UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

ALYSSON HIKARU SHIRAI

ESTUDO E IMPLEMENTAÇÃO DE SISTEMAS DE LOCALIZAÇÃO EM HARDWARE DE LÓGICA PROGRAMÁVEL PARA UTILIZAÇÃO EM REDE DE SENSORES SEM FIO

DISSERTAÇÃO

CURITIBA
2013

ALYSSON HIKARU SHIRAI

ESTUDO E IMPLEMENTAÇÃO DE SISTEMAS DE LOCALIZAÇÃO EM HARDWARE DE LÓGICA PROGRAMÁVEL PARA UTILIZAÇÃO EM REDE DE SENSORES SEM FIO

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do grau de "Mestre em Ciências" - Área de Concentração: Engenharia de Automação e Sistemas.

Orientador: Prof. Dr. Volnei Antonio Pedroni

CURITIBA
2013

Dados Internacionais de Catalogação na Publicação

S558 Shirai, Alysson Hikaru
Estudo e implementação de sistemas de localização em hardware de lógica programável para utilização em rede de sensores sem fio / Alysson Hikaru Shirai. – 2013.
133 f. : il. ; 30 cm

Orientador: Volnei Antonio Pedroni.
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2013.
Bibliografia: f. 100-107.

1. Algoritmos. 2. Problemas de localização (Programação). 3. Redes de sensores sem fio. 4. Sistemas de comunicação sem fio. 5. Arranjos de lógica programável em campo. 6. Métodos de simulação. 7. Engenharia elétrica – Dissertações. I. Pedroni, Volnei Antonio, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD (22. ed.) 621.3

Biblioteca Central da UTFPR, Câmpus Curitiba

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

Título da Dissertação Nº. 629

"Estudo e Implementação de Sistemas de Localização em *Hardware* de Lógica Programável para Utilização em Rede de Sensores sem Fio."

por

Alysson Hikaru Shirai

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: Engenharia de Automação e Sistemas, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Curitiba, às 10h do dia 22 de fevereiro de 2013. O trabalho foi aprovado pela Banca Examinadora, composta pelos doutores:

Prof. Volnei Antonio Pedroni, Dr.
(Presidente – UTFPR)

Prof. Marcelo Eduardo Pellenz, Dr.
(PUC-PR)

Prof. Carlos Raimundo Erig Lima, Dr.
(UTFPR)

Visto da coordenação:

Prof. Ricardo Lüders, Dr.
(Coordenador do CPGEI)

“Se enxerguei mais longe,

foi porque me apoiei em ombros de gigantes”.

(Sir Issac Newton)

AGRADECIMENTOS

A minha família, por todo o apoio e incentivo que recebi durante toda a minha vida e por me encorajar a vencer todos os desafios.

Ao meu orientador, Professor Dr. Volnei Pedroni, pela sabedoria, atenção, paciência e, principalmente, pela confiança depositada desde o início deste trabalho.

Ao Ricardo Jasinski, por toda a sua experiência e contribuições que melhoraram muito este trabalho.

Aos colegas do laboratório LME, pela amizade, ajuda e companhia em todos os momentos que estivemos juntos.

A todos os professores e colegas do mestrado, pelos momentos de estudo, pelo compartilhamento de seus conhecimentos e pelas valiosas discussões que me fizeram aprender mais.

A UTFPR, pelas instalações e os materiais disponibilizados durante o desenvolvimento desta dissertação de mestrado.

A CAPES, pela bolsa de estudos e oportunidade concedida.

RESUMO

SHIRAI, Alysson Hikaru. Estudo e Implementação de Sistemas de Localização em Hardware de Lógica Programável para Utilização em Rede de Sensores sem Fio. 133 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

Redes de sensores sem fio (RSSF) têm sido tema central de diversos estudos na atualidade. Em certas aplicações, como, por exemplo, as que necessitam saber de onde os dados estão sendo enviados ou em casos em que o próprio nó sensor precisa saber sua posição para executar alguma ação, mecanismos de localização se tornam imprescindíveis. Porém, a execução deste tipo de algoritmo é custosa para os nós sensores. Concomitantemente, o advento das *low power* FPGAs têm viabilizado a aplicação de dispositivos programáveis em RSSFs e aplicações envolvendo reconfiguração dinâmica de FPGA em nós sensores têm aumentado o uso destes dispositivos nestas redes. Unindo-se estas demandas, o objetivo desta dissertação é estudar e implementar sistemas de localização em hardware de lógica programável, visando atender aplicações voltadas a RSSF. Utilizando-se no nó sensor um bloco de hardware dedicado para realizar os cálculos de posição minimiza a utilização de seu CPU, podendo este hardware, inclusive, ser apenas uma parte de um sistema maior implementado na FPGA. O processo de localização baseia-se na utilização das distâncias entre o nó de posição desconhecida e os nós de referência, determinadas através de medição de RSSI, e o uso de algoritmos específicos que calculam a posição desejada. As principais etapas foram: revisão da literatura, modelagem do comportamento das medições de RSSI, análise do desempenho dos algoritmos e projeto de hardware. Através das simulações realizadas pôde-se desenvolver metodologias e ferramentas para a geração otimizada do hardware de localização. O desenvolvimento deste trabalho possibilitou analisar a aplicabilidade do ponto flutuante e ponto fixo, definir a arquitetura adequada para o hardware e o dimensionamento adequado da quantidade de bits necessária nas implementações.

Palavras-chave: Algoritmos de localização, Rede de sensores sem fio, Lógica reconfigurável

ABSTRACT

SHIRAI, Alysson Hikaru. Study and Implementation of Locating Systems in Programmable Logic Hardware to Use in Wireless Sensor Network. 133 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

Wireless sensor networks (WSN) have been the central theme of many researches in actuality. In certain applications, like, for example, the ones that need to know from where the data is being sent or in cases which the sensor node need to know its own position to perform some action, location mechanism is indispensable. However, the execution of these algorithms is costly for the sensor nodes. Concomitantly, the advent of low power FPGAs made feasible the application of programmable devices in WSNs and applications involving dynamic reconfiguration of FPGA in sensor nodes increased the use of these devices in WSNs. Joining these demands, the goal of this master thesis is to study and implement locating systems in programmable logic hardware, aiming at meeting applications in WSN. Employing a dedicated hardware block in sensor node to compute the position minimizes its CPU usage, and this hardware can even be just a part of a larger system implemented in FPGA. The localization process is based on the use of distances, measured between the sensor node with unknown position and the reference nodes, determined from RSSI measurements, and the use of specific algorithms that calculate the desired position. The main steps were: review of the literature, modeling the behavior of the RSSI measurements, performance analysis of the algorithms and hardware design. Through the performed simulations it was possible to develop methodologies and tools to generate optimized locating hardware. The development of this work allowed to evaluate the feasibility of the floating point and fixed point, to set the appropriate architecture for the hardware and to find the proper dimension of the number of bits required in the implementations.

Keywords: Locating algorithms, Wireless sensor network, Reconfigurable logic

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AOA	*Angle-of-arrival*
ASIC	*Application-Specific Integrated Circuit*
CPLD	*Complex Programmable Logic Devices*
CRB	Cramér-Rao *Bound*
FDA	Função Distribuição Acumulada
FSM	*Finite State Machine*
FPGA	*Field Programmable Gate Array*
GAL	*General Array Logic*
GPS	*Global Positioning System*
HDL	*Hardware Description Language*
IEEE	*Institute of Electrical and Electronics Engineers*
I/O	*Input/Output*
ISM	Industrial, Científico e Médico
MLE	*Maximum Likelihood Estimation*
NaN	*Not a Number*
NPD	Nó de posição desconhecida
NR	Nó de referência
PAL	*Programmable Array of Logic*
PWM	*Pulse-Width Modulation*
QDSP	*Quantized DSP Simulation Toolbox*
RF	Radiofrequência
RSS	*Received Signal Strength*
RSSF	Rede de Sensor sem Fio
RSSI	*Received Signal Strength Indication*
SOC	Sistema-em-um-Chip
SPLD	*Simple Programmable Logic Device*
TDOA	*Time-difference-of-arrival*
TOA	*Time-of-arrival*
UART	*Universal Asynchronous Receiver/Transmitter*
UWB	*Ultra-wideband*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÕES

Atualmente, Redes de Sensores sem Fio (RSSF) têm sido tema central de diversos estudos. O avanço das tecnologias de comunicação sem fio, assim como do processo de fabricação de sensores, tem contribuído para a construção de dispositivos cada vez menores, com consumo de energia e preços também menores, viabilizando uma enorme gama de aplicações (NAKAMURA et al., 2007; IYENGAR et al., 2011).

Em muitas destas aplicações existe a necessidade de se saber a origem das informações, ou seja, de onde os dados estão sendo enviados. Em outras aplicações, o próprio nó sensor precisa saber sua posição para executar alguma ação. Nestes cenários, mecanismos de localização são imprescindíveis. Alguns exemplos de aplicações onde informações de localização são importantes: agricultura de precisão, monitoramento ambiental, navegação autônoma, rastreamento de objetos, operações militares, entre outros (MAO et al., 2007; CHRAIBI, 2005; LUTHY, 2009).

A figura 1 apresenta um breve resumo das diversas técnicas de localização que têm sido utilizadas ao longo do tempo. A necessidade de se localizar é antiga e as primeiras soluções utilizando sextantes surgiram antes de 1950, para utilização em navegações por terra e ar. Pouco depois surge a necessidade de rastrear os satélites no espaço. Nesta época as bases terrestres eram usadas como referência na determinação da posição dos satélites. Em torno da década de 80 surge o GPS (*Global Positioning System*) e, desta vez, os satélites são usados como referências para o cálculo da posição de um ponto na superfície da terra. Recentemente, surgiram também aplicações de localização em redes *ad-hoc* e RSSF, que é o foco desta pesquisa (REGHELIN, 2007).

Entre as técnicas de localização tradicionais, duas delas podem ser destacadas pela sua grande utilização: o GPS e o sistema de localização da Rede Móvel Celular (*Cell Based Positioning*). Entretanto, nenhuma dessas duas técnicas é viável em aplicações de RSSF. O GPS requer a utilização de receptores que consomem muita energia e pode também apresentar

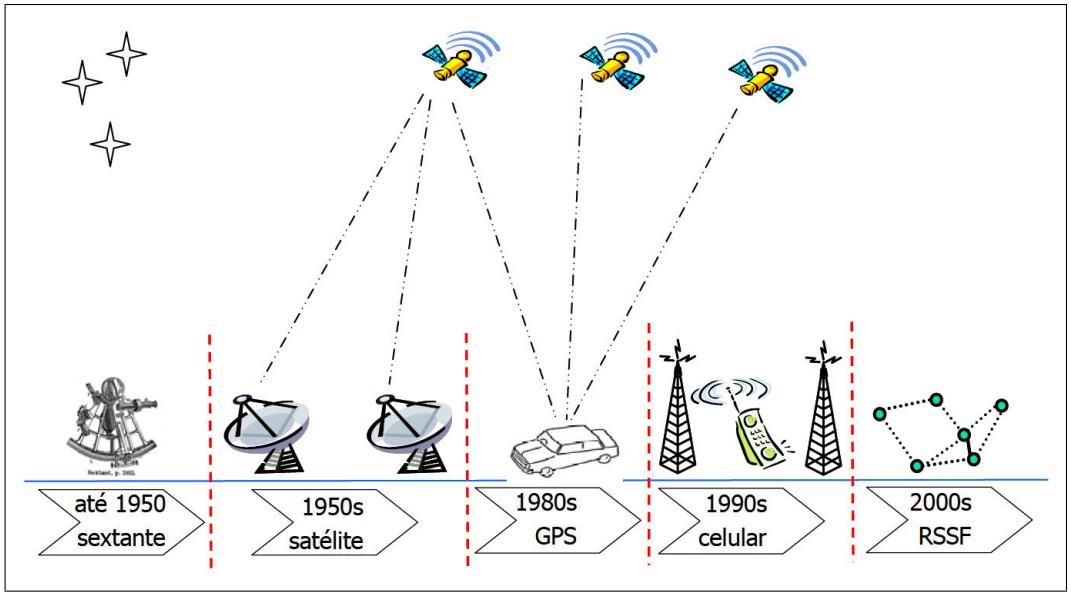

Figura 1: Utilização dos sistemas de localização

Fonte: Reghelin (2007)

dimensões grandes demais para a maioria das aplicações de RSSF; além disso, o sinal de GPS pode não estar disponível em ambientes fechados, como no interior de indústrias e escritórios. Outro fator agravante é o custo de se ter este dispositivo instalado em cada nó sensor da rede. Já a localização realizada por rede móvel celular, que é feita através da triangulação do sinal das estações rádio base, é inviável em aplicações onde não se tem nenhuma estação base na proximidade (TREVISAN, 2009; MAO et al., 2007).

Analisando as técnicas de localização aplicáveis em RSSF, muito estudo tem sido feito com o emprego de indicadores de intensidade do sinal de rádio recebido na localização, o que é interessante em RSSF, visto que, em geral, nenhum hardware adicional é necessário para a sua implementação, pois muitos dos chips destinados aos protocolos que utilizam este parâmetro já possuem hardware específico para tal. Muitos protocolos disponibilizam o valor da intensidade do sinal recebido através de um indicador denominado RSSI (*Received Signal Strength Indication*). Os protocolos IEEE 802.11 (*Institute of Electrical and Electronics Engineers* - IEEE) utilizam este indicador, por exemplo, para definir políticas de associação (WANG et al., 2009), e o IEEE 802.15.4 utiliza este indicador para verificar a qualidade do enlace de comunicação sem fio (LEE et al., 2010).

Pode-se, a partir de medições de RSSI, estimar os parâmetros do canal de comunicação sem fio e obter um modelo de canal que relaciona a potência do sinal recebido com a distância entre o nó transmissor e receptor. Juntamente com este modelo, são necessários também nós de referência, com posição conhecida, para que a localização de um nó na rede seja possível.

Com todos estes dados, pode-se, então, utilizar algoritmos para estimar a posição do nó de interesse. Um exemplo desta metodologia de localização está ilustrado na figura 2. Embora pareça simples, realizar todas estas etapas exige muito processamento por parte do nó sensor móvel, que na maioria das vezes apresenta pouco poder de processamento. Assim, devido a esta limitação, aliado ao seu consumo restrito de energia, muitas abordagens sofisticadas de localização são inviabilizadas em aplicações de RSSF.

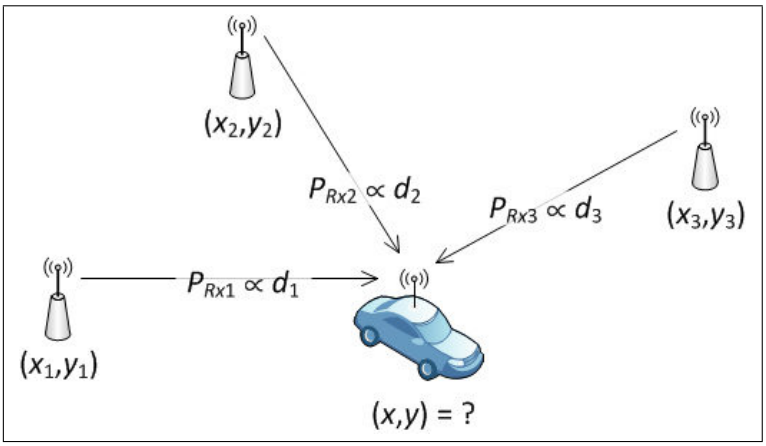

Figura 2: Exemplo de aplicação dos algoritmos de localização

Fonte: Shirai et al. (2012)

Costuma-se, em muitas situações, empregar sistemas centralizados para realizar a localização, visto que, em tese, o elemento que centraliza as informações não possui as restrições citadas há pouco. Entretanto, esta abordagem faz com que o tráfego de mensagens na rede aumente e isto pode ser indesejável em redes com grande densidade de dispositivos sensores (CHEN et al., 2009).

Uma solução seria utilizar no nó sensor um bloco de hardware dedicado para realizar os cálculos de posição. Este bloco, agindo como um coprocessador, minimizaria ao máximo a necessidade de intervenção do CPU no processo de localização. Seria uma situação semelhante aos mecanismos de criptografia, que exigem considerável processamento e que costumam ser realizados por hardware coprocessadores, como no caso do chip CC2431 (OTTOY et al., 2010; TEXAS INSTRUMENTS, 2009).

A partir das pesquisas realizadas, foi constatado que esta área de pesquisa ainda foi pouco explorada, tanto na literatura quanto comercialmente. Muitos algoritmos são propostos, mas poucos trabalhos focam o estudo destas implementações em hardware. Já no aspecto comercial, existe, praticamente, apenas um chip de nó sensor disponível no mercado que calcula a sua posição em hardware (chip CC2431). Além disso, são soluções cara se comparados aos nós sensores tradicionais.

Outro fato que motivou o desenvolvimento desta dissertação é o advento das *low power* FPGAs (*Field Programmable Gate Array*), que têm viabilizado a aplicação de dispositivos programáveis em RSSFs. Diversas arquiteturas de nós sensores têm sido implementadas, sendo elas, basicamente, implementadas somente com FPGAs e outras implementadas juntamente com microcontroladores tradicionais (ROSELLO et al., 2011; DE LA PIEDRA et al., 2012). Além disso, as recentes pesquisas na área de computação reconfigurável têm contribuído para o aumento de estudos de aplicações envolvendo reconfiguração dinâmica de FPGA em RSSF (CHAROENPANYASAK; SUNTIAMORNTUT, 2011). A utilização de FPGA, dada a sua flexibilidade, permite que um hardware possa ser estudado e testado antes de sua implementação em ASICs (*Application-Specific Integrated Circuit*). Além do mais, usando-se uma FPGA, o sistema de localização pode ser apenas uma parte de um sistema maior.

Neste contexto, motivado pelas constatações comentadas, este trabalho tem por objetivo estudar e implementar sistemas de localização em hardware de lógica programável, focando nos algoritmos de localização bidimensional (2D) que permitam ao nó sensor determinar a sua posição através de informações obtidas dos nós de referência. Além disso, serão pesquisados algoritmos distribuídos e de salto simples. Desta forma, nem algoritmos centralizados nem algoritmos baseados em múltiplos saltos serão enfatizados neste trabalho.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

O objetivo geral deste trabalho é estudar e implementar sistemas de localização em hardware de lógica programável, especialmente FPGA, visando aplicações em redes de sensores sem fio.

1.2.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são:

- Estudar os métodos de localização disponíveis na literatura, focando em métodos que permitam a um nó sensor descobrir a sua posição, assim como analisar quais destes métodos são viáveis em RSSF.
- Verificar o erro médio de posição dos algoritmos e verificar como este erro varia com a quantidade de nós de referência.

- Aprofundar os estudos nos algoritmos que sejam viáveis para implementação em dispositivos de lógica programável.
- Avaliar o impacto no sistema de localização devido ao tipo de dado empregado nos cálculos, analisando sua precisão e demanda de hardware.
- Analisar a quantidade de hardware necessária nas implementações e verificar quais fatores influenciam no tamanho final do sistema.

1.3 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada em cinco capítulos. No capítulo 2 está apresentada uma revisão da literatura sobre os métodos de localização utilizados em RSSF, assim como implementações de sistemas de localização em hardware. O capítulo 3 descreve o desenvolvimento desde trabalho. O capítulo 4 apresenta os resultados obtidos. E, por fim, no Capítulo 5 estão presentes as discussões dos resultados, as conclusões, publicação e as propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTRODUÇÃO

Este capítulo tem por objetivo apresentar a fundamentação teórica deste trabalho, mostrando a revisão da literatura acerca dos algoritmos de localização e sua implementação em hardware, assim como apresentar os aspectos principais dos tópicos relacionados com este trabalho, tais como os conceitos de comunicação sem fio e Redes de sensores sem fio, técnicas de localização, algoritmos de localização e uma introdução a respeito dos dispositivos programáveis.

2.2 REVISÃO DA LITERATURA

Na literatura existem trabalhos correlatos com o foco desta pesquisa. Muitos dos trabalhos relacionados a sistemas de localização se encontram no escopo da robótica, sistemas militares e serviços de telefonia móvel (CHENG et al., 2011; CHRAIBI, 2005; GUEDES, 2003). Muitas dessas ideias podem ser estendidas também à área de RSSF. Entretanto, visto que a capacidade de processamento dos nós sensores é pequena se comparado a destes outros sistemas, muitos destes métodos podem ser inviáveis no contexto das RSSF.

A localização de um dispositivo na rede baseia-se, em geral, na determinação da posição de um nó a partir de dados conhecidos, tal como a posição absoluta de outros nós sensores. Na literatura, os nós sensores que sabem sua posição costuma ser denominado *anchor node* ou *beacon*, que doravante será denominado "nó de referência", ou simplesmente NR. Por outro lado, o nó sensor que não possui conhecimento de sua posição, ou aquele que se deseja determinar a sua posição, costuma ter denominações como *blind node* ou *target node*. Neste trabalho, estes nós serão mencionados como "nós de posição desconhecida" ou NPD. Como comentado, os NRs possuem posições conhecidas, sendo estas relativas a referências globais (obtidas via GPS, por exemplo) ou, então, relativas a sistemas de referência locais. A última situação é mais interessante, visto que em muitos casos não é possível utilizar dispositivos GPS,

principalmente em ambientes internos, sem visada com satélites (MAO et al., 2007).

O processo de localização pode ser realizado através de diversas técnicas de medição, sendo as mais usuais a medição do ângulo de chegada (*Angle-of-Arrival* - AOA), do tempo de chegada (*Time-of-Arrival* - TOA), da diferença no tempo de chegada (*Time Difference-of-Arrival* - TDOA) e da intensidade do sinal recebido (*Received Signal Strength* - RSS). Este trabalho foca apenas na utilização de informações de intensidade do sinal recebido. A partir destes dados é possível determinar a distância entre o NPD e NR e, posteriormente, a posição do NPD.

A localização utilizando apenas informações de distância pode ser resolvida através de lateração. Do ponto de vista geométrico, a posição de um nó sensor pode ser entendida como a interseção de circunferências, onde cada nó de referência gera uma circunferência com raio igual a distância entre ele e o nó a ser localizado. Para o caso de duas dimensões, a posição de um nó é determinada unicamente quando existem ao menos três nós de referência não colineares. Para o caso tridimensional, ao menos quatro nós de referência são necessários e estes devem ser não-coplanares (HIGHTOWER; BORRIELLO, 2001). Contudo, devido às variações no canal de comunicação e no ambiente de propagação, os ruídos e erros de medição fazem com que as circunferências não se interceptem em um único ponto, mas criando uma área onde o NPD provavelmente estará (ZANCA et al., 2008).

Na prática, costuma-se utilizar mais nós de referência do que os citados, com o intuito de amenizar os efeitos das interferências, como o sombreamento, que ocorrem frequentemente na comunicação sem fio. No caso de duas dimensões, quando mais do que três nós de referência são considerados, forma-se um sistema de equações superdeterminado. Nestes casos costuma-se utilizar o método dos mínimos quadrados para estimar as coordenadas que minimizem o erro quadrático das distâncias medidas. Contudo, a otimização por este método, como mostrado em Chen et al. (2009), exige o cálculo da inversa de uma matriz, além de uma série de multiplicações de matrizes, o que exige bastante processamento. É importante mencionar que o processo de inversão de matriz, assim como sua implementação em hardware, é uma tarefa complexa e, ainda hoje, têm sido tema de muitos estudos (GARCIA, 2010).

Neste contexto, uma alternativa para o método anterior é o método Min-Max (*Min-Max Method*), apresentado em Savvides et al. (2002). Neste método as circunferências são aproximadas por quadrados e a posição do NPD é determinada como sendo o centroide da área sobreposta por todos os quadrados. Como mostrado em Shirai et al. (2012), este método é interessante para implementação em hardware, visto que apenas operações simples estão envolvidas e, assim, menos elementos lógicos são necessários para a sua sintetização. A desvantagem,

como mostrado no mesmo trabalho, é que o erro médio apresentado pode ser considerável, dada a aproximação das circunferências por quadrados.

Em outra linha de pesquisa, diferentes algoritmos têm sido propostos tentando modelar o processo de localização estatisticamente. Entre estes algoritmos está o Método do Estimador da Máxima Verossimilhança (*Maximum Likelihood Estimation* - MLE). Nesta abordagem assume-se que as medições de RSS são modeladas como variáveis aleatórias do tipo Log-Normal ou distribuição exponencial. Assim, baseando-se nestes modelos, pode-se obter a função de verossimilhança e, então, formular o processo de localização através do Método de Máxima Verossimilhança, onde se encontra o valor do parâmetro que maximize a função verossimilhança ou, então, minimize o negativo da função log-verossimilhança. Estas formulações podem ser encontradas em Patwari et al. (2001). Os experimentos e comparações feitas em Zanca et al. (2008) mostram que o MLE apresenta resultados melhores que o algoritmo baseado em lateração e método Min-Max. Entretanto, o MLE é mais complexo que os outros dois métodos, pois é necessário maximizar funções (ou minimizar, se for tomado o negativo do logaritmo das funções). Nesse sentido, a utilização de algoritmos de otimização numérica se torna necessária. Entre estes podem-se citar o Método de Nelder-Mead, o qual é um método de busca direto e que não requer o cálculo de derivadas de funções (MATHEWS; FINK, 2004). Outra solução seria calcular a derivada das funções e utilizar algoritmos de resolução de sistemas de equações não lineares. Entre estes se destacam: Método de Newton, Método da Secante (Quasi-Newton) e Método do Gradiente Conjugado.

Outras abordagens analisam estatisticamente apenas as estimativas de distâncias. Uma destas abordagens é a utilização do determinante de Cayley-Menger. A ideia deste determinante é explorar as relações geométricas e algébricas das distâncias entre os nós sensores, permitindo obter relações quadráticas que relacionam as distâncias verdadeiras com as distâncias obtidas através das medições. Com estas relações, busca-se estimar o conjunto de erros associados na medição de distâncias que, quando compensados, produzam valores de distâncias as mais próximas possíveis das verdadeiras distâncias. A solução deste problema de otimização corresponde ao conjunto de distâncias que sejam consistentes com o fato de que os nós sensores estejam no mesmo plano, para o caso bidimensional, ou no mesmo espaço, no caso tridimensional (CAO et al., 2006).

O filtro de Kalman é outra alternativa para amenizar os efeitos dos ruídos nas medições de distância. Trata-se de um filtro estatístico que faz estimativas do estado interno de um sistema dinâmico utilizando uma série de medidas da saída do sistema, sendo que estas medidas estão corrompidas por ruídos e outras incertezas (PARANHOS, 2009). O filtro de Kalman tem sido

largamente aplicado em problemas de rastreamentos (KOKS, 2007). Este filtro pode ser escrito em uma única equação, porém, ele é geralmente descrito em duas fases distintas: Predição e Atualização. Na fase de predição um estimador é construído, baseando-se nos dados do passo anterior, para prever a distribuição do vetor de estado um passo a frente. Esta predição é conhecida como estimativa "*a priori*" e não inclui a informação vinda da observação do estado atual. Na fase de atualização, a predição "a priori" e a observação atual são utilizadas para obter uma estimativa refinada para o estado atual. A estimativa refinada é também denominada estimativa "*a posteriori*". Assim, o processo se repete com a estimativa "*a posteriori*" sendo utilizada para predizer a nova estimativa "*a priori*" (KARTHICK et al., 2009; VIJ; MEHRA, 2011). Este filtro, dada a sua robustez, tem sido aplicado também na área de RSSF e Vij e Mehra (2011) comenta, inclusive, sobre a implementação deste filtro em FPGA. Este mesmo trabalho cita também algumas variantes deste filtro, como o EKF (*Extended Kalman Filter*) e o UKF (*Unscented Kalman Filter*), que são utilizadas quando a dinâmica do sistema é não linear.

Um exemplo de sistema de localização implementado em hardware é o chip CC2431 da Texas Instruments, que é um sistema-em-um-chip (*System-on-a-Chip* - SOC) designado para aplicações em redes ZigBee e IEEE 802.15.4. Este chip integra, basicamente, um transceiver RF, MCU de arquitetura 8051, memória FLASH e RAM, e um hardware de localização dedicado, denominado *Location Engine*. O *Location Engine* estima a localização bidimensional do NPD através de um algoritmo distribuído baseado em informações de RSSI, dos sinais obtidos dos NR. Este algoritmo utiliza de 3 a 16 NRs nos cálculos, necessitando de 50μs a 13ms (com *clock* do sistema de 32 MHz) para estimar a localização do NPD. A especificação das posições dos NRs é feita utilizando estrutura de ponto fixo sem sinal de 8 bits, sendo a parte fracionária representada pelos 2 bits menos significativos e a parte inteira pelos outros 6 bits. Assim, os NRs podem estar distribuídos em uma área de até, aproximadamente, 64×64 metros, sendo de 0,25 metro a resolução de suas coordenadas (AAMODT, 2006; TEXAS INSTRUMENTS, 2009).

Este sistema implementa um algoritmo proprietário baseado no Método do Estimador de Máxima Verossimilhança (CHIPCON, 2005). Como apresentado em (TEXAS INSTRUMENTS, 2009), este sistema requer os seguintes parâmetros de entrada para realizar a localização:

- **Definição da área de busca:** Com o intuito de reduzir o erro e acelerar o cálculo da posição, deve-se definir a área retangular que mais se aproxima da posição do NPD. Os valores limites das coordenadas x e y deve estar na faixa de 0 a 63,75 metros. Deve-se sempre especificar a área de busca, mesmo quando não se sabe a estimativa da área de

busca. Nesta situação, deve-se especificar a área máxima, ou seja, a área formada pelo quadrado de vértices: (0;0), (0;63,75), (63,75;0) e (63,75;63,75).

- **Coordenadas de referência:** As coordenadas de referência $[x_0, y_0, x_1, y_1, ..., x_{15}, y_{15}]$ expressam a posição dos respectivos NRs. Como comentado há pouco, a representação é feita utilizando estrutura de ponto fixo sem sinal de 8 bits, podendo utilizar valores de 0 a 63,75 com resolução de 0,25 metro. Deve-se sempre especificar 16 pares de coordenadas, sendo que quando se deseja utilizar menos NRs, recomenda-se escrever 0.0 no registrador em questão e no respectivo registrador de RSSI.

- **Valores de RSSI:** Estes valores representam os valores de RSSI medidos a partir dos respectivos NRs. Os valores aceitos devem estar na faixa de $[-40dBm; -95dBm]$, com precisão de 0,5 dBm. Para a escrita no registrador o sinal negativo é removido. Deve-se sempre especificar 16 valores de RSSI correspondentes aos NRs, sendo que o valor 0.0 deve ser especificado nos registradores dos NRs não utilizados.

- **Parâmetros do ambiente de propagação:** O valor de RSSI é função da potência de transmissão e da distância entre o transmissor e o receptor. A formulação considerada para o RSSI é dada pela equação 1. Esta equação baseia-se no modelo *Log-distance*, onde n representa o expoente de perda de percurso e A a potência recebida na distância de referência. Os valores de n e A devem ser estimados empiricamente e são também parâmetros de entradas para o cálculo da posição.

$$RSSI = -[10.n.log(d) + A] \tag{1}$$

Outro trabalho envolvendo a implementação de sistemas de localização em hardware, desta vez especificamente em FPGA, é apresentado em (BUCHER; MISRA, 2002). Neste artigo os autores implementaram um algoritmo em VHDL que computa a posição tridimensional de um móvel, dadas as posições de quatro referências fixas e as respectivas medidas de TOA entre o móvel e os pontos de referência. Os pontos de referência considerados são os satélites GPS. As informações de tempo de chegada devem ser fornecidas em nanosegundo e o próprio algoritmo as converte em informações de distância. Todos os cálculos são feitos com representações binárias, assim, para que a precisão dos cálculos não seja comprometida, multiplicações adicionais são realizadas após as operações de divisão. Neste contexto, até mesmo vetores com 200 bits são utilizados neste modelo, devido às sucessivas operações de multiplicação. O mesmo algoritmo foi implementado em C++ para comparar os resultados de ambas metodologias. Este

trabalho mostra que o modelo VHDL implementado apresentou os mesmos resultados do algoritmo em C++ implementado com ponto flutuante de 32 bits.

2.3 REDES DE SENSORES SEM FIO

Atualmente, as Redes de Sensores sem Fio (RSSF) têm despertado o interesse em diversas áreas de estudo. O avanço das tecnologias de comunicação sem fio, assim como do processo de fabricação de sensores, tem contribuído para a construção de dispositivos cada vez menores, com consumo de energia e preços também menores, podendo-se então englobar, em um único chip, vários sensores, assim como um processador e uma interface de comunicação sem fio, viabilizando uma enorme gama de aplicações. Cada unidade desta costuma ser denominado de nó sensor ou nodo (NAKAMURA et al., 2007; LOUREIRO et al., 2003).

Uma RSSF pode ser entendida como um tipo especial de redes *ad-hoc*, composta por um grande número de nodos sensores equipados com diferentes sensores (NAKAMURA et al., 2007). As RSSFs diferem das redes de computadores tradicionais em diversos aspectos (BASAGNI et al., 2008; LOUREIRO et al., 2003):

- Normalmente, RSSFs possuem um grande volume de nodos distribuídos (da ordem de milhares, dependendo da aplicação).
- Têm limitações de recursos, tanto de energia quanto de memória e capacidade de processamento.
- Diferentemente de uma rede *ad-hoc* tradicional, a qual, tipicamente, troca dados através de comunicação *peer-to-peer*, em uma RSSF a ênfase da comunicação está no transporte de dados de um nó sensor para um nodo especial coletor de dados, conhecidos como *sinks*. Normalmente, quando um evento de interesse é detectado pelo nó sensor, este cria um pacote de dado e o envia para o *sink*. A comunicação pode ocorrer através de múltiplos saltos, ou seja, com os nodos vizinhos atuando como roteadores.
- RSSF deve possuir mecanismos que permitam auto-configuração e adaptação devido às possíveis falhas de comunicação e perda de nodos que possam ocorrer. Nesse sentido, uma RSSF tende a ser autônoma, necessitando alto grau de cooperação para executar as tarefas concebidas para a rede.

A figura 3 exemplifica uma RSSF com os seus principais componente. Nesta figura está ilustrado também o *Gateway* que, diferentemente dos outros nodos, é um componente que faz a interface da rede de sensores com outras redes, como a internet (LOUREIRO et al., 2003).

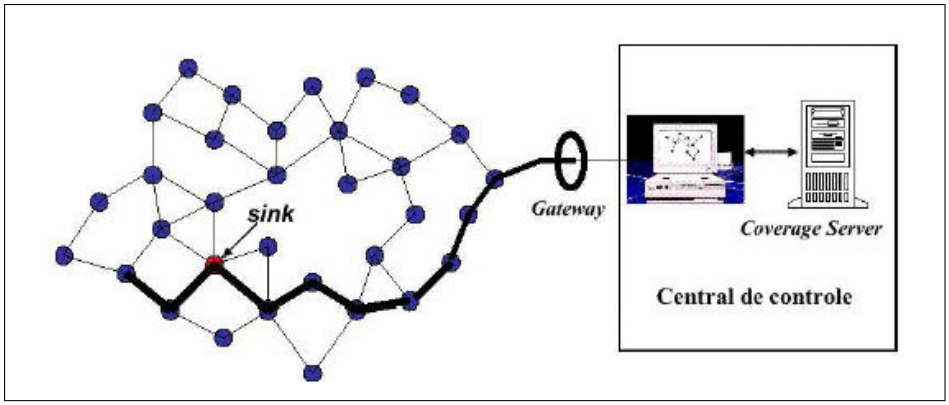

Figura 3: Ilustração de uma RSSF

Fonte: Loureiro et al. (2003)

Entre as diversas aplicações das RSSFs podem-se mencionar as seguintes áreas (IYENGAR et al., 2011; ARAMPATZIS et al., 2005; BASAGNI et al., 2008; LOUREIRO et al., 2003):

- Militar: Na coleta de informações de ambientes, rastreamento de inimigos, vigilância de territórios, entre outros.
- Monitoramento ambiental: Aplicações na agricultura, acompanhamento climático, acompanhamento da variação de umidade, pressão, temperatura de ambientes, entre outras aplicações.
- Suporte em logística: Gerenciamento de mercadorias e equipamentos, segurança de centros comerciais e estacionamentos, entre outras aplicações.
- Medicina/Biologia: No monitoramento do funcionamento de órgãos, como o coração, detecção da presença de substâncias que indicam o surgimento de um problema biológico, detecção da presença de substâncias específicas no corpo, entre outros monitoramentos.
- Robótica: Utilização das informações obtidas da rede de sensores para a tomada de decisão de robôs autônomos, sendo isto aplicado em diversas áreas.

Muitas das RSSFs monitoram apenas fenômenos físicos como temperatura, pressão e umidade. Em geral, estas redes são projetadas para aplicações que toleram certo atraso e que não exijam muita banda de comunicação. Nesse sentido, muitas pesquisas têm focado redes com estas características. Atualmente, porém, com o avanço das pesquisas surgiram novos paradigmas de redes de sensores. Vuran et al. (2009) apresenta as recentes evoluções das redes de sensores e aponta as novas aplicações que surgiram. Estes autores apresentam também

alguns dos novos paradigmas de redes de sensores que têm sido pesquisados, tais como: Redes de Sensores Multimídia sem Fios (*Wireless Multimedia Sensor Networks*), Redes Acústicas de Sensores Subaquáticos (*Underwater Acoustic Sensor Networks*) e Redes sem Fio de Sensores Subterrâneos (*Wireless Underground Sensor Networks*).

2.4 TÉCNICAS DE LOCALIZAÇÃO ATRAVÉS DE SINAIS DE RF

Existem diversas técnicas que utilizam sinais de radiofrequência (RF) para fazer a localização. Nesta seção será dado enfoque maior à técnica baseada em medição de potência, que é um dos pilares deste trabalho.

Pode-se afirmar, contudo, que os métodos baseados em AOA são mais precisos que os baseados em medições de potência recebida, embora os custos envolvidos sejam maiores para o AOA (MAO et al., 2007; PATWARI et al., 2001).

Analisando-se a precisão das medições através do limite de Cramér-Rao (Cramér-Rao *Bound* - CRB), mostra-se também que as estimativas baseadas em RSS tendem a ser piores que outras técnicas de localização como os baseados em AOA e TOA (PATWARI et al., 2005a).

O uso de sinais de bandas ultralargas (*Ultra-wideband* - UWB) também têm sido utilizados para a determinação de distância, visto que são pouco sensíveis ao efeito de multipercurso e espalhamento. É um método que apresenta bom desempenho, porém, ao custo de um consumo maior de energia (ANGELIS et al., 2009; SHIMIZU; SANADA, 2003).

Existe também na literatura classes de algoritmos híbridos que utilizam diferentes formas de medições para realizar a estimativa da localização. Esta metodologia baseia-se no fato de que utilizar informações de diferentes formas de medição pode aumentar o desempenho final, dado que os erros inerentes de cada medição surgem de diferentes fontes. A independência entre as diferentes medições pode ser utilizada no desenvolvimento de estimadores melhores do que os baseados em apenas um tipo de medição. Entre estas técnicas híbridas, a fusão entre medições de TOA e RSS parecem ser interessantes em redes de sensores sem fio, dada a simplicidade do *hardware* requerido (MAO et al., 2007).

A seguir serão discutidas as seguintes técnicas de localização: pelo ângulo de chegada (*Angle-of-Arrival* - AOA), pelo tempo de chegada (*Time-of-Arrival* - TOA), pela diferença no tempo de chegada (*Time Difference-of-Arrival* - TDOA) e pela intensidade do sinal recebido (*Received Signal Strength* - RSS).

2.4.1 ÂNGULO DE CHEGADA (*ANGLE-OF-ARRIVAL* - AOA)

A técnica de localização baseada no ângulo de chegada pode ser dividida em duas categorias principais: através da utilização da amplitude da resposta na antena receptora e a outra através da resposta de fase (MAO et al., 2007).

A primeira técnica é baseada na utilização de técnicas de *beamforming*, onde se aumenta o ganho no sentido dos sinais de interesse e diminui no sentido da interferência e do ruído (HANZO et al., 2008). Para esta finalidade, pode-se utilizar antenas anisotrópicas, como a ilustrada na figura 4. Uma maneira de determinar o AOA é rotacionar o feixe direcional da antena e verificar a direção que corresponde ao máximo sinal recebido. Esta direção pode ser estimada para a direção do transmissor, entretanto, pequenos erros na medição da potência recebida podem acarretar em erros grandes de AOA. Uma forma de minimizar os problemas devido à variação dos sinais recebidos é utilizar várias antenas com padrões anisotrópicas conhecidas. Fazendo a análise dos padrões gerados em cada antena e comparando os sinais recebidos, pode-se melhorar as estimativas à medida que mais antenas são utilizadas. Valores típicos de precisão utilizando quatro antenas são de 10-15°. Com seis antenas pode-se chegar na ordem de 5° e com oito antenas 2° (KOKS, 2007).

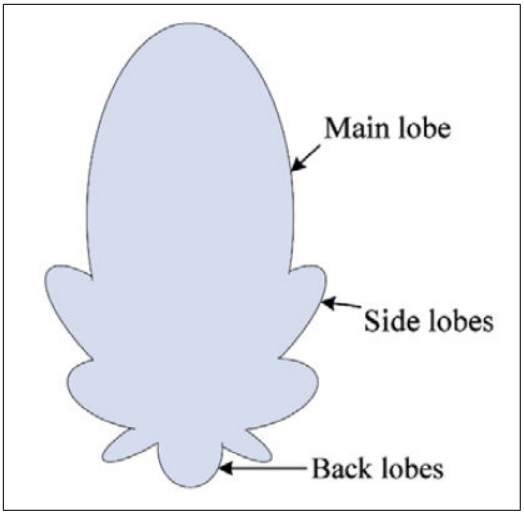

Figura 4: Exemplo de padrão de resposta em uma antena anisotrópica
Fonte: Mao et al. (2007)

A outra técnica de AOA, conhecida como interferômetro de fase (*Phase Interferometry*), utiliza a diferença de fase na chegada de uma frente de onda. Este método requer antenas de recepção maiores em relação ao comprimento de onda emitida pelo transmissor, ao contrário do método anterior. Outra alternativa é utilizar redes de antenas. A figura 5 ilustra uma rede de antenas com N elementos separados uniformemente. A distância entre o transmissor e a i-ésima

antena pode ser aproximada pela equação 2, onde R_0 é a distância entre a 0-ésima antena e o transmissor. Os sinais recebidos pelas antenas adjacentes possuem diferença de fase, dada pela equação 3, onde λ é o comprimento de onda do sinal transmitido. Assim, através da diferença de fase pode-se determinar a localização do transmissor. Nas equações abaixo, *Ri* representa a distância entre o transmissor e a i-ésima antena e $\Delta\varphi$ é a diferença de fase entre as antenas adjacentes (MAO et al., 2007).

$$Ri \approx R_0 - i.d.cos(\theta) \tag{2}$$

$$\Delta\varphi = 2\pi.\frac{d.cos(\theta)}{\lambda} \tag{3}$$

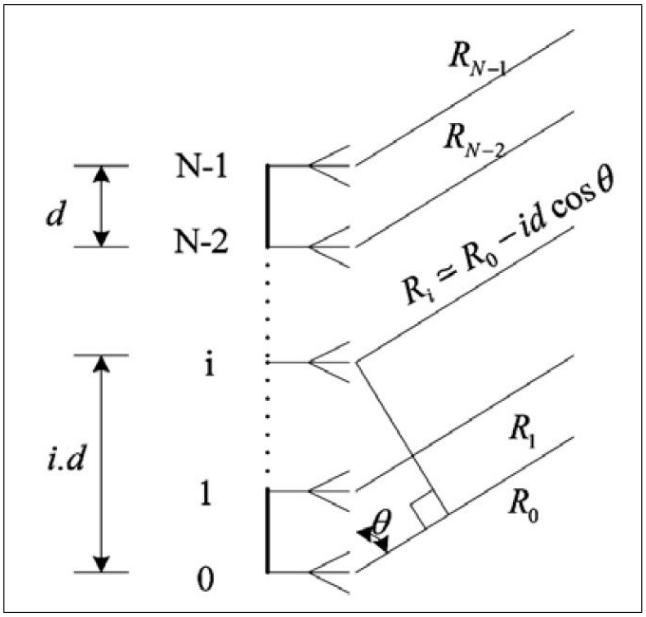

Figura 5: Rede de N antenas
Fonte: Adaptado de Mao et al. (2007)

Estimar posição através de direções (*bearing lines*) já é uma área bastante estudada, na qual Stanfield é um dos pioneiros (STANFIELD, 1947). A precisão das medidas de AOA depende do direcionamento das antenas e também do fato de ter ou não linha de visada. Os impactos causados por sombreamentos e propagação multipercurso nas medidas de AOA têm sido bastante estudado (MAO et al., 2007).

A abordagem através do estimador de máxima verossimilhança é um dos métodos bastante empregado para estimar a posição através das medições de AOA. Considerando o caso de duas dimensões, o estimador de máxima verossimilhança pode ser dada pela equação 4, onde

x_t representa a coordenada real, N é o número de receptores, ε correspondem a ruídos do tipo aditivo e com distribuição gaussiana, β são as direções determinadas nos receptores, $\theta(x)$ são os ângulos calculados pela equação 7 e S é uma matriz de covariância $N \times N$ que modela o ruído nas medições, assumindo uma distribuição gaussiana com média zero (VAGHEFI, 2011; MAO et al., 2007).

$$\widehat{x_t} = arg\,min\left\{\frac{1}{2}\left[\theta(\widehat{x_t}) - \beta\right]^T .S^{-1}.\left[\theta(\widehat{x_t}) - \beta\right]\right\} = arg\,min\frac{1}{2}\sum_{i=1}^{N}\left(\frac{\theta_i(\widehat{x_t}) - \beta_i}{\sigma_i}\right)^2 \tag{4}$$

$$x_t = [x, y]^T \tag{5}$$

$$\theta(x) = [\theta_1(x), ..., \theta_N(x)]^T \tag{6}$$

$$\theta_i(x) = arctg\frac{y_t - y_i}{x_t - x_i} \tag{7}$$

$$\varepsilon = [\varepsilon_1, \varepsilon_2, ..., \varepsilon_N]^T \tag{8}$$

$$\beta = \theta(\widehat{x_t}) + \varepsilon \tag{9}$$

$$S = diag\left\{\sigma_1^2, ..., \sigma_N^2\right\} \tag{10}$$

Na abordagem de Stanfield assume-se que os erros de medições são pequenos o suficiente, de forma que a equação 4 pode ser escrita na equação 11, que pode ser finalmente reescrita em uma equação fechada, mostrada na equação 16 (MAO et al., 2007).

$$\widehat{x_t} = \frac{1}{2}\sum_{i=1}^{N}\frac{sen^2\left(\theta_i(\widehat{x_t}) - \beta_i\right)}{\sigma_i^2} \tag{11}$$

$$A = \begin{bmatrix} sen(\beta_1) & -cos(\beta_1) \\ \vdots & \vdots \\ sen(\beta_N) & -cos(\beta_N) \end{bmatrix} \tag{12}$$

$p \times k$

$$b = \begin{bmatrix} x_1.sen(\beta_1) & -y_1.cos(\beta_1) \\ \vdots & \vdots \\ x_N.sen(\beta_N) & -y_N.cos(\beta_N) \end{bmatrix} \tag{13}$$

$N \times 2$

$$R = diag\left\{r_1^2, ..., r_N^2\right\} \tag{14}$$

$$r_i = \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \tag{15}$$

$$\widehat{x_t} = \left(A^T R^{-1} S^{-1} A\right)^{-1} A^T R^{-1} S^{-1} b \tag{16}$$

2.4.2 TEMPO DE CHEGADA (*TIME-OF-ARRIVAL* - TOA)

Esta técnica busca estimar o tempo de propagação dos sinais para o cálculo da distância. As medições podem ocorrer em apenas uma direção, ou seja, apenas os tempos de propagação do transmissor e do receptor são considerados. Assim, fazendo-se a diferença entre o instante em que o sinal foi enviado e o instante em que o mesmo foi recebido, pode-se determinar o tempo de propagação. Nesta abordagem, deve-se ter uma sincronização rígida entre o transmissor e o receptor a fim de determinar precisamente qual foi o intervalo de tempo gasto. Isto pode encarecer bastante o sistema, sendo um método de localização não muito atrativa em RSSF (MAO et al., 2007).

Outra abordagem é realizar a medição do intervalo de tempo entre a transmissão e a recepção do retorno deste sinal, no mesmo dispositivo de origem. Como a base de tempo é a mesma, tanto na transmissão como na recepção, não se tem o problema de sincronização como na abordagem anterior. Todavia, deve-se saber à priori qual é o atraso de processamento dentro do dispositivo de destino, visto que o sinal enviado é recebido e processado no dispositivo de destino e só depois é reenviado para a origem (MAO et al., 2007).

Aplicação e implementação destas técnicas podem ser encontradas em McCrady et al. (2000).

2.4.3 DIFERENÇA NO TEMPO DE CHEGADA (*TIME DIFFERENCE-OF-ARRIVAL* - TDOA)

Esta técnica faz a localização a partir da diferença de tempo de chegada de um sinal enviado por um transmissor a vários receptores. A figura 6 ilustra esta situação, onde os receptores estão representados por AP1, AP2 e AP3. Geometricamente, como será mostrada posteriormente, a posição do transmissor é determinada pela intersecção das curvas hiperbólicas. A precisão do TDOA aumenta quando a separação entre os receptores aumenta, pois assim aumenta-se também a diferença dos tempos de chegada em cada receptor.

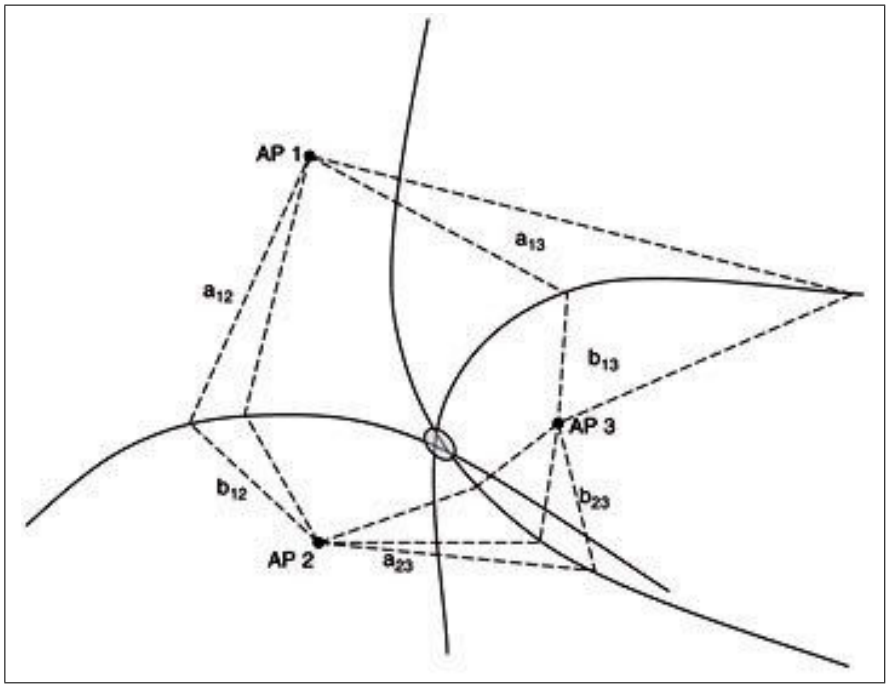

Figura 6: TDOA utilizado na localização do emissor
Fonte: Dawson et al. (2007)

Nesta técnica, a sincronização também é importante. Em tecnologias celular, a sincronização costuma ser feita através de receptores GPS e em redes Wi-Fi a sincronização já está disponível na rede (DAWSON et al., 2007). A medição de TDOA de um sinal em receptores diferentes pode ser determinada através de funções de correlação cruzada (*Cross-Correlation*).

A diferença de tempo de chegada entre os receptores i e j pode ser calculada pela equação 17. Geometricamente, esta equação define uma hipérbole, onde os receptores i e j são os focos. Nesta equação, o operador $\|.\|$ é a norma e c é a velocidade de propagação da luz. No espaço $\mathbb{R}^2$, no mínimo três medições de TDOA, de receptores diferentes, são necessárias para se determinar, de forma única, a posição do transmissor. De forma geral, a partir de N receptores pode-se obter $N-1$ equações linearmente independentes para a TDOA, como

expressa na equação 18.

$$\Delta t_{ij} = t_i - t_j = \frac{1}{c}\left(\|r_i - r_t\| - \left\|r_j - r_t\right\|\right), i \neq j \tag{17}$$

$$\begin{bmatrix} \|r_1 - r_t\| - \|r_N - r_t\| - c.\Delta t_{1,N} \\ \vdots \\ \|r_{N-1} - r_t\| - \|r_N - r_t\| - c.\Delta t_{N-1,N} \end{bmatrix} = 0 \tag{18}$$

Considerando ruído do tipo aditivo, com distribuição gaussiana e média nula, pode-se então expressar a TDOA como sendo a equação 19 e a equação 18 pode ser reescrita na equação 20.

$$\Delta\widetilde{t}_{ij} = \Delta t_{ij} + n_{ij} \tag{19}$$

$$\begin{bmatrix} \Delta\widetilde{t}_{1,N} \\ \vdots \\ \Delta\widetilde{t}_{N-1,N} \end{bmatrix} = \begin{bmatrix} \frac{\|r_1 - r_t\| - \|r_N - r_t\|}{c} \\ \vdots \\ \frac{\|r_{N-1} - r_t\| - \|r_N - r_t\|}{c} \end{bmatrix} + \begin{bmatrix} \varepsilon_{1,N} \\ \vdots \\ \varepsilon_{N-1,N} \end{bmatrix} \tag{20}$$

A resolução deste sistema de equações envolve equações não lineares e, quando o ruído é considerado, estas podem não apresentar solução. Assim, costuma-se, assim como no caso do AOA, utilizar soluções numéricas.

2.4.4 INTENSIDADE DO SINAL RECEBIDO (*RECEIVED SIGNAL STRENGTH* - RSS)

Nesta categoria, as estimativas de distâncias são feitas a partir de informações de potência recebida. Esta técnica é atrativa em RSSF, visto que diversos transceivers utilizados em RSSF já incorporam medidores da intensidade do sinal recebido (RSS, *Received Signal Strength*). Uma forma de realizar esta conversão é através da equação de Friis para o espaço livre, dada pela equação 21. Nesta equação, *Pt* representa a potência transmitida, *Pr* a potência recebida, *Gt* o ganho da antena transmissora, *Gr* o ganho da antena receptora, λ o comprimento de onda do sinal e *d* a distância entre o transmissor e o receptor (GOLDSMITH, 2005).

$$P_r = \frac{Pt.Gt.Gr.\lambda^2}{(4.\pi.d)^2} \tag{21}$$

Contudo, esta modelagem não considera os efeitos causados pelo ambiente de propaga-

ção, tais como reflexão, difração e espalhamento, considerando apenas que a potência recebida decresce com o quadrado do aumento na distância. Um modelo simplificado que considera os efeitos das perdas de percurso é dada na equação 22. Nesta equação Pr é a potência recebida (dBm), Pt é a potência transmitida (dBm), K é uma constante que depende das características da antena e da atenuação média do canal (dado em dB), d é a distância entre o transmissor e receptor (dada em metro), d_0 é a distância de referência (dada em metro) e γ é o expoente de perda de percurso. O valor de $(P_t + K)$ pode ser determinado empiricamente fazendo-se a medição da potência recebida, P_0, a uma distância de referência d_0 selecionada (GOLDSMITH, 2005).

$$P_r(d) = P_t + K - 10.\gamma.log_{10}\left(\frac{d}{d_0}\right) = P_0 - 10.\gamma.log_{10}\left(\frac{d}{d_0}\right) \tag{22}$$

A equação 22 considera que a queda na potência recebida é proporcional a d^γ e não fixado em d^2 como sugere a Equação de Friis. A equação 22 modela bem a queda da potência recebida com o aumento da distância, porém, não considera efeitos como o sombreamento [1]. O sombreamento ocorre devido à presença de obstáculos na comunicação sem fio, que fazem com que o sinal recebido no destino varie aleatoriamente, para medidas realizadas no mesmo lugar. A tabela 1 ilustra alguns valores típicos de γ em diferentes ambientes.

Tabela 1: Valores típicos do expoente de perda de percurso

Ambiente	**Valores típicos de γ**
Macrocélula urbana	3,7 - 6,5
Microcélula urbana	2.7 - 3.5
Escritório (mesmo andar)	1.6 - 3.5
Escritório (vários andares)	2 - 6
Armazém / Loja	1.8 - 2.2
Fábrica	1.6 - 3.3
Residência / Casa	3

Fonte: Goldsmith (2005)

Em Goldsmith (2005) é apresentado um modelo mais completo, o qual combina os efeitos da perda de percurso com o do sombreamento. Neste modelo, o sombreamento é considerado ter uma distribuição normal em escala logarítmica, ou seja, segue uma distribuição Log-Normal. Esta consideração foi verificada empiricamente e consegue modelar satisfatoriamente as variações na potência recebida. O modelo com a equação combinada está apresentado na equação 23, onde ψ_{dB} modela o efeito do sombreamento com uma variável aleatória com distribuição normal, média zero e variância σ^2_ψ.

[1] Maiores detalhes sobre os efeitos no canal de comunicação são dados na seção 2.6.1.

$$P_r(d) = P_t + K - 10.\gamma.log_{10}\left(\frac{d}{d_0}\right) - \psi_{dB} \quad (23)$$

Devido ao *overhead* no cálculo dos parâmetros da equação 23, algumas aplicações fazem o mapeamentos prévio da força do sinal recebido em uma área, com o intuito de montar uma tabela com a distância associada àquele nível de sinal recebido. Todavia, esta abordagem requer considerável quantidade de amostras e depende também das condições ambientais do momento, podendo sofrer grandes alterações nos valores ao longo do tempo. Recentemente, tem sido proposta a análise *online* da área de forma a acelerar o processo de levantamento dos dados antes da distribuição dos nós sensores. Entretanto, esta operação envolve a utilização de dispositivos adicionais na rede (dispositivos *sniffers*), que são distribuídos em lugares estratégicos, para o constante levantamento do modelo RSS da área em questão.

2.5 ALGORITMOS DE LOCALIZAÇÃO

2.5.1 LATERAÇÃO

Realizar a localização através de informações de distância, a partir de múltiplos pontos de referências, é também conhecido como lateração[2] (*Lateration*) e a determinação de uma posição através de informação de ângulos é conhecida como angulação (*Angulation*). Considerando a localização em duas dimensões ao menos três medições de distância, a partir de três pontos de referência não colineares, são necessários para a localização ser única. Para o caso de três dimensões, distâncias a partir de quatro pontos de referências não coplanares são necessários. Geometricamente, para o caso 2D, trata-se de um problema de encontrar a interseção de circunferências centradas nos pontos de referência, de onde surge o termo trilateração (NOBLES et al., 2011; HIGHTOWER; BORRIELLO, 2001).

A figura 7 ilustra um exemplo de localização de um dispositivo através do processo de trilateração. Esta figura ilustra um caso ideal, com todas as informações de distância corretas. Exemplo de utilização desta técnica são os dispositivos GPS. Estes dispositivos receptores estimam sua própria posição a partir de distâncias medidas de diferentes satélites (JUSTICE, 2009).

Diferentemente da trilateração, a triangulação utiliza informações de ângulos para se determinar a posição de um nó desconhecido. Para isso, aplica-se as relações trigonométricas fundamentais de senos e cossenos, desde que se conheçam as posições dos nós de referência.

[2]Este método de localização será, doravante, denominado algoritmo da Lateração ou método da Lateração

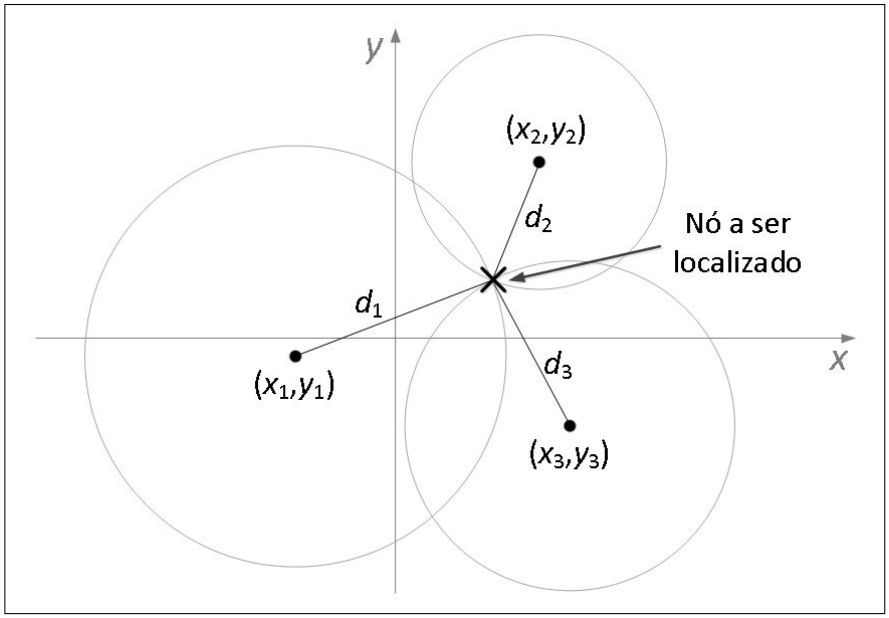

Figura 7: Localização de um nó sensor através de trilateração

Fonte: Adaptado de Shirai et al. (2012)

Para esta metodologia apenas dois nós de referências são necessários para se estimar a posição do terceiro nó. A figura 8 ilustra este conceito.

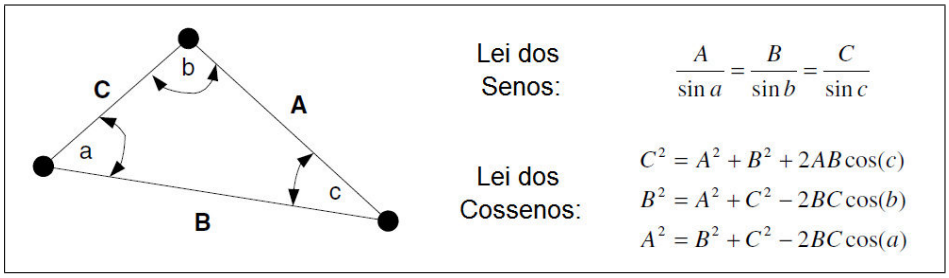

Figura 8: Localização de um nó sensor através de triangulação

Fonte: Adaptado de Savvides e Srivastava (2004)

A multilateração é outra metodologia utilizada para estimar a posição de um dispositivo, a qual utiliza informações de diferença de distância entre pontos de referências, ou de medições de TDOA entre o emissor e múltiplos receptores, para estimar a posição do dispositivo móvel. A multilateração é bastante utilizada em aplicações militares como, por exemplo, vigilância aérea e rastreamento de aeronaves (XU et al., 2010).

Chen et al. (2009) apresenta as equações para determinar a posição almejada, utilizando-se as medições de distâncias entre o transmissor e o receptor, representadas por $d_1, d_2, ..., d_N$, e as coordenadas de referência, dadas por $(x_i, y_i), (x_2, y_2), ..., (x_N, y_N)$, onde N representa o número de nós de referência. O equacionamento está apresentado a seguir:

$$\begin{cases} (x_1 - x)^2 + (y_1 - y)^2 = d_1^2 \\ \vdots \\ (x_N - x)^2 + (y_N - y)^2 = d_N^2 \end{cases} \tag{24}$$

$$\begin{cases} x_1^2 - 2.x_1.x + x^2 + y_1^2 - 2.y_1.y + y^2 = d_1^2 \\ \vdots \\ x_N^2 - 2.x_N.x + x^2 + y_N^2 - 2.y_N.y + y^2 = d_N^2 \end{cases} \tag{25}$$

$$\begin{cases} x_1^2 - x_N^2 - 2(x_1 - x_N)x + y_1^2 - y_N^2 - 2(y_1 - y_N)y = d_1^2 - d_N^2 \\ \vdots \\ x_{N-1}^2 - x_N^2 - 2(x_{N-1} - x_N)x + y_{N-1}^2 - y_N^2 - 2(y_{N-1} - y_N)y + y^2 = d_{N-1}^2 - d_N^2 \end{cases} \tag{26}$$

A partir do sistema de equações acima, pode-se agrupar as equações na notação matricial $A.X = b$, onde A, X e b estão especificados abaixo:

$$A = \begin{bmatrix} 2(x_1 - x_N) & 2(y_1 - y_N) \\ 2(x_2 - x_N) & 2(y_2 - y_N) \\ \vdots & \vdots \\ 2(x_{N-1} - x_N) & 2(y_{N-1} - y_N) \end{bmatrix} \tag{27}$$

$$X = \begin{bmatrix} x \\ y \end{bmatrix} \tag{28}$$

$$b = \begin{bmatrix} x_1^2 - x_N^2 + y_1^2 - y_N^2 + d_N^2 - d_1^2 \\ x_2^2 - x_N^2 + y_2^2 - y_N^2 + d_N^2 - d_2^2 \\ \vdots \\ x_{N-1}^2 - x_N^2 + y_{N-1}^2 - y_N^2 + d_N^2 - d_{N-1}^2 \end{bmatrix} \tag{29}$$

Para o caso especial de três NRs, a matriz A é quadrada (2 x 2) e o sistema de equações pode ser resolvido de forma única. Shirai et al. (2012) partiu da relação $X = A^{-1}b$ e resolveu este sistema de equações de forma qualitativa, organizando as equações de forma a facilitar a sua implementação em hardware. Esta formulação está apresentada a seguir:

$$x = \frac{(y_3 - y_2).k_1 - (y_3 - y_1).k_2}{k3} \tag{30}$$

$$y = \frac{(x_2 - x_3).k_1 - (x_1 - x_3).k_2}{k3} \tag{31}$$

$$k_1 = x_1^2 - x_3^2 + y_1^2 - y_3^2 + d_3^2 - d_1^2 \tag{32}$$

$$k_2 = x_2^2 - x_3^2 + y_2^2 - y_3^2 + d_3^2 - d_2^2 \tag{33}$$

$$k_3 = 2[(y_1 - y_3).(x_2 - x_3) - (x_1 - x_3).(y_2 - y_3)] \tag{34}$$

Em casos com mais de três NRs, a partir da equações 27 a 28 pode-se constatar que o sistema será superdeterminado, visto que apresentará mais equações do que variáveis. Além disso, considerando que os valores de medições são imprecisos, o sistema de equações pode nem mesmo apresentar solução. Nesta situação costuma-se empregar o método dos mínimos quadrados para encontrar uma solução $\overline{X} = [\overline{x}, \overline{y}]$, tal que este valor venha a satisfazer, da melhor forma possível, todas as equações envolvidas (WILLIAMS, 1990; FÉO, 2005).

A utilização do método da lateração para múltiplos NRs, baseando-se no método dos mínimos quadrados, é apresentada em (CHEN et al., 2009) e é dada pela equação 35.

$$\overline{X} = (A^T A)^{-1} A^T b \tag{35}$$

A utilização de vários NRs pode melhorar a qualidade das estimativas de posição. Entretanto, para o caso do método dos mínimos quadrados, existe a necessidade do cálculo da inversa de uma matriz e do cálculo de várias multiplicações de matrizes, o que pode ser custoso, tanto em quantidade de operações, quanto em recursos de hardware. Outro ponto importante é que este método não leva em conta as características das medições de distância, que podem apresentar erros consideráveis, e simplesmente supõe que tais valores de distância são as corretas. Métodos que tratam o processo de localização estatisticamente podem apresentar melhores resultados (CHANG; LIAO, 2009).

2.5.2 MÉTODO MIN-MAX

Uma abordagem similar ao algoritmo da Lateração, conhecido método Min-Max, é apresentado em Savvides et al. (2002). Este método consiste em se aproximar as circunferências do método da Lateração por quadrados. A posição do NPD é estimada como sendo o centroide

da área superposta por todos os quadrados. A figura 9 ilustra este método.

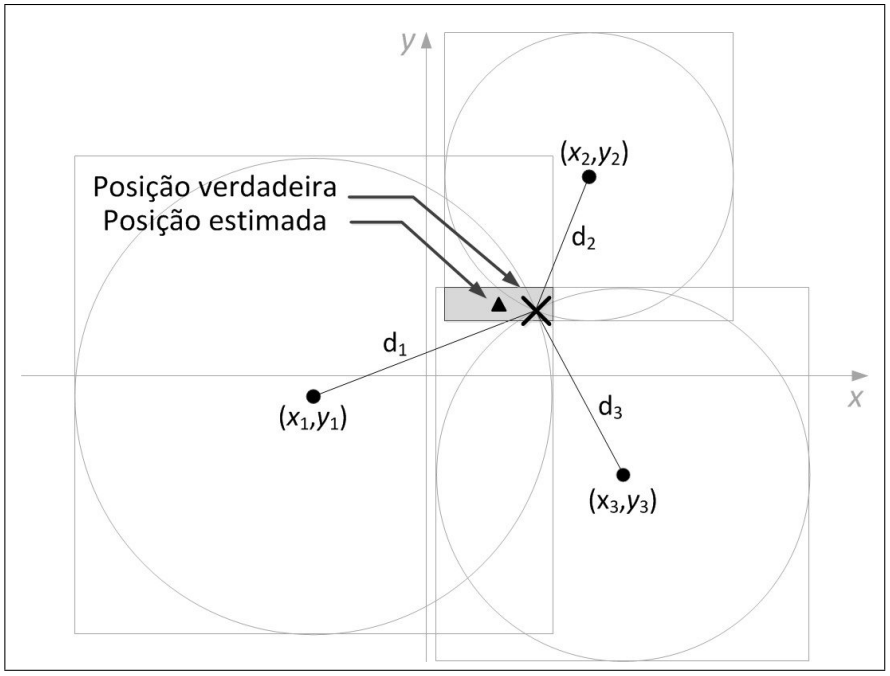

Figura 9: Localização de um nó sensor através do método Min-Max
Fonte: Adaptado de Shirai et al. (2012)

A simplicidade deste método é seu grande diferencial, além de ser interessante do ponto de vista de implementação em hardware, pois seu cálculo envolve, basicamente, comparações para encontrar valores máximos e mínimos, além de algumas outras operações matemáticas simples. A posição (x,y) do NPD, considerando-se N NRs, pode ser determinada por este método conforme as equações 36 a 40. Os operadores $max(.)$ e $min(.)$ correspondem, respectivamente, ao valor máximo e mínimo do vetor em questão.

$$X = [x_1, x_2, ..., x_N] \tag{36}$$

$$Y = [y_1, y_2, ..., y_N] \tag{37}$$

$$D = [d_1, d_2, ..., d_N] \tag{38}$$

$$x = \frac{max(X - D) + min(X + D)}{2} \tag{39}$$

$$y = \frac{max(Y-D)+min(Y+D)}{2} \tag{40}$$

Embora de fácil cálculo, a aproximação das circunferências por quadrados pode acarretar em grandes erros, sobretudo quando poucos NRs são considerados, já que maior será a área de interseção e, assim, pior poderá ser a estimativa de posição. Neste sentido, apesar de sua simplicidade, dependendo da configuração da rede, este método pode não ser o mais adequado para o cálculo da posição (SHIRAI et al., 2012).

2.5.3 MÉTODO DO ESTIMADOR DE MÁXIMA VEROSSIMILHANÇA

A utilização de RSS no processo de localização é atraente em RSSFs, pois vários protocolos utilizados nestas aplicações já implementam meios de se estimar o nível de potência recebida. Neste sentido, nenhum hardware adicional é necessário. Todavia, os valores de RSS podem variar bastante no tempo e são suscetíveis a diversas fontes de interferências, que são inerentes na comunicação sem fio. Neste contexto, os valores imprecisos de RSS produzem estimativas imprecisas de distância, o que influencia diretamente o cálculo da posição. Assim, pode ocorrer do algoritmo de localização apresentar bons resultados teóricos, porém estimar valores de posição longe da real, quando os valores imprecisos de distância são fornecidos como entrada no algoritmo (PATWARI et al., 2001; ZANCA et al., 2008).

Neste sentido, muitos trabalhos têm modelado os valores de RSS estatisticamente, a partir de funções de distribuição de probabilidade condicionada à distância. Na literatura, duas modelagem destas tem sido frequentemente utilizadas: distribuição Log-Normal e distribuição Exponencial. Em ambos casos o canal de propagação entre um NPD e um NR é considerado independente dos demais canais formados entre os outros NPDs e NRs. Assim, a função densidade de probabilidade conjunta pode ser formulada através da equação 41, onde d_i representa as distâncias entre o NPD e os NRs. Nesta equação θ é a posição do NPD (fixo) e p_i é uma variável aleatória (TAKETSUGU; YAMAKITA, 2007; PORTUGAL, 2001).

$$p(P_1, P_2, ..., P_N \mid \theta) = \prod_{i=1}^{N} p(P_i \mid \theta) = \prod_{i=1}^{N} p(P_i \mid d_i) \tag{41}$$

O Método de Máxima Verossimilhança estima o parâmetro θ baseado na função densidade de probabilidade $p(P_1, P_2, ..., P_N \mid \theta)$ e das observações $(P_1, P_2, ..., P_n)$. Uma vez que as observações $(P_1, P_2, ..., P_n)$ são conhecidas, $p(P_1, P_2, ..., P_N \mid \theta)$ fica apenas em função de θ, onde se define a função verossimilhança.

Muitas vezes é mais fácil trabalhar com a função Log-verossimilhança, $L(\theta)$, do que a função verossimilhança em si, visto que em muitas situações é mais fácil trabalhar com a soma de termos do que com a multiplicação delas, além disso, a função verossimilhança pode conter termos exponenciais, que tornam a equação mais simples ao se aplicar o logaritmo. O processo de otimização numérica pode ficar mais simples também. A função Log-verossimilhança é dada pela equação 42.

$$L(\theta) = log \prod_{i=1}^{N} p(P_i \mid \theta) = \sum_{i=1}^{N} log\left[p(P_i \mid \theta_i)\right] \tag{42}$$

O Método da Máxima Verossimilhança (MLE) consiste em encontrar o valor de $\widehat{\theta} = [\widehat{x}, \widehat{y}]$ que provavelmente ocasionou a observação das potências medidas. Trata-se de um método sofisticado, entretanto, apresenta a desvantagem de necessitar de métodos de otimização numérica para realizar a minimização ou maximização de equações.

Para realizar a minimização da equação 42, pode-se utilizar métodos como o do Gradiente Conjugado e Método de Newton, que exigem cálculo de derivadas, ou o Método de Nelder-Mead, que realiza a busca da solução sem necessitar do cálculo de derivadas. O desempenho da utilização do Método de Nelder-Mead em uma proposta de algoritmo de localização foi mostrado em Trevisan (2009).

2.5.3.1 VEROSSIMILHANÇA COM DISTRIBUIÇÃO LOG-NORMAL

A modelagem do RSS pela distribuição Log-Normal está apresentada na equação 43, onde $\overline{P}_{dBm}$ representa o valor médio de RSS. Em outras palavras, quando um NPD está separado do NR de uma distância d, o valor de RSS, medido em dBm, apresenta distribuição gaussiana com média $\overline{P}_{dBm}$ e variância σ^2 (PATWARI et al., 2001).

$$p\left(P[dBm] \mid d\right) = \frac{1}{\sqrt{2\pi}\sigma} exp\left[\frac{\left(P - \overline{P}_{dBm(d)}\right)}{2\sigma^2}\right] \tag{43}$$

Considerando a modelagem do RSS através da distribuição Log-Normal, Patwari et al. (2001) propôs uma formulação para estimar a posição $\widehat{\theta}$. Esta formulação está apresentada nas equações 44 e 45. A coordenada $\widehat{\theta}$ corresponde às coordenadas de maior probabilidade do nó se encontrar.

$$\widehat{\theta} = \underset{X,Y}{arg\,min}\left[f(x_k, y_k)\right] \tag{44}$$

$$f(x_k,y_k) = \frac{b^2}{8}\sum_{i=1}^{N}\sum_{j\in H_i, j\neq i} ln^2\left(\frac{\widehat{d_{i,j}^2}}{d_{i,j}^2}\right) - \sum_{i=1}^{N}\sum_{j\notin H_i, j\neq i} ln\left[Q\left(\frac{b}{2}ln\frac{d_{thr}^2}{d_{i,j}^2}\right)\right] \tag{45}$$

A equação 45 é um caso de localização conjunta e costuma ser utilizada em sistemas centralizados. Nesta formulação supõe-se a existência de N dispositivos, os quais medem as potências recebidas dos nós adjacentes e os enviam para o nó que centraliza o processamento, que então estima as coordenadas dos N dispositivos. Nesta equação H_i representa o conjunto de nós vizinhos, podendo tanto ser um NR ou outro NPD, que o nó i consegue detectar. Assume-se que se a potência recebida estiver abaixo de um valor limite, denotado por P_{thr}, o nó transmissor correspondente não será detectado. O termo do lado direito da equação representa a probabilidade de que a potência recebida para $j \neq Hi$ seja menor que P_{thr}, considerando que a posição candidata seja a verdadeira. A função $Q(x)$ representa a área sob a curva de distribuição normal, x desvios padrão de distância a partir da média. As demais variáveis estão apresentadas nas equações abaixo:

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{46}$$

$$\widehat{d_{i,j}} = d_0.10^{\frac{p_0 - \widehat{p}_{i,j}}{10.n}} = d_{i,j}.10^{\frac{X_\sigma}{10.n}} \tag{47}$$

$$b = \frac{10.n}{ln(10).\sigma_{dB}} \tag{48}$$

$$d_{thr} = d_0.10^{\frac{p_0 - p_{thr}}{10.n}} \tag{49}$$

Trevisan (2009) apresenta uma formulação simplificada, que não leva em consideração o termo do lado direito da equação 45. Nesta equação, N corresponde ao número de NPDs e m o número de NRs presentes na rede e todos os nós se comunicam uns com os outros. Esta formulação está apresentada na equação 50.

$$f(x_k,y_k) = \sum_{i=1}^{N}\sum_{j=1, j\neq i}^{N+m} ln^2\left(\frac{\widehat{d_{i,j}^2}}{d_{i,j}^2}\right) \tag{50}$$

A partir desta última equação, se for considerada a localização de apenas um NPD ou, alternativamente, se os NPDs não se comunicarem entre si, mas apenas com os NRs, a

equação 50 pode ser simplificada na equação 51. Contudo, como mostrado em Patwari et al. (2001), a utilização de mais NPDs nos cálculos melhora a qualidade da localização. Assim, a equação 50, com as duas simplificações feitas, acaba apresentando resultados inferiores aos da equação 45, porém, é a formulação que se adéqua ao cenário de estudo do presente trabalho.

$$f(x_k, y_k) = \sum_{i=1}^{N} ln^2 \left(\frac{\widehat{d_i^2}}{d_i^2} \right) \tag{51}$$

2.5.3.2 VEROSSIMILHANÇA COM DISTRIBUIÇÃO EXPONENCIAL

Por outro lado, considerando que o RSS apresenta distribuição Exponencial, a modelagem pode então ser feita através da equação 52. Nesta equação, $\overline{P}_{mW(d)}$ é a média dos valores de RSS (HARA et al., 2005). Na equação 53, α e β são parâmetros do ambiente e devem ser estimados à priori.

$$p\left(P[mW] \mid d\right) = \frac{1}{\overline{P}_{mW(d)}} exp\left[-\frac{P}{\overline{P}_{mW(d)}} \right] \tag{52}$$

$$\overline{P}_{mW(d)} = \alpha . d^{-\beta} \tag{53}$$

Aplicando a equação 53 na equação 42, pode-se obter o estimador de máxima verossimilhança para o caso deste tipo de distribuição de RSS. Considerando que os nós e posição desconhecida recebem informações apenas dos nós de referência (não recebem informação de outros nós de posição desconhecida), pode-se obter a formulação do MLE apresentada nas equações 54 e 55 (ZEMEK et al., 2008).

$$\widehat{\theta} = \underset{X,Y}{arg\,max} \left[f(x_k, y_k) \right] \tag{54}$$

$$f(x_k, y_k) = \sum_{i=1}^{N} \left[ln\left(\frac{1}{\alpha . d_i^{-\beta}} \right) - \frac{P_i}{\alpha . d_i^{-\beta}} \right] \tag{55}$$

2.6 TÉCNICAS PARA REFINAR ESTIMATIVAS DE DISTÂNCIAS

A determinação de distância através de RSS é uma técnica sujeita a diversas interferências. Em geral, um algoritmo de localização apresentará resultados tanto melhores quanto

melhores forem os dados de entrada. Nesse sentido, melhorando-se as estimativas de distâncias, melhora-se também o desempenho dos algoritmos.

Nesta seção serão estudadas as principais fontes de erro na comunicação sem fio e na medição de RSS. Serão apresentadas também duas técnicas encontradas na literatura para se refinar as estimativas de distâncias, que são afetadas pelo comportamento do canal de comunicação sem fio. Estas técnicas são: método baseado no determinante de Cayley-Menger e o Filtro de Kalman.

2.6.1 FONTES DE ERRO NA COMUNICAÇÃO SEM FIO

Uma comunicação sem fio costuma ser mais difícil de realizar em comparação com a comunicação em sistemas cabeados. A comunicação sem fio está sujeita a ruídos, interferências, obstáculos e, além disso, estas variáveis se alteram no tempo de forma não previsível. O comportamento do sinal transmitido geralmente é estudado dividindo-se em duas componentes: desvanecimento em larga escala e desvanecimento em pequena escala.

O desvanecimento em larga escala está ligado às características de propagação que manifestam seus efeitos no sinal, ao longo de médias e grandes distâncias, comparadas com o comprimento de onda transmitida. Neste tipo de desvanecimento ocorre, basicamente, os efeitos da perda de percurso e do sombreamento. A perda de percurso costuma ser modelado com uma função que decresce logaritmicamente com a distância. O Sombreamento é causado por obstáculos presentes entre o transmissor e o receptor que atenuam o sinal através dos efeitos de absorção, reflexão, espalhamento e difração (GOLDSMITH, 2005; DA COSTA, 2003).

Já o desvanecimento em pequena escala é causado pelo comportamento aleatório das componentes do sinal que chegam ao receptor, que se interagem de forma construtiva ou destrutiva. Estes efeitos são verificados a curtas distâncias, da ordem no comprimento de onda do sinal. Este comportamento é devido à multiplicidade de percurso feito pelas várias componentes do sinal, que chegam ao receptor com diferentes amplitudes e defasagens entre si (GOLDSMITH, 2005; DA COSTA, 2003)

A figura 10 ilustra o efeito conjunto do desvanecimento em larga escala e do desvanecimento em pequena escala.

No caso de medições com RSS o multipercurso e o sombreamento são os efeitos mais significativos. O efeito do multipercurso pode ser reduzido através de transmissões com espalhamento espectral. Já o sombreamento depende muito dos obstáculos presentes entre os nós sensores (PATWARI et al., 2005b).

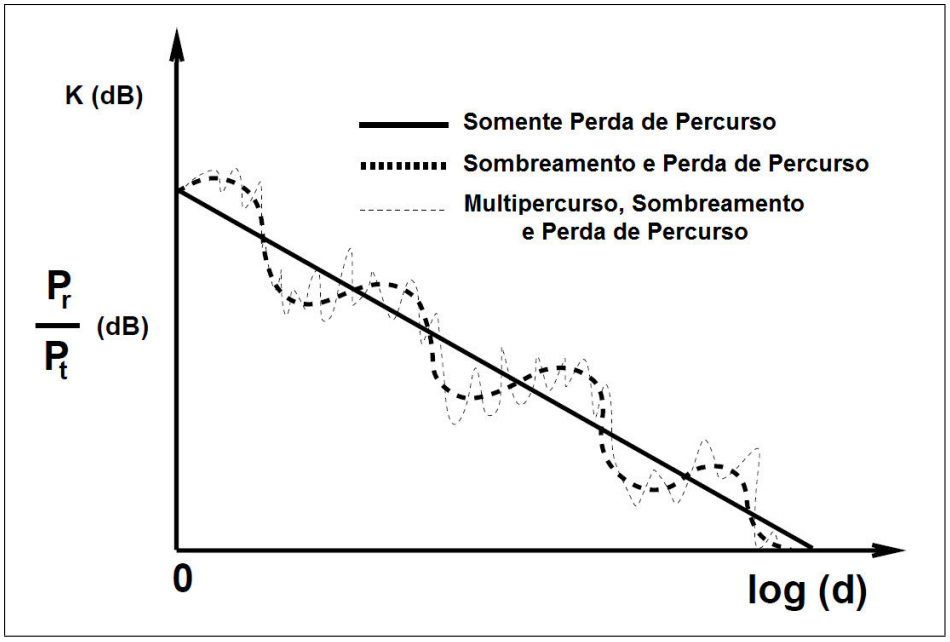

Figura 10: Efeitos que ocorrem na comunicação sem fio

Fonte: Adaptado de Goldsmith (2005)

Assim, considerando os efeitos comentados, a cobertura do sinal de um nó sensor tende a ser irregular. No caso das RSSFs esta irregularidade depende de fatores como: característica das antenas dos nós sensores, ambiente de instalação da rede, padrão de distribuição dos nós de referência, densidade de nós de referência, entre outros (ZHANG et al., 2011; ZHOU et al., 2006).

2.6.2 DETERMINANTE DE CAYLEY-MENGER

Uma técnica que pode ser empregada para a redução do impacto causado pelos erros de medição no algoritmo de localização é a utilização dos determinantes de Cayley-Menger. Considere a figura 11, a qual ilustra o caso de duas dimensões, onde se tem três NRs localizados em P_1, P_2 e P_3 e o NPD localizado em P_0. Para este caso, como mostrado em Cao et al. (2006), a matriz de Cayley-Menger pode ser escrita como consta na equação 56:

$$M(p_0,p_1,p_2,p_3) = \begin{bmatrix} 0 & d_{01}^2 & d_{02}^2 & d_{03}^2 & 1 \\ d_{01}^2 & 0 & d_{12}^2 & d_{13}^2 & 1 \\ d_{02}^2 & d_{12}^2 & 0 & d_{23}^2 & 1 \\ d_{03}^2 & d_{13}^2 & d_{23}^2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{56}$$

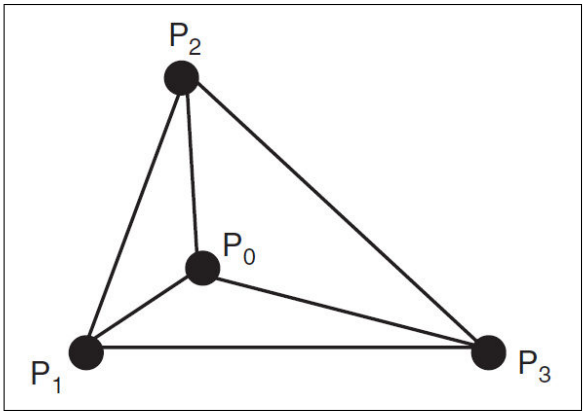

Figura 11: Nó P_0, de posição desconhecida, e três nós de referência, P_1, P_2 e P_3

Fonte: Cao et al. (2006)

Cao et al. (2006) mostra que $det\,M(p_0,p_1,p_2,p_3) = 0$ para o caso de duas dimensões. Na equação 56 considera-se que as distâncias d_{ij} podem ser inferidas a partir das posições dos *anchor nodes*, visto que seus posicionamentos são estabelecidos a priori. Através da equação 57, pode-se relacionar estas distâncias fixas, d_{ij}, com as distâncias medidas na prática, denotadas por $\overline{d}_{ij}$, através de uma relação quadrática. Nesta equação, ε_i denota o erro associado na medição.

$$\overline{d}_{0,i}^2 = d_{0,i}^2 - \varepsilon_i, (1 \leq i \leq 3) \tag{57}$$

É mostrado em Cao et al. (2006) que o erro da equação anterior pode ser escrito na equação 58, onde o vetor $\varepsilon = [\varepsilon_1, \varepsilon_2, \varepsilon_3]^T$, a matriz A, o vetor b e o vetor c podem ser expressos através das distâncias d_{12}, d_{13} e d_{23} (entre os *anchor nodes*) e $\overline{d}_{01}$, $\overline{d}_{02}$, $\overline{d}_{03}$ (distâncias medidas pelo *blind node*).

$$\varepsilon^T.A.\varepsilon + \varepsilon^T.b + c = 0 \tag{58}$$

$$A = \begin{bmatrix} 2.d_{23}^2 & d_{12}^2 - d_{13}^2 - d_{23}^2 & d_{13}^2 - d_{23}^2 - d_{12}^2 \\ d_{12}^2 - d_{13}^2 - d_{23}^2 & 2.d_{13}^2 & d_{23}^2 - d_{12}^2 - d_{13}^2 \\ d_{13}^2 - d_{12}^2 - d_{23}^2 & d_{23}^2 - d_{12}^2 - d_{13}^2 & 2.d_{12}^2 \end{bmatrix} \tag{59}$$

$$\begin{aligned} b_1 = 4.d_{23}^2.\overline{d}_{01}^2 + 2.(d_{12}^2 - d_{13}^2 - d_{23}^2).\overline{d}_{02}^2 + \\ 2.(d_{13}^2 - d_{12}^2 - d_{23}^2).\overline{d}_{03}^2 + 2.d_{23}^2.(d_{23}^2 - d_{12}^2 - d_{13}^2) \end{aligned} \tag{60}$$

$$b_2 = 4.d_{13}^2.\overline{d}_{02}^2 + 2.(d_{12}^2 - d_{13}^2 - d_{23}^2).\overline{d}_{01}^2 + 2.(d_{23}^2 - d_{12}^2 - d_{13}^2).\overline{d}_{03}^2 + 2.d_{13}^2.(d_{13}^2 - d_{12}^2 - d_{23}^2) \quad (61)$$

$$b_3 = 4.d_{12}^2.\overline{d}_{03}^2 + 2.(d_{13}^2 - d_{12}^2 - d_{23}^2).\overline{d}_{01}^2 + 2.(d_{23}^2 - d_{12}^2 - d_{13}^2).\overline{d}_{02}^2 + 2.d_{12}^2.(d_{12}^2 - d_{13}^2 - d_{23}^2) \quad (62)$$

$$\begin{aligned} c = {} & 2.d_{12}^2.d_{13}^2.d_{23}^2 + 2.d_{23}^2.\overline{d}_{01}^4 + 2.d_{13}^2.\overline{d}_{02}^4 + 2.d_{12}^2.\overline{d}_{03}^4 + \\ & 2.(d_{12}^2 - d_{13}^2 - d_{23}^2).\overline{d}_{01}^2.\overline{d}_{02}^2 + 2.(d_{13}^2 - d_{12}^2 - d_{23}^2).\overline{d}_{01}^2.\overline{d}_{03}^2 + \\ & 2.(d_{23}^2 - d_{12}^2 - d_{13}^2).\overline{d}_{02}^2.\overline{d}_{03}^2 + 2.d_{23}^2.(d_{23}^2 - d_{12}^2 - d_{13}^2).\overline{d}_{01}^2 + \\ & 2.d_{13}^2.(d_{13}^2 - d_{12}^2 - d_{23}^2).\overline{d}_{02}^2 + 2.d_{12}^2.(d_{12}^2 - d_{13}^2 - d_{23}^2).\overline{d}_{03}^2 \end{aligned} \quad (63)$$

Finalmente, combinando-se as equações resultantes e minimizando-se a equação do erro, pode-se, através da equação 57, obter estimativas melhoradas para as distâncias medidas, melhorando, assim, a determinação da posição. Esta abordagem pode ser estendida também para os algoritmos de localização baseado em AOA e TDOA (CAO et al., 2006).

As formulações apresentadas permitem relacionar as distâncias entre os nós sensores através de um conjunto de restrições quadráticas. Como mostrado, necessita-se minimizar o erro quadrático e, diferentemente do método baseado no MLE, trata-se de um problema de minimização restrita. Neste caso, a otimização pode ser abordada pelo método dos Multiplicadores de Lagrange (RAU, 2003; CAO et al., 2006).

O trabalho de Taraktas et al. (2010) mostra que, para três nós de referência, o algoritmo da Lateração com os refinamentos desta técnica apresentou resultados melhores que os obtidos com o chip CC2431, da Texas Instruments.

2.6.3 FILTRO DE KALMAN

O filtro de Kalman é um filtro estatístico que faz estimativas do estado interno de um sistema dinâmico utilizando uma série de medidas da saída do sistema, sendo que estas medidas estão corrompidas por ruídos e outras incertezas. Matematicamente, é um conjunto de equações que estima o estado do processo, de forma recursiva, no sentido de minimizar seu erro quadrático (PARANHOS, 2009; WELCH; BISHOP, 2006).

Este filtro é, geralmente, descrito em duas fases distintas: Predição e Atualização/Correção. Na fase de predição um estimador é construído, baseando-se nos dados do passo anterior, para prever a distribuição do vetor de estado um passo a frente. Esta predição não in-

clui a informação vinda da observação do estado atual e é denominada estimativa "*a priori*" (KARTHICK et al., 2009; WELCH; BISHOP, 2006). As seguintes equações descrevem esta fase:

$$\bar{\hat{x}}_k = A\hat{x}_{k-1} + Bu_{k-1} \tag{64}$$

$$\bar{P}_k = AP_{k-1}A^T + Q \tag{65}$$

Na equação 64, A corresponde à uma matriz $n \times n$ que representa a relação existente entre o estado atual, x_k, e o estado anterior, x_{k-1}, devendo esta relação ser linear. Já a matriz B, de dimensão $n \times l$, relaciona as entradas do sistema, u_k, com o estado x. Na equação 65, $\bar{P}_k$ e P_k, correspondem, respectivamente, à covariância do erro de medição *a priori* e à covariância do erro de medição *a posteriori*, dadas pelas equações 66 e 67. Nestas últimas equações, $E[.]$ denota o operador do valor esperado e utilizou-se a notação "$-$" para representar o estado *a priori*. Ainda na equação 65, Q representa a matriz de covariância do ruído do processo (WELCH; BISHOP, 2006; DE SAMPAIO; XAVIER, 2011).

$$\bar{P}_k = E\left[\bar{e}_k \bar{e}_k^T\right] \tag{66}$$

$$P_k = E\left[e_k e_k^T\right] \tag{67}$$

Na fase de atualização ou correção, a predição "*a priori*" e a observação atual são utilizadas para obter uma estimativa refinada para o estado atual. A estimativa refinada é também denominada estimativa "a posteriori". Assim, o processo se repete recursivamente, com a estimativa "a posteriori" sendo utilizada para predizer a nova estimativa "a priori" (WELCH; BISHOP, 2006; DE SAMPAIO; XAVIER, 2011; KARTHICK et al., 2009). As seguintes equações descrevem esta fase:

$$K_k = \bar{P}_k H^T (H\bar{P}_k H^T + R)^{-1} = \frac{\bar{P}_k H^T}{H\bar{P}_k H^T + R} \tag{68}$$

$$\hat{x}_k = \bar{\hat{x}}_k + K_k(z_k - H\bar{\hat{x}}_k) \tag{69}$$

$$P_k = (I - K_k H)\bar{P_k} \tag{70}$$

A etapa de atualização inicia-se com o cálculo do ganho de Kalman, K_k, dada pela equação 68. Este ganho tem como objetivo minimizar a covariância do erro *a posteriori*. Nesta equação, a matriz H, de dimensão *mxn*, relaciona o estado x com as medidas atuais z_k e R representa a matriz de covariância do ruído de medição. A equação 69 faz a estimativa do estado *a posteriori*, combinando a estimativa *a priori* com o ganho de Kalman e as medições atuais. Por fim, a equação 70 calcula a estimativa da covariância do erro *a posteriori* (WELCH; BISHOP, 2006; DE SAMPAIO; XAVIER, 2011).

O funcionamento completo deste filtro está esquematizado na figura 12.

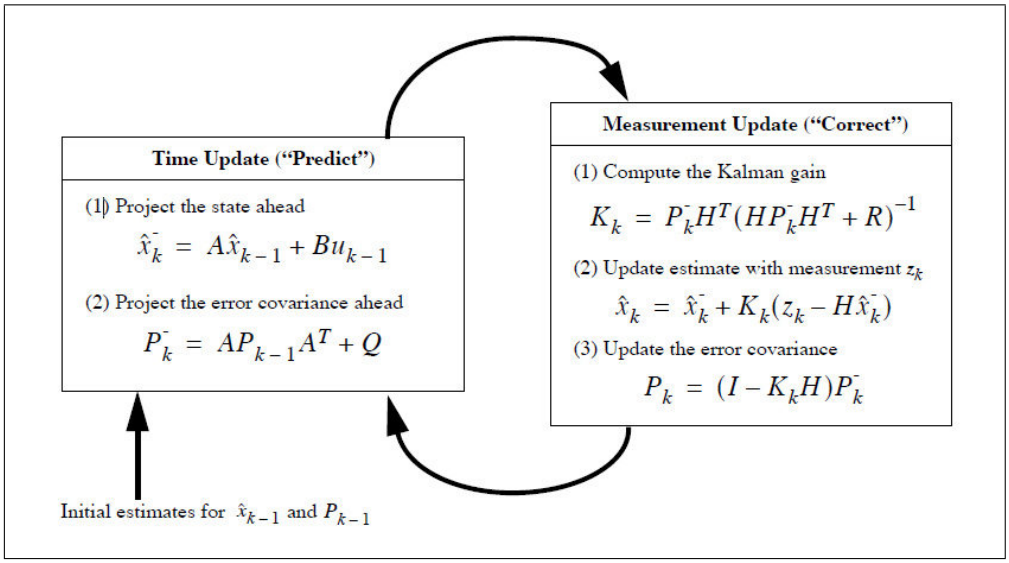

Figura 12: Funcionamento do filtro de Kalman

Fonte: Welch e Bishop (2006)

O filtro de Kalman tem sido largamente aplicado em problemas de rastreamentos (KOKS, 2007). Este filtro também tem sido aplicado em rastreamentos que utilizam RSSI, como apresentado Masiero (2007) e Skibniewski e Jang (2009).

Em outra linha de pesquisa, Vij e Mehra (2011) comenta sobre a implementação deste filtro em FPGA. Este mesmo trabalho cita também algumas variantes deste filtro, como o EKF (*Extended Kalman Filter*) e o UKF (*Unscented Kalman Filter*), que são utilizadas quando a dinâmica do sistema é não linear.

2.7 DISPOSITIVOS PROGRAMÁVEIS

2.7.1 INTRODUÇÃO

Dispositivo lógico programável é um termo genérico para designar os componentes eletrônicos utilizados para construir circuitos digitais programáveis pelo usuário. A lógica de programação necessita tanto de hardware como de software. Nesta seção serão apresentados os hardware mais utilizados atualmente e, na próxima seção, será tratada a parte de software (FLOYD, 2006; BROWN; ROSE, 1996).

A grande vantagem dos dispositivos lógicos programáveis frente aos de lógica fixa é sua flexibilidade em se poder alterar sua funcionalidade com facilidade, sem alterações físicas no hardware ou mesmo substituição de componentes. Além disso, um projeto com estes dispositivos pode ser implementado mais rapidamente e com menor custo do que projetos com função fixa, ou seja, com circuitos integrados e componentes discretos. Entre os dispositivos de lógica programáveis, podem-se citar: SPLD (*Simple Programmable Logic Device*), CPLD (*Complex Programmable Logic Devices*) e FPGA (*Field Programmable Gate Array*). Nesta dissertação será trabalhado apenas com FPGAs (FLOYD, 2006; BROWN; ROSE, 1996).

Os SPLDs costumam ser divididos em PAL (*Programmable Array of Logic*) e GAL (*General Array Logic*). Um dispositivo PAL possui um arranjo programável de portas AND e um arranjo fixo de portas OR. Estes dispositivos podem ser programados apenas uma única vez. Já um dispositivo GAL possui um arranjo programável de portas AND, um arranjo fixo de portas OR e lógica de saída programável, podendo ser reprogramado várias vezes. Já um CPLD é um dispositivo que contém múltiplos SPLDs (FLOYD, 2006; BROWN; ROSE, 1996).

As FPGAs, por sua vez, são dispositivos mais complexos e com densidade de circuitos maior que os CPLDs. Uma FPGA contém um conjunto de blocos lógicos programáveis e uma hierarquia de interconexões reconfiguráveis que permitem ligar fisicamente estes blocos. Os blocos lógicos são circuitos que contém a representação da função lógica desejada, que podem ser desde complexas funções combinacionais ou mesmo simples operações AND ou XOR, por exemplo. Outros elementos presentes nas FPGAs são os blocos de entrada/saída, que realizam a interface de entrada e saída dos blocos lógicos e o mundo externo. A figura 13 ilustra a arquitetura típica de uma FPGA. Nas FPGAs atuais pode-se encontrar também transmissores/receptores de alta velocidade, elementos de memória, blocos de entrada e saída de alta velocidade, blocos de processamento de sinais digitais, entre outros elementos (JODAS, 2010; ALTERA, 2012; FLOYD, 2006; BROWN; ROSE, 1996).

Existem diversos fabricantes de FPGAs, dentre elas: ALTERA, Xilinx, Actel, Quic-

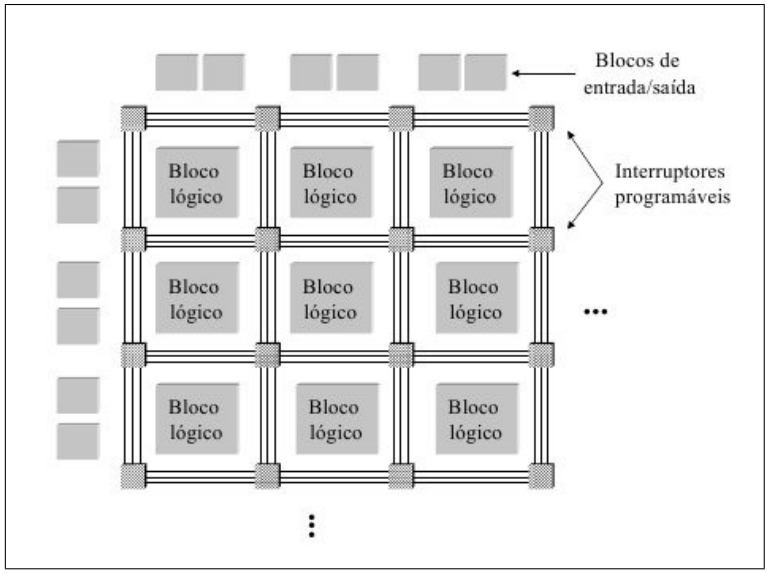

Figura 13: Arquitetura de uma FPGA

Fonte: Jodas (2010)

klogic, Cypress, entre outros (BROWN; ROSE, 1996). As FPGAs utilizadas neste trabalho são as da Altera, devido à grande experiência do grupo de pesquisa com estas FPGAs, além da grande gama de kits didáticos da Altera que o laboratório possui.

Atualmente, muitas pesquisas têm sido feitas para o emprego de FPGAs em RSSFs, dada a flexibilidade destes dispositivos e a execução mais rápida de certos algoritmos. Aliado a isto, o advento das *Low Power* FPGAs, como as plataformas IGLOO, da Actel, têm viabilizado ainda mais tais pesquisas (ROSELLO et al., 2011; DE LA PIEDRA et al., 2012).

2.7.2 LINGUAGEM DE DESCRIÇÃO DE HARDWARE

Uma linguagem de descrição de hardware, também denominada de HDL (*Hardware Description Language*), é um tipo especial de linguagem para se modelar hardware. Um sistema digital é, normalmente, constituído de partes menores que se interligam. Quando um sinal da entrada ou saída de uma destas partes se altera, as partes conectadas por este sinal sofrem as devidas mudanças. Estas mudanças ocorrem de forma concorrente e cada operação leva um certo tempo para completar, ou seja, o sinal leva um tempo para se propagar. Este tempo depende dos atrasos internos de cada parte. Estes aspectos ressaltam as características dos sistemas digitais, que não podem ser descritos com as linguagens de programação tradicionais, como C e JAVA, as quais modelam os sistemas de forma sequencial, ou seja, as operações ocorrem em sequência, respeitando uma ordem (CHU, 2006).

Um sistema digital pode ser descrito a partir de diferentes níveis de abstração e de diferentes pontos de vista. Uma HDL deve modelar fielmente e precisamente o circuito descrito. Entre as linguagens de descrição de hardware mais utilizadas podem-se citar: VHDL e Verilog (CHU, 2006).

Neste trabalho optou-se pela utilização do VHDL (*VHSIC* [3] *Hardware Description Language*). Esta linguagem de descrição de hardware surgiu na década de 1980, através de iniciativas do departamento de defesa dos Estados Unidos e hoje é amplamente utilizado também fora da área militar. Trata-se de um padrão aberto e independente de fabricante e é uma linguagem padronizada pelo IEEE. Sua primeira versão foi denominada VHDL 87 (IEEE 1076), que foi atualizada para a versão VHDL 93 (IEEE 1164) e VHDL 2002 e, atualmente, a última versão é o VHDL 2008 (PEDRONI, 2010; BROWN; VRANESIC, 2000).

Um projeto de hardware em VHDL é, geralmente, estruturado em três seções: declaração das bibliotecas (*library*) e pacotes (*package*), entidade (*entity*) e arquitetura (*architecture*). Na primeira seção adicionam-se as bibliotecas e pacotes, conjunto de declarações e subprogramas, que serão utilizados no projeto. Na entidade especificam-se as entradas e saídas do circuito projetado, assim como algumas constantes opcionais. A arquitetura, por sua vez, contém o código de descrição do hardware propriamente dito, onde se descreve como o circuito deve funcionar (PEDRONI, 2010; BROWN; VRANESIC, 2000).

Após o desenvolvimento do código VHDL, seguindo as especificações do projeto, deve-se utilizar um software específico para realizar as simulações e, posteriormente, a sintetização em hardware. Estes softwares, ao final do processo, geram arquivos específicos que são utilizados para gravar no dispositivo lógico programável (PEDRONI, 2010).

[3] VHSIC - *Very High Speed Integrated Circuits*

3 DESENVOLVIMENTO

3.1 INTRODUÇÃO

Um dos objetivos específicos desta dissertação é estudar os métodos de localização disponíveis na literatura, focando em métodos que sejam viáveis em RSSF e que permitam a um nó sensor da rede, de posição previamente desconhecida, descobrir a sua própria posição. Para que o nó sensor consiga determinar sua própria posição, o mesmo utiliza os valores de distâncias aos nós de referências que estão a seu alcance. Os valores de distância são determinados a partir da leitura do RSSI.

Neste capítulo será abordada, inicialmente, a modelagem do canal de comunicação sem fio. Em seguida, serão descritos os valores típicos de RSSI obtidos na prática e como estes valores podem ser utilizados para se determinar a distância entre os nós transmissor e receptor.

Através dos valores típicos de RSSI, o comportamento dos valores de distâncias foi analisado e, a partir de uma modelagem proposta para as variações nos valores de distâncias, foram feitas as avaliações dos desempenhos dos algoritmos.

Após a verificação das características, complexidade e desempenhos dos algoritmos, partiu-se, então, para a sua implementação em hardware de lógica programável. Nesta última seção serão apresentadas as análises de viabilidade das implementações dos algoritmos, análises dos tipos de dados empregados nas implementações e os aspectos relacionadas às abordagens combinacional e sequencial dos algoritmos e seu impacto no tempo de execução do processo de localização.

3.2 MODELAGEM DO CANAL DE COMUNICAÇÃO SEM FIO

A modelagem do canal de comunicação sem fio deve ser realizada antes da utilização dos algoritmos de localização, visto que é nesta etapa que são obtidas as estimativas de distâncias, que são as informações de entrada dos algoritmos. Neste sentido, deve-se ter um conhecimento a priori dos parâmetros de propagação antes de se estimar a posição. O chip CC2431, da

Texas Instruments, requer também que se especifiquem todos os parâmetros de propagação do canal antes de se aplicar o algoritmo de localização.

Zemek et al. (2008) apresenta uma metodologia para contornar essa necessidade do conhecimento prévio dos parâmetros de propagação, propondo um método iterativo para se estimar tais parâmetros. Em outra abordagem, Tennina et al. (2008) apresenta formas de se estimar estes parâmetros continuamente, para aplicação em ambientes com bastante movimentação. Todavia, neste trabalho será abordada a metodologia tradicional que supõe o conhecimento prévio dos parâmetros de propagação.

Neste contexto, esta seção tem como objetivos principais:

- Analisar os valores de medições reais de RSSI.
- Verificar a ordem de grandeza dos parâmetros de propagação do canal de comunicação sem fio.
- Verificar a ordem de grandeza dos erros nas estimativas de distâncias, para se ter uma ideia dos valores de erros a serem considerados nas simulações.

3.2.1 MEDIÇÃO DE RSSI

Muitos protocolos utilizam o valor do RSSI. Os protocolos IEEE 802.11, por exemplo, utilizam este indicador para definir políticas de associação (WANG et al., 2009), e o IEEE 802.15.4 utiliza este indicador para verificar a qualidade do enlace de comunicação sem fio (LEE et al., 2010).

Com o intuito de observar como a medição de RSSI pode ser feito na prática, escolheu-se um sistema disponível no mercado para realizar alguns experimentos. O sistema escolhido foi o XBee, um produto da empresa norte-americana Digi International compatível com a tecnologia de comunicação sem fio ZigBee.

A tecnologia ZigBee é um padrão de comunicação padronizado pela ZigBee Alliance e se destina às aplicações de redes de sensores sem fio de baixo custo e baixo consumo de potência, suportando redes com topologia em árvore ou do tipo *mesh*. Apresenta também baixa taxa de transmissão de dados e pode operar na faixa ISM (Industrial, Científico e Médico) de 2.4 GHz (PINEDO-FRAUSTO; GARCIA-MACIAS, 2008).

Estes módulos foram escolhidos principalmente devido à sua facilidade de configuração e utilização. As versões da linha XBee escolhidas foram o XBee S2 e o XBee-PRO

S2. Ambos hardware possuem estruturas similares, contendo, basicamente, um microcontrolador dedicado para executar suas tarefas internas, transceivers para a comunicação sem fio e pinos de I/O (*Input*/*Output*). A principal diferença entre estes módulos é quanto à potência de transmissão. O XBee-PRO S2 apresenta tamanho maior que o XBee S2 e pode transmitir com mais potência (até 50 mW para o XBee-PRO S2 e até 2 mW no caso do último módulo). A configuração, entretanto, é feita da mesma forma para ambas versões. Como ilustrado na figura 14, os módulos podem ser utilizados com diferentes antenas, tal como a antena *Whip* (antena Wire), antena *Chip*, antena PCB, e conector U.FL (FALUDI, 2010; DIGI INTERNATIONAL, 2010).

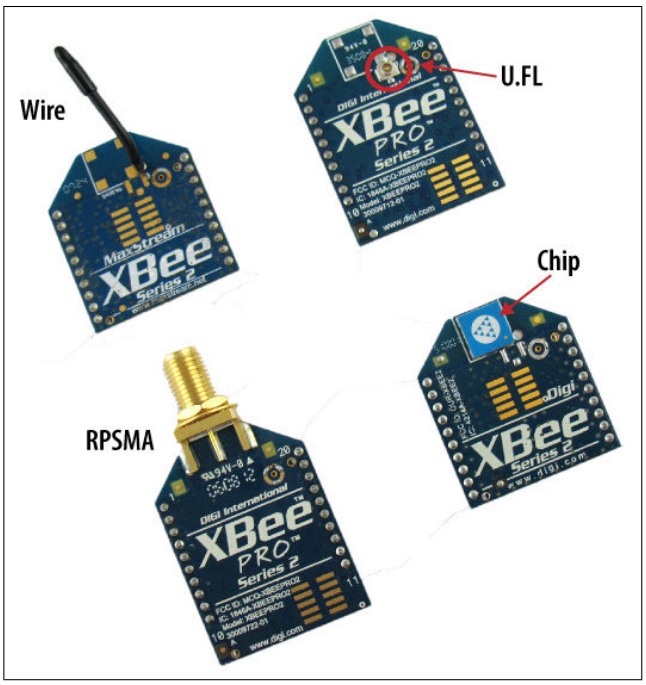

Figura 14: Variantes do XBee

Fonte: Faludi (2010)

Quanto à configuração dos módulos o software utilizado para realizar a interface com o XBee é o X-CTU. Neste software é possível alterar o firmware dos módulos XBee, configurando, por exemplo, se o módulo XBee terá função de coordenador da rede, roteador ou dispositivo final. Estas funções se referem aos elementos de uma rede ZigBee. Além disso, a partir do XCTU pode-se configurar a potência de transmissão, definir as funções dos pinos de I/O, realizar testes de conexão, verificar a topologia da rede, entres outras funções.

A interface física entre o módulo XBee é feita através de UART (*Universal Asynchronous Receiver*/*Transmitter*) e o desenvolvimento de software de aplicação é facilitado, dadas as diversas bibliotecas abertas disponíveis. Alguns exemplos são o MFToolkit, para a plataforma

.NET, e o xbee-api, para se programar com linguagem C e JAVA.

Quanto ao valor do RSSI, existem, basicamente, duas maneiras de obtê-los dos módulos: a primeira delas é através do envio de um comando ao XBee (comando DB), o qual retorna o valor do RSSI em 8 bits. A outra maneira de verificar este indicador é através de um sinal PWM (*Pulse-Width Modulation*) disponibilizado por um pino de I/O do XBee. Nesta abordagem o valor do RSSI é calculado através de uma equação matemática fornecida pelo fabricante, envolvendo o ciclo de trabalho do sinal PWM. Ambos métodos disponibilizam apenas o valor do RSSI correspondente ao último salto da comunicação, ou seja, nenhuma destas abordagens disponibilizam indicações da situação de todo o caminho percorrido em uma comunicação com múltiplos saltos (DIGI INTERNATIONAL, 2010).

De forma a facilitar a verificação da conectividade dos módulos, o software X-CTU disponibiliza a ferramenta Range Test. Para utilizar esta ferramenta foram necessários os seguintes elementos: um módulo XBee conectado a um computador com o X-CTU instalado e um outro módulo XBee, equipado com bateria, que será movimentado. Esta ferramenta faz com que o módulo XBee conectado no computador envie pacotes do tipo *loopback*, fazendo com que o módulo que receba este pacote reenvie este pacote para o módulo de origem. Desta forma, quando o módulo de origem recebe o pacote de volta, a ferramenta Range Test do X-CTU consegue obter o valor do RSSI da transmissão feita pelo nó móvel ao nó conectado ao computador. A tela de interface do X-CTU para se visualizar o valor do RSSI está indicado na figura 15 (DIGI, 2009; DIGI INTERNATIONAL, 2010).

Quanto aos locais, foram feitas medições em dois ambientes diferentes. O primeiro foi um experimento em um ambiente interno, mais especificamente, em uma sala de aula com computadores. O outro ambiente foi um corredor livre. Para ambos locais foram utilizados dois tipos de XBee: um XBee S2 com antena Wire e o outro um XBee-PRO S2 com antena RPSMA.

As medições na sala de aula foram feitas colocando-se o módulo móvel sobre as mesas, ficando na mesma altura do módulo conectado ao computador. Foram feitas medições em linha reta, de um lado da parede até a parede oposta. Além disso, as medições foram feitas com a sala vazia. Foram feitas medições aproveitando as distâncias dos espaçamentos entre as mesas, visto que os módulos ficavam em cima da mesas. Para cada valor de distância, foram coletadas três medidas e feito uma média aritmética. Os resultados obtidos estão ilustrados na tabela 2.

O outro experimento, agora em lugar aberto, foi feito em um corredor da UTFPR. As medições foram feitas com o corredor sem pessoas transitando e os módulos com linha de visada direta. Tanto o módulo móvel como o conectado ao computador (um *netbook*) foram segurados por uma pessoa. Assim como no outro experimento, foram coletadas três medidas

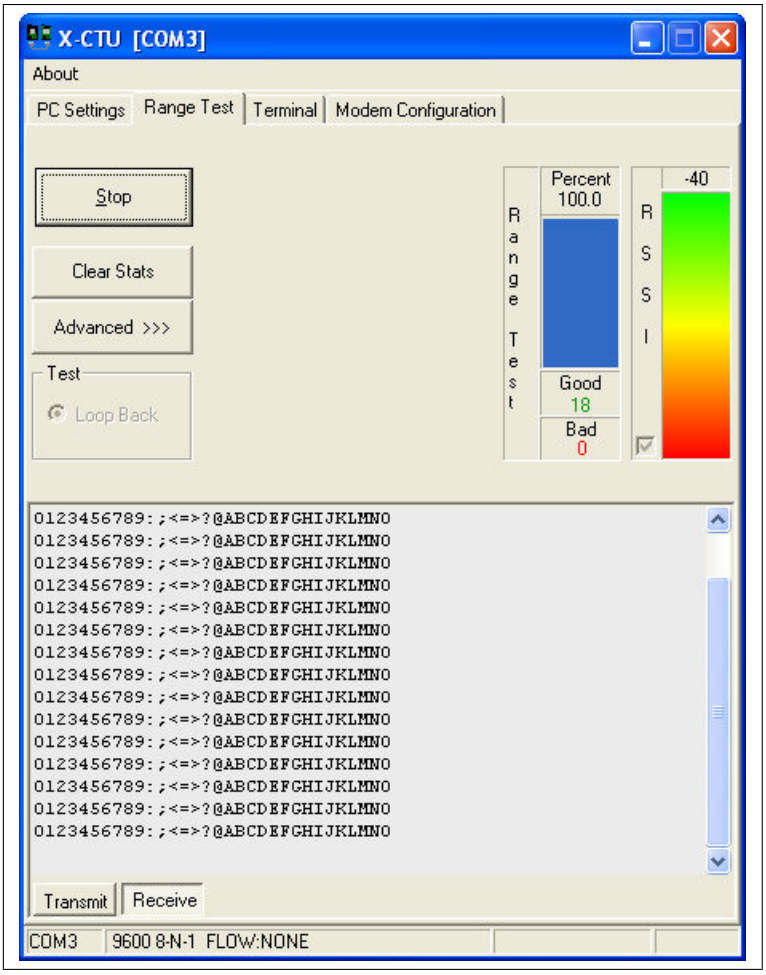

Figura 15: Tela de interface com o usuário para a verificação do RSSI

Fonte: Digi (2009)

Tabela 2: Medição de RSSI na sala de aula

Distância (m)	RSSI (dBm)	
	XBee S2	XBee-PRO S2
2,80	-46	-61
4,42	-55	-65
6,07	-60	-68
7,75	-63	-71

para cada valor de distância e foi calculada a média aritmética dos valores. Estes valores estão ilustrados na tabela 3.

Com o intuito de analisar o comportamento do RSSI com a distância, plotou-se nas figuras 16 e 17 o gráfico RSSI x distância para os cenários analisados. Na verdade, plotou-se os gráficos de RSSI x 10.log(distância), como sugerido na equação 22 da seção 2.4.4. Neste modelo, o expoente de perda de percurso, representado pela letra γ, indica a inclinação da curva. Os valores de γ obtidos nestes experimentos estão coerentes com os indicados na tabela 1. O

Tabela 3: Medição de RSSI no corredor

Distância (m)	RSSI (dBm)	
	XBee S2	XBee-PRO S2
6,30	-50	-42
12,30	-58	-46
21,70	-61	-49
27,70	-68	-55
37,00	-75	-64
43,30	-67	-51
51,30	-85	-74

ajuste dos pontos medidos através de regressão linear indicou valores de R^2 (Coeficiente de Determinação) de 0,9907 para o XBee S2 e 0.9947 para o XBee-PRO S2, no experimento na sala de aulas, e 0,9124 para o XBee S2 e 0,8398 XBee-PRO S2, nas medições no corredor. Entretanto, este bom ajuste foi possível graças ao fato das medições terem sido realizadas com os ambientes vazios e em pequenas distâncias. Além disso, poucos pontos foram medidos. Apesar disso, os experimentos realizados proporcionaram comprovar na prática o comportamento do RSSI. Como obter informações de distâncias a partir dos valores de RSSI será mais bem trabalhado na seção 3.2.2.

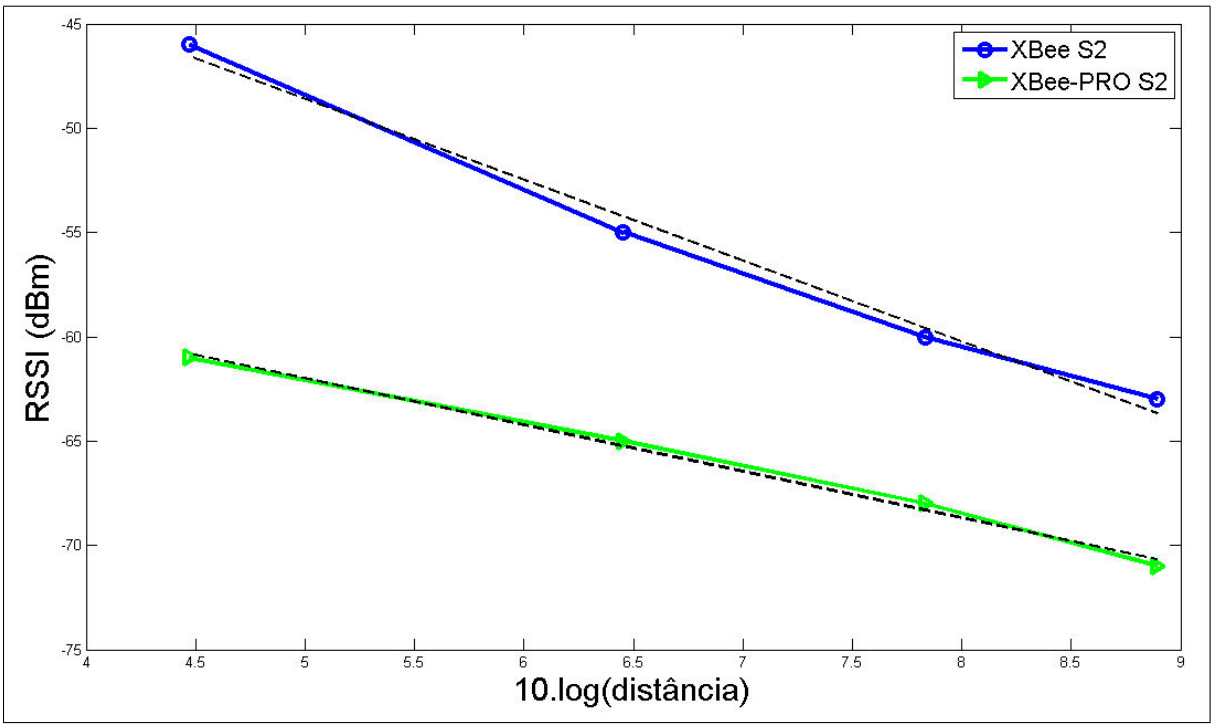

Figura 16: Medição de RSSI realizada na sala de aula

Como mostrado nas figuras 16 e 17, o valor de RSSI tende a diminuir com o aumento da distância, porém, não apresenta um comportamento perfeitamente linear, desejável para se ter boas estimativas de distâncias. Outro ponto é que foram coletadas poucas amostras, pois

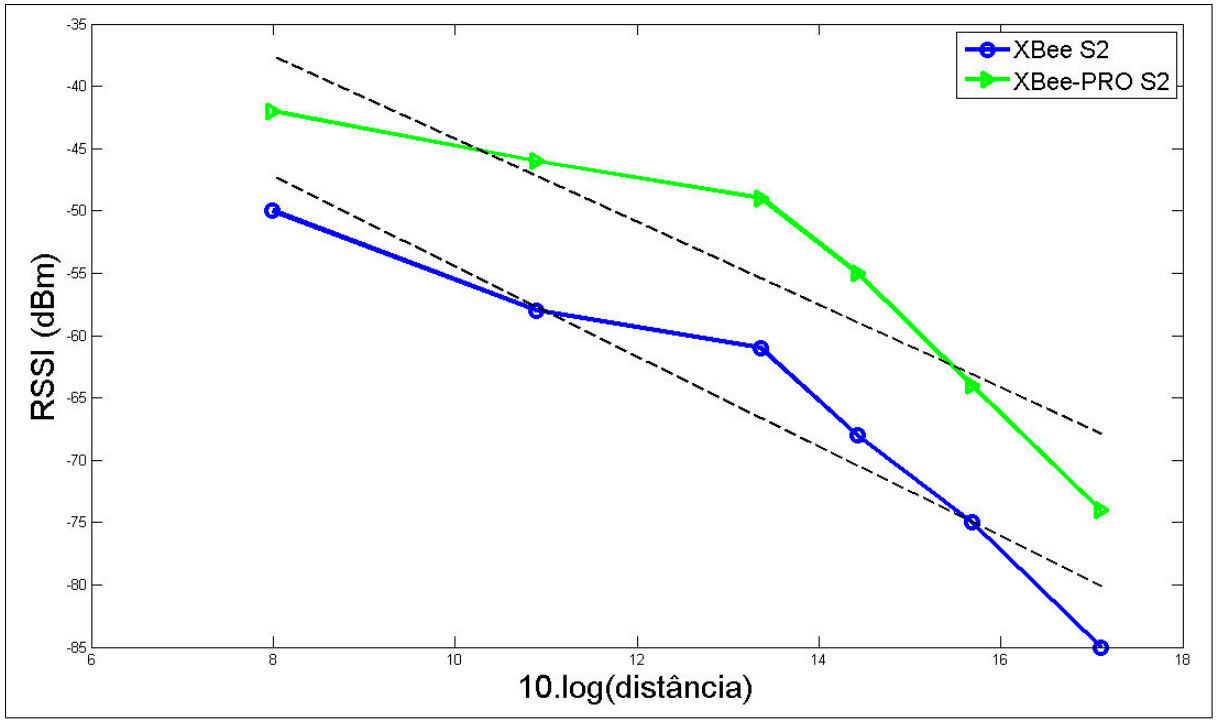

Figura 17: Medição de RSSI realizada no corredor

o objetivo não era realizar o mapeamento completo dos locais, nem o aprofundamento do estudo do ambiente. Vale ressaltar que os experimentos com o XBee teve como objetivo apenas exemplificar o processo de medição de RSSI, apresentando a facilidade com que a informação é disponibilizada, visto que o dado já é utilizado no protocolo de comunicação.

Comprovou-se também que a medição de RSSI apresenta grande flutuação de valores. Outro problema é a pouca resolução das medidas, pois o valor de RSSI costuma ser especificado com 8 bits, além da forte dependência com a dinâmica do ambiente. Análises considerando-se a dinâmica do ambiente não está no escopo deste trabalho, todavia, maiores informações podem ser encontradas, por exemplo, em Zemek et al. (2008).

Para realizar um estudo mais aprofundado do processo de conversão de RSSI em informação de distâncias, que são os parâmetros de entrada dos algoritmos de localização deste trabalho, necessita-se de maior volume de medições. Nesse sentido, nas simulações e estudos deste trabalho utilizou-se os valores medidos por Zanca et al. (2012). Neste trabalho mencionado, com equipamentos apropriados, foram coletadas grandes quantidades de medição de RSSI, permitindo realizar um maior mapeamento dos ambientes. O experimento escolhido para a análise foi realizado no interior do Laboratório IGNET Research, da Universidade de Padova. As medições foram feitas com 48 nós sensores EYES-IFXv2 instalados a 50 centímetros do teto. Foram coletadas no total 77.331 medidas de potências recebidas.

3.2.2 CONVERSÃO DE RSSI EM INFORMAÇÃO DE DISTÂNCIA

Como apresentado na seção 2.4.4, a equação 23 pode ser utilizada para modelar os efeitos da perda de percurso e do sombreamento que ocorrem no canal de comunicação sem fio. Para se estimar a distância a partir dos valores de RSS, determina-se, primeiramente, os parâmetros γ e $Pt + K$ da equação 22, obtendo-se o modelo do canal apenas com as perdas de percurso. Considerando múltiplas leituras de RSS para um mesmo ponto, calcula-se, então, a média destes valores para amenizar as variações das leituras das potências recebidas. Estes valores médios são os pontos utilizados para o cálculo da equação da reta que melhor se ajusta às medidas, por exemplo, através de regressão linear.

O termo relativo ao sombreamento, ψ_{dB}, responsável pela variação aleatória nos valores de potências recebidas, pode ser estimado a partir da equação 71, onde $Pr_{medido}(d_i)$ é o valor da potência recebida na i-ésima posição e $Pr'(d_i)$ é a potência estimada para a i-ésima posição, utilizando-se a equação da perda de percurso determinada no passo anterior.

$$\sigma_{\psi}^2 = \frac{1}{N}\sum_{i=1}^{N}\left[Pr_{medido}(d_i) - Pr'(d_i)\right]^2 \tag{71}$$

Partindo da equação 23, dada uma medição de RSS, pode-se calcular a distância entre o transmissor e o receptor de acordo com a equação 73. Nesta equação, d_{real} é a distância que resultaria se as medições não fossem afetadas pelo sombreamento.

$$d_{real} = d_0.10^{(P_t+K-P_r)/10\gamma} \tag{72}$$

$$\widehat{d} = d_0.10^{\frac{P_t+K+\psi-P_r}{10\gamma}} = d_{real}.10^{\frac{\psi}{10\gamma}} \tag{73}$$

Como mostrado em Mao et al. (2007) e Patwari et al. (2003), a equação 73 é um estimador enviesado (não centrado). Um estimador não enviesado, apresentado nestes trabalhos citados, é dado pela equação 74.

$$d_{est} = \widehat{d}\,.\,e^{-0,5\sigma_{\psi}^2\left(\frac{ln10}{10\gamma}\right)^2} \tag{74}$$

Para ilustrar a utilização das formulações apresentadas nesta seção e verificar os resultados do ajuste, foram utilizadas as medições de RSSI feitas por Zanca et al. (2012), como comentado no final da seção 3.2.1. Considerando estas medições, foi realizado o ajuste da reta

por regressão linear e os parâmetros encontrados foram: $\gamma = 1.64$, $P_t + K = -44,23$ dBm e $\sigma_\psi = 7,30$ dB. O valor de γ encontrado está coerente com os valores indicados na tabela 1, para ambientes similares a escritórios. O ajuste, juntamente com as medições, está ilustrado na figura 18.

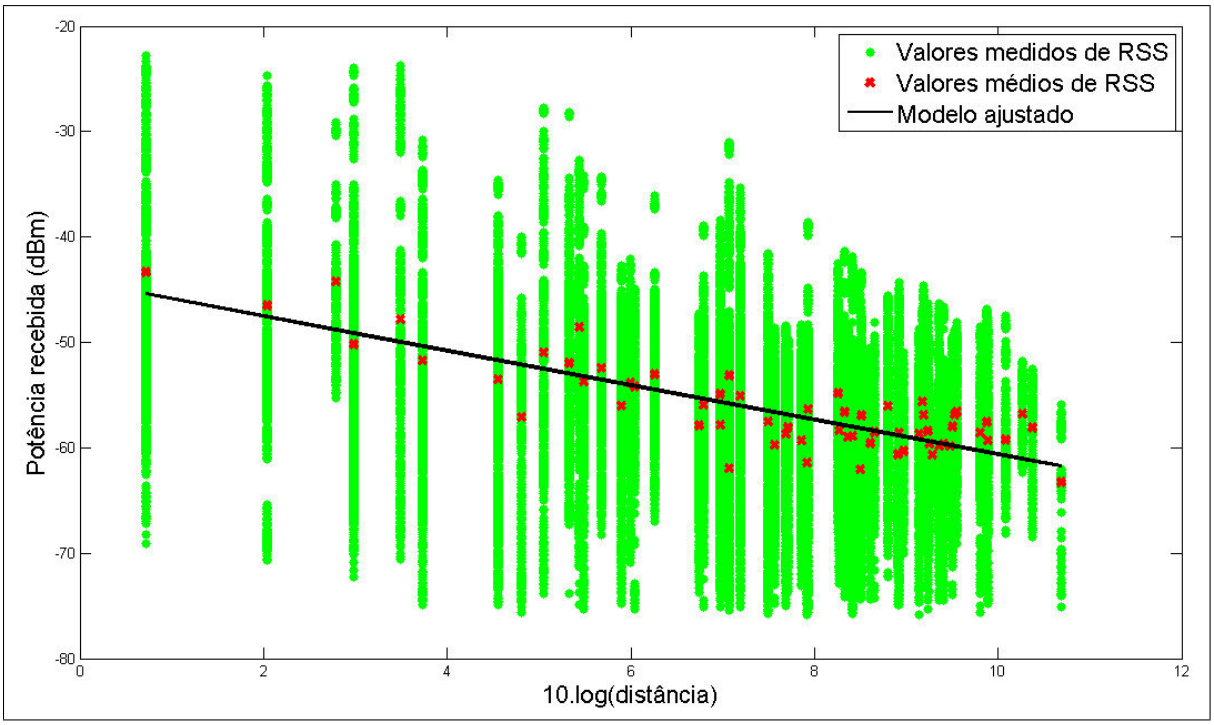

Figura 18: Modelo para estimar as distâncias entre o nó transmissor e receptor

Os erros de distâncias, comparando-se os valores medidos com os valores estimados pelo modelo, estão apresentados na figura 19. Nota-se que o modelo ajustado consegue estimar com erro da ordem de 2 metros para a maioria dos pontos, contudo, ocorrem também erros maiores em vários pontos. Para estimar a variância da amostra, utilizou-se a equação 75 e obteve-se o valor $s_d^2 = 10,2$ para a amostra de distâncias considerada.

$$s_d^2 = \frac{1}{n-1} \sum_{i=1}^{n} \left(d_i - \overline{d}\right)^2 \tag{75}$$

Neste sentido, os algoritmos de localização devem estar preparados para suportar as variações nos valores de distâncias, caso contrário, as estimativas de posição podem se tornar muito imprecisas. A figura 19 pode ser utilizada para se obter um modelo matemático para o erro nos valores de distâncias. Tendo-se um modelo matemático, pode-se aumentar ou diminuir a intensidade do erro nos valores de distâncias e verificar a sensibilidade e robustez dos algoritmos com esse erro.

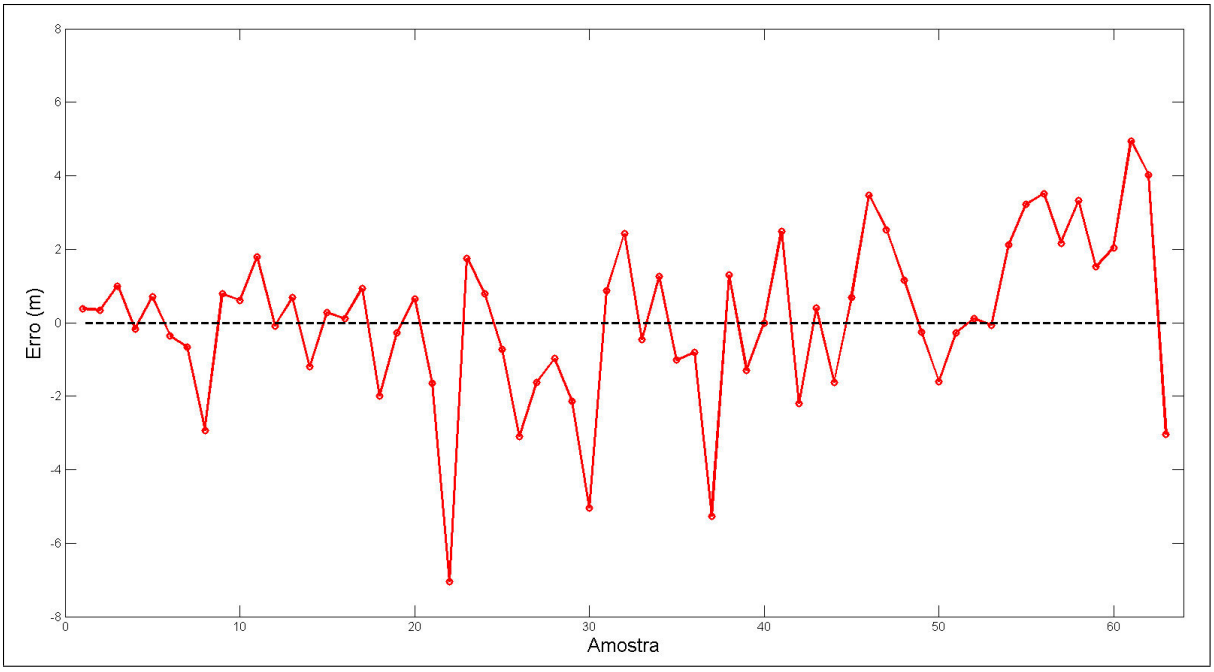

Figura 19: Erros das estimativas de distâncias

3.3 AVALIAÇÃO DO DESEMPENHO DOS ALGORITMOS

Antes de implementar os algoritmos em linguagem de descrição de hardware, uma etapa de avaliação dos algoritmos é crucial a fim de se verificar a complexidade de execução dos algoritmos e os erros no cálculo de posição de cada abordagem. Os algoritmos foram avaliados utilizando-se o software MATLAB.

As simulações desta seção têm como objetivo principal a análise do comportamento dos algoritmos em três cenários de teste:

1. Situações hipotéticas em que as estimativas de distâncias não apresentam erros.
2. Situações em que as estimativas de distâncias apresentam erros aleatórios.
3. Análise do desempenho dos algoritmos quando utilizados refinamentos nas estimativas de distâncias.

3.3.1 IMPLEMENTAÇÃO DOS ALGORITMOS

O algoritmo baseado em Lateração foi implementado utilizando-se a abordagem dos mínimos quadrados, apresentado na equação 35. Vale ressaltar, que para o caso de três NRs a equação 35 recai nas nas equações 30 a 34.

O algoritmo Min-Max foi implementado conforme explicado na seção 2.5.2. A implementação foi simples, pois já existem funções do MATLAB para o cálculo de máximos e mínimos de um conjunto de valores, que são as operações mais custosas deste algoritmo.

Para a avaliação do algoritmo baseado no MLE foi utilizada a abordagem simplificada com distribuição Log-Normal, conforme apresentado na equação 51. Para realizar a minimização das equações foi utilizado o método de Nelder-Mead, disponível na biblioteca do MATLAB. Este método de otimização simplifica o processo de minimização, pois realiza a busca direta pelo valor, sem a necessidade de se computar a derivada das funções. Para este processo de otimização foi configurado o número máximo de iterações para 10^5 e, para a condição de parada, foi considerada a tolerância de 10^{-8} no valor da função.

Todos estes algoritmos implementados estão disponibilizados no apêndice A.

3.3.2 CENÁRIO DE TESTES

Para a avaliação destas situações citadas foi considerado o mesmo cenário de testes para todos os algoritmos, apresentado a seguir:

- Os NRs foram posicionados uniformemente sobre uma circunferência de raio R. Esta imposição é interessante, visto que em todos os casos existirão ao menos três pontos não colineares, evitando que o posicionamento dos NRs necessite de verificações adicionais, diminuindo, assim, o tempo de simulação.
- O NPD, por sua vez, foi sempre sorteado aleatoriamente dentro da circunferência.
- As distâncias reais do NPD aos NRs foram calculadas através da equação 76 e armazenadas em um vetor para comparações posteriores.

$$d_{real} = \sqrt{(x_{NR} - x_{NPD})^2 + (y_{NR} - y_{NPD})^2} \tag{76}$$

- Inicialmente, as distâncias utilizadas nos testes foram as reais. Posteriormente, foram utilizados valores de distâncias com erros, com o intuito de verificar a robustez e o funcionamento destes algoritmos em cenários mais próximos do que ocorre na prática.
- Com estes dados, os algoritmos foram aplicados e os erros de posicionamento foram calculados através da equação 77, que é a distância entre o ponto esperado e o ponto estimado.

$$Erro = \sqrt{(x_{NPD} - x_{estimado})^2 + (y_{NPD} - y_{estimado})^2} \tag{77}$$

- As simulações foram realizadas empregando-se de 3 a 35 NRs. Para acrescentar erro nos valores de distâncias foram somados a estes valores erros com distribuição normal e média nula. Em cada caso foram executadas 10.000 iterações e o erro médio de cada algoritmo foi calculado para realizar as comparações.

A figura 20 ilustra o cenário de testes descrito para o caso com 3 NRs e raio de circunferência igual a 10 metros. Como indica a figura, o NPD é aleatoriamente sorteado dentro da circunferência e as distâncias medidas pelo NPD em relação ao NR_1, NR_2 e NR_3 são, respectivamente, d_1, d_2 e d_3.

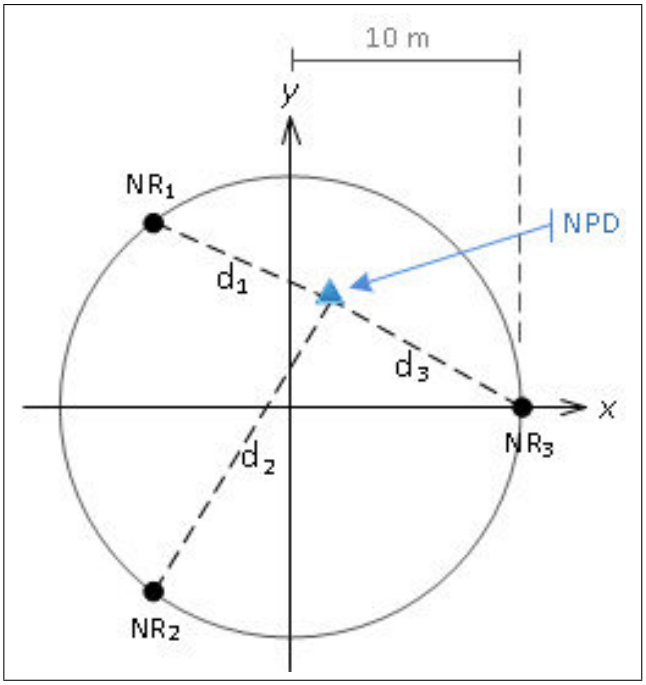

Figura 20: Cenário de testes com 3 nós de referência

3.3.3 SIMULAÇÕES COM VALORES IDEAIS

As simulações com valores ideais retratam as situações nas quais o NPD estima perfeitamente todos os valores de distâncias entre os NRs. O objetivo destas simulações é, principalmente, analisar se os algoritmos considerados conseguem estimar sem erro a posição do NPD, dadas estimativas corretas de distâncias. Alternativamente, deseja-se verificar se estes algoritmos, tendo como entradas as distâncias corretas, são capazes de calcular corretamente a posição do NPD. Estas simulações foram realizadas utilizando-se os algoritmos descritos na seção 3.3.1 e configurando-se os erros de distâncias para zero.

Nesta situação, verificou-se que o método baseado na Lateração conseguiu estimar exatamente a posição do NPD, como era esperado analiticamente.

O algoritmo MLE também conseguiu estimar corretamente a posição do NPD. A rigor, o MLE estimou corretamente a posição do NPD de acordo com a tolerância configurada para o critério de parada da otimização. Nestes testes os erros apresentados ficaram na ordem de 10^{-6} e 10^{-7}, ou seja, praticamente nulo. A tolerância poderia ter sido configurada com valores ainda menores, todavia, isto aumentaria consideravelmente o tempo de simulação.

Já o método Min-Max não conseguiu estimar corretamente a posição do NPD, mesmo com todas as entradas de distâncias corretas. Isto se deve ao fato da aproximação das circunferências por quadrados acabar gerando grandes áreas de superposição. Os erros, contudo, diminuíram à medida que mais NRs foram considerados e chegaram a valores próximos de zero. Esta situação está ilustrada na figura 21.

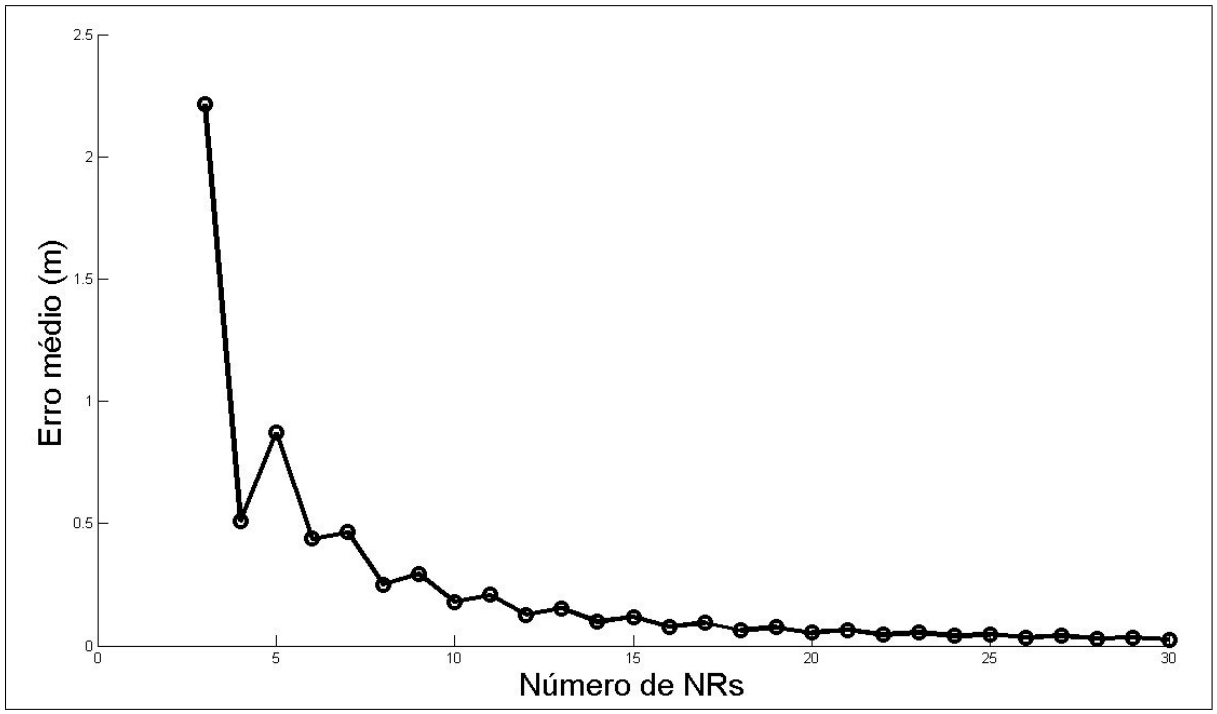

Figura 21: Desempenho do método Min-Max com valores ideais de distâncias

Observa-se na figura 21 que, no caso do método Min-Max, o comportamento do erro é instável quando poucos NRs estão presentes, com o erro ora aumentando e ora diminuindo. Estes erros tenderam a se estabilizar à medida que mais NRs foram considerados. Nota-se também que, mesmo fornecendo estimativas boas de distâncias (ideais neste caso), o método Min-Max pode não apresentar desempenhos interessantes, especialmente com baixas quantidades de NRs. Contudo, observou-se também que este algoritmo mostra bons resultados quando vários NRs são considerados.

3.3.4 SIMULAÇÕES COM ERRO NOS VALORES DE DISTÂNCIAS

Esta seção tem por objetivo investigar o comportamento dos algoritmos quando os valores de distâncias de entrada apresentam erros. Deseja-se verificar a grandeza dos erros de posição, assim como o quão robusto é cada algoritmo. Para tanto serão considerados os mesmos algoritmos da seção anterior, mas agora com os valores de entrada corrompidos com erro.

Para gerar os valores de erros de distância que sejam consistentes com os que ocorrem em cenários reais, serão considerados os comportamentos observados na seção 3.2.2. Analisando-se a figura 19, pode-se verificar que os erros causaram tanto variações positivas quanto negativas nas estimativas de distância. De forma a simplificar a modelagem matemática do erro, considerou-se que o erro apresenta distribuição gaussiana e média nula, embora se observe que, na realidade, os erros das estimativas de distância apresentaram também erro sistemático não nulo. Considerando a distribuição do erro com média nula, encontrou-se um valor maior para a variância.

Considerando novamente a figura 19, simulou-se o comportamento dos algoritmos quando sujeitos à erros daquela ordem de grandeza. Para estas simulações foi considerado que os erros de distâncias se distribuem segundo distribuição gaussiana com média nula e variância igual a 11, parâmetros estes estimados com as amostras da figura 19. Os resultados das simulações dos algoritmos, observando-se a variação do erro médio com o aumento dos NRs está ilustrado na figura 22.

Como publicado em diversos trabalhos correlatos, as estimativas de posição tendem a melhorar à medida que mais NRs são considerados. Destas simulações pode-se verificar que o algoritmo baseado no MLE é o que apresentou os melhores resultados, seguido do método Min-Max e, por fim, o método baseado na Lateração. Desempenhos similares foram também verificados nas simulações feitas em Zanca et al. (2008). No trabalho anterior, porém, o algoritmo baseado no MLE apresentou desempenho superior ao ilustrado nas presentes simulações. Duas situações poderiam explicar este acontecimento: a primeira consideração se faz devido ao uso da equação simplificada (equação 51) e não a equação completa apresentada em Patwari et al. (2001), de onde os autores Zanca et al. (2008) se basearam; a outra consideração é que em muitas iterações o algoritmo implementado não conseguiu convergir para o valor com as especificações desejadas, mesmo após realizar 10^5 iterações. Estas duas situações certamente influenciaram no desempenho do algoritmo MLE implementado neste trabalho.

Outro fato observado no trabalho de Zanca et al. (2008), e também nas simulações desta seção, é que para poucos NRs o erro diminui mais rapidamente com o aumento deste,

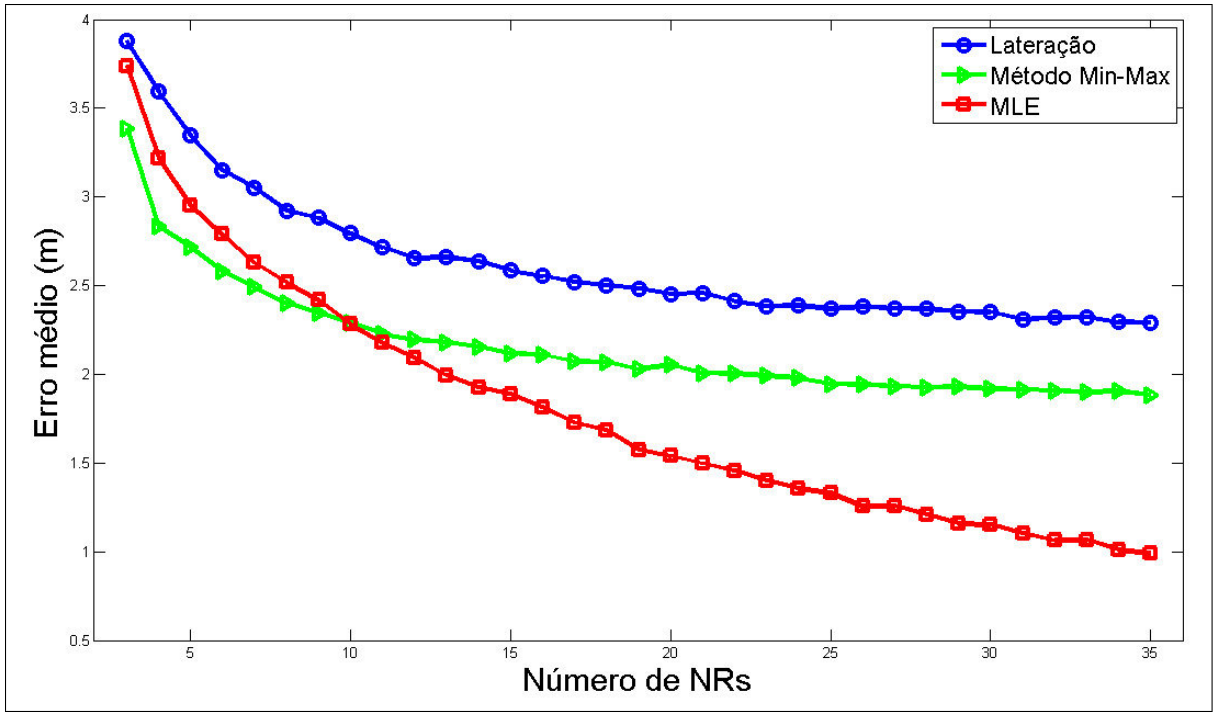

Figura 22: Comparação do desempenho dos algoritmos de localização

contudo, o desempenho dos algoritmos tende a saturar em um patamar e não apresenta mais melhora significativa com o aumento do número de NRs.

De forma a avaliar a sensibilidade dos algoritmos com a amplitude do erro, configurou-se situações em que os algoritmos estivessem sujeitas a erros com variâncias 1, 5 e 10. Os resultados obtidos estão ilustrados na figura 23. As curvas em azul correspondem às simulações com variância 1, as curvas em vermelho correspondem às simulações com variância 5 e as verdes são as simulações com variância 10.

Como esperado, ao passo que se aumenta a variância do erro, as curvas tendem a se deslocar para cima, apresentando erros de posição cada vez maiores. Nota-se que o método baseado na Lateração é bastante sensível ao erro das medições de distância. Observa-se também que, ao passo que o erro das distâncias se torna maior, o algoritmo MLE apresenta desempenhos cada vez mais superiores frente ao método da Lateração e Min-Max.

Analisando-se o desempenho dos algoritmos estatisticamente e observando-se o comportamento de suas Funções Distribuição Acumulada (FDA), foi possível ter uma ideia de como são as ocorrências dos erros em cada algoritmo. A FDA de uma variável aleatória X associa a cada valor de X a probabilidade deste valor ser menor ou igual a x. Pode-se denotar esta situação como sendo: $F(x) = P(X \leq x)$ (GONÇALVES JÚNIOR, 2012). Todavia não se sabe a priori qual é o tipo de distribuição das amostras de erro obtidas. Neste sentido, foram analisadas as Funções de Distribuição Acumulada Empíricas, que são as Funções Distribuição Acumulada

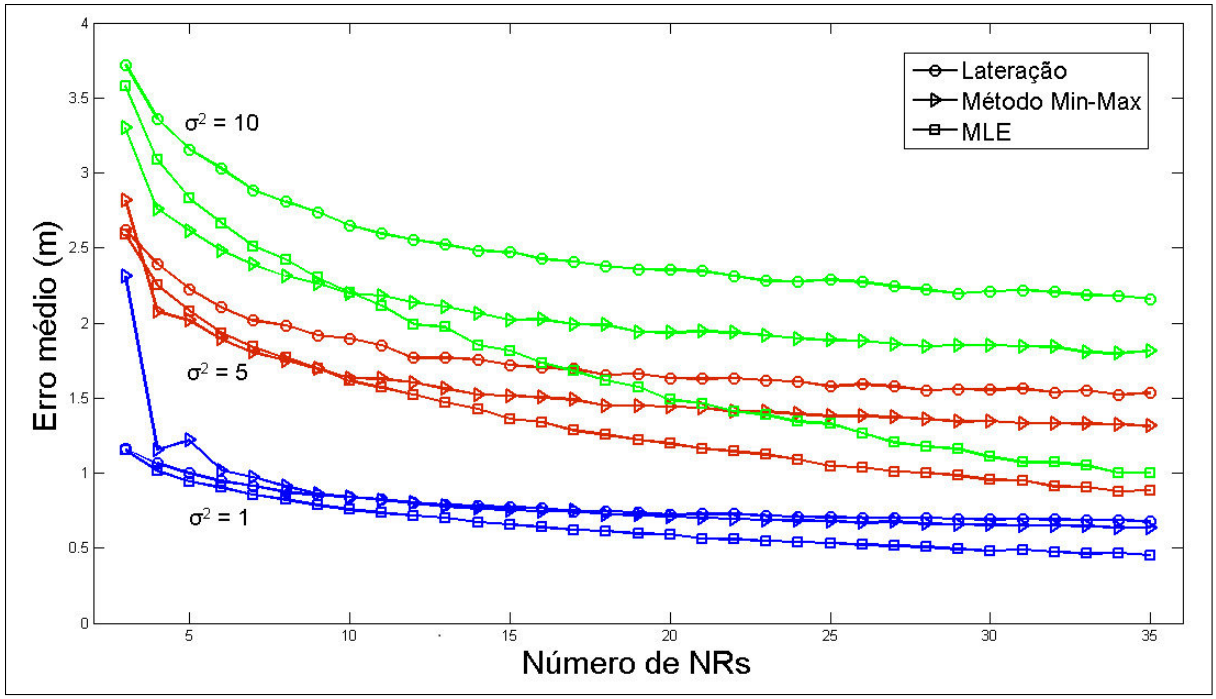

Figura 23: Desempenho dos algoritmos com o aumento da intensidade do erro

associadas com as medidas empíricas. Neste contexto, utilizou-se no MATLAB a função *ecdf*, de modo a estimar a Função Distribuição Acumulada Empírica das amostras, através da estimativa de Kaplan-Meier (THE MATHWORKS, 2012a; MARTÍNEZ-ESPINOSA et al., 2004).

A figura 24 ilustra as curvas de FDA Empíricas obtidas para as simulações com o erro apresentando distribuição gaussiana de média nula e variância 11. Análises similares são apresentadas em Luo et al. (2010) e Zanca et al. (2008).

A partir da figura 24 pode-se verificar que o algoritmo MLE é o que consegue estimar as posições com os menores erros. Para estas simulações, o menor erro de distância obtido pelo algoritmo MLE foi de 0,99 metro. Para o método Min-Max o menor erro encontrado foi de 1,88 metros e para o algoritmo da Lateração o menor erro foi de 2,29 metros.

De acordo com as análises realizadas nesta seção pode-se concluir que o algoritmo baseado na Lateração é o que apresentou os piores resultados.

Destas análises pôde-se observar também que o método Min-Max apresenta resultados interessantes quando vários NRs são considerados, chegando a apresentar resultados próximos ao do MLE para valores entre 5 e 10 NRs. Uma característica interessante do algoritmo Min-Max é a sua tendência de localizar o NPD no centro da área do retângulo formado pelos valores de distâncias, limitando-se, assim, o erro máximo de localização para a metade da distância entre os dois NRs mais afastados (ZANCA et al., 2008).

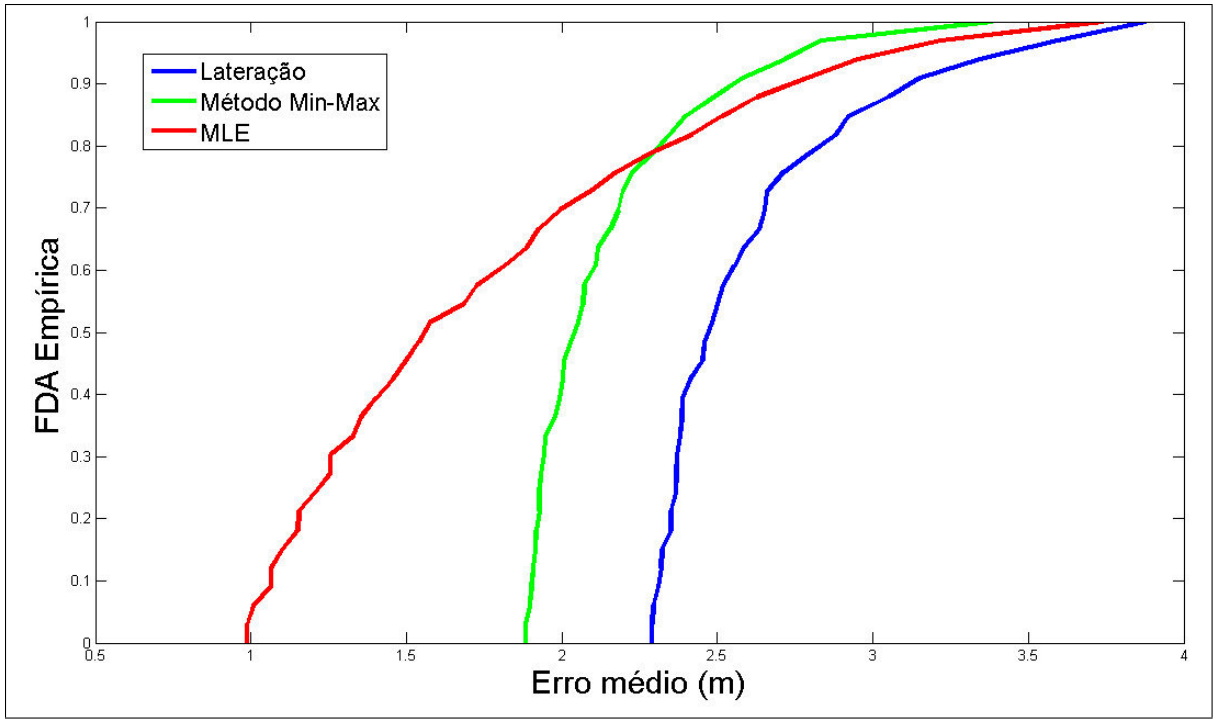

Figura 24: Curvas das Funções Distribuição Acumulada Empírica

Quanto ao algoritmo MLE, foi verificado que este é o algoritmo que apresentou os melhores resultados. Contudo, o MLE pode vir a ser inviável em diversos sistemas, dado o seu custo de implementação e tempo de execução. Além disso, para valores práticos de NRs, em torno de 10 NRs, este algoritmo apresenta resultados pouco melhores que os resultados do método Min-Max, que é de implementação muito mais simples e que exige muito menos processamento. Desta forma, a implementação do algoritmo MLE pode não ser a melhor opção para aplicações em RSSF.

De forma geral, os resultados desta seção estão comparáveis ao método de localização *indoor GPS*, que permite realizar a localização com erros da ordem de 1,5 a 2,0 metros (LUO et al., 2010).

3.3.5 SIMULAÇÕES COM REFINAMENTOS NOS VALORES DE DISTÂNCIAS

Esta seção tem como objetivo verificar a melhora no desempenho dos algoritmos quando as estimativas das entradas são melhoradas com a aplicação de algoritmos específicos. Na seção 2.6 foram abordados dois métodos: Filtro de Kalman e a utilização dos Determinantes de Cayley-Menger.

Embora o filtro de Kalman seja bastante conhecido e frequentemente utilizado em aplicações envolvendo rastreamentos, é uma abordagem que requer várias multiplicações de

matriz, o que é inviável para o nível de processamento dos nós de uma RSSF. Neste sentido, para as simulações desta seção foi optado pelo método dos Determinantes de Cayley-Menger, por ser de implementação mais simples e, também, por se ter poucas informações disponíveis na literatura a respeito de seu desempenho, motivando uma investigação.

O código MATLAB implementado, utilizando-se agora as distâncias melhoradas pela utilização dos Determinantes de Cayley-Menger, também está disponível no apêndice A. O código foi baseado no exemplo de aplicação apresentado pelos autores Cao et al. (2006).

Uma desvantagem desta abordagem é a necessidade de se minimizar funções, recaindo no mesmo problema do algoritmo MLE. Contudo, é uma boa oportunidade para verificar a capacidade desta abordagem de melhorar as estimativas de distância, visto que melhorando-se estas estimativas, o desempenho dos algoritmos de localização também tende a melhorar, como mostrado na seção 3.3.4.

A figura 25 ilustra o quanto de ganho se obtém com a utilização dos Determinantes de Cayley-Menger, ao se variar a variância do erro de medição. As curvas em azul, verde e vermelho correspondem, respectivamente, aos resultados obtidos pelo algoritmo da Lateração, método Min-Max e algoritmo MLE.

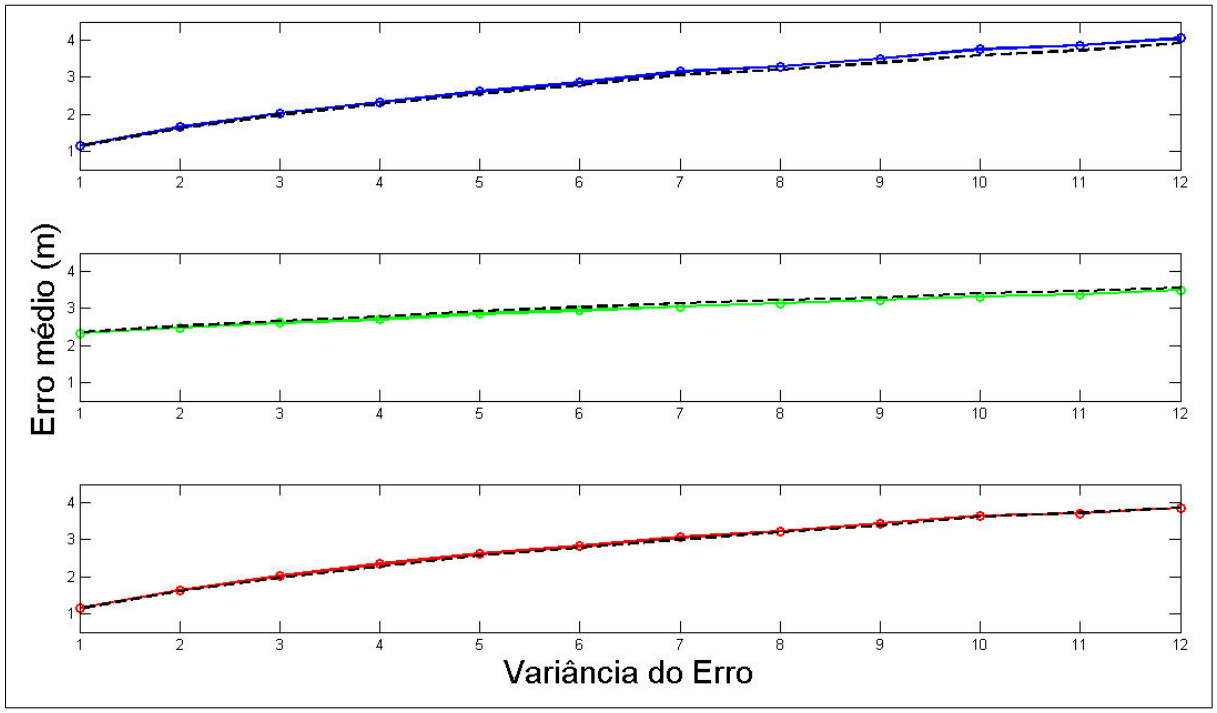

Figura 25: Aplicação dos Determinantes de Cayley-Menger para três NRs

Como mostrado, não houve melhora significativa na estimativa da posição do NPD com esta abordagem e, inclusive, em muitos casos a estimativa de posição piorou com a utilização desta abordagem. Observa-se que a melhora nas estimativas de distâncias não refletiram em

mesma proporção na saída do algoritmo, ou seja, na realidade, as estimativas de distâncias de fato foram melhoradas, mas não o suficiente para afetar significativamente o resultado dos algoritmos. Além disso, deve-se ressaltar que as simulações foram feitas considerando-se apenas três NRs, pois ainda não foram encontrados na literatura meios de estender esta metodologia para mais NRs. Assim, não se pode afirmar como seria o desempenho se mais NRs fossem considerados.

3.4 IMPLEMENTAÇÃO EM HARDWARE

Nesta seção serão discutidas as características dos algoritmos escolhidos neste trabalho, analisando-se mais a fundo as operações matemáticas necessárias em cada método, com o objetivo de verificar o custo e a viabilidade de suas implementações em hardware. Será estudado o impacto no sistema de localização devido ao tipo de dado empregado nos cálculos e, também, os principais aspectos a se considerar nas implementações dos algoritmos em hardware. Após estas análises, os algoritmos interessantes quanto ao custo benefício em termos de hardware e erro de localização serão escolhidos para a sintetização em hardware de lógica programável, para análises no próximo capítulo.

3.4.1 ANÁLISE DA VIABILIDADE DE IMPLEMENTAÇÃO DOS ALGORITMOS

Analisando-se as operações matemáticas exigidas em cada algoritmo, juntamente com as simulações realizadas na seção 3.3, pode-se propor alguns algoritmos que são viáveis de se implementar em FPGAs. A seguir serão analisados os algoritmos MLE, algoritmo da Lateração e o método Min-Max.

O algoritmo MLE é o que apresentou os melhores resultados dentre todos os algoritmos simulados neste trabalho. Trata-se de um algoritmo que leva em consideração os aspectos estatísticos da comunicação sem fio, mas que, ao mesmo tempo, requer muito processamento por parte do NPD, visto que necessita calcular o mínimo de uma função não linear. A escolha dos métodos de otimização numérica não é trivial, visto que muitos algoritmos necessitam de uma estimativa inicial do ponto de interesse e, dependendo do caso, o algoritmo pode nem mesmo convergir para o ponto desejado (JÚDICE, 2005). Além disso, o número de iterações pode ser elevado, como mostrado no trabalho de Trevisan (2009) e também observado nas simulações da seção 3.3. Estas características o torna um algoritmo pouco interessante para implementação em hardware.

O algoritmo da Lateração possui a vantagem de apresentar resultado exato quando

as entradas não possuem erros. A implementação para 3 NRs é viável em hardware, dada as operações simples envolvidas, como soma, subtração, multiplicação e divisão. Para mais de 3 NRs e utilizando-se o método dos mínimos quadrados, o algoritmo exige o cálculo da inversa de uma matriz, o que é uma tarefa computacionalmente custosa e suas implementações em hardware são também complexas (GARCIA, 2010).

Do ponto de vista de hardware, o método Min-Max é o mais adequado para implementação em dispositivos de lógica programável, pois necessita apenas de operações simples como comparação, soma, subtração e divisão. Entretanto, das simulações observa-se que o método é instável quando há poucos NRs, mesmo quando se aplicam entradas de distâncias sem erros. Contudo, ao mesmo tempo, no trabalho de Luo et al. (2010) foi mostrado que os resultados do método Min-Max foram melhores que os do MLE, quando foram considerado poucos NRs ($0{,}089\ \mathrm{NR}/m^2 . 44{,}87 m^2 = 4$ NRs). Esta situação foi também observado nas simulações da seção 3.3 que, para quantidades pequenas de NRs, o método Min-Max foi superior ao MLE. Na figura 22, para quantidades de NRs menores que 10 nós, observa-se que o método Min-Max apresentou resultados melhores que o do MLE, o que novamente torna este último algoritmo pouco atraente, dado o custo de implementação e tempo de execução e, agora, o desempenho inferior ao método Min-Max, quando poucos NRs são considerados.

Segue abaixo um resumo com as principais conclusões das últimas seções deste trabalho, considerando os resultados obtidos nas simulações:

1. Para três NRs e com boas estimativas de distâncias, pode-se obter boas estimativas de posição com o método da Lateração. Dado que o mesmo utiliza a formulação exata, pode-se obter estimativas de posição isenta de erro quando as estimativas de distâncias são perfeitas, ao contrário do método Min-Max.

2. Para quatro ou mais NRs, o método Min-Max começa a apresentar resultados interessantes, além de não aumentar, de forma significativa, a complexidade do método com o aumento de NRs.

3. O algoritmo MLE não é muito aconselhável para implementação em hardware, dada a complexidade do algoritmo e resultado pouco recompensador para poucos NRs. Apesar disso, é o método que apresentou os melhores resultados.

Assim, considerando todas as análises desta seção, decidiu-se por implementar em hardware o método Min-Max e o algoritmo da Lateração para 3 NRs.

3.4.2 ANÁLISE DOS TIPOS DE DADOS

Uma vez analisados os algoritmos que são viáveis de se implementar em hardware e selecionados quais implementar neste trabalho, fez-se necessário então analisar a fundo o projeto de hardware. O primeiro estudo está relacionado ao tipo de dado a ser empregado nos cálculos, pois o cálculo de posição está sujeito a aproximações, dadas as operações matemáticas requeridas, como, por exemplo, as operações de divisão.

O objetivo desta seção é analisar os tipos de dados que poderiam ser empregados nos cálculos, analisando o impacto desta escolha, tanto na precisão dos cálculos, como na quantidade de hardware necessária para implementar o sistema. Assim, deseja-se encontrar a menor combinaçao de bits para se atingir um erro com a ordem de grandeza considerada aceitável, de forma a se obter o menor hardware.

Em VHDL existem várias formas de se representar números e existem diversos tipos de dados pré-definidos. Além disso, a faixa de valores previsto para o tipo de dado pode ser configurado. Para o contexto deste trabalho considerou-se os seguintes tipos de dados: Ponto Fixo e Ponto Flutuante. O tipo inteiro não foi considerado, como será comentado a seguir.

Uma maneira de representar números em VHDL é através da utilização de representações binárias/hexadecimal, como, por exemplo, o tipo *STD_LOGIC_VECTOR*. Estes tipos são úteis para representar variáveis físicas, todavia, o tipo de dado inteiro (*INTEGER*), que é mais fácil de se utilizar, pode também ser empregado. Em VHDL o tipo de dado inteiro pode ser definido especificando-se a faixa de valores envolvidos. Caso não especificado, o inteiro é representado, por padrão, como sendo de 32 bits (AMARAL; MEHTA, 2003; PEDRONI, 2010).

No trabalho de Bucher e Misra (2002), os autores implementaram em VHDL um sistema de localização através do método hiperbólico e utilizando medições de TOA. Neste sistema, os autores utilizaram representações binárias para realizar o cálculo de posição, especificando vetores de bits. Assim, para que a precisão dos cálculos não fosse comprometida com as operações de divisão, foram feitas multiplicações adicionais após cada divisão. Neste contexto, até mesmo vetores com 200 bits foram necessários para que as multiplicações adicionais pudessem ser realizadas. Apesar de não se tratar, exatamente, do mesmo tipo de sistema de localização desta dissertação, o trabalho de Bucher e Misra (2002) mostra importantes aspectos de projeto e as dificuldades de se manter a precisão dos cálculos com a utilização de vetores de bits, que também se aplicam ao tipo de dado inteiro. Considerando estas dificuldades e as facilidades que, por exemplo, o ponto fixo oferece, o tipo de dado inteiro não será utilizado nos

algoritmos deste trabalho.

Com as representações binárias só é possível descrever um conjunto determinado de valores inteiros, ou seja, não é possível descrever valores fracionários. Quando um número não coincide com um dos valores do conjunto este será arredondado, inserindo um erro, também denominado ruído de quantização (TOMASELLI, 2001).

O ponto fixo é um intermediário entre os números inteiros e os de ponto flutuante. Este tipo de dado apresenta processamento de suas operações quase tão rápidas que o dos inteiros, pois o mesmo é baseado em aritmética de inteiros, e, além disso, permite representar números menores do que 1. Em VHDL, o ponto fixo é especificado por uma quantidade de bits para a parte inteira e uma quantidade de bits para a parte fracionária, como apresentado em Pedroni (2010). Dependendo da precisão requerida nos cálculos, este tipo de dado pode ser uma boa alternativa (BISHOP, 2010).

A padronização do ponto flutuante foi possível graças ao surgimento do padrão IEEE 754. Entretanto, devido ao padrão apresenta algumas flexibilidades, nem todas as implementações de ponto flutuante apresentam resultados idênticos. Porém, tem-se uma uniformidade quanto ao funcionamento do ponto flutuante (MOLER, 2004). Neste padrão o ponto flutuantes possui a representação apresentada na figura 26. O valor do ponto flutuante é, então, calculado pela equação 78, onde x é o valor do ponto flutuante, S é um bit que representa o sinal do número (0 para positivo e 1 para negativo), E é o valor armazenado no campo de expoentes, N é um fator de normalização, apresentado na equação 79. Nesta última equação, E_{max} é o valor máximo representável no campo de expoente. A fração no formato IEEE compreende um bit pressuposto a 1, conhecida como *bit escondido*. O 1 somado à fração fica implícito, ou seja, não é armazenado nos bits. Assim, na verdade, a precisão que a estrutura representa é a precisão de um bit a mais que o armazenado (IEEE, 1985; PEDRONI, 2010; WILLRICH, 2002).

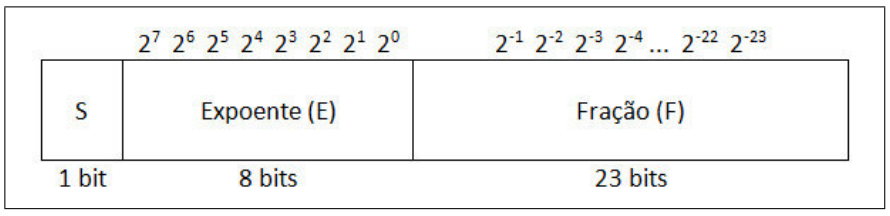

Figura 26: Representação do ponto flutuante *single precision*

Fonte: Adaptado de Pedroni (2010)

$$x = (-1)^S.(1+F).2^{E-N} \tag{78}$$

$$N = \frac{E_{max} + 1}{2} - 1 \tag{79}$$

Em VHDL o tipo *REAL* pode ser utilizado para descrever números fracionários, contudo, possui limitações quanto à sintetização física e tem utilidade apenas para simulações. Uma forma de implementar o ponto flutuante em VHDL é utilizar as bibliotecas sugeridas em (PEDRONI, 2010) e disponíveis para *download* em: *www.vhdl.org/fphdl/vhdl.html*. O ponto flutuante é o tipo de dado que oferece a melhor precisão, porém, o seu uso é, de certa forma, impeditiva em diversas aplicações, visto que pode ser necessário até três vezes mais hardware em comparação às implementações com aritméticas de ponto fixo (BISHOP, 2009).

Com o intuito de verificar o impacto do tipo de dado nos cálculos, assim como para se ter uma ideia da quantidade de bits necessária nas implementações de hardware, buscou-se formas de simular o comportamento dos algoritmos com a utilização de diferentes tipos de dados. Para este propósito, priorizou-se os seguintes pontos:

- Integração com o MATLAB, para se poder utilizar os códigos já implementados e testados na seção 3.3.
- Ambientes que permitam configurar com facilidade os tipos de dados e que permitam bom aproveitamento da estrutura para simulações com os diferentes tipos de dados.
- Que os tipos de dados possam ser alterados em código, permitindo a execução em *loop*, pois a simulação ficará muito demorada se os tipos de dados necessitarem ser alterados manualmente a cada iteração.
- Ferramentas gratuitas.

Considerando estes pontos, foram pesquisadas e propostas duas abordagens para se emular o ponto flutuante e ponto fixo de configurações arbitrárias: Simulink e ferramenta *Quantized DSP Simulation Toolbox*. A seguir serão apresentadas estas duas abordagens.

3.4.2.1 FERRAMENTAS DE ANÁLISE

O Simulink é uma ferramenta desenvolvida pela companhia The MathWorks e está integrado ao software MATLAB. Esta ferramenta permite modelar, realizar simulações, testar e analisar sistemas dinâmicos. Os projetos são feitos através de um ambiente de desenvolvimento gráfico baseado na utilização de um conjunto de bibliotecas com blocos customizáveis.

O Simulink é amplamente utilizado, tal como na teoria de controle, telecomunicação, processamento digital de sinais, processamento de imagens, entre outras áreas (THE MATHWORKS, 2012c).

O ponto flutuante no MATLAB segue o formato do IEEE 754 e, internamente, utiliza o formato *IEEE double precision*, de 64 bits. O software suporta também a utilização do formato *single precision*, em casos onde deseja-se salvar espaços de memória. Contudo, não há muito ganho de velocidade de processamento em máquinas modernas. Há também a possibilidade de utilizar o formato *extended precision* no MATLAB (MOLER, 2004).

A utilização do Simulink permite integração direta com o MATLAB e utilizando-se a abordagem por blocos garante-se que as operações sejam realizadas com o tipo de dado configurado. A configuração do ponto flutuante pode ser feita especificando-se o número total de bits e o número de bits de expoente na função *float(TotalBits, ExpBits)*. No número total de bits não se leva em consideração o bit escondido. Desta forma, por exemplo, ao se executar o comando *float(16, 5)*, criar-se-á uma estrutura de ponto flutuante com 1 bit de sinal, 10 bits de mantissa e 5 bits de expoente (THE MATHWORKS, 2012d).

No Simulink pode-se criar blocos que englobem outros blocos, permitindo melhor organização. A figura 27 ilustra a implementação do método da Lateração através de blocos customizáveis.

No código MATLAB, a chamada ao bloco implementado se faz através do comando: *sim(Nome do bloco)*. O Simulink consegue enxergar as variáveis locais do MATLAB e, através disso, faz-se a interação entre o sistema criado no Simulink e o código MATLAB. Para se especificar o tipo de dado dos blocos deve-se configurar o campo *Output data type* dos blocos. A figura 28 destaca a configuração de um multiplicador com o tipo de dado *my_float*, que é um tipo de dado criado durante as simulações. Deste modo, é possível que as simulações ocorram em *loops*, não necessitando que se especifique manualmente o tipo de dado em cada iteração.

Embora esta abordagem satisfaça os pontos destacados na introdução desta seção, a mesma possui algumas desvantagens. A primeira delas refere-se ao processo trabalhoso inerente a este método, pois o mesmo baseia-se na utilização de elementos discretos. Outro inconveniente é que a configuração do ponto flutuante através da função *float(TotalBits, ExpBits)* não tem mais suporte em versões mais atuais do MATLAB (THE MATHWORKS, 2012d). A versão do MATLAB utilizada nestas simulações foi o MATLAB R2008a. Nesse sentido, foi dado continuidade na pesquisa por outras abordagens mais convenientes e não se deu continuidade na utilização desta ferramenta para as simulações desta dissertação.

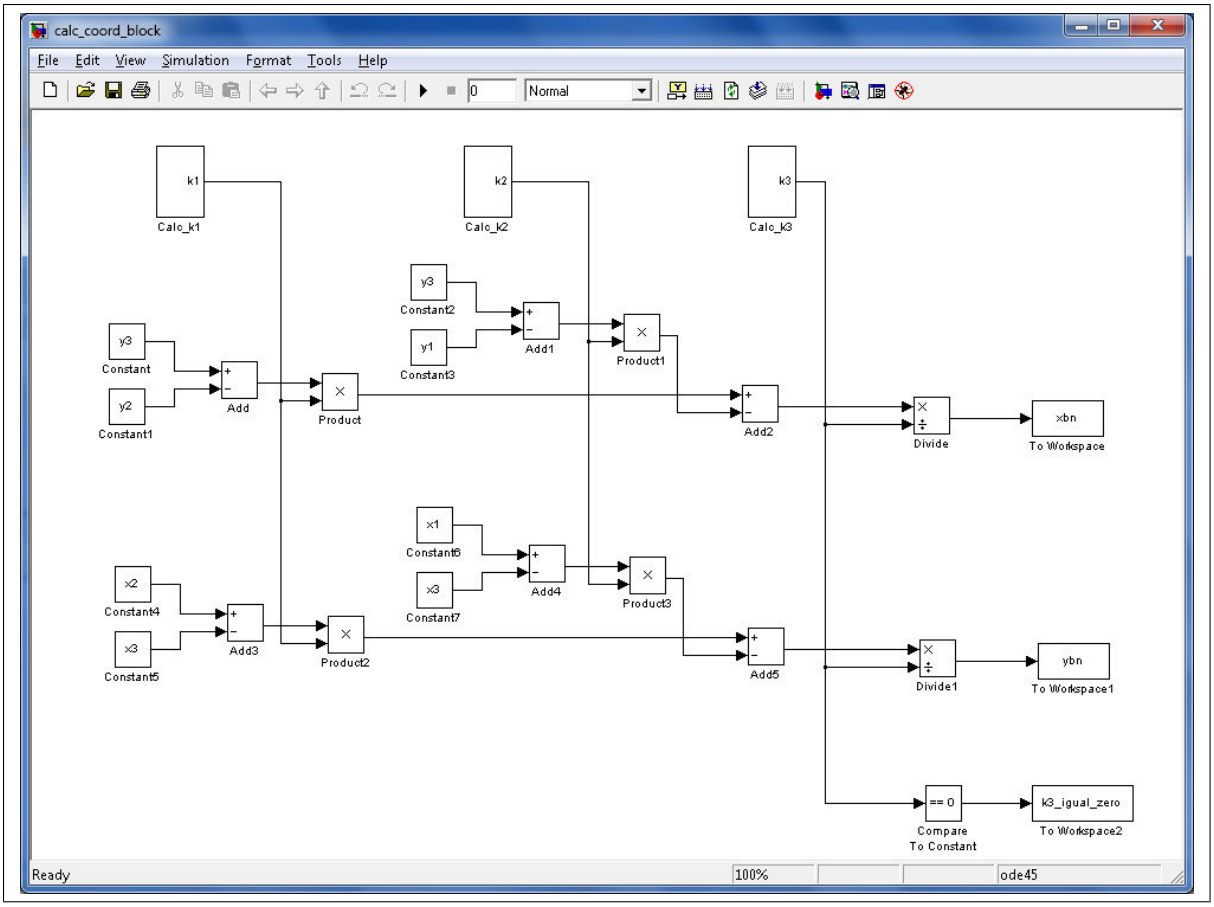

Figura 27: Método da Lateração implementado por blocos no Simulink

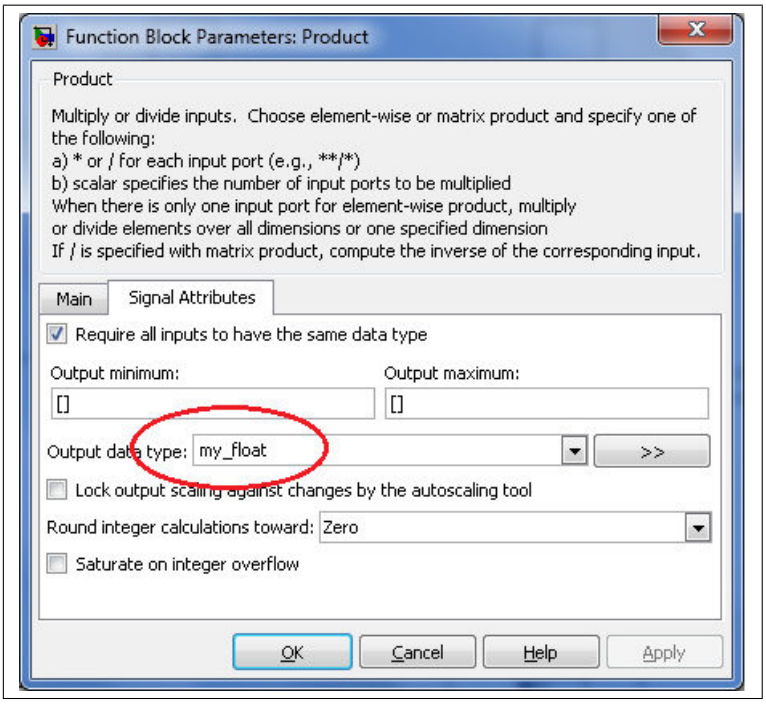

Figura 28: Especificação do tipo de dado no multiplicador

Outros métodos de analisar o erro devido aos tipos de dado empregados nos cálculos é através das ferramentas de simulações desenvolvidas por Widrow e Kollár (2008). Estas ferramentas foram desenvolvidas no MATLAB e, desta forma, permite também integração direta com os códigos desenvolvidos na seção 3.3. Estas ferramentas podem ser encontradas em (KOLLÁR, 2012).

Widrow e Kollár (2008) apresenta várias formas de simular os cálculos com ponto fixo e ponto flutuante configuráveis. O autor apresenta também as limitações inerentes destas implementações. Seguem abaixo os métodos citados neste trabalho:

- Arredondamento manual
- Arredondamento manual através da utilização de quantizadores
- Ferramenta *DSP Simulation Toolbox*
- Ferramenta *Fixed-Point Toolbox*, disponibilizada pela The MathWorks

O primeiro método consiste em realizar o arredondamento manual do resultado, através da equação descrita abaixo, escrita em linguagem MATLAB. Os códigos MATLAB apresentados nesta seção estão disponíveis em Widrow e Kollár (2008)

```
xq=q*round(x/q)
```

O comando *round* faz o arrendondamento de um número para o inteiro mais próximo e q representa a amplitude da quantização (*quantum size*). Na simulação do ponto flutuante, q representaria a menor variação possível da estrutura de dado simulada.

Outro método de arredondamento manual, mais sofisticado que o anterior, consiste em realizar o arredondamento considerando o valor da mantissa a ser simulada e não o *quantum size*. O código MATLAB apresentado abaixo ilustra esta metodologia. Neste código, p representa a quantidade de bits de fração do ponto flutuante simulado.

```
[f, e] = log2(x);
dxp = sign(x).*pow2(max(e, -1021) + 52 - p);
xqfl = (x + dxp) - dxp;
```

Estes métodos são formas simples de se simular cálculos com o ponto flutuante desejado. Porém, estes apresentam algumas desvantagens, tal como a dificuldade de se utilizar diferentes estratégias e critérios de arredondamento, simulação restrita a, no máximo, 52 bits de fração (o que, contudo, é muito mais que o suficiente para muitas aplicações), impossibilidade

de simular saturação (*overflow*), entre outras restrições. Outro inconveniente é a necessidade de se fazer o arredondamento depois de cada operação, o que ocasionalmente pode ser esquecido.

Um método mais abrangente consiste em se utilizar estruturas de dados que descrevem o tipo de dado desejado, assim como as especificações das estratégias de arredondamento, codificações, entre outros parâmetros. Nesta ferramenta, também denominada pelos autores de quantizadores (*quantizers*), a configuração do tipo de dado é feita através do comando *qstruct*. Uma vez definida a estrutura, utiliza-se a função *roundq* para arredondar o número em questão, de acordo com a estrutura definida.

```
Q = qstruct('p', 24);
a = roundq(1.1, Q);
b = roundq(2.3, Q);
c = roundq(a*b, Q);
```

Apesar da abrangência deste método, ainda existe a necessidade de se fazer chamadas explícitas ao método de arredondamento após todos os cálculos. Além disso, não é possível também simular pontos flutuantes com mais de 52 bits de fração.

O terceiro método é a ferramenta mais precisa disponibilizada por Widrow e Kollár (2008) e é chamada pelos autores de *Quantized DSP Simulation Toolbox* (QDSP). Trata-se de uma ferramenta versátil para simular o comportamento do ponto flutuante e ponto fixo, baseada na utilização de objetos. Para utilizar esta ferramenta, define-se, primeiramente, uma estrutura de dado com o comando *qmake*. Nesta função deve-se especificar dados como a quantidade de bits de expoente e bits fração. Depois disso, associa-se a variável à estrutura de dado criada, através do comando *qdata*. Após isso, todas as operações com estas variáveis serão feitas de acordo com o tipo de dado definido. Segue abaixo um exemplo de utilização desta ferramenta:

```
Q = qmake('name','Qs','type','floating-point','precision','6');
a = qdata([1.22  3.3], Q);
b = qdata(2.1, Q);
c = a*b;
```

Usando-se esta ferramenta, não se necessita mais fazer chamadas explícitas de funções de arredondamento, além de ser possível simular, com mais eficiência, o comportamento do tipo de dado desejado. Entretanto, todos estes ganhos acarretam em uma execução mais lenta. Esta desvantagem, porém, é minimizada nos computadores modernos.

O último método citado é utilizar a ferramenta *Fixed-Point Toolbox*, comercializada com o MATLAB. Esta ferramenta apresenta utilização similar ao QDSP, onde é feita a definição da estrutura do quantizador e associações desta estrutura com as variáveis de interesse. Esta

ferramenta, voltada apenas para a utilização do ponto fixo, permite que os algoritmos criados com a linguagem MATLAB sejam executados na velocidade de um código C compilado (THE MATHWORKS, 2012b). O código abaixo exemplifica a utilização do *Fixed-Point Toolbox*:

```
Q16 = quantizer(struct('mode','fixed','format',[16,15], ...
        'roundmode','convergent'));
a = fi(0.6, Q16);
```

Apesar desta ferramenta já vir integrada com o MATLAB, a mesma não permite analisar o comportamento do ponto flutuante, apenas do ponto fixo.

Dos métodos e ferramentas citados nesta seção, a ferramenta QDSP é a mais atrativa, visto que se trata de uma ferramenta gratuita e pode-se utiliza-la tanto para a simulação com o ponto flutuante como para o ponto fixo. Além disso, a configuração do tipo de dado é simples e conta com diversas customizações configuráveis. Nesse contexto, esta será a ferramenta a ser utilizada nas simulações posteriores.

3.4.2.2 IMPLEMENTAÇÃO DAS SIMULAÇÕES COM O *QUANTIZED DSP SIMULATION TOOLBOX*

Como citado anteriormente, a ferramenta QDSP, desenvolvida por Widrow e Kollár (2008), permite simular o comportamento de um ponto flutuante ou ponto fixo com as configurações desejadas, possibilitando a análise do erro inserido nos cálculos, devido ao tipo de dado utilizado.

Para utilizar esta ferramenta não foi necessário mudanças significativas nos códigos desenvolvidos na seção 3.3. Os principais comandos necessários foram: definir a estrutura do tipo de dado, através do comando *qmake*, e associar as variáveis com sua respectiva estrutura, utilizando-se o comando *qdata*.

Teve-se o cuidado de não utilizar funções do MATLAB que empregam o ponto flutuante interno do MATLAB, de forma a não interferir nos resultados dos arredondamentos.

Houve problemas na utilização desta ferramenta no sistema operacional Windows 7, de 64 bits. Entretanto, a mesma funcionou perfeitamente no sistema operacional Windows XP.

Levando-se em conta as análises feitas na seção 3.4.1, decidiu-se implementar o algoritmo da Lateração para 3 NRs e o método Min-Max para vários valores de NRs. Os códigos desenvolvidos nesta seção encontram-se disponível no Apêndice A e os resultados das simulações serão apresentados e discutidos no capítulo 4.

3.4.3 ABORDAGENS COMBINACIONAL E SEQUENCIAL

Após analisar o impacto da escolha do tipo de dado a ser empregado nos cálculos, partiu-se para a análise dos aspectos ligados a arquitetura do sistema. Em VHDL pode-se implementar tanto circuitos combinacionais como sequenciais. Naturalmente, um sistema pode englobar os dois tipos de circuito.

Os circuitos combinacionais são aqueles que possuem saídas que dependem apenas do estado das entradas. Em contraste, as saídas dos circuitos sequenciais dependem não somente do estado das entradas, mas também do estado anterior. Os circuitos sequenciais possuem, desta forma, elementos de memória que são, geralmente, flip-flops D. Além disso, os circuitos sequenciais geralmente utilizam um sinal de relógio para controlar a evolução do sistema (PEDRONI, 2010).

Os circuitos combinacionais são mais rápidos que os sequenciais, visto que os circuitos combinacionais não necessitam de elementos de memória para armazenar os estados. Isto influencia diretamente no tempo de execução dos algoritmos (SAHA; MANNA, 2007).

Por outro lado, com circuitos sequenciais é possível obter-se circuitos de tamanhos menores. Através do reuso de estruturas de hardware com máquina de estados finitos (*Finite State Machine* - FSM) pode-se reduzir o tamanho final do hardware, como mostrado em Shirai et al. (2012). Uma máquina de estados finitos é um modelo utilizado para representar circuitos lógicos sequenciais. Do ponto de vista de hardware, uma FSM pode ser representado como ilustra a figura 29, a qual é constituída de uma parte combinacional e uma parte sequencial, onde estão os elementos de memória. Esta abordagem é útil em projeto de circuitos que possuem uma lista de estados bem definidos, apresentando também condições bem definidas que façam mover de um estado para outro, assim como as saídas que cada estado deve produzir. Controladores digitais são exemplos clássicos desta abordagem (PEDRONI, 2010). Além disso, existem certos tipos de algoritmos que só são possíveis de ser implementados com lógica sequencial. Entre estes, pode-se citar os algoritmos iterativos. Alguns exemplos de algoritmos iterativos seriam os inversão de matrizes e os de determinação de máximos e mínimos.

Shirai et al. (2012) compararam as implementações em hardware do algoritmo da Lateração e método Min-Max, apresentando os fatores que influenciam no tamanho final do hardware, assim como as análises de erro. Com as otimizações os autores conseguiram reduções de até 88% no tamanho do hardware.

Para o contexto desta dissertação, é de interesse analisar tanto as abordagens combinacional e sequencial. As simulações e análises de Shirai et al. (2012) consideraram também

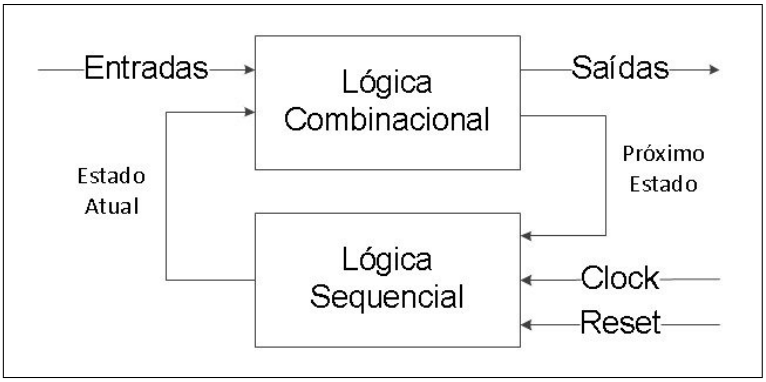

Figura 29: Representação de uma máquina de estados finitos

Fonte: Baseado em Pedroni (2010)

ambas abordagens, contudo, os algoritmos foram implementados apenas com ponto flutuante, ou seja, não foram feitos análises com outros tipos de dado. A abordagem combinacional, mesmo demandando, à princípio, um hardware maior, pode vir a ser viável dependendo do tipo de dado utilizado, fato que não foi analisado no trabalho citado.

3.4.4 ALGORITMOS IMPLEMENTADOS

Após estudar os principais aspectos do projeto de hardware avançou-se, finalmente, para a implementação do código VHDL. Considerando as análises feitas na seção 3.4.1, decidiu-se implementar o algoritmo da Lateração para 3 NRs e o método Min-Max para vários valores de NRs.

Em VHDL o código descrito pode ser executado de forma concorrente, ou seja, de forma paralela, ou então sequencialmente. Existem para este propósito palavras-chaves adequadas para cada caso. Os códigos concorrentes são utilizados para descrever circuitos combinacionais, enquanto que os códigos sequenciais podem descrever tanto circuitos sequenciais como os combinacionais (PEDRONI, 2010).

O algoritmo da Lateração foi implementado apenas para o caso de 3 NRs, pois como ressaltado na seção 2.5.1, utilizando-se o algoritmo da Lateração para mais do que 3 NRs, ter-se-á a necessidade de calcular a inversa de uma matriz de ordem $n \times n$, onde $n = NRs - 1$. Isto tornaria o projeto inviável, dada a complexidade da implementação, além de não apresentar desempenho superior aos outros métodos. Todavia, o caso de 3 NRs é interessante, visto que com boas estimativas de distâncias, o método consegue obter boas estimativas de posição.

Foram feitas três implementações: uma versão puramente combinacional e outras duas baseadas em FSM. A implementação da versão puramente combinacional foi direta, seguindo

as equações 30 a 34. Como mostrado em Shirai et al. (2012), a quantidade de elementos lógicos requeridos para sintetizar este hardware com ponto flutuante é muito grande, o que torna esta implementação inviável, pelo menos com ponto flutuante.

Com o intuito de minimizar a demanda de hardware, buscou-se reaproveitar estruturas de hardware que poderiam ser utilizadas em várias etapas do algoritmo e sincronizar seu uso através de uma FSM. Contudo, deve-se frisar que a diminuição de hardware por esta metodologia acarreta em um tempo de execução do algoritmo maior, com respostas mais lentas que as com circuitos combinacionais, visto que o algoritmo será executado em vários ciclos de *clock*. Todavia, este acréscimo no tempo de execução não é crítico, pois as FPGAs oferecem tempo de execução suficientemente rápida em comparação com a maioria das aplicações de RSSFs. Nesse sentido, o maior gargalo seria a quantidade de hardware requerida nas implementações.

A primeira versão com FSM foi elaborada conforme ilustra o fluxograma da figura 30. Nesta implementação procurou-se aproveitar as similaridades entre as equações 32 e 33 e fazer o reuso deste hardware, calculando k_1 em um ciclo de relógio e K_2 no outro. O mesmo é feito com as equações 30 e 31. Assim, nesta abordagem, o cálculo da posição é realizada em 4 ciclos de *clock*.

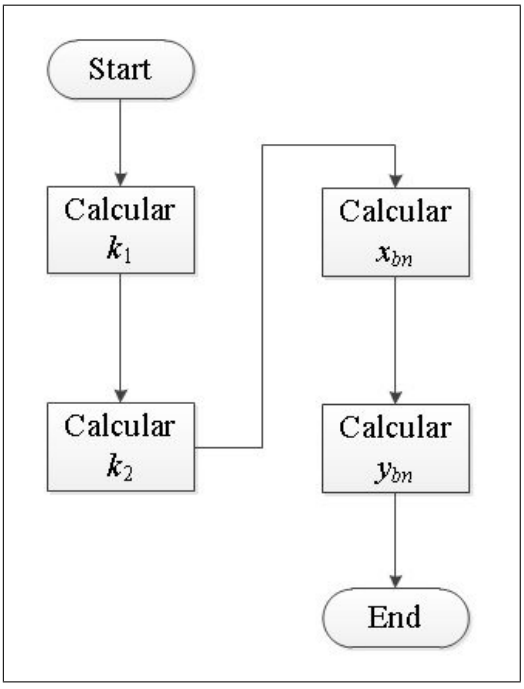

Figura 30: Implementação do algoritmo da Lateração com 4 estados

A outra versão, mais otimizada, foi implementada com o intuito de se utilizar apenas um multiplicador e um divisor, de forma que todas as operações necessitam, agora, ser serializadas. O funcionamento desta implementação está ilustrado na figura 31. Para esta abordagem, o cálculo da posição é realizado em 17 ciclos de *clock*. Esta implementação tende a apresen-

tar hardware menor que o da versão anterior, contudo, necessita de mais ciclos de *clock* para concluir os cálculos.

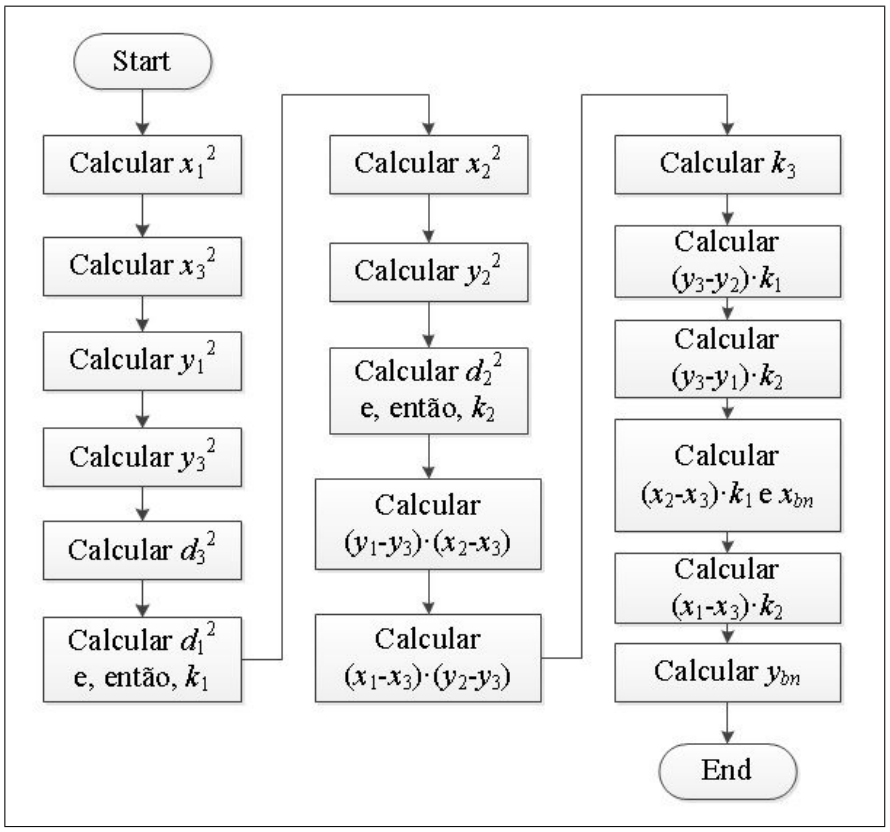

Figura 31: Implementação do algoritmo da Lateração com 17 estados

Para a implementação do método Min-Max foram utilizados códigos sequenciais (*process*), visto que existe uma sequência determinada de cálculos a ser realizada. Primeiramente, determinou-se as coordenadas da área superposta pelos quadrados. Após isto, estimou-se a posição do NPD pelo centroide da área superposta, utilizando-se as coordenadas encontradas no passo anterior. Neste método, dada a sua simplicidade, decidiu-se por implementa-lo sem utilizar FSM. Nesse sentido, algumas otimizações de tamanho de hardware ainda poderiam ser implementadas. Por fim, vale frisar que, apesar do sistema ser descrito com códigos que são executados sequencialmente, o sistema em si é combinacional, ou seja, o sinal de saída deste sistema depende apenas de uma combinação dos sinais de entradas.

Para agrupar as funções utilizadas e os tipos de dados criados para o projeto, foi criada uma *PACKAGE* em um arquivo separado, denominada *rssi_loc_types*. Utilizar *PACKAGE* é uma forma de agrupar as declarações e os recursos que as diferentes unidades de projeto possam partilhar, permitindo também organizar os recursos (IEEE, 1994).

Para trabalhar com o ponto fixo e ponto flutuante, todos os algoritmos foram implementados utilizando-se a biblioteca disponível em Bishop (2010). Estes arquivos destinam-se

a fazer uma ponte entre as versões VHDL-93 e VHDL-2008, porém, não são versões oficiais do IEEE. A biblioteca é denominada *IEEE proposed* e os arquivos são isentos de restrições autorais. Esta biblioteca foi desenvolvida para se criar circuitos sintetizáveis em VHDL-93. Os arquivos necessários são: *fixed_pkg_c.vhd* e *fixed_float_types_c.vhd* para se trabalhar com o ponto fixo e, para se trabalhar com ponto flutuante, necessita-se, além destes, do arquivo *float_pkg_c.vhd* (BISHOP, 2010; PEDRONI, 2010).

Para configurar o tipo de dado utilizado definiu-se o tipo *my_float*, com o intuito de facilitar a configuração do tipo de dado durante as simulações.

Os códigos VHDL desenvolvidos nesta seção encontram-se disponível no Apêndice B.

Após a implementação do código, passou-se para a etapa de simulação funcional, para verificar o funcionamento do circuito obtido. Para testar o hardware descrito, utilizou-se o software Modelsim, da Mentor Graphics Corporation. Este software é uma ferramenta bastante utilizada para simulação de circuitos digitais. Esta ferramenta permite fazer verificação e *debug* de sistemas digitais descritos em diversas linguagens como, por exemplo, Verilog e VHDL. A versão do software utilizada foi o Modelsim Altera Starter Edition 6.6d. Esta versão já integra as bibliotecas das FPGAs da Altera, necessárias para realizar as simulações (ALTERA, 2012).

Primeiramente, foram realizadas simulações funcionais manuais ou simulações do tipo I, como descrito em Pedroni (2010). Nestas simulações os atrasos internos dos circuitos não são levados em consideração e a saída é verificada manualmente (visualmente, neste caso). A seguir será apresentado um exemplo destes testes, considerando a seguinte configuração:

- Três nós de referência.
- $NR_1 = (-3;0)$; $NR_2 = (4,5;5)$ e $NR_3 = (5;-2,5)$.
- Os valores de distâncias aos NRs, com duas casas decimais: $d_1 = 6,18$; $d_2 = 3,81$ e $d_3 = 4,47$.
- Ponto flutuante com 5 bits de expoente e 5 bits de fração.

O resultado esperado é: $NPD_{real} = (3;1,5)$. Calculando-se pelo método Min-Max, o valor de posição esperado é: $NPD_calc = (1,94;1,58)$. A figura 32 ilustra este exemplo.

Considerando que será utilizado um ponto flutuante com apenas 5 bits de fração e o fato dos valores de distâncias já terem sido arredondados com 2 casas decimais, a posição calculada pode não corresponder ao valor real. Com o intuito de verificar o resultado esperado em um hardware de ponto flutuante com as configurações deste exemplo, utilizou-se as funções da

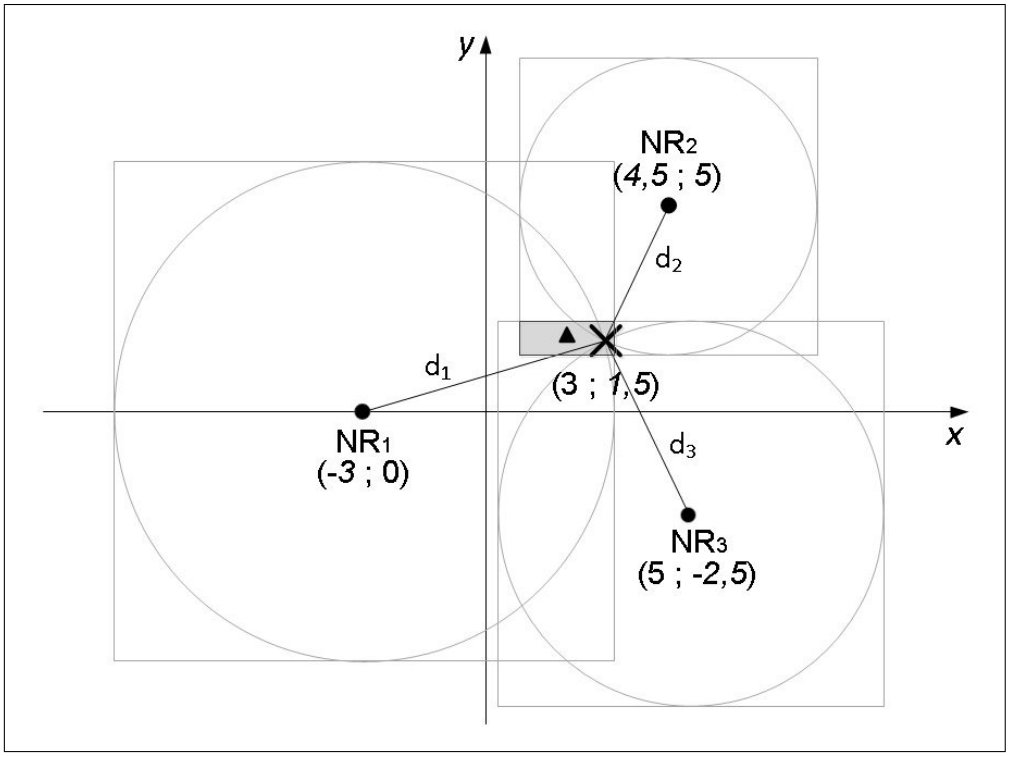

Figura 32: Cenário simulado no Modelsim

biblioteca QDSP e os códigos desenvolvidos na seção 3.4.2.2 para verificar o resultado esperado. A fim de realizar esta simulação, implementou-se e executou-se no MATLAB o código abaixo:

```
% Adiciona o caminho para a biblioteca
addpath(strcat(pwd,'\qdsp'), '-end');

% Localizacao real do NPD (blind node)
bn = [3 1.5]

% Gera um ponto de teste
Refs = [-3 0; 4.5 5; 5 -2.5];
Dist = [6.18; 3.81; 4.47];

% Gera uma estrutura de ponto-flutuante
Qfloat = setup_floating_point(5, 5 + 1);

% Calcula a posicao pelo Método Min-Max e o erro
bn_est = find_point_by_min_max_quantized(Refs, Dist, Qfloat);
error1 = sqrt( (bn(1)-bn_est(1))^2 + (bn(2)-bn_est(2))^2 )

% Calcula a posicao pelo algoritmo da Lateracao e o erro
bn_est = find_point_by_trilateration_quantized(Refs, Dist, Qfloat);
```

```
error2 = sqrt( (bn(1)-bn_est(1))^2 + (bn(2)-bn_est(2))^2 )
```

O ponto flutuante deve ter a fração acrescida de 1, visto que a biblioteca QDSP considera a precisão contando com o bit escondido. Após a execução, os seguintes valores foram obtidos:

- Método Min-Max: $xbn = 0\,01111\,11101 = 1,91$; $ybn = 0\,01111\,10011 = 1,59$; $error_1 = 1,10$
- Algoritmo da Lateração: $xbn = 0\ 10000\ 10000 = 3,0$; $ybn = 0\ 01111\ 10000 = 1,5$; $error_2 = 0$

Observa-se que o algoritmo da Lateração encontrou a posição correta. Contudo, dado que houve arredondamento nos dados de entrada, além do ponto flutuante empregado não ser de alta precisão, esperava-se que o resultado apresentasse um erro. O fato de este algoritmo ter determinado a posição exata foi uma mera coincidência.

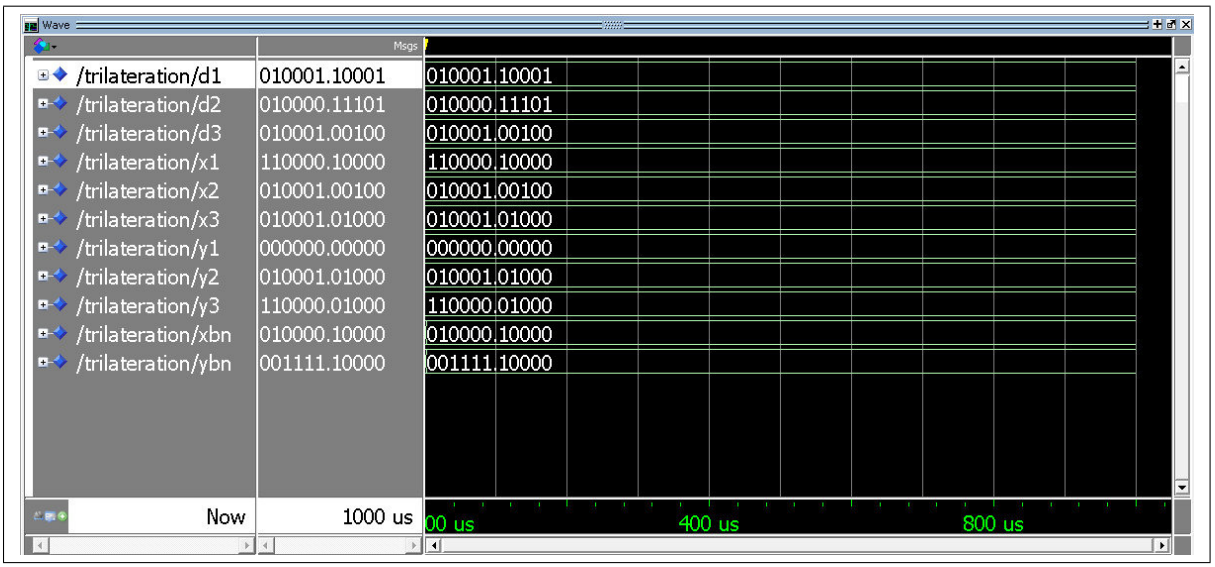

Figura 33: Simulação do algoritmo da Lateração no Modelsim

Conforme ilustrado nas figuras 33, 34 e 35, as três abordagens do método da Lateração apresentaram os mesmos resultados. Além disso, apresentaram os mesmos resultados da simulação feita com a biblioteca QDSP. A figura 36 ilustra a saída obtida do hardware do método Min-Max. Os resultados também foram idênticos aos obtidos na simulação com a biblioteca QDSP.

Além destas simulações, foram feitas também simulações funcionais automatizadas, ou simulações do tipo III, como descrito em Pedroni (2010). Nestas simulações os atrasos

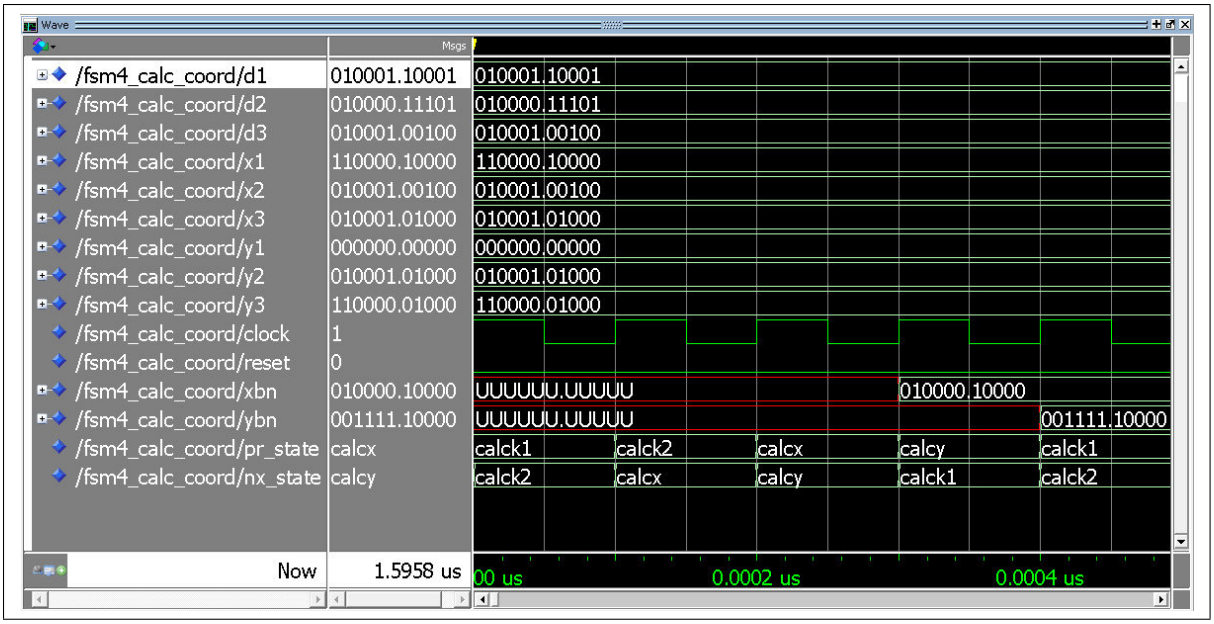

Figura 34: Simulação do algoritmo da Lateração com FSM de 4 estados

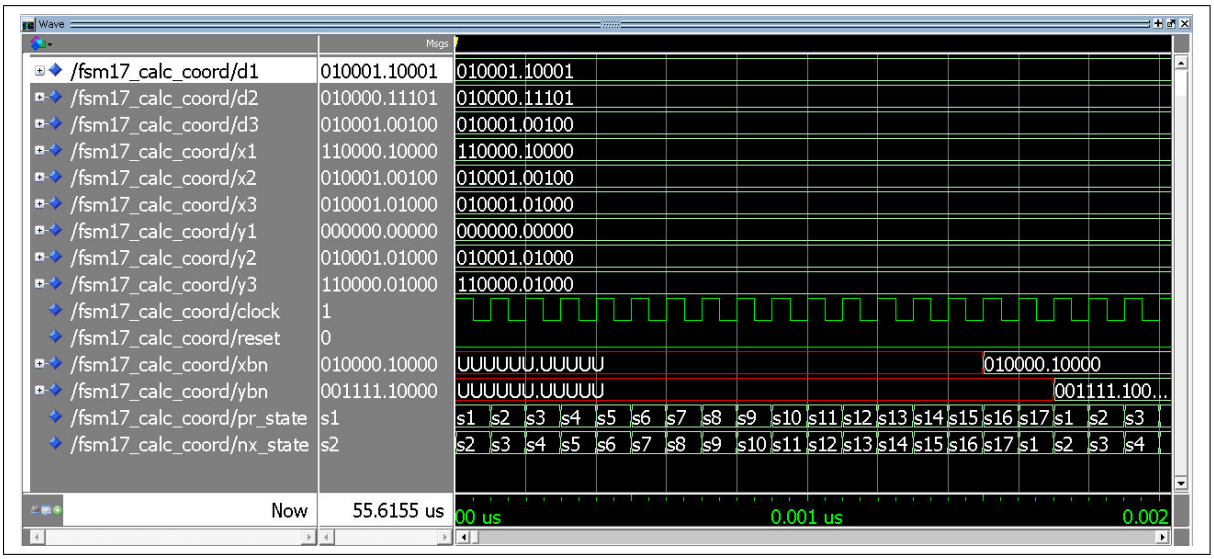

Figura 35: Simulação do algoritmo da Lateração com FSM de 17 estados

internos dos circuitos continuam sendo desconsiderados, porém, as saídas são automaticamente verificadas. Para tanto, foram utilizados comandos de VHDL específicos para simulação. Caso a saída, para uma dada entrada, não corresponda ao valor esperado, são lançadas mensagens de erro.

A partir de todas estas simulações, mostrou-se que os resultados das simulações com a biblioteca QDSP e as saídas obtidas dos hardware implementados estão equivalentes. O próximo passo será analisar a ordem de grandeza dos erros devido à utilização de determinados tipos de dado e, a partir destes resultados, verificar a quantidade de hardware necessária em cada uma das implementações. Os resultados serão apresentados e discutidos no capítulo 4.

Figura 36: Simulação do método Min-Max no Modelsim

4 RESULTADOS

4.1 INTRODUÇÃO

Este capítulo tem como objetivo apresentar os resultados finais obtidos nesta dissertação, assim como servir de base para as discussões e conclusões do capítulo 5. O presente capítulo está dividido em duas partes, como explanado a seguir:

Primeiramente, serão apresentados os resultados das simulações feitas no MATLAB, na investigação da ordem de grandeza dos erros acrescentados nos algoritmos de localização, devido ao tipo de dado utilizado nos cálculos. As metodologias seguidas e os códigos implementados foram apresentados na seção 3.4.2.

A segunda parte expõe os resultados do levantamento da quantidade de hardware necessária nas implementações dos algoritmos implementados na seção 3.4.4, considerando as diversas configurações dos tipos de dado. As simulações foram feitas no software Quartus II.

4.2 ERRO DEVIDO AO TIPO DE DADO UTILIZADO

Nesta seção serão apresentados os resultados das simulações feitas no MATLAB quanto aos erros acrescentados nos algoritmos de localização, devido ao tipo de dado utilizado nos cálculos. Nas simulações foram analisados os algoritmo da Lateração com três NRs e o método Min-Max, conforme feito na seção 3.4.4. Considerou-se tanto o ponto fixo como o ponto flutuante, com diferentes quantidades de bits de expoente e fração. O erro foi calculado comparando-se o resultado encontrado com o tipo de dado criado com o valor calculado com o ponto flutuante interno do MATLAB, de 64 bits. Foram realizadas 1000 iterações em cada tipo de dado e calculou-se o erro de quantização médio, utilizando-se a equação 80. Nesta equação, a coordenada encontrada com o ponto flutuante de 64 bits é representada por (x_{double}, y_{double}) e a coordenada encontrada com o tipo de dado simulado é representada por $(x_{simulador}, y_{simulador})$.

$$Erro = \sqrt{(x_{double} - x_{simulador})^2 + (y_{double} - y_{simulador})^2} \tag{80}$$

Na ocorrência de *overflow*, ou seja, quando a faixa de valores de expoente é ultrapassada, a ferramenta QDSP permite tratar esta situação de diversas maneiras. Nestas simulações configurou-se que, em caso de *overflow*, o valor da variável mantém o maior ou menor valor representável, dependendo se, respectivamente, o valor máximo ou mínimo de expoente é ultrapassado. Isto foi necessário, visto que, por padrão, a biblioteca trata o *overflow* como NaN (*Not a Number*), fazendo com que ao se somar os erros individuais no cálculo do erro médio, a soma resulta em NaN, na ocorrência de qualquer NaN.

Na simulação com ponto flutuante considerou-se, inicialmente, o número mínimo de 7 bits (1 para sinal, 3 para expoente e 3 para fração), conforme suportado pela biblioteca de ponto flutuante do IEEE, e aumentou-se gradativamente os bits de expoente e fração. Estes resultados estão apresentados nas tabela 4 e 5, para o algoritmo da Lateração e método Min-Max, respectivamente. Os valores em negrito ilustram as configurações de bits, cujo erro de quantização é menor do que 10 cm (PEDRONI, 2010).

Tabela 4: Algoritmo da Lateração: Erro x Qtd. de bits do ponto flutuante

		Bits de fração							
		3	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Bits de exp.	**3**	4,896	4,833	5,008	4,761	4,846	5,041	4,944	4,921
	4	4,191	4,229	4,218	4,131	4,144	4,086	4,111	3,993
	5	0,552	0,290	0,142	**0,074**	**0,038**	**0,018**	**0,011**	**0,005**
	6	0,574	0,303	0,144	**0,076**	**0,038**	**0,018**	**0,011**	**0,005**
	7	0,559	0,289	0,148	**0,075**	**0,039**	**0,017**	**0,011**	**0,005**

Tabela 5: Método Min-Max: Erro x Qtd. de bits do ponto flutuante

		Bits de fração							
		3	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Bits de exp.	**3**	0,531	0,375	0,306	0,239	0,223	0,226	0,232	0,214
	4	0,264	0,135	**0,068**	**0,033**	**0,017**	**0,008**	**0,004**	**0,002**
	5	0,264	0,130	**0,067**	**0,033**	**0,017**	**0,008**	**0,004**	**0,002**
	6	0,261	0,132	**0,066**	**0,032**	**0,017**	**0,008**	**0,004**	**0,002**
	7	0,262	0,129	**0,066**	**0,034**	**0,017**	**0,008**	**0,004**	**0,002**

Para o método Min-Max analisou-se também o erro em função do aumento do número de NRs. O aumento do número de NRs faz com que mais operações de adição e subtração sejam necessárias, tendendo a aumentar o erro de quantização. Para verificar isto, fixou-se a configuração do ponto flutuante com 4 bits de expoente e 5 de fração e realizou-se a simulação. Os resultados estão ilustrados na tabela 6.

Analogamente às simulações com o ponto flutuante, foram feitas as simulações com

Tabela 6: Método Min-Max: Número de NRs x Erro - Ponto flutuante

Número de NRs	Erro
3	0,068
4	0,074
5	0,079
6	0,071
7	0,073
8	0,075
9	0,086
10	0,071

o ponto fixo. Os resultados estão apresentados nas tabelas 7 e 8. Novamente, os valores em negrito ilustram as configurações de bits, cujo erro de quantização é menor do que 10 cm.

Tabela 7: Algoritmo da Lateração: Erro x Qtd. de bits do ponto fixo

		Bits de fração							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Bits de inteiro	**11**	3,292	3,298	3,293	3,231	3,227	3,174	3,191	3,081
	12	1,829	1,836	1,768	1,735	1,857	1,841	1,829	1,905
	13	0,379	0,306	0,197	0,219	0,194	0,169	0,180	0,183
	14	0,277	0,135	**0,067**	**0,036**	**0,016**	**0,008**	**0,005**	**0,002**
	15	0,270	0,131	**0,068**	**0,036**	**0,016**	**0,008**	**0,005**	**0,002**
	16	0,271	0,132	**0,066**	**0,036**	**0,016**	**0,008**	**0,005**	**0,002**
	17	0,276	0,134	**0,066**	**0,036**	**0,016**	**0,008**	**0,005**	**0,002**

Tabela 8: Método Min-Max: Erro x Qtd. de bits do ponto fixo

		Bits de fração							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Bits de inteiro	**11**	0,129	**0,063**	**0,031**	**0,016**	**0,008**	**0,004**	**0,002**	**0,001**
	12	0,131	**0,063**	**0,032**	**0,016**	**0,008**	**0,004**	**0,002**	**0,001**
	13	0,126	**0,063**	**0,031**	**0,016**	**0,008**	**0,004**	**0,002**	**0,001**
	14	0,129	**0,063**	**0,032**	**0,016**	**0,008**	**0,004**	**0,002**	**0,001**
	15	0,125	**0,063**	**0,031**	**0,016**	**0,008**	**0,004**	**0,002**	**0,001**
	16	0,124	**0,063**	**0,031**	**0,016**	**0,008**	**0,004**	**0,002**	**0,001**
	17	0,128	**0,064**	**0,031**	**0,016**	**0,008**	**0,004**	**0,002**	**0,001**

Como feito nas simulações com o ponto flutuante, analisando-se a tabela 7, fixou-se a configuração do ponto fixo em 11 bits para inteiros e 2 bits para fração e verificou-se o comportamento do método Min-Max com o aumento do número de NRs. Estes resultados estão apresentados na tabela 9.

Tabela 9: Método Min-Max: Número de NRs x Erro - Ponto fixo

Número de NRs	Erro
3	0,063
4	0,066
5	0,079
6	0,069
7	0,069
8	0,068
9	0,077
10	0,069

4.3 QUANTIDADE DE HARDWARE NECESSÁRIA

Para verificar a quantidade de hardware necessária utilizou-se o software Quartus II, versão 11.1 de 64 bits, e as tabelas de erros da seção 4.2. Para utilizar a biblioteca IEEE *proposed* no Quartus II, seguiu-se os passos citados em Bishop (2010).

Nas simulações com o Quartus II utilizou-se a FPGA da Altera, da família Cyclone IV E e modelo EP4CE30F23A7. A análise da quantidade de hardware foi feita a partir da quantidade de elementos lógicos utilizados para a sintetização do circuito em hardware. Nestas simulações foi desabilitado o uso de multiplicadores dedicados, focando-se na comparação da quantidade de elementos lógicos requerida.

Consultando as tabelas 4 e 5 e considerando erros de quantização menores que 10 centímetros, montou-se a tabela 10 comparando-se os hardware dos seguintes métodos: Lateração com lógica combinacional pura, Lateração com FSM de 4 estados, Lateração com FSM de 17 estados e o método Min-Max.

Tabela 10: Quantidades de elementos lógicos necessários - Ponto flutuante

Bits do Ponto flutuante	Algoritmo da Lateração			Método Min-Max
	Lógica combinacional pura	FSM de 4 estados	FSM de 17 estados	
5 de exp. e 6 de fração	15.142	10.003	6.236	5.755
5 de exp. e 7 de fração	17.178	11.311	6.623	6.260
5 de exp. e 8 de fração	18.430	12.053	7.087	6.598
5 de exp. e 9 de fração	20.992	13.631	7.826	7.233
5 de exp. e 10 de fração	22.942	14.585	10.367	7.688

Para o método Min-Max analisou-se também a quantidade de elementos lógicos necessários em função do aumento do número de NRs. Para verificar isto, fixou-se a configuração do ponto flutuante com 4 bits de expoente e 5 de fração e realizou-se a simulação. Os resultados estão apresentados na tabela 11 e a figura 37 ilustra estes resultados graficamente.

Tabela 11: Método Min-Max: Número de NRs x Qtd. de hardware - Ponto flutuante

Número de NRs	Quantidade de Elementos Lógicos
3	4.510
4	5.752
5	6.952
6	8.151
7	9.384
8	10.587
9	11.795
10	13.037

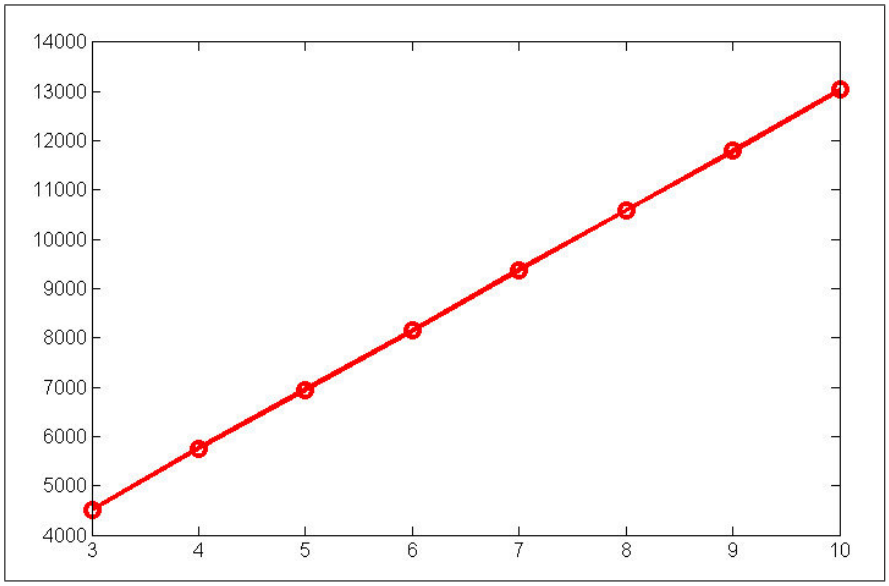

Figura 37: Quantidade de hardware x Número de NRs (Ponto Flutuante)

De forma similar, foram feitas simulações com o ponto fixo, simulando-se as configurações que apresentam erros semelhantes aos dos ponto flutuantes utilizados na tabela 10. Para o ponto fixo, necessitou-se acrescentar o comando *resize* em todas as operações que envolviam as variáveis de ponto fixo. Isto foi necessário para evitar erros de compilação, sem alterar significativamente o código VHDL utilizado no ponto flutuante. Os resultados obtidos estão ilustrados na tabela 12. Para algumas configurações de ponto fixo não foi possível, utilizando-se o mesmo código base do ponto fixo, gerar o hardware desejado. Nestes casos a operação de divisão não foi suportado pela biblioteca de ponto fixo, devidas às otimizações feitas pelo compilador.

Para a análise da quantidade de elementos lógicos necessários com o aumento do número de NRs, no método Min-Max, utilizou-se a configuração de 11 bits de inteiros e 2 bits de fração. Os resultados estão apresentados na tabela 13. Nesta tabela, para os valores destacados com asterisco (para 8, 9 e 10 NRs) foram feitos apenas a compilação de análise e

síntese. A compilação completa não pôde ser feita, pois ultrapassou-se quantidade de pinos existentes. Por fim, a figura 38 ilustra o gráfico de crescimento da quantidade de hardware em função do aumento do número de NRs.

Tabela 12: Quantidades de elementos lógicos necessários - Ponto fixo

Bits do Ponto fixo	Algoritmo da Lateração			Método Min-Max
	Lógica combinacional pura	FSM de 4 estados	FSM de 17 estados	
14 de inteiros e 3 de fração	10.290	6.354	2.239	716
14 de inteiros e 4 de fração	11.606	7.166	2.444	758
14 de inteiros e 5 de fração	13.541	8.367	2.724	800
14 de inteiros e 6 de fração	-	-	2.970	842
14 de inteiros e 7 de fração	-	-	3.245	884

Tabela 13: Método Min-Max: Número de NRs x Qtd. de hardware - Ponto fixo

Número de NRs	Quantidade de Elementos Lógicos
3	548
4	756
5	964
6	1.172
7	1.380
8	1.588 (*)
9	1.796 (*)
10	2.004 (*)

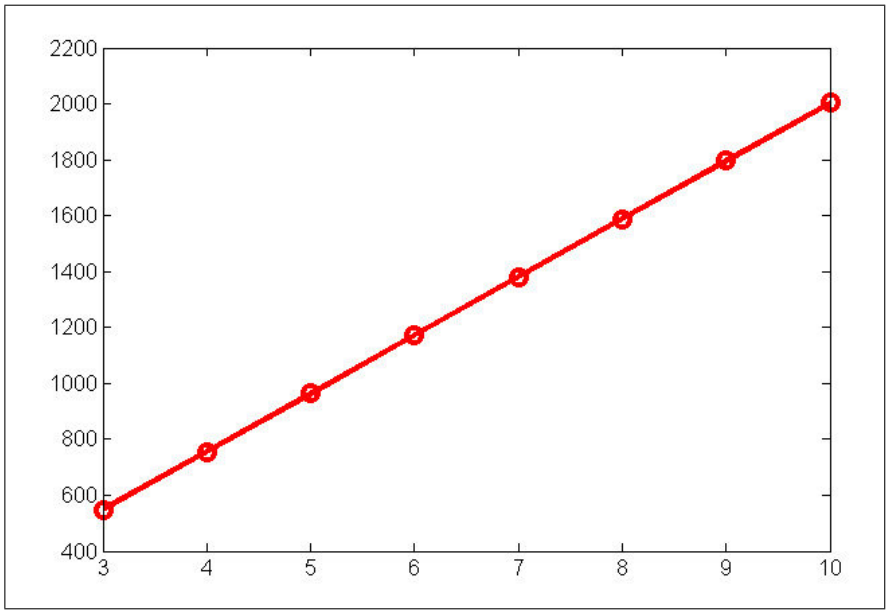

Figura 38: Quantidade de hardware x Número de NRs (Ponto Fixo)

5 CONSIDERAÇÕES FINAIS

5.1 ANÁLISE DOS RESULTADOS

As simulações finais para se verificar o erro devido ao tipo de dado utilizado foram importantes para se definir a configuração do tipo de dado a ser utilizado nas sintetizações em hardware.

As tabelas 4 e 5 apresentam, respectivamente, o comportamento do algoritmo da Lateração e do método Min-Max com a utilização do ponto flutuante. O grande erro observado nos valores com poucos bits de expoente deve-se ao fato da ocorrência de *overflow* nos cálculos intermediários, acarretando grande erro na localização final. Observa-se, que depois de certa quantidade de bits de expoente, o erro só vai diminuir com o aumento do número de bits de fração. Observa-se também que o Método Min-Max necessitou de menos bits de expoente para evitar o *overflow* intermediário e menos bits de fração para atingir o mesmo erro do algoritmo da Lateração. Caso os cálculos sejam tratados em metro, observa-se que é possível obter erros da ordem de centímetros, sem a necessidade de utilizar ponto flutuante ou ponto fixo com excessivos bits de fração.

Nas simulações do método Min-Max com o aumento do número de NRs observa-se que, apesar do número de operações matemáticas aumentarem e, consequentemente, mais arredondamentos serem realizados, os erros apresentados foram praticamente os mesmos. Nesse sentido, pode-se aumentar a precisão da localização aumentando-se o número de NRs e mantendo-se o mesmo tipo de dado, sem acrescentar erro adicional progressivo.

Analogamente, fizeram-se as simulações com o ponto fixo. As tabelas 7 e 8 apresentam, respectivamente, os resultados apresentados pelo algoritmo da Lateração e pelo método Min-Max. Os resultados apresentados foram similares aos da simulação com ponto flutuante. O método Min-Max novamente apresentou melhores resultados.

Comparando-se as simulações com o ponto flutuante e ponto fixo, observou-se que o ponto fixo necessitou de mais bits na estrutura de dado que o ponto flutuante, para se obter erros

de mesma grandeza. A quantidade de bits necessária impacta diretamente no número de pinos de entrada e saída necessários na FPGA. Em todas as simulações, o método Min-Max necessitou de menos bits que o algoritmo da Lateração, para apresentar erro de mesma grandeza.

Na última etapa deste trabalho, utilizando-se de todas as informações obtidas com as simulações anteriores, verificou-se a quantidade de hardware requerida para implementar os algoritmos de localização. A tabela 10 apresenta os resultados obtidos. Observa-se, nesta tabela, que o método Min-Max é o que demanda menos hardware. Além disso, pode-se verificar que, com as otimizações no projeto de hardware do algoritmo da Lateração, pôde-se obter boa economia de hardware. Para a configuração de 5 bits de expoente e 6 bits de fração, o algoritmo da Lateração com FSM de 17 estados apresentou hardware muito menor que a versão combinacional pura (cerca de 2,43 vezes menor) e pouco maior que o do método Min-Max (apenas cerca de 8% maior).

Para o método Min-Max foram feitas simulações para se verificar o crescimento do hardware em função do aumento do número de NRs. Observa-se na figura 37 que a quantidade de hardware aumenta praticamente de forma linear com o aumento do número de NRs.

De forma similar ao que foi feito com o ponto flutuante, foram feitas simulações com o ponto fixo. A tabela 12 compara as quantidades de elementos lógicos necessários para se implementar os respectivos algoritmos. Destas simulações observou-se redução na quantidade de hardware em todos os métodos, quando comparados com os de ponto flutuante. Todavia, o método Min-Max foi o que apresentou a maior redução de hardware frente aos demais. Como comentado, com hardware de ponto flutuante, o algoritmo da Lateração com FSM de 17 estados aproximou-se, em tamanho, do hardware do método Min-Max, contudo, utilizando-se hardware de ponto fixo a diferença tornou-se muito maior. Esta diferença poderia ser até maior se fossem feitas otimizações de hardware no método Min-Max. Por fim, assim como nas simulações com o ponto flutuante, observou-se também que o crescimento do hardware do método Min-Max com ponto fixo se dá de forma linear.

Em algumas configurações, mesmo a compilação ter falhada ou feita apenas as etapas de análise e síntese, estes casos não influenciaram conclusivamente, pois apenas com as simulações que correram normalmente já foi possível verificar que o método Min-Max apresenta o menor hardware e que mesmo a implementação mais otimizada do algoritmo da Lateração apresentou hardware muito maior, diferentemente do que ocorreu com o ponto flutuante.

De forma geral, observou-se que com a utilização do ponto fixo é possível obter-se hardware muito menores, em comparação com os gerados com ponto flutuante. Todavia, o hardware com ponto fixo necessita de mais bits na estrutura de dados (cerca de 50 % mais que

o ponto flutuante) e, consequentemente, as variáveis de entrada e saída exigirão mais pinos da FPGA.

5.2 CONCLUSÕES

Este trabalho teve como objetivo principal estudar e implementar sistemas de localização em hardware de lógica programável, especialmente em FPGA, visando aplicações em RSSF.

Ao longo do trabalho, foram estudados os métodos de localização disponíveis na literatura, focando em métodos que permitam a um nó sensor descobrir a sua posição e analisando também quais destes métodos são viáveis em RSSF. Observou-se que muitos dos trabalhos relacionados a sistemas de localização se encontram no escopo da robótica, sistemas militares e serviços de telefonia móvel (CHENG et al., 2011; CHRAIBI, 2005; GUEDES, 2003). Durante a pesquisa, foram encontrados dois exemplos de implementação em hardware, correlatos com o objetivo desta dissertação. A primeira delas é o chip CC2431, da Texas Instruments, que possui um hardware de localização dedicado, denominado *Location Engine*, o qual estima a localização bidimensional do NPD, através de um algoritmo distribuído baseado em informações de RSSI e de um algoritmo proprietário baseado no MLE (CHIPCON, 2005). O outro trabalho, desta vez, especificamente em FPGA, é apresentado em (BUCHER; MISRA, 2002). Neste artigo os autores implementaram em VHDL um algoritmo que computa a posição tridimensional de um móvel, tendo-se como entradas as posições de quatro referências fixas e as respectivas medidas de TOA entre o móvel e os pontos de referência.

Conforme o andar da pesquisa, observou-se que a maioria dos algoritmos utilizam como entradas as informações de distâncias entre o objeto móvel e os pontos de referência. A determinação destas distâncias pode ser realizada através de diversas técnicas de medição. As técnicas mais usuais são: medição do AOA, do TOA, do TDOA e da RSS. Existem também métodos mais sofisticados para se estimar a distância entre os nós, tais como o UWB e CSS. Este trabalho focou apenas na utilização de informações de intensidade do sinal recebido, mais especificamente do RSSI, que é disponibilizado pela maioria dos nós sensores comerciais. A utilização do RSSI é interessante, visto que nenhum hardware adicional é necessário. Porém, estas medições podem apresentar grandes variações.

Para se verificar o funcionamento, na prática, destas medições e o comportamento dos valores de RSSI, foram feitos experimentos e coleta de dados com o kit XBee. Nestes experimentos, foi verificado o comportamento destes valores, assim como a facilidade da aquisição

dos dados com este kit. Todavia, para as simulações com os algoritmos de localização, optou-se por utilizar os valores medidos por Zanca et al. (2012), visto que neste trabalho mencionado, com equipamentos apropriados, foram coletadas grandes quantidades de medição de RSSI, permitindo realizar um maior mapeamento dos ambientes.

Com os valores reais de RSSI e as pesquisas na literatura, foi possível realizar a conversão destes valores em informações de distâncias. Observando-se o comportamento destes valores e comparando-os com o valor verdadeiro, pôde-se verificar que os erros causaram tanto variações positivas quanto negativas nas estimativas de distância. De forma simplificada, modelou-se este erro através de uma distribuição gaussiana com média nula e variância com valores equivalentes aos apresentados pelas amostras reais. Considerando estes valores de distâncias corrompidas com este erro modelado, simulou-se o comportamento dos algoritmos quando sujeitos à erros daquela ordem de grandeza. As simulações com valores reais de RSSI são importantes para verificar como o sistema de localização se comporta na prática. Como mencionado em Zanca et al. (2008), mesmo algoritmos sofisticados e complexos não apresentam desempenhos extraordinários, pois ficam limitadas às estimativas de distâncias, que são severamente afetadas pelo comportamento não previsível do canal de comunicação sem fio, especialmente em ambientes internos.

A partir das informações de distâncias foram pesquisadas formas de se inferir a posição do NPD a partir destes valores. A forma mais conhecida é através do método da Lateração, que necessita pelo menos de 3 NRs. Mas na prática, costuma-se utilizar vários NRs, com o intuito de amenizar os efeitos das interferências. No caso de duas dimensões, quando mais do que 3 NRs são considerados, costuma-se resolver o problema da alteração utilizando-se o método dos mínimos quadrados, para estimar as coordenadas que minimizem o erro quadrático das distâncias medidas. Contudo, esta abordagem, como mostrado em Chen et al. (2009), exige o cálculo da inversa de uma matriz, além de uma série de multiplicações de matrizes, o que exige muito processamento por parte do nó sensor. Uma alternativa encontrada para o método anterior foi o método Min-Max, apresentado em Savvides et al. (2002). Outra forma, em outra linha de pesquisa, é modelar o processo de localização estatisticamente. Na classe destes algoritmos encontra-se o Método do Estimador da Máxima Verossimilhança, ou, simplificadamente, MLE.

Com o intuito de avaliar o desempenho destes algoritmos, ou seja, o erro médio da posição calculada, assim como verificar como este erro varia com a quantidade de NRs, foram feitas simulações no MATLAB. Antes, porém, foi necessário buscar na literatura algum método de se modelar o canal de comunicação sem fio. Esta modelagem é necessária, visto que são a partir delas que se obtêm as estimativas de distâncias, que são as informações de entrada dos

algoritmos. Neste estudo, considerou-se que os parâmetros de propagação são conhecidos a priori, ou seja, que experimentos práticos já foram feitos para determiná-los. Esta consideração é feita também com o chip CC2431, da Texas Instruments, o qual requer que se especifiquem todos os parâmetros de propagação do canal antes de se utilizar o *Location Engine*. Existe, entretanto, métodos que contornam estas situações, como o apresentado em Zemek et al. (2008) e Tennina et al. (2008). Mas estes métodos ficam apenas como citação, pois estes fogem ao escopo deste trabalho.

Como publicado em diversos trabalhos correlatos, as estimativas de posição tendem a melhorar à medida que mais NRs são considerados. Das simulações pôde-se verificar que o algoritmo baseado no MLE é o que apresenta os melhores resultados, seguido do método Min-Max e, por fim, o método baseado na Lateração. Desempenhos similares foram também verificados em Zanca et al. (2008). Embora o método da Lateração permita obter a posição exata, quando se tem estimativas de distâncias sem erro, o mesmo apresenta-se muito sensível aos erros de distâncias. Diferentemente, apesar do método Min-Max apresentar desempenho ruim para3 NR, este se mostrou mais robusto, inclusive, apresentando desempenhos similares aos do MLE, quando cerca de 10 NRs são utilizados.

Continuando com as simulações no MATLAB, foram estudadas formas de se melhorar as estimativas de distâncias. Uma das abordagens foi a utilização do determinante de Cayley-Menger. A ideia deste determinante é explorar as relações geométricas e algébricas das distâncias entre os nós sensores, permitindo obter relações quadráticas que relacionam as verdadeiras distâncias com aquelas obtidas através das medições. Com estas relações matemáticas, busca-se estimar o conjunto de erros associados na medição de distâncias que, quando compensados, produzam valores de distâncias as mais próximas possíveis das verdadeiras distâncias (CAO et al., 2006).

O filtro de Kalman foi outra alternativa estudada. Trata-se de um filtro para amenizar os efeitos dos ruídos nas medições de distância, técnica esta largamente aplicada em problemas de rastreamentos, como ressaltado em Koks (2007). Trata-se de um filtro estatístico que, em termos matemáticos, estima o estado do processo de forma preditiva e recursiva, no sentido de minimizar seu erro quadrático (PARANHOS, 2009; WELCH; BISHOP, 2006).

Optou-se pela utilização do determinante de Cayley-Menger, principalmente devido à oportunidade de se verificar seu desempenho neste contexto, devido à falta de informações aprofundadas na literatura. Contudo, os resultados obtidos não foram satisfatórios. Apesar da melhora nas estimativas de distâncias, esta melhora não foi o suficiente para alterar significativamente o cálculo de posição. Deve-se, entretanto, ressaltar que estas simulações foram feitas

apenas com 3 NRs, pois ainda não foi encontrado na literatura métodos de se expandir esta técnica para mais de 3 NRs.

Após todos os estudos e simulações com os algoritmos, foram feitas análises da viabilidade de se implementar estes algoritmos em dispositivos de lógica programável, através de linguagens de descrição de hardware. Como justificado na seção 3.4.1, decidiu-se por implementar, em hardware, o método Min-Max e o algoritmo da Lateração para 3 NRs.

Após selecionar quais algoritmos implementar em hardware, analisou-se o projeto de hardware. O primeiro estudo foi relacionado ao tipo de dado a ser empregado nos cálculos, pois o cálculo de posição está sujeito a aproximações, dadas as operações matemáticas envolvidas. Considerou-se, inicialmente, o uso do inteiro, ponto fixo e ponto flutuante. Porém, dadas as dificuldades de se manter a precisão com o inteiro, o mesmo foi posteriormente desconsiderado. As simulações para se analisar o erro devido ao tipo de dado utilizado foram feitas, inicialmente, com o Simulink. Constatou-se, contudo, que este processo tornou-se muito trabalhoso de ser feito com o Simulink, pois o mesmo baseia-se na utilização de elementos discretos. Outro inconveniente é que a configuração do ponto flutuante através da função *float(TotalBits, ExpBits)* não tem mais suporte em versões mais atuais do MATLAB (THE MATHWORKS, 2012d).

Nesse contexto, a melhor alternativa encontrada, que satisfazia as especificações descritas no capítulo 3.4.2, foi a ferramenta *Quantized DSP Simulation Toolbox*, desenvolvida por Widrow e Kollár (2008). Utilizando-se esta ferramenta foi possível encontrar as menores combinações de bits da estrutura de dado para se atingir um erro com a ordem de grandeza, considerada aceitável.

Após analisar o impacto da escolha do tipo de dado, partiu-se para a análise dos aspectos ligados a arquitetura do sistema. Como visto, em VHDL existem duas arquiteturas de sistema: baseadas em circuitos combinacionais e em circuitos sequenciais. A abordagem dos algoritmos com circuitos combinacionais foi de mais fácil implementação. Por outro lado, com circuitos sequenciais e através do reuso de estruturas de hardware com FSM, pôde-se reduzir o tamanho final do hardware, assim como relatado em Shirai et al. (2012) e também verificado no final desta dissertação. Deve-se ressaltar, porém, que essa redução de hardware veio com o custo do aumento no tempo de execução do algoritmo. Após implementar todos estes algoritmos, foram feitas simulações funcionais manuais e simulações funcionais automatizadas para verificar se o hardware implementado correspondia ao que foi descrito com o código VHDL.

Após estas verificações e com as informações da quantidade de bits a ser utilizada na estrutura de dados, fez-se o levantamento da quantidade de hardware necessária nas implementações dos algoritmos implementados na seção 3.4.4, considerando as diversas configura-

ções dos tipos de dado simulados no MATLAB. As sintetizações em hardware foram feitas com o software Quartus II.

De forma geral, observou-se que com a utilização do ponto fixo é possível obter-se hardware muito menores, em comparação com os gerados com ponto flutuante. Todavia, o hardware com ponto fixo necessita de mais bits na estrutura de dado e, consequentemente, as variáveis de entrada e saída exigirão mais pinos da FPGA. Constatou-se também que, com o ponto flutuante, a implementação mais otimizada aproximou-se do tamanho do hardware do método Min-Max. Entretanto, para o ponto-fixo, o hardware do método Min-Max foi sempre consideravelmente menor. Por fim, verificou-se que o crescimento do hardware do método Min-Max, tanto com ponto flutuante ou com ponto fixo, se deu de forma linear.

Sintetizando todas as informações, pode-se afirmar que o método Min-Max com ponto fixo mostrou ser o algoritmo de localização mais adequado para implementação em FPGA. Porém, deve-se atentar à quantidade de pinos requeridos, ou optar-se por acrescentar um hardware adicional para receber os dados de distâncias serialmente. Finalmente, apesar do chip CC2431 utilizar o algoritmo de melhor desempenho na localização, baseado no MLE, o método Min-Max conseguiu oferecer um desempenho de localização próximo ao do MLE, quando poucos NRs foram considerados, e com a grande vantagem de apresentar um hardware pequeno. Ou seja, a utilização do método Min-Max proporciona bom desempenho na localização e, em eventual implementação em ASIC, ocuparia também pouca área de silício.

5.3 TRABALHOS FUTUROS

Como trabalho futuro pode se propor um estudo mais aprofundado com o determinante de Cayley-Menger para se poder estender o método para a utilização com mais de 3 NRs. Em outra linha de pesquisa, seria desejável verificar, na prática, a viabilidade de se implementar outros tipos de algoritmos, como os algoritmos iterativos de refinamentos de posição, comparando-os com os hardware implementados nesta dissertação.

5.4 PUBLICAÇÃO

O desenvolvimento desta dissertação de mestrado proporcionou, até o momento, a aceitação do seguinte artigo:

- SHIRAI, A. H. et al. Hardware-efficient Location Discovery Based on Received Signal Strength Indication (RSSI). In: **LASCAS 2012**. [S.l.: s.n.], 2012.

REFERÊNCIAS

AAMODT, K. **Application Note AN042 - CC2431 Location Engine**. 2006. Disponível em: <http://www.ti.com/lit/an/swra095/swra095.pdf>. Acesso em: 04 jan. 2012.

ALTERA. **FPGAs**. 2012. Disponível em: <http://www.altera.com/products/fpga.html>. Acesso em: 29 out. 2011.

ALTERA. **Modelsim Simulation Software**. 2012. Disponível em: <http://www.altera.com/education/univ/software/modelsim/unv-modelsim.html>. Acesso em: 17 set. 2012.

AMARAL, J. N.; MEHTA, P. **VHDL Syntax Reference**. 2003. Disponível em: <http://webdocs.cs.ualberta.ca/~amaral/courses/329/labs/VHDL_Reference.html>. Acesso em: 15 ago. 2012.

ANGELIS, A. D. et al. A low-cost ultra-wideband indoor ranging system. **Instrumentation and Measurement, IEEE Transactions on**, v. 58, n. 12, p. 3935–3942, dez. 2009. ISSN 0018-9456.

ARAMPATZIS, T.; LYGEROS, J.; MANESIS, S. A survey of applications of wireless sensors and wireless sensor networks. In: **Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation**. [S.l.: s.n.], 2005. p. 719–724. ISSN 2158-9860.

BASAGNI, S.; CAROSI, A.; PETRIOLI, C. Mobility in wireless sensor networks. In: BOUKERCHE, A. (Ed.). **Algorithms and Protocols for Wireless Sensor Networks**. Hoboken, EUA: Wiley-IEEE Press, 2008. cap. 10, p. 267–305.

BISHOP, D. **Fixed point package user's guide**. 2010. Disponível em: <http://www.vhdl.org/fphdl/Fixed_ug.pdf>. Acesso em: 15 ago. 2012.

BISHOP, D. W. **Floating point package user's guide**. 2009. Disponível em: <http://www.vhdl.org/fphdl/Float_ug.pdf>. Acesso em: 17 ago. 2012.

BISHOP, D. W. **VHDL-2008 Support Library**. 2010. Disponível em: <http://www.eda.org/fphdl>. Acesso em: 28 set. 2012.

BROWN, S.; ROSE, J. Architecture of FPGAs and CPLDs: A tutorial. In: **IEEE Design and Test of Computers**. [S.l.: s.n.], 1996. v. 13, n. 2, p. 42–57.

BROWN, S.; VRANESIC, Z. **Fundamentals of Digital Logic With VHDL Design**. 2. ed. New York: McGraw-Hill, 2000.

BUCHER, R.; MISRA, D. A synthesizable VHDL model of the exact solution for three-dimensional hyperbolic positioning system. **VLSI Design**, v. 15, n. 2, p. 507–520, 2002. Disponível em: <http://journalsonline.tandf.co.uk/Index/10.1080/1065514021000012129>.

CAO, M.; ANDERSON, B. D. O.; MORSE, A. S. Sensor network localization with imprecise distances. **Systems & Control Letters**, v. 55, n. 11, p. 887–893, 2006.

CHANG, C.-H.; LIAO, W. Network localization in partially localizable networks. In: **IEEE International Conference on Communications, 2009. ICC '09.** Dresden, DE: [s.n.], 2009. p. 1–5. ISSN 1938-1883.

CHAROENPANYASAK, S.; SUNTIAMORNTUT, W. The next generation of sensor node in wireless sensor networks. **Journal of Telecommunications**, v. 9, n. 2, p. 6–9, jul. 2011.

CHEN, J. et al. Improved RSSI-based localization algorithm for park lighting control and child location tracking. In: **International Conference on Information and Automation, 2009. ICIA '09.** Zhuhai/Macau, China: [s.n.], 2009. p. 1526–1531.

CHENG, L.; WU, C.-D.; ZHANG, Y.-Z. Indoor robot localization based on wireless sensor networks. **IEEE Transactions on Consumer Electronics**, v. 57, n. 3, p. 1099–1104, ago. 2011. ISSN 0098-3063.

CHIPCON. **Chipcon Introduces CC2431: The World's First System-on-Chip Solution with location estimation capability targeting ZigBee/IEEE 802.15.4 Low Power Wireless Sensor Networks**. 2005. Disponível em: <https://docs.zigbee.org/zigbee-docs/dcn/05-3976.pdf>. Acesso em: 09 jan. 2012.

CHRAIBI, Y. **Localization in Wireless Sensor Networks**. Dissertação (Mestrado) — KTH Vetenskap Och Konst, Stockholm, Suécia, 2005.

CHU, P. **RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability**. Hoboken, EUA: Wiley-IEEE Press, 2006.

DA COSTA, R. B. F. **Estudo e Simulação de Técnicas de Localização de Terminais em Ambientes Microcelulares**. Dissertação (Mestrado em Engenharia Elétrica) — PUC-Rio, Rio de Janeiro, 2003.

DAWSON, M.; WINTERBOTTOM, J.; THOMSON, M. **IP Location**. New York: McGraw-Hill, 2007.

DE LA PIEDRA, A.; BRAEKEN, A.; TOUHAFI, A. Sensor systems based on FPGAs and their applications: A survey. **Sensors**, v. 12, n. 9, p. 12235–12264, 2012. ISSN 1424-8220. Disponível em: <http://www.mdpi.com/1424-8220/12/9/12235>.

DE SAMPAIO, J. E.; XAVIER, C. **Aplicação do Filtro de Kalman na correcção de dados provenientes de um sistema de Localização baseado em RFID**. Dissertação (Mestrado Integrado em Engenharia Electrotécnica e de Computadores) — Faculdade de Engenharia da Universidade do Porto - FEUP, Porto - Portugal, 2011.

DIGI. **How to perform a Range Test**. 2009. Disponível em: <http://ftp1.digi.com/support/documentation/90001067_a.pdf>. Acesso em: 12 maio 2012.

DIGI INTERNATIONAL. **XBee/XBee-PRO ZB RF Modules**. 2010. Disponível em: <http://ftp1.digi.com/support/documentation/90000976_G.pdf>. Acesso em: 18 jul 2012.

FALUDI, R. **Building Wireless Sensor Networks**. Sebastopol, EUA: O'Reilly Media, 2010.

FLOYD, T. **Sistemas Digitais: Fundamentos e Aplicações**. 9. ed. São Paulo: Bookman, 2006.

FÉO, A. E. Resolução de sistemas superdeterminados usando mínimos quadrados e eliminação de Gauss. In: **II Simpósio de Excelência em Gestão e Tecnologia, SEGeT'2005**. Resende: SEGeT, 2005. p. 1139–1144. Disponível em: <http://www.aedb.br/seget/artigos05/16_TP3_SEGet.pdf>.

GARCIA, J. A. **Implementação em FPGA de uma Biblioteca Parametrizável para Inversão de Matrizes Baseada no Algoritmo Gauss-Jordan, usando Representação em Ponto Flutuante**. Dissertação (Mestrado em Sistemas Mecatrônicos) — Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, 2010.

GOLDSMITH, A. **Wireless Communications**. New York, EUA: Cambridge University Press, 2005.

GONÇALVES JÚNIOR, A. A. **EMC 6421 - Fundamentos de Metrologia e Estatística**. 2012. Disponível em: <http://www.labmetro.ufsc.br/Disciplinas/EMC6421>. Acesso em: 31 maio 2012.

GUEDES, E. M. P. **Estudo de Técnica Híbrida de Localização de Estações Móveis baseada em TDoA e AoA**. Dissertação (Mestrado em Engenharia Elétrica) — Instituto Militar de Engenharia, Rio de Janeiro, 2003.

HANZO, L.; BLOGH, J.; NI, S. **3G, HSPA and FDD versus TDD Networking: Smart Antennas and Adaptive Modulation**. West Sussex, England: Wiley-IEEE Press, 2008.

HARA, S. et al. Propagation characteristics of IEEE 802.15.4 radio signal and their application for location estimation. In: **Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st**. [S.l.: s.n.], 2005. v. 1, p. 97–101. ISSN 1550-2252.

HIGHTOWER, J.; BORRIELLO, G. **Location Sensing Techniques**. Washington, EUA, jul. 2001.

IEEE. IEEE standard for binary floating-point arithmetic. **ANSI IEEE Std 754-1985**, p. 20, 1985.

IEEE. IEEE Standard VHDL Language Reference Manual. **ANSI/IEEE Std 1076-1993**, p. i, 1994.

IYENGAR, S. S. et al. **Fundamentals of Sensor Network Programming: Applications and Technology**. New Jersey, EUA: John Wiley & Sons, 2011.

JÚDICE, J. J. **Equações Não Lineares e Optimização Unidimensional**. 2005. Disponível em: <http://www.co.it.pt/~judice/Articles/Optimizacao.pdf>. Acesso em: 10 jul 2012.

JODAS, D. S. **Utilização de FPGA em aplicações de alto desempenho**. 2010. Disponível em: <http://www.dcce.ibilce.unesp.br/~aleardo/cursos/hpc/Danilo.pdf>. Acesso em: 05 nov. 2012.

JUSTICE, R. E. F. **Police Technology - Geographic Information**. 2009. Disponível em: <http://www.hitechcj.com/powerpoints/chapter_five_geographic_information.ppt>. Acesso em: 22 set. 2012.

KARTHICK, N. et al. Location estimation using RSSI and application of extended Kalman filter in wireless sensor networks. In: **Proceedings of the 2009 International Conference on Advanced Computer Control**. Washington, DC, USA: IEEE Computer Society, 2009. p. 337–341. ISBN 978-0-7695-3516-6. Disponível em: <http://dl.acm.org/citation.cfm?id=1510523.1510693>.

KOKS, D. **Numerical calculations for passive geolocation scenarios**. Edinburgh, Austrália, maio 2007.

KOLLÁR, I. **Simulation tools for arbitrary-precision floating-point and fixed-point roundoff**. 2012. Disponível em: <http://oldweb.mit.bme.hu/books/quantization/Matlab-files.html>. Acesso em: 24 fev. 2012.

LEE, Y.-D.; JEONG, D.-U.; LEE, H.-J. Performance analysis of wireless link quality in wireless sensor networks. In: **5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), 2010**. [S.l.: s.n.], 2010. p. 1006 –1010.

LOUREIRO, A. A. F. et al. Redes de sensores sem fio. In: **XXI Simpósio Brasileiro de Redes de Computadores (SBRC'03)**. Natal, RN: [s.n.], 2003. p. 179–226.

LUO, X.; O'BRIEN, W. J.; JULIEN, C. L. Comparative evaluation of Received Signal-Strength Index (RSSI) based indoor localization techniques for construction jobsites. **Advanced Engineering Informatics**, v. 25, n. 2, p. 355–363, 2010. ISSN 1474-0346. ¡ce:title¿Information mining and retrieval in design¡/ce:title¿. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1474034610000984>.

LUTHY, K. A. **Mobile Robotic Navigation and Control for Large-Scale Wireless Sensor Network Repair**. Tese (Doutorado em Engenharia da Computação) — Graduate Faculty of North Carolina State University, Raleigh, North Carolina, EUA, 2009.

MAO, G.; FIDAN, B.; ANDERSON, B. D. O. Wireless sensor network localization techniques. **Computer Networks: The International Journal of Computer and Telecommunications Networking**, Elsevier North-Holland, Inc., New York, NY, USA, v. 51, p. 2529–2553, jul. 2007. ISSN 1389-1286. Disponível em: <http://dl.acm.org/citation.cfm?id=1242848.1243139>.

MARTÍNEZ-ESPINOSA, M.; CALIL JÚNIOR, C.; LAHR, F. A. R. Métodos paramétricos e não-paramétricos para determinar o valor característico em resultados de ensaio de madeira. **Scientia Forestalis**, v. 66, p. 76–83, 2004.

MASIERO, R. **RSSI Based Tracking Algorithms for Wireless Sensor Networks: Theoretical Aspects and Performance Evaluation**. Dissertação (Mestrado) — University of Padova, Padova, Itália, 2007.

MATHEWS, J. H.; FINK, K. K. **Numerical Methods Using MATLAB**. Upper Saddle River, New Jersey, USA: Prentice-Hall, 2004. 430–434 p.

MCCRADY, D. et al. Mobile ranging using low-accuracy clocks. **IEEE Transactions on Microwave Theory and Techniques**, v. 48, n. 6, p. 951–958, jun. 2000. ISSN 0018-9480.

MOLER, C. B. **Numerical Computing with MATLAB**. Philadelphia, EUA: Society for Industrial and Applied Mathematics (SIAM), 2004.

NAKAMURA, E. F.; LOUREIRO, A. A. F.; FRERY, A. C. Information fusion for wireless sensor networks. **ACM Computing Surveys**, ACM, v. 39, p. 9–55, 2007. Disponível em: <http://portal.acm.org/citation.cfm?doid=1267070.1267073>.

NOBLES, P.; ALI, S.; CHIVERS, H. Improved estimation of trilateration distances for indoor wireless intrusion detection. **Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)**, v. 2, n. 1, p. 93–102, mar. 2011.

OTTOY, G. et al. AES data encryption in a ZigBee network: Software or hardware? **Security and Privacy in Mobile Information and Communication Systems**, Springer, p. 163–173, 2010. Disponível em: <http://www.springerlink.com/index/M36206L24616417P.pdf>.

PARANHOS, P. M. **Localização em ambientes externos através da fusão de sensores GPS e inercial por um filtro de Kalman**. Dissertação (Mestrado em Engenharia Mecânica) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2009.

PATWARI, N. et al. Locating the nodes: Cooperative localization in wireless sensor networks. **Signal Processing Magazine, IEEE**, v. 22, n. 4, p. 54–69, jul. 2005. ISSN 1053-5888.

PATWARI, N. et al. Locating the nodes: Cooperative localization in wireless sensor networks. **IEEE SIGNAL PROCESSING MAGAZINE**, v. 22, n. 4, p. 54–69, jul. 2005.

PATWARI, N. et al. Relative location estimation in wireless sensor networks. **IEEE Transactions on Signal Processing**, v. 51, n. 8, p. 2137–2148, ago. 2003. ISSN 1053-587X.

PATWARI, N.; O'DEA, R.; WANG, Y. Relative location in wireless networks. In: **Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd**. [S.l.: s.n.], 2001. v. 2, p. 1149–1153.

PEDRONI, V. A. **Circuit Design and Simulation with VHDL**. 2. ed. Cambridge: MIT Press, 2010.

PINEDO-FRAUSTO, E.; GARCIA-MACIAS, J. An experimental analysis of zigbee networks. In: **Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on**. [S.l.: s.n.], 2008. p. 723 –729.

PORTUGAL, M. S. **Notas Introdutórias Sobre o Princípio de Máxima Verossimilhança: Estimação e Teste de Hipóteses**. 2001. Disponível em: <http://www.ufrgs.br/decon/publionline/textosdidaticos/Textodid04.pdf>. Acesso em: 21 jan. 2012.

RAU, N. **Optimization Principles:Practical Applications to the Operation and Markets of the Electric Power Industry**. Hoboken, EUA: Wiley-IEEE Press, 2003.

REGHELIN, R. **Um Algoritmo Descentralizado de Localização para Rede de Sensores Sem Fio Usando Calibragem Cooperativa e Heurísticas**. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Santa Catarina, Florianópolis, 2007.

ROSELLO, V.; PORTILLA, J.; RIESGO, T. Ultra low power fpga-based architecture for wake-up radio in wireless sensor networks. In: **IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society**. [S.l.: s.n.], 2011. p. 3826–3831. ISSN 1553-572X.

SAHA, A.; MANNA, N. **Digital Principles and Logic Design**. Hingham, EUA: Jones & Bartlett Publishers, 2007.

SAVVIDES, A.; PARK, H.; SRIVASTAVA, M. B. The bits and flops of the n-hop multilateration primitive for node localization problems. In: **Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications WSNA 02**. [S.l.]: ACM Press, 2002. p. 112–121.

SAVVIDES, A.; SRIVASTAVA, M. B. Location discovery. In: BASAGNI, S. et al. (Ed.). **Mobile Ad Hoc Networking**. 1. ed. Hoboken, NJ, EUA: Wiley-IEEE Press, 2004. cap. 8, p. 231–254.

SHIMIZU, Y.; SANADA, Y. Accuracy of relative distance measurement with ultra wideband system. In: **IEEE Conference on Ultra Wideband Systems and Technologies**. Reston, VA, EUA: [s.n.], 2003. p. 374–378.

SHIRAI, A. H. et al. Hardware-efficient location discovery based on received signal strength indication (RSSI). In: **LASCAS 2012**. [S.l.: s.n.], 2012.

SKIBNIEWSKI, M. J.; JANG, W.-S. Simulation of accuracy performance for wireless sensor-based construction asset tracking. **Computer-Aided Civil and Infrastructure Engineering**, Blackwell Publishing Inc, v. 24, n. 5, p. 335–345, 2009. ISSN 1467-8667. Disponível em: <http://dx.doi.org/10.1111/j.1467-8667.2009.00592.x>.

STANFIELD, R. G. Statistical theory of df finding. **Journal of IEE**, v. 94, n. 5, p. 762–770, 1947.

TAKETSUGU, J.; YAMAKITA, J. A modification strategy of maximum likelihood method for location estimation based on received signal strength in sensor networks. **IEICE Trans. Fundam. Electron. Commun. Comput. Sci.**, Oxford University Press, Oxford, UK, E90-A, p. 1093–1104, maio 2007. ISSN 0916-8508. Disponível em: <http://dl.acm.org/citation.cfm?id=1521092.1521120>.

TARAKTAS, K. F.; CEYLAN, O.; YAGCI, B. Received signal strength technique performance in sensor network localization application. In: **Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications**. Washington, DC, USA: IEEE Computer Society, 2010. (BWCCA '10), p. 357–362. ISBN 978-0-7695-4236-2. Disponível em: <http://dx.doi.org/10.1109/BWCCA.2010.96>.

TENNINA, S. et al. Locating ZigBee nodes using the TIs CC2431 location engine: a testbed platform and new solutions for positioning estimation of wsns in dynamic indoor environments. In: **Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments**. New York, NY, USA: ACM, 2008. (MELT '08), p. 37–42. ISBN 978-1-60558-189-7. Disponível em: <http://doi.acm.org/10.1145/1410012.1410022>.

TEXAS INSTRUMENTS. **CC2431: System-on-Chip for 2.4 GHz ZigBee/IEEE 802.15.4 with Location Engine**. 2009. Disponível em: <http://www.ti.com/lit/ds/symlink/cc2431.pdf>. Acesso em: 04 jan. 2012.

THE MATHWORKS. **ecdf - Empirical cumulative distribution function**. 2012. Disponível em: <http://www.mathworks.com/help/toolbox/stats/ecdf.html>. Acesso em: 30 maio 2012.

THE MATHWORKS. **Fixed-Point Toolbox**. 2012. Disponível em: <http://www.mathworks.com/products/fixed>. Acesso em: 15 set. 2012.

THE MATHWORKS. **Simulink - Simulation and Model-Based Design**. 2012. Disponível em: <http://www.mathworks.com/products/simulink>. Acesso em: 29 jul. 2012.

THE MATHWORKS. **Version 7.4 (R2009b) Simulink Software**. 2012. Disponível em: <http://www.mathworks.com/products/simulink>. Acesso em: 11 ago. 2012.

TOMASELLI, L. C. **Controle de um Pré-Regulador com Alto Fator de Potência Utilizando o Controlador DSP TMS320F243**. Dissertação (Mestrado em Engenharia Elétrica) — Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, 2001.

TREVISAN, L. M. **Um Algoritmo de Localização de Nós em Redes Sem-Fio usando Nível de Potência do Sinal**. Dissertação (Mestrado em Informática) — Programa de Pós-Graduação em Informática, Pontifícia Universidade Católica do Paraná, Curitiba, 2009.

VAGHEFI, S. R. M. **Single Node Localization in Wireless Sensor Networks**. Dissertação (Mestrado) — Department of Signals and Systems, Chalmers University of Technology, Gothenburg, 2011.

VIJ, V.; MEHRA, R. FPGA based Kalman filter for wireless sensor networks. **International Journal of Computer Technology and Applications**, v. 2, p. 155–159, 2011. ISSN 22296093. Disponível em: <http://www.doaj.org/doaj?func=abstract&id=686449>.

VURAN, M. C.; POMPILI, D.; MELODIA, T. Future trends in wireless sensor networks. In: ZHENG, J.; JAMALIPOUR, A. (Ed.). **Wireless Sensor Networks:A Networking Perspective**. Hoboken, EUA: Wiley-IEEE Press, 2009. cap. 14, p. 433–470.

WANG, H. et al. Dynamic association in IEEE 802.11 based wireless mesh networks. In: **6th International Symposium on Wireless Communication Systems, 2009. ISWCS 2009.** [S.l.: s.n.], 2009. p. 81–85.

WELCH, G.; BISHOP, G. **An Introduction to the Kalman Filter**. 2006. Disponível em: <http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html>. Acesso em: 24 nov. 2012.

WIDROW, B.; KOLLÁR, I. **Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications**. Cambridge, UK: Cambridge University Press, 2008. 711–719 p. ISBN 9780521886710.

WILLIAMS, G. Overdetermined systems of linear equations. **The American Mathematical Monthly**, Mathematical Association of America, v. 97, n. 6, p. 511–513, 1990. ISSN 00029890. Disponível em: <http://www.jstor.org/stable/2323837>.

WILLRICH, R. **Sistemas Numéricos e a Representação Interna dos Dados no Computador**. 2002. Disponível em: <http://www.inf.ufsc.br/~willrich/Ensino/INE5602/restrito/ii-cap2.PDF>. Acesso em: 18 ago. 2012.

XU, N. et al. Performance assessment of multilateration systems - a solution to nextgen surveillance. In: **Integrated Communications Navigation and Surveillance Conference (ICNS), 2010**. [S.l.: s.n.], 2010. p. D2–1 –D2–8. ISSN 2155-4943.

ZANCA, G. et al. **RSSI measurements for indoor WSN**. 2012. Disponível em: <http://telecom.dei.unipd.it/pages/read/59>. Acesso em: 03 fev. 2012.

ZANCA, G. et al. Experimental comparison of RSSI-based localization algorithms for indoor wireless sensor networks. In: **Proceedings of the workshop on Real-world wireless sensor networks**. New York, NY, USA: ACM, 2008. (REALWSN '08), p. 1–5. ISBN 978-1-60558-123-1. Disponível em: <http://doi.acm.org/10.1145/1435473.1435475>.

ZEMEK, R. et al. RSSI-based localization without a prior knowledge of channel model parameters. **International Journal of Wireless Information Networks**, Springer Netherlands, v. 15, p. 128–136, 2008. ISSN 1068-9605. 10.1007/s10776-008-0085-6. Disponível em: <http://dx.doi.org/10.1007/s10776-008-0085-6>.

ZHANG, R.-B. et al. Environmental-adaptive indoor radio path loss model for wireless sensor networks localization. **AEU - International Journal of Electronics and Communications**, v. 65, n. 12, p. 1023–1031, 2011. ISSN 1434-8411. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1434841111000859>.

ZHOU, G. et al. Models and solutions for radio irregularity in wireless sensor networks. **ACM Trans. Sen. Netw.**, ACM, New York, NY, USA, v. 2, n. 2, p. 221–262, maio 2006. ISSN 1550-4859. Disponível em: <http://doi.acm.org/10.1145/1149283.1149287>.

APÊNDICE A – CÓDIGOS MATLAB

```
1  function run_experiment()
2      clear all;
3      close all;
4      rand('twister', 1);
5
6      % Tenta usar vários processadores
7      if matlabpool('size') == 0
8          matlabpool open
9      end
10
11     iterations_count = 1E4;
12     radius = 10;
13
14     for refCount = 3:35
15         % Calcula o erro do algoritmo de localização
16         error = calculate_error(iterations_count, radius, refCount);
17         mean_error = sum(error)/iterations_count;
18         strDisp = sprintf('RefCount = %d; MeanError = %e', refCount, mean_error);
19         disp(strDisp);
20
21         % Salva as variáveis
22         fileName = sprintf('%dNRs_LS', refCount);
23         %fileName = sprintf('%dNRs_Min_Max', refCount);
24         %fileName = sprintf('%dNRs_MLE', refCount);
25         save(fileName, 'mean_error', 'error');
26     end
27 end
28
29 function distances = calculate_error(iterations_count, radius, refCount)
30     variance = 10;
31     distances = zeros(iterations_count,1);
32
33     parfor i = 1:iterations_count
34         [bn_real, Refs, Dist] = setup_points(radius, refCount);
```

```
35
36          % Contamina os valores de distâncias com ruído AWGN
37          % (a distância resultante deve ser maior ou igual a zero)
38          noisyDist = Dist + sqrt(variance)*randn(refCount, 1);
39          noisyDist = (noisyDist & noisyDist > 0).*noisyDist;
40
41          % Estima a posição do blind node
42          bn_est = find_point_by_least_squares(Refs, noisyDist, bn_real);
43          %bn_est = find_point_by_min_max(Refs, noisyDist, bn_real);
44          %bn_est = find_point_by_mle(Refs, noisyDist, bn_real);
45
46          % Calcula o erro
47          distances(i) = pdist([bn_real; bn_est]);
48      end
49  end
```

```
1   function point = rand_point(radius_max)
2       radius = radius_max*rand();
3       angle = 2*pi*rand();
4       x = radius*cos(angle);
5       y = radius*sin(angle);
6       point = [x y];
7   end
```

```
1   function [bn, Refs, Dist] = setup_points(radius, refCount)
2       % O blind node será sorteado aleatoriamento dentro da circunferência
3       bn = rand_point(radius);
4
5       % Ref. nodes serão posicionados uniformemente sobre o círculo com
6       % diâmetro igual ao Range. Matematicamente x^2 + y^2 = (Range/2)^2
7       % Dessa forma, garante-se que os pontos nunca serão colineares
8       Refs = zeros(refCount,2);
9       Dist = zeros(refCount,1);
10      for i = 1:refCount
11          Refs(i,1) = radius*cos(i*2*pi/refCount);
12          Refs(i,2) = radius*sin(i*2*pi/refCount);
13          Dist(i) = pdist([Refs(i,1) Refs(i,2); bn]);
14      end;
15  end
```

```
1   function bn_est = find_point_by_trilateration(Refs, Dist)
2       X = [Refs(1,1) Refs(2,1) Refs(3,1)];
3       Y = [Refs(1,2) Refs(2,2) Refs(3,2)];
4       D = [Dist(1) Dist(2) Dist(3)];
```

```
5
6       K1 = X(1)^2 - X(3)^2 + Y(1)^2 - Y(3)^2 + D(3)^2 - D(1)^2;
7       K2 = X(2)^2 - X(3)^2 + Y(2)^2 - Y(3)^2 + D(3)^2 - D(2)^2;
8       K3 = 2*( (Y(1)-Y(3))*(X(2)-X(3)) - (X(1)-X(3))*(Y(2)-Y(3)) );
9
10      xbn = -(Y(2)-Y(3))*K1 + (Y(1)-Y(3))*K2;
11      xbn = xbn/K3;
12
13      ybn = (X(2)-X(3))*K1 - (X(1)-X(3))*K2;
14      ybn = ybn/K3;
15      bn_est = [xbn ybn];
16  end
```

```
1   function bn_est = find_point_by_least_squares(Refs, Dist)
2       % x = [x1 x2 ... xn]
3       % y = [y1 y2 ... yn]
4       % d = [d1 d2 ... dn]
5
6       x = Refs(:,1);
7       y = Refs(:,2);
8       d = Dist;
9
10      N = length(x);
11      A = zeros(N-1, 2);
12      b = zeros(N-1, 1);
13      for (i = 1:N-1)
14          A(i,1) = 2*(x(i) - x(end));
15          A(i,2) = 2*(y(i) - y(end));
16          b(i) = x(i)^2 - x(end)^2 + y(i)^2 - y(end)^2 + d(end)^2 - d(i)^2;
17      end;
18
19      X = ((A.'*A)^-1) * A.' * b;
20      bn_est = [X(1) X(2)];
21  end
```

```
1   function bn_est = find_point_by_min_max(Refs, Dist)
2       Xref = Refs(:,1);
3       Yref = Refs(:,2);
4       Dref = Dist;
5
6       Xil = Xref - Dref;
7       Xir = Xref + Dref;
8       Yib = Yref - Dref;
9       Yit = Yref + Dref;
```

```
10
11      xc = (max(Xil) + min(Xir))/2;
12      yc = (max(Yib) + min(Yit))/2;
13      bn_est = [xc yc];
14  end
```

```
1   function bn_est = find_point_by_mle(Refs, Dist, bn_real)
2       % X: Abscissa dos nós de referência
3       % Y: Ordenada dos nós de referência
4       % D: Valor das distâncias estimadas
5       X = Refs(:,1);
6       Y = Refs(:,2);
7       D = Dist;
8
9       % Escreve a função anônima em string (strFunc)
10      strFunc = sprintf('@(x) ');
11
12      for i = 1:length(X) - 1
13          aux = sprintf('log( %e^2 / ( (x(1)-%e)^2 + (x(2)-%e)^2) )^2 + ', ...
14              D(i), X(i), Y(i));
15          strFunc = strcat(strFunc, aux);
16      end
17      aux = sprintf('log( %e^2 / ( (x(1)-%e)^2 + (x(2)-%e)^2) )^2', ...
18          D(end), X(end), Y(end));
19      strFunc = strcat(strFunc, aux);
20
21      % Converte a string na função anônima correspondente
22      func = str2func(strFunc);
23
24      options = optimset;
25      options.MaxFunEvals = 1E5;
26      options.MaxIter = 1E5;
27      options.TolX = 1E-8;
28      options.TolFun = 1E-8;
29      options.Display = 'off';
30
31      bn_est = fminsearch(func, bn_real, options);
32  end
```

```
1   function run_experiment_cayley_menger()
2       clear all;
3       close all;
4       rand('twister', 1);
5
```

```
6       % Tenta usar vários processadores
7       if matlabpool('size') == 0
8           matlabpool open
9       end
10
11      iterations_count = 1E4;
12      radius = 10;
13      refCount = 3;
14
15      for variance = 1:12
16          % Calcula o erro do algoritmo de localização
17          [error_cm, error] = calculate_error(iterations_count, ...
18              radius, refCount, variance);
19          mean_error_cm = sum(error_cm)/iterations_count;
20          mean_error = sum(error)/iterations_count;
21          strDisp = sprintf('Variance = %d; MeanErrorCM = %e; MeanError = %e', ...
22              variance, mean_error_cm, mean_error);
23          disp(strDisp);
24
25          % Salva variáveis
26          fileName = sprintf('Variance%d_LS', variance);
27          %fileName = sprintf('Variance%d_Min_Max', variance);
28          %fileName = sprintf('Variance%d_MLE', variance);
29          save(fileName, 'mean_error_cm', 'mean_error', 'error_cm', 'error');
30      end
31  end
32
33  function [distances_cm, distances] = calculate_error( ...
34  iterations_count, radius, refCount, variance)
35
36      distances_cm = zeros(iterations_count,1);
37      distances = zeros(iterations_count,1);
38
39      parfor i = 1:iterations_count
40          [bn_real, Refs, Dist] = setup_points(radius, refCount);
41
42          % Contamina os valores de distâncias com ruído AWGN
43          noisyDist = Dist + sqrt(variance)*randn(refCount, 1);
44          noisyDist = (noisyDist & noisyDist > 0).*noisyDist;
45
46          % Melhora a estimativa da distância através
47          % do método de Cayley-Menger
48          smoothDist = cayley_menger_filter(Refs, noisyDist);
```

```
49
50          % Estima a posição do blind node com oa valores de distância
51          % melhorados e com os valores de distância sem aplicar o
52          % método de Cayley Menger (CM)
53          bn_est_cm = find_point_by_least_squares(Refs, smoothDist);
54          bn_est = find_point_by_least_squares(Refs, noisyDist);
55          %bn_est_cm = find_point_by_min_max(Refs, smoothDist');
56          %bn_est = find_point_by_min_max(Refs, noisyDist);
57          %bn_est_cm = find_point_by_mle(Refs, smoothDist, bn_real);
58          %bn_est = find_point_by_mle(Refs, noisyDist, bn_real);
59
60          % Calcula o erro
61          distances_cm(i) = pdist([bn_real; bn_est_cm]);
62          distances(i) = pdist([bn_real; bn_est]);
63      end
64  end
```

```
1   function Dout = cayley_menger_filter(Ref, Din)
2       % Todas as distâncias são elevadas ao quadrado agora
3       % para não poluir o código com x^2
4       d12 = pdist([Ref(1,:); Ref(2,:)])^2;
5       d13 = pdist([Ref(1,:); Ref(3,:)])^2;
6       d23 = pdist([Ref(2,:); Ref(3,:)])^2;
7       d01 = Din(1)^2;
8       d02 = Din(2)^2;
9       d03 = Din(3)^2;
10
11      A = zeros(3,3);
12      A(1,1) = 2*d23;
13      A(1,2) = d12 - d13 - d23;
14      A(1,3) = d13 - d23 - d12;
15      A(2,1) = d12 - d13 - d23;
16      A(2,2) = 2*d13;
17      A(2,3) = d23 - d12 - d13;
18      A(3,1) = d13 - d12 - d23;
19      A(3,2) = d23 - d12 - d13;
20      A(3,3) = 2*d12;
21
22      % A sequência "..." quebra linha
23      b = zeros(1,3);
24      b(1) = 4*d23*d01 + 2*(d12 - d13 - d23)*d02 ...
25          + 2*(d13 - d12 - d23)*d03 ...
26          + 2*d23*(d23 - d12 - d13);
```

```
27      b(2) = 4*d13*d02 + 2*(d12 - d13 - d23)*d01 ...
28          + 2*(d23 - d12 - d13)*d03 ...
29          + 2*d13*(d13 - d12 - d23);
30      b(3) = 4*d12*d03 + 2*(d13 - d12 - d23)*d01 ...
31          + 2*(d23 - d12 - d13)*d02 ...
32          + 2*d12*(d12 - d13 - d23);
33
34      c = 2*d12*d13*d23 + 2*d23*d01^2 + 2*d13*d02^2 + 2*d12*d03^2 ...
35          + 2*(d12 - d13 - d23)*d01*d02 + 2*(d13 - d12 - d23)*d01*d03 ...
36          + 2*(d23 - d12 - d13)*d02*d03 + 2*d23*(d23 - d12 - d13)*d01 ...
37          + 2*d13*(d13 - d12 - d23)*d02 + 2*d12*(d12 - d13 - d23)*d03;
38
39      cte = [A(1,1) A(1,2) A(1,3) A(2,1) A(2,2) A(2,3) ...
40          A(3,1) A(3,2) A(3,3) b(1) b(2) b(3) c];
41      x0 = [0 0 0 0];
42      options = optimset;
43      options.MaxFunEvals = 1E7;
44      options.MaxIter = 1E7;
45      options.TolX = 1E-8;
46      options.TolFun = 1E-8;
47      options.Display = 'off';
48      res = fsolve(@(x) ErrorFunction(x, cte), x0, options);
49
50      Dout = zeros(1,3);
51      Dout(1) = sqrt(Din(1)^2 + res(1));
52      Dout(2) = sqrt(Din(2)^2 + res(2));
53      Dout(3) = sqrt(Din(3)^2 + res(3));
54  end
```

```
1   function Qfloat = setup_floating_point(exponentBits, fractionBits)
2       expBias = (2^exponentBits)/2 - 1;
3       expMax = (2^exponentBits - 1 - expBias) - 1;
4       expMin = 1 - expBias;
5
6       Qfloat = qmake('name','Qfloat','type','floating-point', ...
7           'precision',sprintf('%d',fractionBits), ...
8           'expbias',sprintf('%d',expBias), ...
9           'emax',sprintf('%d',expMax), ...
10          'emin',sprintf('%d',expMin), ...
11          'overflow', 'clip' );
12  end
```

```
1   function Qfixed = setup_fixed_point(integerBits, fractionBits)
2       % Supoe-se codificacao em complemento de 2
```

```
3       totalBits = integerBits + fractionBits;
4       Qfixed = qmake('name','Qfixed','type','fixed-point', ...
5           'bits',sprintf('%d',totalBits), ...
6           'fractionbits',sprintf('%d',fractionBits) );
7   end
```

```
1   function bn_est = find_point_by_trilateration_quantized(Refs, Dist, Qfloat)
2       % Configura as variaveis com a estrutura de dado fornecida
3       X = qdata(Refs(:,1), Qfloat);
4       Y = qdata(Refs(:,2), Qfloat);
5       D = qdata(Dist, Qfloat);
6
7       K1 = X(1)*X(1) - X(3)*X(3) + Y(1)*Y(1) - Y(3)*Y(3) + D(3)*D(3) - D(1)*D(1);
8       K2 = X(2)*X(2) - X(3)*X(3) + Y(2)*Y(2) - Y(3)*Y(3) + D(3)*D(3) - D(2)*D(2);
9       K3 = 2*( (Y(1)-Y(3))*(X(2)-X(3)) - (X(1)-X(3))*(Y(2)-Y(3)) );
10
11      xbn = (Y(3)-Y(2))*K1 - (Y(3)-Y(1))*K2;
12      xbn = xbn/K3;
13
14      ybn = (X(2)-X(3))*K1 - (X(1)-X(3))*K2;
15      ybn = ybn/K3;
16      bn_est = [xbn.data(1,:) ybn.data(1,:)];
17  end
```

```
1   function bn_est = find_point_by_min_max_quantized(Refs, Dist, Qfloat)
2       % Configura as variaveis com a estrutura de dado fornecida
3       Xref = qdata(Refs(:,1), Qfloat);
4       Yref = qdata(Refs(:,2), Qfloat);
5       Dref = qdata(Dist, Qfloat);
6
7       Xil = Xref - Dref;
8       Xir = Xref + Dref;
9       Yib = Yref - Dref;
10      Yit = Yref + Dref;
11
12      % Use as linhas abaixo caso deseje verificar os bits do resultado
13      [Xil_max, Xil_max_i] = max(Xil.data(1,:));
14      [Xir_min, Xir_min_i] = min(Xir.data(1,:));
15      [Yib_max, Yib_max_i] = max(Yib.data(1,:));
16      [Yit_min, Yit_min_i] = min(Yit.data(1,:));
17
18      xc_bin = (Xil(Xil_max_i) + Xir(Xir_min_i))/2;
19      yc_bin = (Yib(Yib_max_i) + Yit(Yit_min_i))/2;
20      bn_est = [xc_bin.data yc_bin.data];
```

```
21
22      % Use as linhas abaixo caso nao deseje verificar os bits do resultado
23      %xc = (max(Xil.data(1,:)) + min(Xir.data(1,:)))/2;
24      %yc = (max(Yib.data(1,:)) + min(Yit.data(1,:)))/2;
25      %bn_est = [xc yc];
26  end

 1  function run_experiment_quantization()
 2      % Adiciona o caminho para a biblioteca
 3      addpath(strcat(pwd,'\qdsp'), '-end');
 4
 5      clear all;
 6      close all;
 7      rand('twister', 1);
 8
 9      iterations_count = 1000;
10      radius = 10;
11      refCount_min = 3;
12      refCount_max = 10;
13      exp_bits_min = 3;
14      exp_bits_max = 7;
15      %int_bits_min = 3;
16      %int_bits_max = 10;
17      fr_bits_min = 3;
18      fr_bits_max = 10;
19
20      % [refCount, exp_bits, fr_bits]
21      error_matrix = zeros(refCount_max - refCount_min + 1, ...
22          exp_bits_max - exp_bits_min + 1, fr_bits_max - fr_bits_min + 1);
23
24      for refCount = refCount_min:refCount_max
25          for exp_bits = exp_bits_min : exp_bits_max
26              for fr_bits = fr_bits_min : fr_bits_max
27                  % Gera uma estrutura de ponto-fixo
28                  %Qfixed = setup_fixed_point(int_bits, fr_bits)
29
30                  % Gera uma estrutura de ponto-flutuante
31                  % (considerando hidden bit)
32                  Qfloat = setup_floating_point(exp_bits, fr_bits + 1);
33
34                  error = calculate_error(iterations_count, radius, ...
35                      refCount, Qfloat);
36                  mean_error = sum(error)/iterations_count;
```

```
37
38                  error_matrix(refCount - refCount_min + 1, ...
39                      exp_bits - exp_bits_min + 1, ...
40                      fr_bits - fr_bits_min + 1) = mean_error;
41
42                  strDisp = sprintf('%d:%d,%d = %f', refCount, exp_bits, ...
43                      fr_bits, mean_error);
44                  disp(strDisp);
45              end
46          end
47      end
48
49      % Salva variáveis
50      save('error_matrix.mat', 'error_matrix');
51  end
52
53  % Calcula o erro, em distancia
54  function distances = calculate_error(iterations_count, radius, refCount, Qfloat)
55      distances = zeros(iterations_count, 1);
56
57      for i = 1:iterations_count
58          % Gera um ponto de teste
59          [bn_real, Refs, Dist] = setup_points(radius, refCount);
60
61          % Método da Lateracao
62          %bn_est = find_point_by_trilateration_quantized(Refs, Dist, Qfloat);
63          %distances(i) = pdist([bn_real; bn_est]);
64
65          % Método Min-Max
66          bn_est = find_point_by_min_max_quantized(Refs, Dist, Qfloat);
67          bn_est_double = find_point_by_min_max(Refs, Dist);
68
69          distances(i) = pdist([bn_est; bn_est_double]);
70      end
71  end
```

APÊNDICE B – CÓDIGOS VHDL

```
1  library ieee_proposed;
2  use ieee_proposed.float_pkg.all;
3  use ieee_proposed.fixed_pkg.all;
4  use work.fixed_float_types.all;
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  package rssi_loc_types is
9     constant EXPONENT_WIDTH: integer := 5;
10    constant FRACTION_WIDTH: integer := 6;
11    constant INTEGER_WIDTH: integer := 14;
12
13    -- Floating Point functions --
14    subtype my_float is float( EXPONENT_WIDTH downto -FRACTION_WIDTH );
15    type my_float_vector is array(natural range <>) of my_float;
16    function to_my_float(r: real) return my_float;
17
18    function find_max(vector: my_float_vector) return my_float;
19    function find_min(vector: my_float_vector) return my_float;
20
21    function "+" (L: my_float_vector; R: my_float_vector) return my_float_vector;
22    function "-" (L: my_float_vector; R: my_float_vector) return my_float_vector;
23    function "-" (L: my_float_vector; R: my_float) return my_float_vector;
24    function "/" (L: my_float_vector; R: my_float_vector) return my_float_vector;
25
26    -- Fixed Point functions --
27    subtype my_fixed is sfixed( INTEGER_WIDTH - 1 downto -FRACTION_WIDTH );
28    type my_fixed_vector is array(natural range <>) of my_fixed;
29    function to_my_fixed(r: real) return my_fixed;
30
31    function find_max(vector: my_fixed_vector) return my_fixed;
32    function find_min(vector: my_fixed_vector) return my_fixed;
33
34    function "+" (L: my_fixed_vector; R: my_fixed_vector) return my_fixed_vector;
```

```
35     function "-" (L: my_fixed_vector; R: my_fixed_vector) return my_fixed_vector;
36     function "-" (L: my_fixed_vector; R: my_fixed) return my_fixed_vector;
37     function "/" (L: my_fixed_vector; R: my_fixed_vector) return my_fixed_vector;
38
39 end package;
40
41 package body rssi_loc_types is
42 ------------------------------
43 -- Floating Point functions --
44 ------------------------------
45    -- Converte um valor "real" no formato de ponto flutuante solicitado
46     function to_my_float(r: real) return my_float is begin
47        return to_float( r, EXPONENT_WIDTH, FRACTION_WIDTH );
48     end function;
49
50     function "+" (L: my_float_vector; R: my_float_vector) return my_float_vector is
51        variable ret_vector: my_float_vector(L'range);
52     begin
53        for i in L'range loop
54           ret_vector(i) := L(i) + R(i);
55        end loop;
56        return ret_vector;
57     end function;
58
59     function "-" (L: my_float_vector; R: my_float_vector) return my_float_vector is
60        variable ret_vector: my_float_vector(L'range);
61     begin
62        for i in L'range loop
63           ret_vector(i) := L(i) - R(i);
64        end loop;
65        return ret_vector;
66     end function;
67
68     function "-" (L: my_float_vector; R: my_float) return my_float_vector is
69        variable ret_vector: my_float_vector(L'range);
70     begin
71        for i in L'range loop
72           ret_vector(i) := L(i) - R;
73        end loop;
74        return ret_vector;
75     end function;
76
77     function "/" (L: my_float_vector; R: my_float_vector) return my_float_vector is
```

```
78        variable ret_vector: my_float_vector(L'range);
79    begin
80        for i in L'range loop
81            ret_vector(i) := L(i) / R(i);
82        end loop;
83        return ret_vector;
84    end function;
85
86    function find_max(vector: my_float_vector) return my_float is
87        -- inicializa valor de retorno com primeiro elemento do vetor
88        variable max : my_float := vector( vector'left );
89    begin
90        -- percorre vetor a partir do segundo elemento
91        for i in vector'left + 1 to vector'right loop
92            if vector(i) > max then
93                max := vector(i);
94            end if;
95        end loop;
96        return max;
97    end function;
98
99    function find_min(vector: my_float_vector) return my_float is
100       variable min : my_float := vector( vector'left );
101   begin
102       for i in vector'left + 1 to vector'right loop
103           if vector(i) < min then
104               min := vector(i);
105           end if;
106       end loop;
107       return min;
108   end function;
109
110 ------------------------------
111 -- Fixed Point functions --
112 ------------------------------
113   -- Converte um valor "real" no formato de ponto flutuante solicitado
114   function to_my_fixed(r: real) return my_fixed is begin
115       return to_sfixed( r, INTEGER_WIDTH - 1, -FRACTION_WIDTH );
116   end function;
117
118   function "+" (L: my_fixed_vector; R: my_fixed_vector) return my_fixed_vector is
119       variable ret_vector: my_fixed_vector(L'range);
120   begin
```

```
121       for i in L'range loop
122          ret_vector(i) := resize( arg => L(i) + R(i),
123             left_index     => L(i)'high,
124             right_index    => L(i)'low,
125             round_style    => fixed_truncate,
126             overflow_style => fixed_saturate);
127       end loop;
128       return ret_vector;
129    end function;
130
131    function "-" (L: my_fixed_vector; R: my_fixed_vector) return my_fixed_vector is
132       variable ret_vector: my_fixed_vector(L'range);
133    begin
134       for i in L'range loop
135          ret_vector(i) := resize( arg => L(i) - R(i),
136             left_index     => L(i)'high,
137             right_index    => L(i)'low,
138             round_style    => fixed_truncate,
139             overflow_style => fixed_saturate);
140       end loop;
141       return ret_vector;
142    end function;
143
144    function "-" (L: my_fixed_vector; R: my_fixed) return my_fixed_vector is
145       variable ret_vector: my_fixed_vector(L'range);
146    begin
147       for i in L'range loop
148          ret_vector(i) := resize( arg => L(i) - R,
149             left_index     => L(i)'high,
150             right_index    => L(i)'low,
151             round_style    => fixed_truncate,
152             overflow_style => fixed_saturate);
153       end loop;
154       return ret_vector;
155    end function;
156
157    function "/" (L: my_fixed_vector; R: my_fixed_vector) return my_fixed_vector is
158       variable ret_vector: my_fixed_vector(L'range);
159    begin
160       for i in L'range loop
161          ret_vector(i) := resize( arg => L(i) / R(i),
162             left_index     => L(i)'high,
163             right_index    => L(i)'low,
```

```
164               round_style    => fixed_truncate,
165               overflow_style => fixed_saturate);
166         end loop;
167         return ret_vector;
168     end function;
169
170     function find_max(vector: my_fixed_vector) return my_fixed is
171         -- inicializa valor de retorno com primeiro elemento do vetor
172         variable max : my_fixed := vector( vector'left );
173     begin
174         -- percorre vetor a partir do segundo elemento
175         for i in vector'left + 1 to vector'right loop
176             if vector(i) > max then
177                 max := vector(i);
178             end if;
179         end loop;
180         return max;
181     end function;
182
183     function find_min(vector: my_fixed_vector) return my_fixed is
184         variable min : my_fixed := vector( vector'left );
185     begin
186         for i in vector'left + 1 to vector'right loop
187             if vector(i) < min then
188                 min := vector(i);
189             end if;
190         end loop;
191         return min;
192     end function;
193
194 end package body;
```

```
1  ------------------------------------------------------------
2  library ieee_proposed;
3  use ieee_proposed.float_pkg.all;
4  use work.rssi_loc_types.all;
5  ------------------------------------------------------------
6  entity trilateration IS
7      port (
8          d1, d2, d3: in my_float;
9          x1, x2, x3: in my_float;
10         y1, y2, y3: in my_float;
11         xbn, ybn: out my_float
```

```
12      );
13  end entity;
14  ----------------------------------------------------------------
15  architecture localization of trilateration is
16      signal k1, k2, k3: my_float;
17  begin
18
19      k1 <= (x1 * x1) - (x3 * x3) + (y1 * y1)
20            - (y3 * y3) + (d3 * d3) - (d1 * d1);
21      k2 <= (x2 * x2) - (x3 * x3) + (y2 * y2)
22            - (y3 * y3) + (d3 * d3) - (d2 * d2);
23      k3 <= 2 *( (y1-y3) * (x2-x3) - (x1-x3) * (y2-y3) );
24
25      xbn <= ((y3-y2) * k1 - (y3-y1) * k2)/k3;
26      ybn <= ((x2-x3) * k1 - (x1-x3) * k2)/k3;
27  end architecture;
28  ----------------------------------------------------------------
```

```
1   ----------------------------------------------------------------
2   library ieee_proposed;
3   use ieee_proposed.float_pkg.all;
4   use work.rssi_loc_types.all;
5   library ieee;
6   use ieee.std_logic_1164.all;
7   ----------------------------------------------------------------
8   entity fsm4_calc_coord is
9       port (
10          d1, d2, d3: in my_float;
11          x1, x2, x3: in my_float;
12          y1, y2, y3: in my_float;
13          clock: in std_logic;
14          reset: in std_logic;
15          xbn, ybn: out my_float
16      );
17  end entity;
18  ----------------------------------------------------------------
19  architecture calc of fsm4_calc_coord is
20      constant const: my_float := to_my_float(0.0);
21
22      signal calc_k_a, calc_k_b,
23          calc_k_c, calc_k_result: my_float;
24      signal calc_xy_a, calc_xy_b, calc_xy_c,
25          calc_xy_d, calc_xy_result: my_float;
```

```
26      signal k1, k2, k3: my_float;
27
28      type state is (CalcK1, CalcK2, CalcX, CalcY);
29      signal pr_state, nx_state: state := CalcK1;
30  begin
31
32      calc_k: entity work.fsm4_calc_k1_k2 port map(
33          a=>calc_k_a, b=>calc_k_b, c=>calc_k_c,
34          x3=>x3, y3=>y3, d3=>d3, result=>calc_k_result
35      );
36
37      calc_xy: entity work.fsm4_calc_x_y port map(
38          a=>calc_xy_a, b=>calc_xy_b, c=>calc_xy_c,
39          d=>calc_xy_d, k1=>k1, k2=>k2, k3=>k3,
40          result=>calc_xy_result
41      );
42
43      -- Parte sequencial da máquina de estados
44      process(clock, reset)
45      begin
46          if(reset='1') then
47              pr_state <= CalcK1;
48          elsif(clock'event and clock='1') then
49
50              case pr_state is
51                  when CalcK1 =>
52                      k1 <= calc_k_result;
53                  when CalcK2 =>
54                      k2 <= calc_k_result;
55                  when CalcX =>
56                      xbn <= calc_xy_result;
57                  when CalcY =>
58                      ybn <= calc_xy_result;
59              end case;
60              pr_state <= nx_state;
61
62          end if;
63      end process;
64
65      -- Parte combinacional da máquina de estados
66      process (pr_state, x1, x2, x3, y1, y2, y3, d1, d2, d3)
67      begin
68          case pr_state is
```

```
69              when CalcK1 =>
70                  calc_k_a <= x1;
71                  calc_k_b <= y1;
72                  calc_k_c <= d1;
73                  calc_xy_a <= const;
74                  calc_xy_b <= const;
75                  calc_xy_c <= const;
76                  calc_xy_d <= const;
77                  nx_state <= CalcK2;
78              when CalcK2 =>
79                  calc_k_a <= x2;
80                  calc_k_b <= y2;
81                  calc_k_c <= d2;
82                  calc_xy_a <= const;
83                  calc_xy_b <= const;
84                  calc_xy_c <= const;
85                  calc_xy_d <= const;
86                  nx_state <= CalcX;
87              when CalcX =>
88                  calc_k_a <= const;
89                  calc_k_b <= const;
90                  calc_k_c <= const;
91                  calc_xy_a <= y3;
92                  calc_xy_b <= y2;
93                  calc_xy_c <= y3;
94                  calc_xy_d <= y1;
95                  nx_state <= CalcY;
96              when CalcY =>
97                  calc_k_a <= const;
98                  calc_k_b <= const;
99                  calc_k_c <= const;
100                 calc_xy_a <= x2;
101                 calc_xy_b <= x3;
102                 calc_xy_c <= x1;
103                 calc_xy_d <= x3;
104                 nx_state <= CalcK1;
105         end case;
106
107         k3 <= 2 *( (y1-y3) * (x2-x3) - (x1-x3) * (y2-y3) );
108
109     end process;
110 end architecture;
111 ------------------------------------------------------------------------
```

```
1  ----------------------------------------------------
2  library ieee_proposed;
3  use ieee_proposed.float_pkg.all;
4  use work.rssi_loc_types.all;
5  library ieee;
6  use ieee.std_logic_1164.all;
7  ----------------------------------------------------
8  entity fsm4_calc_k1_k2 is
9     port (
10       a, b, c: in my_float;
11       x3, y3, d3: in my_float;
12       result: out my_float
13    );
14 end entity;
15 ----------------------------------------------------
16 architecture calc of fsm4_calc_k1_k2 is
17 begin
18    result <= (a * a) - (x3 * x3) + (b * b) -
19       (y3 * y3) + (d3 * d3) - (c * c);
20 end architecture;
21 ----------------------------------------------------
```

```
1  ----------------------------------------------
2  library ieee_proposed;
3  use ieee_proposed.float_pkg.all;
4  use work.rssi_loc_types.all;
5  library ieee;
6  use ieee.std_logic_1164.all;
7  ----------------------------------------------
8  entity fsm4_calc_x_y is
9     port (
10       a,  b, c, d: in my_float;
11       k1, k2, k3: in my_float;
12       result: out my_float
13    );
14 end entity;
15 ----------------------------------------------
16 architecture calc of fsm4_calc_x_y is
17 begin
18     result <= ((a-b) * k1 - (c-d) * k2)/k3;
19 end architecture;
20 ----------------------------------------------
```

```
1  ------------------------------------------------------------------------
```

```
2  library ieee_proposed;
3  use ieee_proposed.float_pkg.all;
4  use work.rssi_loc_types.all;
5  library ieee;
6  use ieee.std_logic_1164.all;
7  ----------------------------------------------------------------------
8  entity fsm17_calc_coord is
9     port (
10       d1, d2, d3: in my_float;
11       x1, x2, x3: in my_float;
12       y1, y2, y3: in my_float;
13       clock: in std_logic;
14       reset: in std_logic;
15       xbn, ybn: out my_float
16    );
17 end entity;
18 ----------------------------------------------------------------------
19 architecture calc of fsm17_calc_coord is
20    constant const: my_float := to_my_float(1.0);
21
22    signal mult_a, mult_b, res_mult: my_float;
23    signal div_a, div_b, res_div: my_float;
24    signal temp1, temp2, temp3, temp4, temp5: my_float;
25    signal k1, k2, k3: my_float;
26
27    type state is (S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12,
28       S13, S14, S15, S16, S17);
29    signal pr_state, nx_state: state := S1;
30 begin
31
32    calc_k: entity work.fsm17_divider port map(
33       a=>div_a, b=>div_b, result=>res_div
34    );
35
36    calc_xy: entity work.fsm17_multiplier port map(
37       a=>mult_a, b=>mult_b, result=>res_mult
38    );
39
40    -- Parte Sequencial da Máquina de Estados
41    process(clock, reset)
42    begin
43       if(reset='1') then
44          pr_state <= S1;
```

```
45          elsif(clock'event and clock='1') then
46             -- Registrando os valores que serão utilizados futuramente
47             case pr_state is
48                when S1 =>
49                   temp1 <= res_mult;
50                when S2 =>
51                   temp2 <= res_mult;
52                when S3 =>
53                   temp3 <= res_mult;
54                when S4 =>
55                   temp4 <= res_mult;
56                when S5 =>
57                   temp5 <= res_mult;
58                when S6 =>
59                   k1 <= temp1 - temp2 + temp3
60                         - temp4 + temp5 - res_mult;
61                when S7 =>
62                   temp1 <= res_mult;
63                when S8 =>
64                   temp3 <= res_mult;
65                when S9 =>
66                   k2 <= temp1 - temp2 + temp3
67                         - temp4 + temp5 - res_mult;
68                when S10 =>
69                   temp1 <= res_mult;
70                when S11 =>
71                   temp2 <= res_mult;
72                when S12 =>
73                   k3 <= res_mult;
74                when S13 =>
75                   temp1 <= res_mult;
76                when S14 =>
77                   temp2 <= res_mult;
78                when S15 =>
79                   temp1 <= res_mult;
80                   xbn <= res_div;
81                when S16 =>
82                   temp2 <= res_mult;
83                when S17 =>
84                   ybn <= res_div;
85             end case;
86
87             pr_state <= nx_state;
```

```
88
89          end if;
90      end process;
91
92      -- Parte Combinacional da Máquina de Estados
93      process(pr_state, x1, x2, x3, y1, y2, y3, d1, d2, d3)
94      begin
95          case pr_state is
96              when S1 =>
97                  mult_a <= x1;
98                  mult_b <= x1;
99                  div_a <= const;
100                 div_b <= const;
101                 nx_state <= S2;
102             when S2 =>
103                 mult_a <= x3;
104                 mult_b <= x3;
105                 div_a <= const;
106                 div_b <= const;
107                 nx_state <= S3;
108             when S3 =>
109                 mult_a <= y1;
110                 mult_b <= y1;
111                 div_a <= const;
112                 div_b <= const;
113                 nx_state <= S4;
114             when S4 =>
115                 mult_a <= y3;
116                 mult_b <= y3;
117                 div_a <= const;
118                 div_b <= const;
119                 nx_state <= S5;
120             when S5 =>
121                 mult_a <= d3;
122                 mult_b <= d3;
123                 div_a <= const;
124                 div_b <= const;
125                 nx_state <= S6;
126             when S6 =>
127                 mult_a <= d1;
128                 mult_b <= d1;
129                 div_a <= const;
130                 div_b <= const;
```

```
131             nx_state <= S7;
132         when S7 =>
133             mult_a <= x2;
134             mult_b <= x2;
135             div_a <= const;
136             div_b <= const;
137             nx_state <= S8;
138         when S8 =>
139             mult_a <= y2;
140             mult_b <= y2;
141             div_a <= const;
142             div_b <= const;
143             nx_state <= S9;
144         when S9 =>
145             mult_a <= d2;
146             mult_b <= d2;
147             div_a <= const;
148             div_b <= const;
149             nx_state <= S10;
150         when S10 =>
151             mult_a <= y1-y3;
152             mult_b <= x2-x3;
153             div_a <= const;
154             div_b <= const;
155             nx_state <= S11;
156         when S11 =>
157             mult_a <= x1-x3;
158             mult_b <= y2-y3;
159             div_a <= const;
160             div_b <= const;
161             nx_state <= S12;
162         when S12 =>
163             mult_a <= to_my_float(2.0);
164             mult_b <= temp1-temp2;
165             div_a <= const;
166             div_b <= const;
167             nx_state <= S13;
168         when S13 =>
169             mult_a <= y3-y2;
170             mult_b <= k1;
171             div_a <= const;
172             div_b <= const;
173             nx_state <= S14;
```

```
174             when S14 =>
175                 mult_a <= y3-y1;
176                 mult_b <= k2;
177                 div_a <= const;
178                 div_b <= const;
179                 nx_state <= S15;
180             when S15 =>
181                 mult_a <= x2-x3;
182                 mult_b <= k1;
183                 div_a <= temp1-temp2;
184                 div_b <= k3;
185                 nx_state <= S16;
186             when S16 =>
187                 mult_a <= x1-x3;
188                 mult_b <= k2;
189                 div_a <= const;
190                 div_b <= const;
191                 nx_state <= S17;
192             when S17 =>
193                 mult_a <= const;
194                 mult_b <= const;
195                 div_a <= temp1-temp2;
196                 div_b <= k3;
197                 nx_state <= S1;
198         end case;
199     end process;
200 end architecture;
201 ------------------------------------------------------------------------------
```

```
1  ----------------------------------------
2  library ieee_proposed;
3  use ieee_proposed.float_pkg.all;
4  use work.rssi_loc_types.all;
5  library ieee;
6  use ieee.std_logic_1164.all;
7  ----------------------------------------
8  entity fsm17_divider is
9      port (
10         a, b: in my_float;
11         result: out my_float
12     );
13 end entity;
14 ----------------------------------------
```

```
15 architecture calc of fsm17_divider is
16 begin
17   result <= a / b;
18 end architecture;
19 ----------------------------------------
```

```
1 ----------------------------------------
2 library ieee_proposed;
3 use ieee_proposed.float_pkg.all;
4 use work.rssi_loc_types.all;
5 library ieee;
6 use ieee.std_logic_1164.all;
7 ----------------------------------------
8 entity fsm17_multiplier is
9    port (
10      a, b: in my_float;
11      result: out my_float
12   );
13 end entity;
14 ----------------------------------------
15 architecture calc of fsm17_multiplier is
16 begin
17    result <= a * b;
18 end architecture;
19 ----------------------------------------
```

```
1 ------------------------------------------------------------
2 library ieee_proposed;
3 use ieee_proposed.float_pkg.all;
4 use work.rssi_loc_types.all;
5 ------------------------------------------------------------
6 entity min_max_method is
7    generic (
8      NREF: integer range 0 to 2 := 2
9    );
10   port (
11      D: in my_float_vector(0 to NREF);
12      X: in my_float_vector(0 to NREF);
13      Y: in my_float_vector(0 to NREF);
14      xbn, ybn: out my_float
15   );
16 end entity;
17 ------------------------------------------------------------
18 architecture algorithm of min_max_method is
```

```
19 begin
20    process (D, X, Y)
21       variable iLeft, iRight, iBottom, iTop: my_float;
22       variable xc, yc: my_float;
23    begin
24       -- encontra a interseção dos quadrados circunscritos
25       -- às distâncias
26       iLeft := find_max(X-D);
27       iRight := find_min(X+D);
28       iBottom := find_max(Y-D);
29       iTop := find_min(Y+D);
30
31       -- determina o centroide da interseção
32       xbn <= (iLeft + iRight) / 2;
33       ybn <= (iBottom + iTop) / 2;
34    end process;
35 end architecture;
36 ------------------------------------------------------------------
```