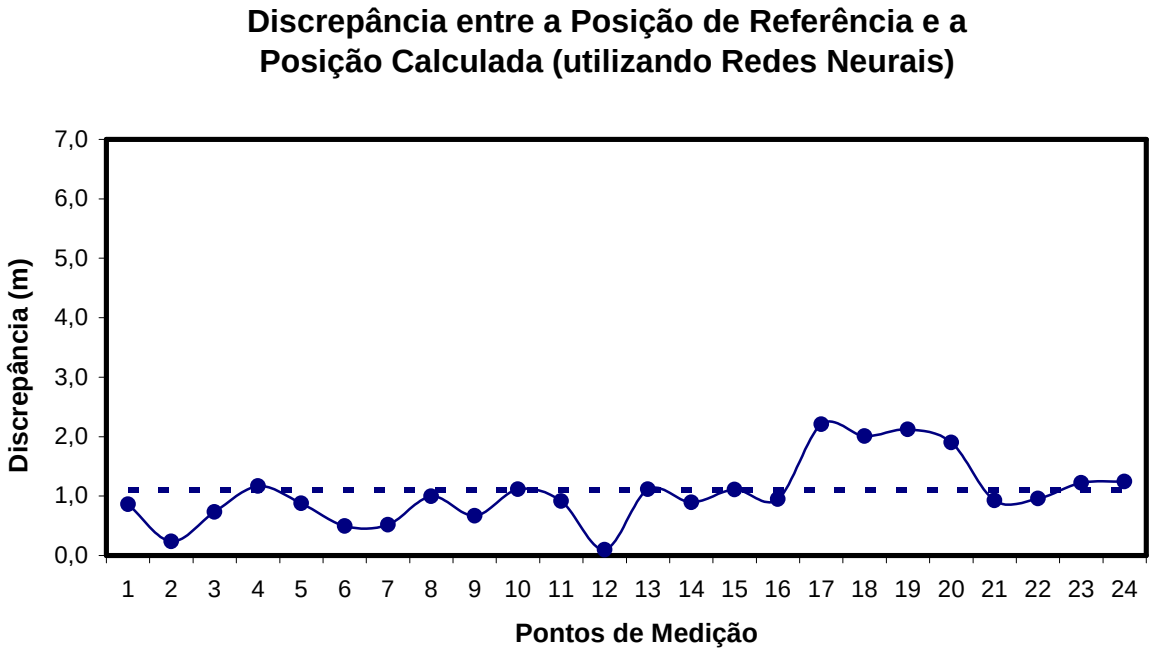


Figura 5.4 – Gráficos das discrepâncias dos pontos calculados em relação à posição de referência (utilizando-se redes neurais).

Entretanto, devemos ressaltar que os mesmos fatores que influenciam negativamente a precisão do sistema de localização por triangulação hiperbólica também estão presentes neste método de localização. De fato, a rede neural considera fatores interventores fixos do ambiente (como a posição e altura dos módulos sensores, paredes, colunas, divisórias e estantes), mas ainda sim, o experimento está sujeito a outros fatores aleatórios que influenciam na precisão das medidas de sinal, principalmente, a angulação relativa entre o módulo remoto e os módulos sensores e obstáculos móveis, como pessoas.

6 Conclusões e Perspectivas

O presente trabalho teve como objetivo a aplicação e a comparação dos principais métodos de localização estudados na literatura. O desenvolvimento de métodos de localização baseados na medição do parâmetro RSSI revelou-se uma tarefa bastante complexa. Pelo fato de diferentes variáveis presentes em ambientes fechados influenciarem na intensidade do sinal de rádio, a precisão das leituras é sensivelmente deteriorada. Por conta desta falta de precisão, a utilização, somente, do método analítico de propagação de rádio não apresentou resultados satisfatórios. O método de localização utilizando redes neurais, por considerar as interferências sistemáticas do ambiente, como paredes, estantes e mesas, apresentou melhores resultados. No entanto, este método exige o conhecimento prévio do ambiente de implementação. Além disso, o sistema ainda é suscetível as imprecisões de leitura geradas pelas diversas fontes de interferência ao sinal de rádio.

Futuramente, a substituição do módulo por outros dispositivos com transmissores de rádio de melhor qualidade e maior potência poderia incrementar a precisão dos resultados, uma vez que as interferências teriam menor influencia sobre o sinal. Uma opção seria a utilização do outro módulo XBee desenvolvido pela Digi, o XBee-PRO, que possui uma potência de transmissão cerca de 60 vezes maior que a do XBee utilizado. Uma segunda solução seria a combinação das técnicas de localização estudadas fornecendo resultados mais acurados ao sistema de localização. Entretanto, esta segunda solução ainda dependeria do conhecimento prévio do ambiente de implementação. Por fim, utilização de métodos de filtragem estocástica poderia tornar o sistema mais robusto, reduzindo a influência das interferências e imprecisões inevitáveis e de caráter aleatório.

Foi verificado, também, que melhorias podem ser feitas nos circuitos dos módulos sensores, para diminuir ainda mais o consumo de energia destes dispositivos. Com esse intuito, sugestões de modificações para estes circuitos que podem ser aplicadas no futuro são apresentadas no Anexo VI deste documento.

Referências

ATMEL, Corp. *ATmega8(L) Complete Datasheet*. 2004.

***Backpropagation*.** Wikipedia, Wikimedia Foundation, INC (E.UA.), 2008. Autor desconhecido. Disponível em: <http://en.wikipedia.org/wiki/Backpropagation>.

BAUCHSPIESS, Adolfo. **Introdução aos Sistemas Inteligentes**: Aplicações em Engenharia de Redes Neurais Artificiais, Lógica Fuzzy e Sistemas Neuro-Fuzzy. 2004. 71f. Departamento de Engenharia Elétrica, Universidade de Brasília (UNB), Brasília, Brasil.

BAUMANN, Chris. Tips *for selecting a Media Access Controller for ZigBee*. ***Industrial Control Designline: TechOnline Community*.** 28 ago. 2006. Disponível em: <http://www.industrialcontroldesignline.com/192300912;jsessionid=QL0APYY5BZ5A0QSNDLQCKIKCJUNN2JVN?printableArticle=true>.

BORGES, Geovany A.; BO, Antônio Padilha L.; MARTINS, Alexandre S.; COTTA, Leandro C.; FERNANDES, Maurílio; FREITAS, Gabriel; BECKMANN, Ener **Desenvolvimento com Microcontroladores Atmel AVR**. 2006. 34f. Departamento de Engenharia Elétrica, Universidade de Brasília (UNB), Brasília, Brasil.

CARVALHO, Paula M.; PASSARELA, Lucas; SANTOS, Daniel R. **ZigBee.** 2006. 8 f. Universidade de Brasília (UnB), Brasília.

DANKWA, Nana *A.* ***An Evaluation of Transmit Power Levels for Node Localization on the Mica2 Sensor Node***. 2004. 22f. *Electrical Engineering Department*, Yale University, New Haven, Estados Unidos.

DEMUTH, Howard; BEALE, Mark; HAGAN, Martin. ***Neural Network Toolbox*:** *User's Guide*. 6. ed. Massachusetts: The MathWorks, 2008.

DODD, Mark; WINDSOR, Benjamin. ***A ZigBee Based Wireless Smart Home***. 2006. 60f. Projeto de Graduação em Engenharia Computacional – *Faculty of Engineering, University of Manitoba*, Winnipeg, Canadá.

DVORAK, Joseph. ***IEEE 802.15.4 and Zigbee Overview*.** Motorola, 2005, 26f. Disponível em: <www.media.mit.edu/resenv/classes/MAS961/readings/802-15-4_Tutorial.ppt>.

EDWARDS JR., C. H; PENNEY, D. E. **Introdução à Álgebra Linear**. Rio de Janeiro: LTC Editora, 2000, 406f.

FIGUEREDO, Luis F. C. **Tutorial XBee**. 2008. 79f. Departamento de Engenharia Elétrica, Universidade de Brasília (UnB), Brasília, Brasil.

HECHT, Eugene. **Óptica**. Lisboa: Fundação Calouste Gulbenkian, 2002. 790f.

HEILE, Bob. ***ZigBee Alliance Tutorial*.** 2005. Disponível em: <http://www.cs.ucdavis.edu/~aksoy/course/w06/slides/ZigBeeTutorial_05.ppt>.

INDRIA, Yolla. ***Design of an Individual Mobile Measurement of Thermal Comfort***. 2006. 51f. Tese de Mestrado – *Fachbereich Elektrotechnik und Informationstechnik*, *Technische Universität Kaiserslautern*, Kaiserslautern, Alemanha.

MAO, Guogiang; FIDAN, Baris; ANDERSON, Brian D.O. ***Wireless Sensor Network Localization Techniques***. 2007. *School of Electrical and Information Engineering*, *University of Sydney*, Sydney, Austrália.

MAXSTREAM, Inc. *Product Manual* - ***XBee/XBee-PRO OEM RF Modules, RF Module Operation, RF Module Configuration and Appendices***. 2006.

MAXSTREAM, Inc. ***XBee & XBee PRO OEM RF Module Antenna Considerations***. 2005.

MONSIGNORE, Ferdinando. **Sensoriamento de Ambiente utilizando o padrão Zigbee**. 2007. 74f. Tese de Mestrado – Departamento de Engenharia Elétrica, Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, Brasil.

NEVES, Marcelo Veiga. **Uma (Breve) Introdução às Redes Neurais Artificiais.** 2006. 8 f. Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre.

NUNES, Bruno Astuto Arouche. **Um Sistema de Localização para Redes *Wi-Fi* Baseado em Níveis de Sinal e Modelo Referenciado de Propagação**. 2006. 81 f. Tese de Mestrado – Departamento de Engenharia de Sistemas e Computação, Coordenação de Projetos, Pesquisas e Estudos Tecnológicos (COPPE), Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brasil.

PINHEIRO, José Maurício S. As Redes com ZigBee. **Projeto de Redes**. 27 jul. 2004. Disponível em: <http://www.projetoderedes.com.br/artigos/artigo_zigbee.php>.

RAGHUVANSHI, Manish. ***Implementation of Wireless Sensor Mote***. 2006. 87 f. Tese de Mestrado - *Department of Nuclear Engineering and Technology, Indian Institute of Technology*, Kanpur, Índia.

ROJAS, Raul. *The backpropagation algorithm*. In: ROJAS, Raul. ***Neural Networks*****:** *A Systematic Introduction*. Berlin: Springer-verlag, 1996. Cap. 7, p. 151-184.

SAVVIDES, Andreas; HAN, Chin-Chieh; STRIVASTAVA, Mani B. Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors. 2002. *Computer Architecture Lab, University of California*, Los Angeles, Estados Unidos.

TERWILLIGER, Mark; GUPTA, Ajay; Bhuse, Vijay; Kamal, Zille Huma; Salahuddin, Mohammad Ali. ***A Localization System using Wireless Network Sensors***: *A comparision of Two Techniques*. In: *WORKSHOP ON POSITIONING, NAVIGATION AND COMMUNICATION*, 1., 2004, Hannover. Proceedings… Disponível em: <http://www.wpnc.net/fileadmin/WPNC04/Proceedings/A_Localization_System_using_Wireless_Network_Sensors_A_Comparison_of_Two_Techniques_passend.pdf>.

Anexos

Anexo I – Programa Embarcado do Microcontrolador

Anexo II – Programa de Localização

Anexo III – Descrição do Algoritmo de Localização por Triangulação Hiperbólica

Anexo IV – Descrição do Algoritmo de Localização por Redes Neurais

Anexo V – Códigos-Fonte das bibliotecas auxiliares do programa embarcado

Anexo VI – Sugestões de Alteração dos Módulos Sensores

Anexo I

Este anexo descreve detalhadamente o funcionamento do programa embarcado no microcontrolador Atmel ATmega8. O diagrama de blocos do programa, desenvolvido em C, com todos os seus passo lógicos é representado pelo diagrama de blocos da Figura I.1.

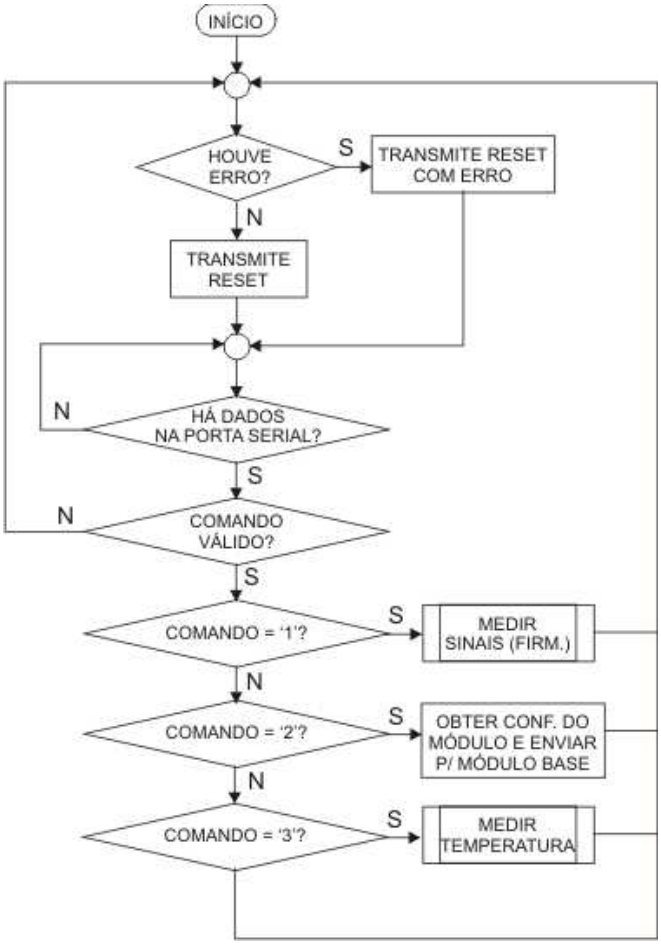


Figura I.1 – Diagrama de blocos do programa embarcado.

É possível observar pela Figura I.1 que o primeiro processo realizado pelo programa é a verificação de erros, onde checa-se a existência de algum imprevisto no último processamento do enlace que forçou sua reinicialização. Este tipo de erro ocorre quando há o recebimento de algum dado corrompido, ou seja, quando se detecta algum *byte* inesperado em um frame API. Caso este erro seja detectado, o programa envia ao módulo base um pacote de seis *bytes* de dados, indicando que houve reinício devido a alguma falha. Este pacote de dados é apresentado na Figura I.2, onde **XXX** indica o ponto no código que gerou esta exceção e os números

representam o valor, em decimal, dos *bytes*. Todavia, caso não seja detectado nenhum erro, o programa envia ao módulo base uma seqüência de dados informando que está pronto para receber novos comandos. Este pacote de dados é apresentado na Figura I.3, onde os números apresentados representam o valor, em decimal, dos *bytes*.

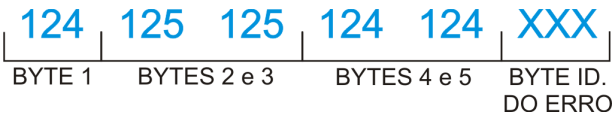


Figura I.2 – Pacote de *bytes* que indicam um erro.

Figura I.3 – Pacote de *bytes* que indicam que o módulo remoto está pronto para receber comandos.

O programa, então, aguarda pela recepção de alguma instrução válida e caso receba algum comando desconhecido, reinicia sem gerar erros. As instruções esperadas são os comandos 49, 50 e 51.

O comando 50 (caractere ‘2’ em ASCII), envia uma série de requisitos ao módulo XBee a fim de obter as suas principais configurações e, então, as transmite uma a uma para o módulo base.

O comando 49 (caractere ‘1’ em ASCII) chama a função para medir a intensidade do nível de sinal dos módulos sensores, representada pelo diagrama de blocos da Figura I.4.

Analisando o diagrama de blocos, verifica-se que, inicialmente, é gerado um vetor de dados que armazena o byte 47 como valor inicial. Em seguida, o programa requisita, ao módulo XBee, o rastreamento dos módulos sensores através do comando ND. Sabendo que cada módulo envia um pacote API ao módulo remoto, este analisa todos os pacotes recebidos até receber um pacote com tamanho igual a um, o que representa o fim da transmissão. Para cada pacote API recebido, o programa analisa apenas o trecho que contém o pacote de dados. Dentro do pacote de dados, o programa obtém apenas o valor do endereço de 16 bits do módulo sensor e seu respectivo valor de intensidade do sinal. Estes valores são, então, armazenados no vetor utilizando o *byte* 46 como separador entre o endereço do sensor, seu valor de intensidade de sinal e os dados do próximo sensor. Ao final do

processo, substitui-se o último separador, representado pelo *byte* 46, pelo indicador de fim de pacote (*byte* 47). O pacote de dados que será transmitido ao módulo base é, então, definido de acordo com a Figura I.5, onde MY representa o endereço do módulo sensor e DB representa a intensidade do sinal lido.

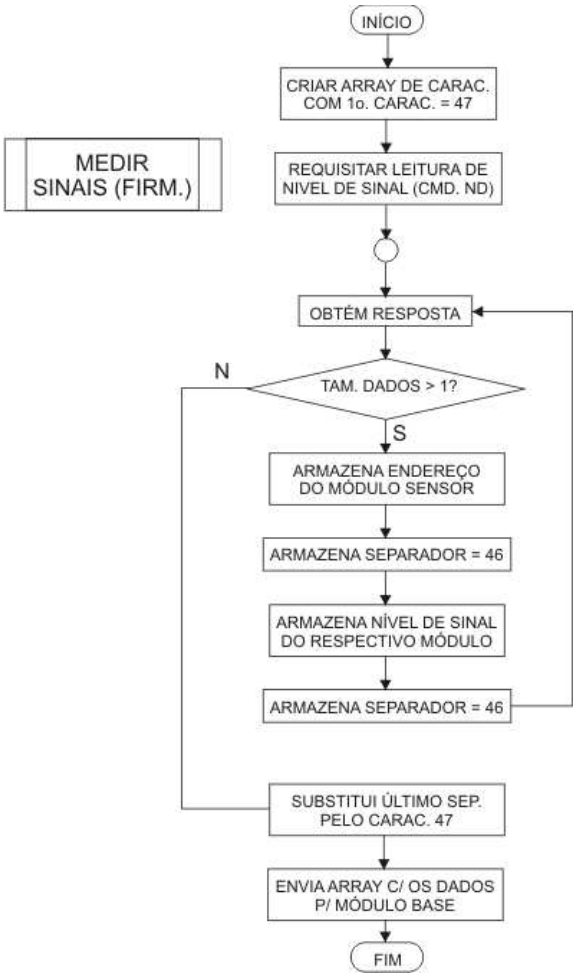


Figura I.4 – Diagrama de blocos da função “Medir Sinais (Firm.)”.

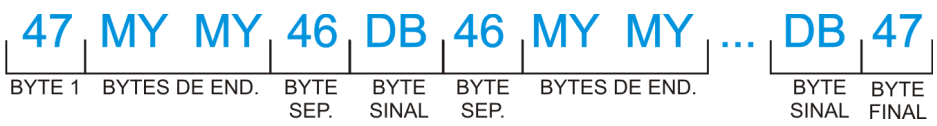


Figura I.5 – Estrutura do pacote de dados transmitido ao módulo base pelo módulo remoto.

O comando 51 (caractere ‘3’ em ASCII) chama a função de medir temperatura. Esta leitura da temperatura ambiente é feita através do processamento dos dados advindos da saída do sensor LM35 e da saída do controlador proporcional, como apresentado na Figura 4.5, do item 4.1.2. Para tanto, uma função de medição dinâmica do ganho K_p do amplificador foi adicionada ao programa do microcontrolador. A idéia é que, a cada leitura feita pelo conversor A/D,

calcule-se o ganho do controlador proporcional, através da simples divisão entre o valor lido da entrada A/D, ligada à saída do amplificador, pelo valor lido da entrada A/D ligada diretamente ao LM35. Após uma série de leituras, o programa processa o conjunto de ganhos e seleciona o ganho mais provável para ser utilizado no cálculo da temperatura. O algoritmo utilizado para este processamento é representado, com todos os seus passos lógicos, pelo diagrama de blocos da Figura I.6. Suas variáveis de maior relevância são:

- Variáveis **G1**, **G2**, **G3**, e **G4**: armazenam os valores de ganho, organizadas em ordem de maior freqüência de ocorrência;
- Variáveis **Gx1**, **Gx2**, **Gx3**, e **Gx4**: representam a quantidade de ocorrência dos ganhos G1, G2, G3, e G4, respectivamente;
- Variável **zerar**: representa a quantidade de ocorrência de ganhos distintos aos ganhos padrões: G1, G2, G3 e G4;
- Variáveis **valor1** e **valor2**: armazenam os valores digitais das saídas do controlador proporcional e do sensor LM35, respectivamente, após a conversão A/D (valores entre 0 e 1023, para um conversor A/D de 10 bits).

Infere-se do diagrama de blocos da Figura I.6 que a cada chamada da função, o programa verifica se os valores de ganhos padrões se encontram em uma faixa válida (entre 9 e 14) . Em seguida, obtém-se o valor da conversão A/D da entrada vinculada ao controlador proporcional (**valor1**). Então, calcula-se a temperatura ambiente, utilizando o valor de **G1** como o ganho do controlador. A temperatura é calculada através de uma simples regra de três (apresentada nas Equações 16, 17 e 18), visto que a tensão de saída do sensor LM35 é diretamente proporcional a temperatura medida. Caso o ganho **G1** não tenha sido inicializado, ou se encontre fora da faixa válida de ganhos, o programa utiliza um ganho constante igual a 10 para o cálculo da temperatura.

Após o cálculo da temperatura, o programa obtém a conversão A/D da entrada vinculada ao sensor LM35 e, então, chama a função que atualiza o vetor de ganhos, representada pelo diagrama de blocos da Figura I.7. Nesta função, calcula-se a razão entre as variáveis **valor1** e **valor2**. Esta razão é o valor de ganho de tensão do amplificador referente àquela leitura. Este ganho é, em seguida, comparado com os ganhos de outras leituras e o valor de ganho que mais se repete

é, então, utilizado como valor correto de ganho do controlador proporcional e será utilizado para o cálculo da temperatura.

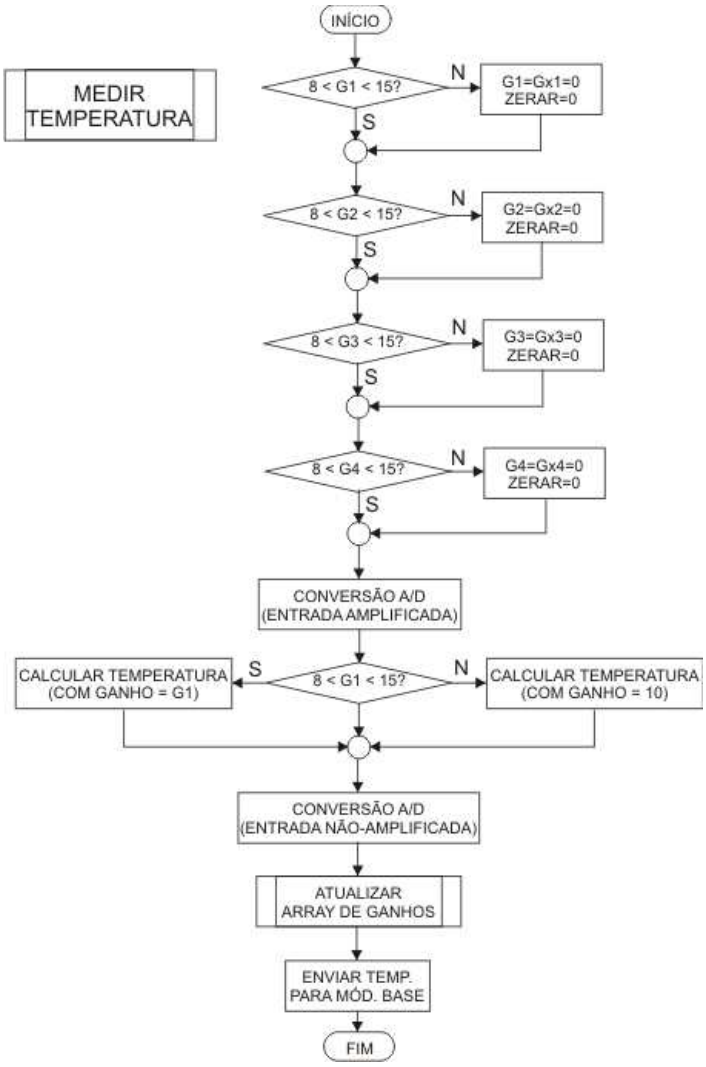


Figura I.6 – Diagrama de blocos da função “Medir Temperatura”.

$$\frac{V_x}{5V} = \frac{\boldsymbol{valor1}}{1023} \tag{16}$$

$$V_x = temperatura \times (0.01\ V \times ganho) \tag{17}$$

$$temperatura = \frac{\boldsymbol{valor1} \times 100}{204.6 \times ganho} \tag{18}$$

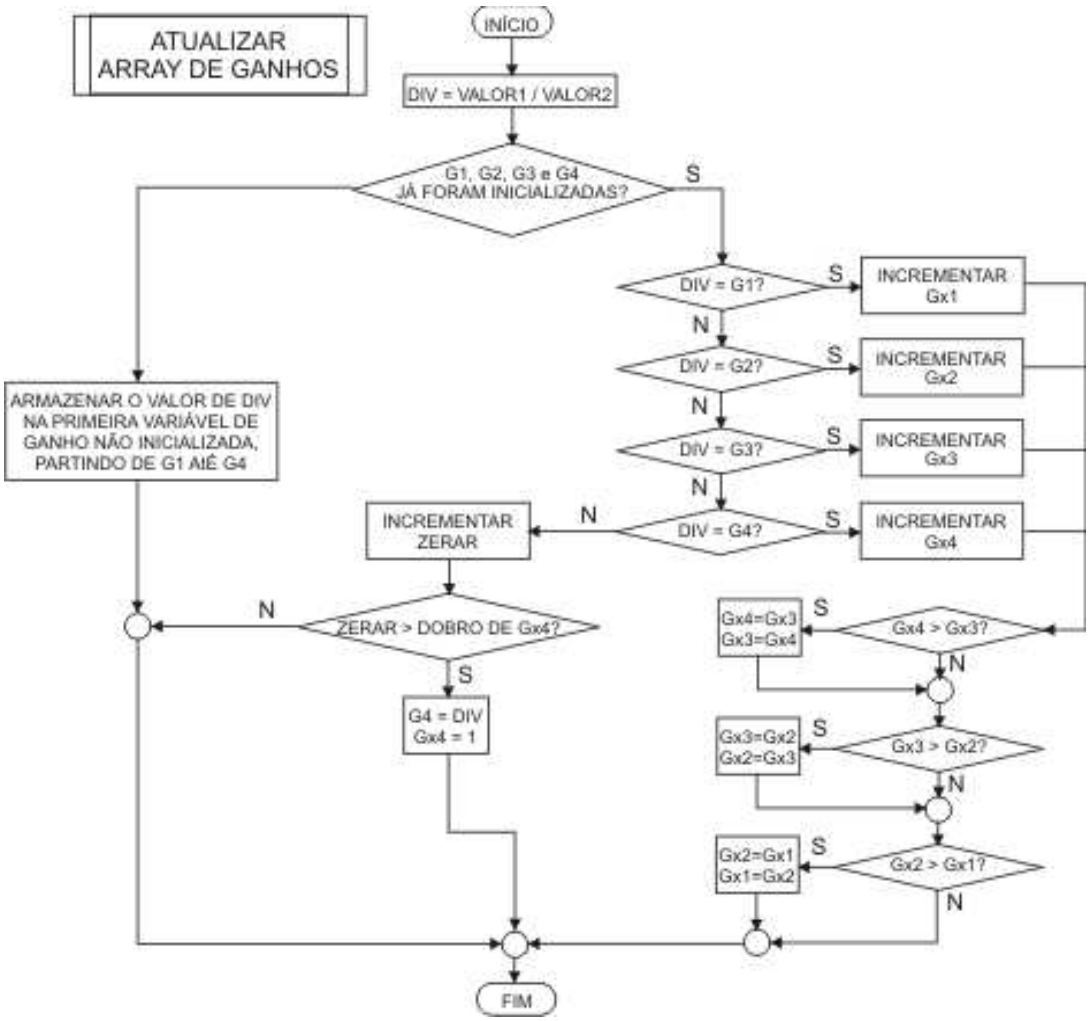


Figura I.7 – Diagrama de blocos da função “Atualizar *Array* de Ganhos”, uma sub-rotina da função “Medir Temperatura”.

Anexo II

Este anexo descreve detalhadamente o funcionamento do programa de localização, desenvolvido em C, que processa as informações transmitidas pelo módulo remoto para o módulo base, como descrito na Figura 4.8 do item 4.1.2. O diagrama de blocos do programa é ilustrado na Figura II.1.

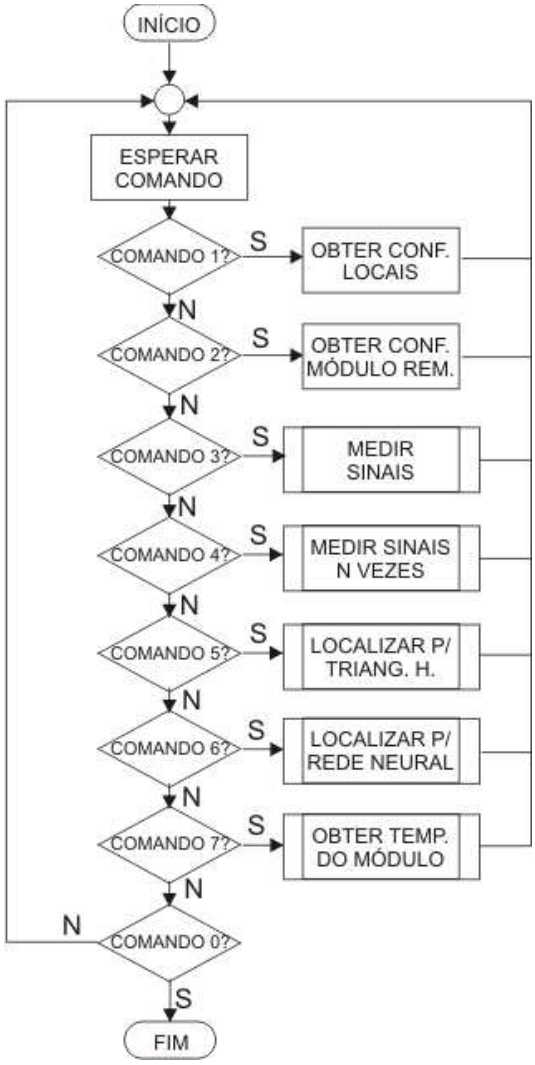


Figura II.1 – Diagrama de blocos do *software* de localização.

A idéia do programa é aguardar algum comando válido ser digitado pelo usuário e então processar o comando. Entre os possíveis comandos, o comando ‘0’ termina o programa e o comando ‘7’ requisita o valor de temperatura ambiente medido pelo módulo remoto. Neste último caso, não há processamento. O

programa, simplesmente, imprime na tela o valor de temperatura enviado pelo módulo remoto.

O comando ‘1’ faz com que o módulo XBee do módulo base entre em modo de comando e, então, requisita uma série de configurações como o endereço de 16bits, o endereço destino, o canal e o número da PAN de trabalho, entre outras. Esta função possui a finalidade de checar as configurações do módulo base.

O comando ‘2’ obtém os mesmos resultados do comando ‘1’, porém referentes ao módulo remoto. Neste caso, transmite-se o comando ‘2’ para o módulo remoto que, após o processamento do comando, responde com suas configurações.

O comando ‘3’ obtém a intensidade do sinal de cada módulo sensor com relação ao módulo remoto. Para tanto, o programa chama a função de medição de sinal, representada pelo diagrama de blocos da Figura II.2.

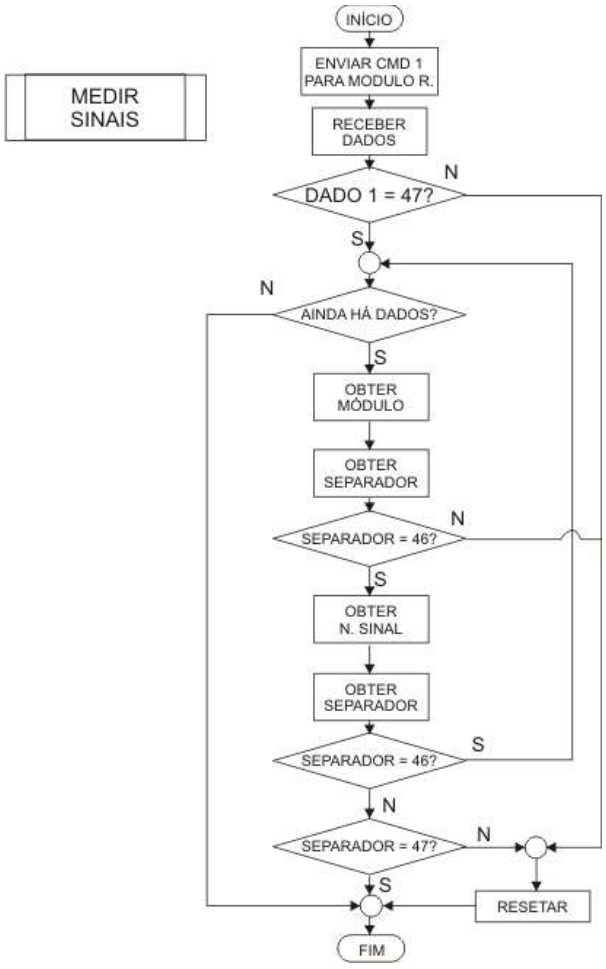


Figura II.2 – Diagrama de blocos da função “Medir Sinais”.

Analisando o diagrama de blocos da Figura II.2, verifica-se que o programa envia o comando ‘1’ para o módulo remoto, que obtém a intensidade dos sinais dos

módulos sensores e envia estes dados ao módulo base no padrão definido na Figura I.5 do Anexo I. Os dados recebidos são checados, para verificar se obedecem ao padrão estabelecido. Assim, o primeiro *byte* é comparado ao *byte* 47 (que é o valor esperado), e então examina-se a existência de outros *bytes* para serem analisados. Se existirem dados, o programa analisa os próximos dois *bytes* para obtenção do valor de endereço do módulo sensor, então testa a existência do separador (*byte* 46) e, em seguida, obtém a intensidade do sinal relativo ao módulo sensor. Por fim, verifica-se a existência de um separador (*byte* 46) ou de um *byte* 47, que indicam, respectivamente, a existência de mais um conjunto de dados ou o fim da transmissão dos dados. Caso haja algum erro na transmissão (por exemplo, o não recebimento de um byte separador), o programa entra em uma função de *reset*, no qual se faz uma limpeza no buffer de transmissão e espera o pacote de dados,descrito na Figura I.3, que o módulo remoto envia quando é reinicializado.

O comando ‘4’ chama uma função que faz “n” leituras da intensidade do nível de sinal dos módulos sensores e então grava o conjunto de dados obtidos em um arquivo. Esta função é representada pelo diagrama de blocos da Figura II.3.

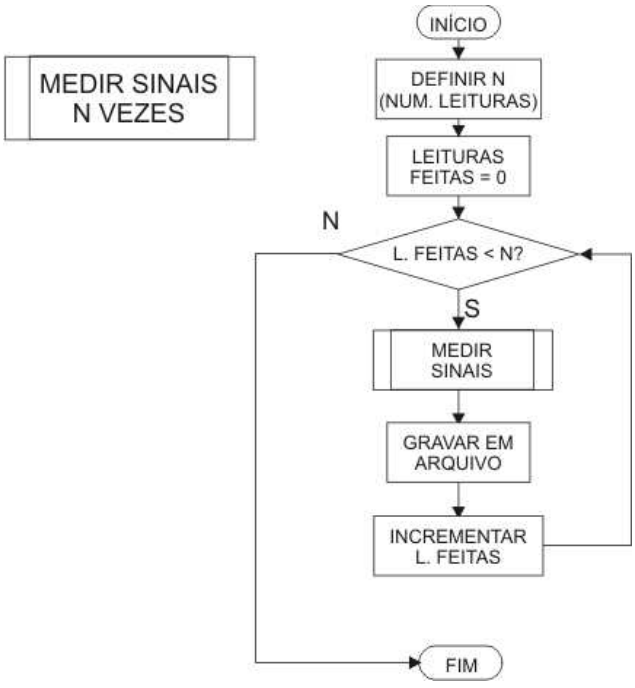


Figura II.3 – Diagrama de blocos da função “Medir Sinais N Vezes”.

Analisando o diagrama de blocos da Figura II.3, verificamos que, inicialmente, o programa solicita o número de leituras de intensidade de sinal que deve realizar, e

então entra em um enlace de leitura até que todas as medições tenham sido feitas. Por fim, grava-se em um arquivo todas as informações obtidas a cada interação.

O comando ‘5’ chama o algoritmo de localização por triangulação hiperbólica e, portanto, somente é analisado no Anexo III. Da mesma forma, o comando ‘6’ é utilizado no método de localização utilizando redes neurais e, portanto, será analisado, somente, no Anexo IV.

Anexo III

A função de localização por triangulação hiperbólica é construída com um código em linguagem C, e possui seus cálculos baseados em relações de geometria analítica, para determinar, de forma simples, a distância radial de cada módulo sensor no recinto. Com base na intersecção destas distâncias, o algoritmo estima a área de localização provável e o ponto que produz o menor erro de distância entre os módulos sensores (de acordo com o procedimento descrito no sub-item 4.2.1.2).

Assim, o recinto de localização é mapeado no programa como uma matriz de valores inteiros inicialmente nulos, dividindo o ambiente em pequenos quadrados de 10 por 10 cm. Uma vez que o ambiente no qual o experimento foi realizado possui 7 por 3 metros, o programa trabalha com uma matriz de 30 linhas e 70 colunas. A matriz foi definida com este tamanho para que cada uma de suas células pudesse ser impressa no gráfico de saída (já que a tela padrão do sistema operacional DOS possui uma largura de 80 caracteres).

O diagrama de blocos do algoritmo de localização por triangulação hiperbólica é apresentado na Figura III.1.

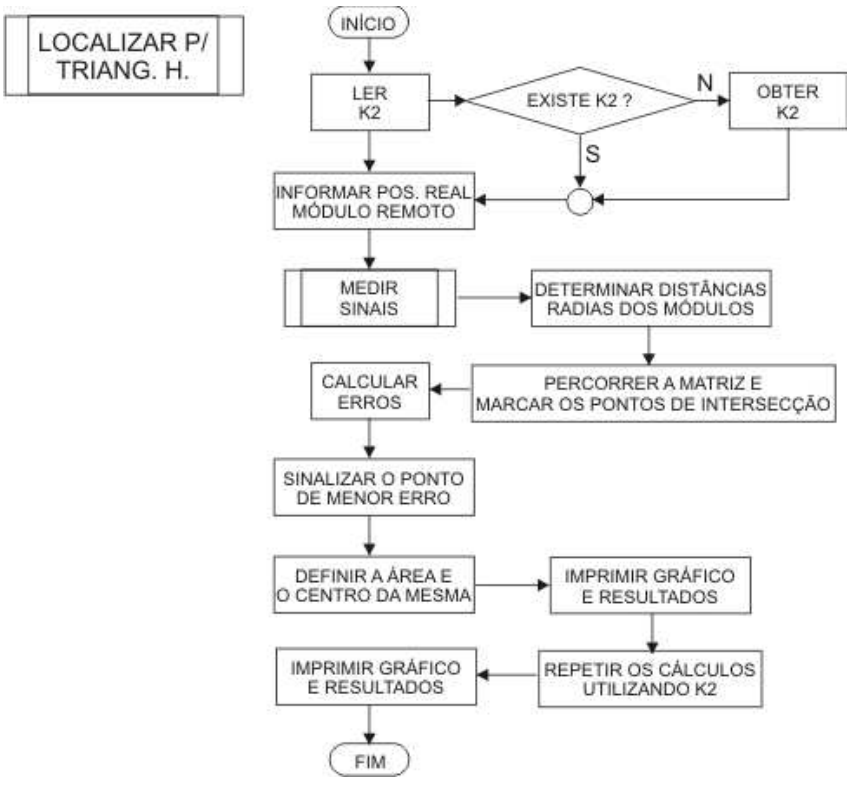


Figura III.1 – Diagrama de blocos da Função "Localizar por Triangulação Hiperbólica".

Primeiramente, obtém-se os dados necessários para os cálculos da função: verifica-se a existência de um arquivo texto que contenha um valor já calculado anteriormente para a constate K_2. Se não, o algoritmo inicia o procedimento para obter e calcular a constante. A posição real do módulo é fornecida pelo usuário apenas para a comparação, pelo algoritmo, do ponto calculado e do ponto real. Para completar a coleta de dados, a função solicita ao módulo remoto os dados de nível de sinal dos módulos sensores.

O próximo passo é percorrer toda a matriz, utilizando seus índices como coordenadas de um gráfico cartesiano. Assim, dado um ponto desta matriz, a distância deste ponto em relação a um dos módulos é calculada, sendo posteriormente comparada com a distância radial obtida da Equação 13 para aquele módulo; se a distância calculada for menor que a distância radial daquele módulo, o valor da matriz naquele ponto é incrementado. Este procedimento é realizado uma vez para cada módulo.

Após este procedimento, os pontos que se encontram nas regiões de intersecção das distâncias radiais possuirão os maiores valores dentro da matriz. Assim, para estes pontos, é calculado o valor de erro descrito na Equação 15, sendo sinalizado o ponto que possuir o maior erro. Este ponto é estimado como sendo a localização mais provável para o módulo remoto.

Em seguida, a função utiliza outra matriz, de 300 linhas e 700 colunas, para estimar o tamanho da área de provável localização. Esta outra matriz é utilizada para se obter um valor mais preciso do tamanho desta área. O procedimento descrito anteriormente é repetido para esta outra matriz, e, em seguida, os seus pontos de maior valor (aqueles que indicam a sobreposição das distâncias radias dos módulos) são somados, fornecendo o valor da área de localização provável. Por fim, a média das coordenadas X e Y destes pontos de intersecção fornecem uma estimativa das coordenadas do centro desta área. Os dados obtidos são apresentados de acordo com as Figuras 4.14 e 4.15, citadas no sub-item 4.2.1.2.

Por fim, toda a função é repetida, utilizando o valor da constante K_2 na Equação 13, ao invés do valor fixo calculado pelo método descrito no sub-item 4.2.1.2.

Anexo IV

Este anexo descreve detalhadamente o funcionamento do algoritmo de localização por redes neurais. Este algoritmo é dividido em duas partes. Inicialmente, a função ‘6’ do programa de localização, descrito no Anexo II, é utilizada para se obter o nível de intensidade de sinal dos módulos sensores e transmitir estes dados para o MATLAB. Em seguida o processamento é feito através das redes neurais desenvolvidas em ambiente MATLAB. Esta segunda etapa é representada pelo diagrama de blocos da Figura IV.2.

A função chamada pelo comando ‘6’ do programa de localização, “Localizar por Rede Neural”, é representada pelo diagrama de blocos da Figura IV.1. Seu funcionamento é relativamente simples. Inicialmente, o programa solicita a posição real do módulo remoto e, então, entra em um enlace que só é finalizado ao se pressionar alguma tecla do computador. Dentro deste enlace, o software requisita ao módulo remoto que faça a leitura da intensidade de sinal dos módulos sensores. Em seguida, o programa grava, dentro do diretório de trabalho do MATLAB, a posição real do módulo remoto e a intensidade do sinal de cada modulo sensor. A função é finalizada quando se define a posição do módulo remoto como 0, tanto para o eixo X como para o eixo Y. Ao finalizar a função, o software cria o arquivo **stop.txt** dentro do diretório de trabalho do MATLAB.

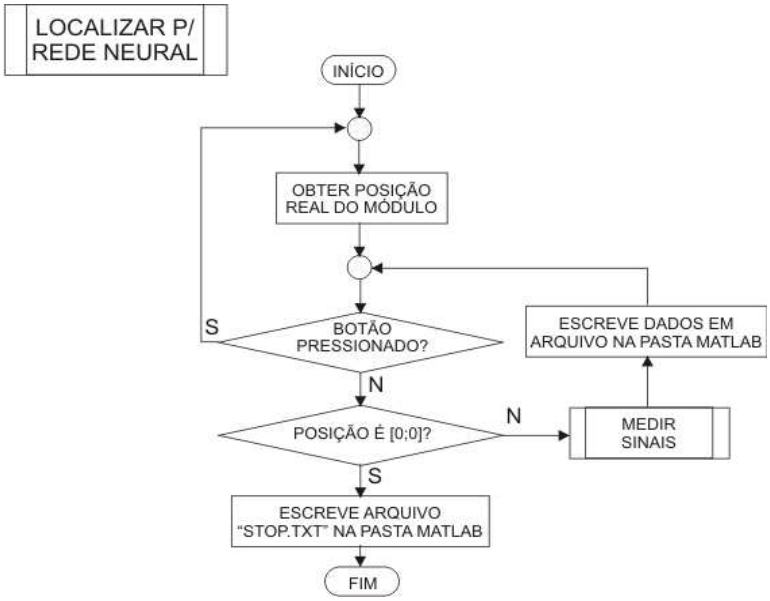


Figura IV.1 – Diagrama de blocos da função “Localizar por Rede Neural”.

Dentro do ambiente de trabalho do MATLAB obtêm-se os valores da intensidade de sinal dos módulos sensores. Em seguida, a saída do sistema é simulada utilizando as redes neurais previamente criadas. Após a simulação, um gráfico (ilustrado na Figura 4.16 do sub-item 4.2.2.4) relacionando a posição do módulo remoto simulada pelas redes neurais com a posição real do módulo é gerado. Este processo é contínuo até que se detecte a presença do arquivo **stop.txt** dentro da pasta de trabalho do MATLAB. O algoritmo descrito é representado pelo diagrama de blocos da Figura IV.1.

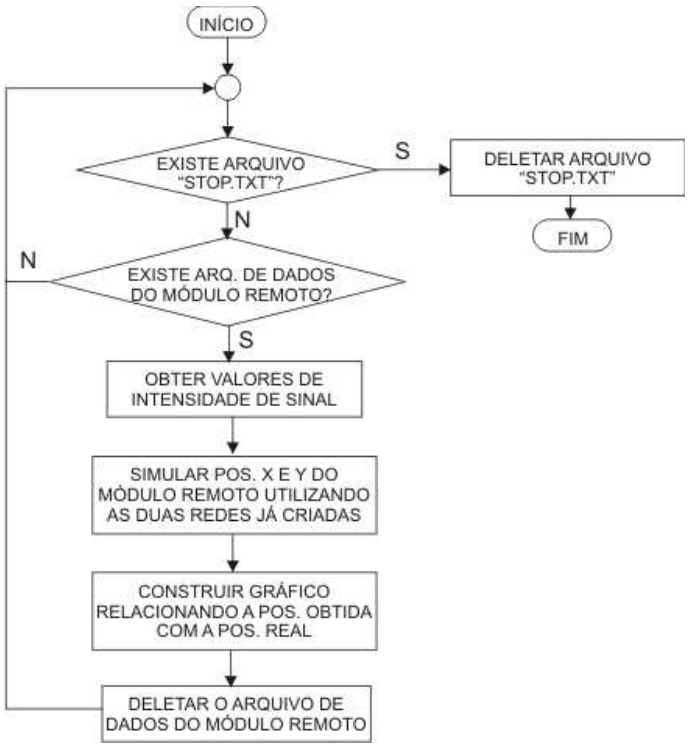


Figura IV.2 – Diagrama de blocos do processo de obtenção dos resultados da localização por redes neurais.

Anexo V

Anexo V.1 – Funções de Entrada e Saída do AVR

```
/*##########################################################################
################*/
/*##########################################################################
################*/
/*##########################################################################
################*/
/*##########################################################################
################*/
/*##########################################################################
################*/
/*###################                                    ###################*/
/*###################     Funções que Fazem a Comunicação Serial
###################*/
/*###################                                    ###################*/
/*##########################################################################
################*/
/*##########################################################################
################*/
/*##########################################################################
################*/
/*##########################################################################
################*/
/*##########################################################################
################*/

void USART_Iniciar(void);


int AVR_printC(unsigned char dados);
void AVR_printString(int tam, char txdBuffer[15], int linha);
void AVR_printLong(long valor);


int AVR_getC (int timelimit);
char *AVR_getString(int digitos);
char *AVR_getSInt(int digitos, int *valor);

void AVR_flushIO(int vezes);


int StringtoInt (char *temp, int expoente);
char InttoString (char *result, int valor, int expoente);
```

```
//Bibliotecas relacionadas ao debug do XBee. Se não for utiliza-las, comente.
char cmdReset;
char cmdERRO;

int loop1, loop2;


/*#########################################################################
################*/
/*#                                                                     #*/
/*#                        FUNÇÃO CONFIGURAÇÃO                          #*/
/*#                                                                     #*/
/*#########################################################################
################*/
/*
UCSRA -> (7)RXC  (6)TXC  (5)UDRE  (4)FE - Frame Error
                (3)DOR - Data OverRun    (2)PE - Parity Error
                (1)U2X - Modo 2x Speed Operation   (0)MPCM - Operacao
MultiProcessos

UCSRB -> (7)RXCIE
                (6)TXCIE
                (5)UDRIE
                [[4]]RXEN - Habilita recepcao  ----------> OBRIGATORIO
                [[3]]TXEN - Habilita Transmissão   -------> OBRIGATORIO
                (2)UCSZ2 - Tam da palavra
                (1)RXB8 - Recepcao 9bit          (0)TXB8 - Transmissao 9bit

UCSRC -> (7)URSEL - Habilita o acesso ao UCSRC [1] ou o UBRRH [0]
                (6)UMSEL - Modo Assinc.[0] ou Sincr.[1]
                (5-4)UPM - Habilita teste de paridade [00 - não] [10 - impar] [11 - par]
                (3)USBS  - Habilita Stop Bit [0 - 1bit] [1 - 2bits]
                (2-1)UCSZ- Tamanho da palavra (com UCSZ2 do UCSRB). [0 1 1]-
>8bits  [1 1 1]->9bits...
                (0)UCPOL   - Modo do clock (apenas no modo síncrono).

UBRRH -> (7)URSEL - Habilita o acesso ao UCSRC [1] ou o UBRRH [0]
                (6-3) Reservados
                (2-0)BAUD-RATE Register
UBRRL -> (7-0)BAUD-RATE Register
                  _______[U2X=0]_____________[U2X=1]________
                 |   [2400]UBRR=25   |  [2400]UBRR=51  |
         PARA 1MHz -->|   [4800]UBRR=12   |  [4800]UBRR=25  |
                 |   [9600]UBRR=6    |  [9600]UBRR=12  |
                 .....................................................

                 _______[U2X=0]_____________[U2X=1]________
                 |   [2400]UBRR=416  |  [2400]UBRR=832 |
        PARA 16MHz -->|   [4800]UBRR=207  |  [4800]UBRR=416 |
                 |   [9600]UBRR=103  |  [9600]UBRR=207 |
                 .....................................................

*/
```

```
void USART_Iniciar(void) {

	UDR=0;
	UCSRA = _BV(U2X);

	UBRRH=0;
	UBRRL=12;

	UCSRB = _BV(RXEN) | _BV(TXEN);

	UCSRC= _BV(URSEL) | _BV(UCSZ1) | _BV(UCSZ0);

	return;
}



/*#########################################################################
################*/
/*#                                                               #*/
/*#                    FUNÇÃO TRANSMITIR - CHAR                   #*/
/*#                     [  AVR --> serial  ]                      #*/
/*#########################################################################
################*/
/*
Só é possível enviar dados via serial, se o registrador UDR (USART Data Register)
estiver vazio.
Quando o UDR está vazio, o bit 5 (UDRE) do registrador UCSRA é setado [1].
Assim, antes de mandar
o byte de dado via serial, verifica-se se o UDRE está setado: while (UDRE==0);
então --> UDR=dados.
*/
int AVR_printC(unsigned char dados) {
	while(!(UCSRA & (1<<UDRE))) { }
	UDR=dados;
	return 0;
}



/*#########################################################################
################*/
/*#                                                               #*/
/*#                     FUNÇÃO RECEBER - CHAR                     #*/
/*#                     [ serial --> AVR ]                        #*/
/*#########################################################################
################*/
/*-------------------------------------------------------------------------------
Se existir algum dado no registrador UDR (USART Data Register), o bit 7 (RXC) do
registrador UCSRA
```

```
é setado (HIGH). Aproveita-se isso para fazer um while (RXC == 0) {} e após
(RXC=1)--> retorna o UDR.
-------------------------------------------------------------------------------------
Se timelimit=0. Espera-se eternamente por um byte no registrador USART.
Caso este não seja seu desejo. Utilize o int timelimit:
        Código esperará cerca de 1,5ms * timelimit.
        Então se não houver um byte no registrador USART, retorna o byte 127 e
seta cmdReset=5;

        O Resto do código deve entender que quando cmdReset=5 significa que o
programa deve ser resetado.
        se não estiver utilizando este protocolo, crie um próprio ou comente as linhas.
 -------------------------------------------------------------------------------------
        -> Codigo de ERRO: 1
 -------------------------------------------------------------------------------------*/
int AVR_getC (int timelimit) {
        loop1=0;
        loop2=0;
        int resultado;

        if (timelimit>0) {
                while(!(UCSRA & (1<<RXC))) {

                        if (loop1>=100) { loop1=0; loop2++; }               //1,5ms
                        if (loop2>=timelimit) {                 //timelimit * 1,5ms.
                                cmdReset=5;                                 //cmdReset=5.
Significa erro. Deve resetar.
                                return 127;
                        }
                        loop1++;

                }
        }
        else {    while(!(UCSRA & (1<<RXC))) { }       }          //sem limites de tempo.

        resultado=UDR;

        //Protocolo de comunicação com xbee remoto. comando |}}}| é reset: (124)
(125) (125) (125) (124);
        //Neste caso, o char global cmdReset será setado como 5. O resto do código
deve entender que é então para resetar.
        //SE NÃO FOR UTILIZAR ESTE PROTOCOLO, COMENTE AS LINHAS
ABAIXO (mantenha o return.)
        if (cmdReset==0) {
                if (resultado==124) cmdReset++;
        }
        if ( (cmdReset>=1) && (cmdReset<=3) ) {
                if (resultado==125) cmdReset++;
                else cmdReset=0;
        }
        if (cmdReset==4) {
                if (resultado==124) { cmdReset=5;  cmdERRO=1; }
```

```
		else cmdReset=0;
	}

	return resultado;				//retorna o resultado.
}


/*#########################################################################
################*/
/*#                                                               #*/
/*#                     FUNÇÃO FLUSH DA SERIAL                    #*/
/*#                       [ serial --> --- ]                      #*/
/*#########################################################################
################*/
/*
Por i vezes, testa se há algum dado no buffer de transmissão (UDR),
UCSRA=HIGH, Então recebe o dado e ignora-o.
*/

void AVR_flushIO(int vezes) {
	int i;
	unsigned char flushtemp;
	for (i=0; i<vezes; i++) {
		while((UCSRA & (1<<RXC))) { flushtemp = UDR; }
	}
	return;
}



/*#########################################################################
################*/
/*#                                                               #*/
/*#                     FUNÇÃO RECEBER - String                   #*/
/*#                       [ serial --> AVR ]                      #*/
/*#########################################################################
################*/
/*
UTILIZAR A SOLUÇAO SEM RETORNO DE PONTEIRO E SEM USAR MALLOC,
PODE ATÉ
PARECER MAIS SIMPLES E ORGANIZADO MAS OCUPA MAIS DE 100BYTES DE
ESPAÇO A MAIS.
-------------------------------------------------*/
char AVR_getString(char *temp, int digitos) {
	int i;
	for (i=0; i<digitos; i++) {
		*(temp+i)=AVR_getC();
		if ( ( *(temp+i)==13 ) || ( *(temp+i)==0 ) ) break;
	}
	*(temp+i)=0;
	return 0;
}
```

```
//---------------------------------------------*/

/*


*/

char *AVR_getString(int digitos) {
        int i;
        char *temp;
        temp=malloc(sizeof(char)*digitos);
        for (i=0; i<digitos; i++) {
                *(temp+i)=AVR_getC(0);
                if ( ( *(temp+i)==13 ) || ( *(temp+i)==0 ) ) break;
        }
        *(temp+i)=0;
        return temp;
}



/*############################################################################
################*/
/*#                                                          #*/
/*#                    FUNÇÃO RECEBER - String e Int                  #*/
/*#                       [ serial --> AVR ]                        #*/
/*############################################################################
################*/
/*


*/

char *AVR_getSInt(int digitos, int *valor) {
        int i;
        char *temp;
        temp=malloc(sizeof(char)*digitos);
        for (i=0; i<digitos; i++) {
                *(temp+i)=AVR_getC(0);
                if ( ( *(temp+i)==13 ) || ( *(temp+i)==0 ) ) break;
        }
        *(temp+i)=0;
        //*valor= strtol(temp,NULL,10);
        *valor=StringtoInt(temp,10);
        return temp;
}



/*############################################################################
################*/
```

```
/*#                                                              #*/
/*#                    FUNÇÃO TRANSMITIR - String                #*/
/*#                      [ AVR --> Serial ]                      #*/
/*##############################################################################*/
/*


*/
void AVR_printString(int tam, char txdBuffer[15], int linha) {
//Adicionar controle se i<tam ou se txdBuffer[i]==0
        int i;
        for (i=0; i<tam; i++) {
                AVR_printC(txdBuffer[i]);
        }
        if (linha==0) { }    //AVR_printC(0);         }
        else { Nova_Linha(); }
        return;
}


/*##############################################################################*/
/*#                                                              #*/
/*#                    FUNÇÃO TRANSMITIR - Integer               #*/
/*#                      [ AVR --> Serial ]                      #*/
/*##############################################################################*/

void AVR_printInt(int valor) {
//Adicionar controle se i<tam ou se txdBuffer[i]==0
        char temp = valor >> 8;
        AVR_printC(temp);
        temp = valor;
        AVR_printC(temp);

        return;
}


void AVR_printLong(long valor) {
//Adicionar controle se i<tam ou se txdBuffer[i]==0
        char temp = valor >> 24;
        AVR_printC(temp);
        char temp = valor >> 16;
        AVR_printC(temp);
        char temp = valor >> 8;
        AVR_printC(temp);
        temp = valor;
        AVR_printC(temp);

        return;
```

```
}



/*##########################################################################
################*/
/*#                                                                    #*/
/*#                FUNÇÃO TRANSMITIR - Nova Linha [ASCII 13]           #*/
/*#                        [ AVR --> Serial ]                          #*/
/*##########################################################################
################*/
/*
Escreve o char 13, equivalente a Carriage Return.
*/
void Nova_Linha(void) {
        AVR_printC(13);
        //AVR_printC(0);
        return;
}


void SOH(void) {
        AVR_printC(124);
        AVR_printC(124);
        AVR_printC(124);
        return;
}

void EOT(void) {
        AVR_printC(124);
        AVR_printC(124);
        AVR_printC(124);
        return;
}


*/



//A FUNÇÃO OCUPA 140 bytes de memoria
//Ao contrário da strtol() que ocupa 800 bytes.
int StringtoInt (char *temp, int expoente) {
        int i,k;
        int tam = strlen(temp);
        int result=0;
        int exp=1;
        char digito;

        for (i=0; i<tam; i++) {
                digito=*(temp+i);
```

```
                if ( (digito>64) && (digito<71) ) { digito=digito-7;}
                exp=1;
                for (k=0; k<(tam-1-i); k++) { exp*=expoente; }
                result+= (digito-48)*exp ;
        }
        return result;
}



//A FUNÇÃO OCUPA exatamente 200 bytes de memoria
//Ao contrário da itoa que ocupa 100 bytes (ihh perdi);

char InttoString (char *result, int valor, int expoente) {
        int i;
        int exp=1;
        int valortemp;

        valortemp=valor;
        for (i=0; valortemp>0; i++) {
                exp=1;
                while ( (exp*expoente)<valortemp ) exp*=expoente;
                *(result+i) = valortemp/exp;
                if ( *(result+i) < 10 ) { *(result+i) += 48; }
                else { *(result+i) += 55; }
                valortemp = valortemp%exp;
                if (exp==1) {
                        *(result+i+1)=0;
                        return 0;
                }
        }
        *(result+i) =0;
        return 0;
}
```

Anexo V.2 – Funções de comunicação API com o XBee

```
/*##################################################################################*/
/*##################################################################################*/
/*##################################################################################*/
/*##################################################################################*/
/*##################################################################################*/
/*##################                                  ##################*/
```

```
/*##################     Funções do PROTOCOLO de Comunicação
##################*/
/*##################             com o XBEE                ##################*/
/*##################                                       ##################*/
/*#########################################################################
################*/
/*#########################################################################
################*/
/*#########################################################################
################*/
/*#########################################################################
################*/
/*#########################################################################
################*/



void enviarReset(void);
void enviarResetERRO(void);


char *getATresposta(int *tam);
char cmdATler(char cmd1, char cmd2);
char cmdATescrever(char cmd1, char cmd2, int tam, unsigned char *param);
char transmitir(char dl1, char dl2, int tam, unsigned char *dados);


int calcularCHKSum(int soma);
void leituraERRO(int erro);

/*#########################################################################
################*/
/*#########################################################################
################*/
/*#########################################################################
################*/
/*#########################################################################
################*/




/*////////////////////////////////////////////////////////////////////////////*/
/*////////////////////////////////////////////////////////////////////////////*/
/*#########################################################################
################*/
/*#########################################################################
################*/
/*#########################################################################
################*/
```

```
/*                                                                    */
/*                    FUNÇÕES BÁSICAS DO XBEE                         */
/*                                                                    */
/*####################################################################
################*/
/*####################################################################
################*/
/*####################################################################
################*/
/*////////////////////////////////////////////////////////////////////*/
/*////////////////////////////////////////////////////////////////////*/



/*####################################################################
################*/
/*#                                                                  #*/
/*#              FUNÇÃO PARA CALCULAR O CHKSUM                       #*/
/*#              a partir da soma dos bytes a enviar                 #*/
/*#                                                                  #*/
/*####################################################################
################*/
char transmitir(char dl1, char dl2, int tam, unsigned char *dados) {
        int soma, cksum, i;


        soma=5+tam;

        cksum=1+dl1+dl2+1;
        for (i=0; i<tam; i++) {
                cksum+=*(dados+i);
        }
        cksum=calcularCHKSum(cksum);


        AVR_printC(126);   //7E
        AVR_printC(0);             //TAM
        AVR_printC(soma); //TAM

        AVR_printC(1);             //Tx com 16bits
        AVR_printC(0);             //Frame ID
        AVR_printC(dl1);   //DL
        AVR_printC(dl2);   //DL
        AVR_printC(1);             //Options

        for (i=0; i<tam; i++) {       AVR_printC(*(dados+i));   }       //dados
        AVR_printC(cksum);        //chksum


return 0;
}
```

```
/*#########################################################################
################*/
/*#                                                                     #*/
/*#              FUNÇÃO PARA LER UM COMANDO ESPECIFICO NO XBEE
#*/
/*#                                                                     #*/
/*#########################################################################
################*/
char cmdATler(char cmd1, char cmd2) {
        int var_chksum;

        delay(10);
        AVR_flushIO(30);

        var_chksum=8+51+cmd1+cmd2;
        var_chksum=calcularCHKSum(var_chksum);

        AVR_printC(126);   //7E
        AVR_printC(0);              //TAM
        AVR_printC(4);              //TAM
        AVR_printC(8);              //COMANDO AT
        AVR_printC(51);             //FRAME ID
        AVR_printC(cmd1);
        AVR_printC(cmd2);
        AVR_printC(var_chksum);         //Checksum

return 0;
}



/*#########################################################################
################*/
/*#                                                                     #*/
/*#              FUNÇÃO PARA LER UM COMANDO ESPECIFICO NO XBEE
#*/
/*#                                                                     #*/
/*#########################################################################
################*/
char cmdATescrever(char cmd1, char cmd2, int tam, unsigned char *param) {
        int soma, cksum, i;

        soma=4+tam;

        cksum=8+cmd1+cmd2;
        for (i=0; i<tam; i++) {
                cksum+=*(param+i);
        }
        cksum=calcularCHKSum(cksum);
```

```
        delay(10);
        AVR_flushIO(30);

        AVR_printC(126);   //7E
        AVR_printC(0);              //TAM
        AVR_printC(soma); //TAM
        AVR_printC(8);              //COMANDO AT
        AVR_printC(00);             //FRAME ID
        AVR_printC(cmd1);
        AVR_printC(cmd2);
        for (i=0; i<tam; i++) {       AVR_printC(*(param+i));   }       //dados
        AVR_printC(cksum);                  //Checksum

return 0;
}




/*##########################################################################
################*/
/*#                                                                  #*/
/*#              FUNÇÃO OBTÉM OS DADOS DESEJADOS DA                  #*/
/*#                   RESPOSTA AT do XBEE                            #*/
/*#                                                                  #*/
/*##########################################################################
################*/
/*---------------------------------------------------------------------------------

        -> Pega os primeiros 7bytes de resposta a um cmd AT. Destes, testa o
1º=0x7E e o 4º=0x88.
        -> Então, alloca a string passada com um tam certo p/ o resto dos dados
(length-4).
        -> Na string alocada, insere todos os dados obtidos como resposta.
        -> Por último, ignora o checksum e testa se o status=0 (OK).
---------------------------------------------------------------------------------
        -> Codigo de erro: 10, 11, 12.
---------------------------------------------------------------------------------*/
char *getATresposta(int *tam) {


        char *resposta;
        char temp;
        int i;

        temp=AVR_getC(1200);    //1ºBYTE -> 7E (126)
        if (temp != 126)  { leituraERRO(10); return NULL; }

        *tam=AVR_getC(300);             //2ºBYTE -> tamanho
        *tam*=256;
```

```
        *tam+=AVR_getC(300);    //3ºBYTE -> tamanho
        *tam=*tam-4;

        temp=AVR_getC(300);             //4ºBYTE -> api ID --> 0x88 = 136 para
resposta de cmd AT
        if (temp != 136)  { leituraERRO(11); return NULL; }  //
        temp=AVR_getC(300);     //5ºBYTE -> frame id.
        temp=AVR_getC(300);             //6ºBYTE -> comando AT
        temp=AVR_getC(300);             //7ºBYTE -> comando AT

        resposta=malloc(sizeof(char)*(*tam));
        for (i=0; i<*tam; i++) {                        //8ºBYTE -> status
                *(resposta+i)=AVR_getC(300);    //9º-nºBYTEs -> resposta
        }
        temp=AVR_getC(300);             //n+1ºBYTE -> checksum

        if ( *(resposta) == 1 ) { leituraERRO(12); return NULL; }


return resposta;
}


/*////////////////////////////////////////////////////////////////////////////*/
/*////////////////////////////////////////////////////////////////////////////*/
/*############################################################################
################*/
/*############################################################################
################*/
/*############################################################################
################*/
/*                                                                          */
/*                  FUNÇÕES DE SUPORTE A TRANSMISSÃO                        */
/*                                                                          */
/*############################################################################
################*/
/*############################################################################
################*/
/*############################################################################
################*/
/*////////////////////////////////////////////////////////////////////////////*/
/*////////////////////////////////////////////////////////////////////////////*/

/*############################################################################
################*/
/*#                                                                         #*/
/*#                FUNÇÃO PARA CALCULAR O CHKSUM                            #*/
/*#               a partir da soma dos bytes a enviar                       #*/
/*#                                                                         #*/
/*############################################################################
################*/
/*----------------------------------------------------------------------------
```

```
	-> Função simples. Checksum = 256 - ultimos 8bits da soma.
----------------------------------------------------------------------------------------*/
int calcularCHKSum(int soma) {
	int tambyte;

	for (tambyte=256; tambyte<soma; tambyte+=256);

	return (tambyte - soma - 1);
}

/*########################################################################
################*/
/*#                                                              #*/
/*#                FUNÇÃO de EVENTO de ERRO                      #*/
/*#                                                              #*/
/*########################################################################
################*/
/*----------------------------------------------------------------------------------------
	-> Função simples. Limpa o buffer de entrada. Seta o reset (cmdReset=5) e
passa o erro.
----------------------------------------------------------------------------------------*/
void leituraERRO(int erro) {
	AVR_flushIO(25);
	delay(30);
	AVR_flushIO(25);
	cmdReset=5;
	cmdERRO=erro;
	return;
}


/*////////////////////////////////////////////////////////////////////////////////*/
/*////////////////////////////////////////////////////////////////////////////////*/
/*########################################################################
################*/
/*########################################################################
################*/
/*########################################################################
################*/
/*                                                              */
/*                FUNÇÕES DE LOG COM XBEE REMOTO                */
/*                                                              */
/*########################################################################
################*/
/*########################################################################
################*/
/*########################################################################
################*/
/*////////////////////////////////////////////////////////////////////////////////*/
/*////////////////////////////////////////////////////////////////////////////////*/
```

```
/*#####################################################################
################*/
/*#                                                                   #*/
/*#                  Transmite um Reset ao XBee Base                  #*/
/*#                                                                   #*/
/*#####################################################################
################*/
/*---------------------------------------------------------------------------
        -> Reset indica ao xbee base que o modulo está no início do código,
esperando comando.
        -> O Reset está definido como (124) (125) (125) (125) (124)
---------------------------------------------------------------------------*/
void enviarReset(void) {

        AVR_flushIO(30);

        AVR_printC(126);   //0x7E
        AVR_printC(0);             //tam
        AVR_printC(10);            //tam

        AVR_printC(1);             //0x01 -> API id -> tx com 16bits
        AVR_printC(0);             //frame id (0) -> sem ack
        AVR_printC(0);             //DL 0
        AVR_printC(9);             //DL 9
        AVR_printC(1);             //options -> 1 sem ack

        AVR_printC(124);   //cmd reset (000)
        AVR_printC(125);   //125 - cmd
        AVR_printC(125);
        AVR_printC(125);
        AVR_printC(124);

        AVR_printC(133);

  return;
}


/*#####################################################################
################*/
/*#                                                                   #*/
/*#              Transmite um Reset com ERRO ao XBee Base             #*/
/*#                                                                   #*/
/*#####################################################################
################*/
/*---------------------------------------------------------------------------
        -> Reset indica ao xbee base que o modulo está no início do código,
esperando comando.
        -> O erro deve indicar em qual parte do código aconteceu o erro.
        -> O Reset com ERRO está definido como (124) (125) (125) (124) (124)
        -> O número do erro deve seguir os 5 bytes de comando.
---------------------------------------------------------------------------*/
```

```
void enviarResetERRO(void) {

	int soma;

	soma = 1+9+1+124+125+125+124+124;

	if ( (cmdERRO>0) && (cmdERRO<100) ) {
		soma+=cmdERRO;
	}
	else {  soma+=127; }
	soma=calcularCHKSum(soma);


	AVR_flushIO(30);

	AVR_printC(126);   //0x7E
	AVR_printC(0);		//tam
	AVR_printC(11);		//tam

	AVR_printC(1);		//0x01 -> API id -> tx com 16bits
	AVR_printC(0);		//frame id (0) -> sem ack
	AVR_printC(0);		//DL 0
	AVR_printC(9);		//DL 9
	AVR_printC(1);		//options -> 1 sem ack

	AVR_printC(124);   //cmd reset (001)
	AVR_printC(125);   // (125 - cmd)
	AVR_printC(125);
	AVR_printC(124);
	AVR_printC(124);

	if ( (cmdERRO>0) && (cmdERRO<100) ) {
		AVR_printC(cmdERRO);
	}
	else {
		AVR_printC(127);
	}

	AVR_printC(soma);

PORTC &= ~(1<<5);

	cmdReset=0;		//ZERA O COMANDO DE RESET
	cmdERRO=127;


   return;
}
```

Anexo VI

Este anexo descreve detalhadamente as alterações propostas para o circuito do módulo sensor, descrito no item 4.1.1. As modificações foram idealizadas visando a simplificação do circuito e a minimização do consumo de energia em certos pontos do circuito onde esta redução ainda era ineficiente.

O *hardware* do módulo mantém os mesmos elementos básicos: o módulo de comunicação XBee e o sensor de temperatura LM35 com um controlador proporcional de ganho K_p igual 6,8. Entretanto, modificam-se algumas resistências do circuito e substitui-se o bloco do circuito responsável por realizar a saturação após a saída do controlador proporcional. Estas alterações são apresentadas no esquemático da Figura VI.1.

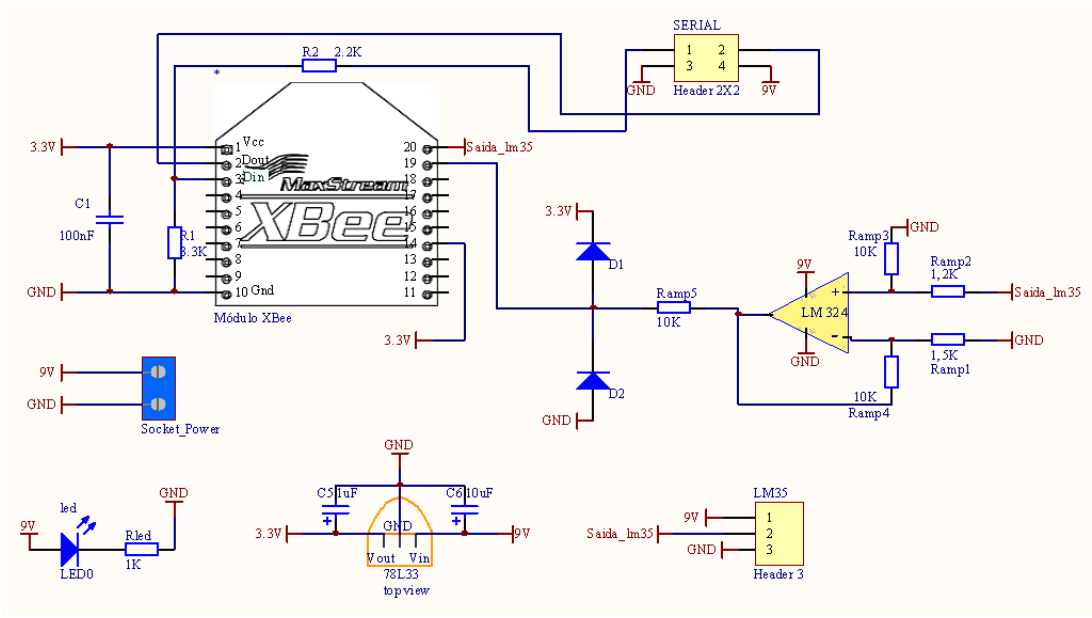


Figura VI.1 – Esquemático do módulo sensor após modificações propostas.

Observa-se do esquemático que as resistências R_{amp5} e R_{led} foram incrementadas de 1KΩ para 10KΩ e de 330Ω para 1KΩ, respectivamente. Estas modificações foram propostas objetivando a redução da potência dissipada nestas resistências. O bloco saturador, que antes utilizava dois amplificadores operacionais, foi substituído por um par de diodos de proteção que, apesar de não atuarem com a mesma precisão do circuito anterior, é capaz de desempenhar a saturação com um consumo de energia inferior ao consumo do bloco saturador.